Universidade do Minho
Escola de Engenharia

João Nuno Cardoso Gonçalves de Abreu

Development of DNA sequence classifiers based on deep learning

October, 2022

Universidade do Minho
Escola de Engenharia

João Nuno Cardoso Gonçalves de Abreu

Development of DNA sequence classifiers based on deep learning

Master's Dissertation
Master's in Informatics Engineering

Work supervised by
Miguel Rocha
Óscar Dias

October, 2022

Acknowledgements

To begin with, I would like to thank my supervisor, Prof. Miguel Rocha for giving me this opportunity and for all the support he provided whenever I needed it. Then, I would also like to thank Ana Marta Sequeira for the continuous support and guidance throughout the whole dissertation, as well as the patience for all my questions. This project was only possible with their help, and I'm forever grateful for it.

I would also like to thank the colleagues from OmniumAI, namely Fernando Cruz, João Capela Ribeiro and Miguel Barros for the huge help regarding the new technologies I had to learn and the challenges I had to face.

Then, I want to thank my parents for providing me all the opportunities and everything I have ever asked for. Thank you for all the support and encouragement throughout my whole life, but especially this last year.

I also want to thank my friends who were present in my life during these last 5 years of university, especially Hugo, Tiago, and Duarte. Thank you all for the amazing moments we have spent together and for being the best friends I have ever had.

Lastly, I would like to thank all the family, friends, and colleagues I did not mention here, but who were present in my life during these last few years. Thank you all for the support and encouragement you have provided me.

Abstract

Development of DNA sequence classifiers based on deep learning

Deoxyribonucleic acid (DNA) is a biological macromolecule whose primary function is to store an individual's genetic information. Because of breakthroughs in sequencing technology, the number of DNA sequences is now growing at an exponential rate. The assignment of a function to these sequences is a great obstacle in Bioinformatics, and current methods rely on homologies, a solution that is slow and less accurate. Machine learning (ML) has been widely employed as it is a relevant tool for processing huge amounts of data by learning on its own without explicit programming. Using ML, it is now possible to speed up and automatically classify DNA sequences into existing categories with the objective of learning their functions.

However, building a machine learning classifier of biological sequences is a tough challenge due to the lack of numerical properties in the sequence that the model requires. Therefore, it is still necessary to apply some pre-processing techniques so that the sequences are properly represented for the model. These techniques include feature extraction and feature selection, and they are the most difficult components because sequences lack explicit features. Deep learning models have recently been developed that not only extract features from input automatically, but also improve the prediction and classification of DNA sequences.

The main goal of this project is to create a tool that can automatically classify DNA sequences using machine and deep learning models and algorithms, followed by its integration into *ProPythia*, a Python package developed by the host group. Automated ML classifiers will also be developed to integrate in *OmniumAI* software platforms. Transcription factor annotation and essential gene determination will be used as case studies for the platform validation. With this study, it is intended to encourage the use of such technologies to develop new tools that can manage vast volumes of biological data, thus boosting DNA prediction understanding.

Keywords: DNA, DNA Sequence Classification, Machine Learning, Deep Learning

Resumo

Desenvolvimento de classificadores de sequências de ADN baseado em deep learning

O ácido desoxirribonucleico (ADN) é uma macromolécula biológica cuja principal função é armazenar a informação genética de um indivíduo. Devido aos avanços na tecnologia de sequenciamento, o número dessas sequências está a crescer a uma taxa exponencial. A atribuição de funções a estas sequências é um grande obstáculo na Bioinformática, e os métodos atuais usam homologias, uma solução lenta e pouco precisa. Machine learning tem sido bastante utilizado, pois é uma ferramenta capaz de processar grandes quantidades de dados aprendendo por conta própria sem programação explícita. Desta maneira, é possível acelerar e classificar automaticamente as sequências de ADN em categorias existentes com o objetivo de aprender as suas funções.

No entanto, construir um classificador de machine learning de sequências biológicas é um grande desafio devido à falta de propriedades numéricas na sequência que o modelo exige. É necessário aplicar algumas técnicas de pré-processamento para que as sequências sejam devidamente representadas para o modelo. Essas técnicas incluem extração e seleção de características, e são os componentes mais difíceis porque as sequências carecem de características explícitas. Modelos de deep learning foram desenvolvidos recentemente que não só extraem características dos dados automaticamente, como também melhoram a previsão e classificação de sequências de ADN.

O principal objetivo deste projeto é criar uma ferramenta capaz de classificar automaticamente sequências de ADN usando modelos e algoritmos de machine e deep learning, seguido da sua integração no *ProPythia*, um Python package desenvolvido pelo grupo anfitrião. Classificadores automáticos de machine learning também serão desenvolvidos para integração em plataformas de software *OmniumAI*. A determinação do fator de transcrição e de genes essenciais serão utilizados como casos de estudo para validação da plataforma. Com este estudo, pretende-se incentivar o uso de tais tecnologias para desenvolver novas ferramentas que consigam lidar com grandes volumes de dados, permitindo avanços na área de previsão de ADN.

Palavras-chave: ADN, Classificação de sequências de ADN, Machine Learning, Deep Learning

Contents

List of Figures

List of Tables

Glossary

genome	An organism's complete set of genetic instructions. Each genome contains all of the information needed to build that organism and allow it to grow and develop.

Acronyms

A	Adenine
ANF	Accumulated Nucleotide Frequency
ANN	Artificial Neural Network
AT	Adenine-Thymine
AutoML	Automated Machine Learning
BPTT	Backpropagation through time
C	Cytosine
CGR	Chaos Game Representation
CKSNAP	Composition of K-spaced Nucleic Acid Pairs
CNN	Convolutional Neural Network
DAC	Dinucleotide-based Auto Covariance
DACC	Dinucleotide-based Auto-Cross Covariance
DBS	DNA-binding domains
DCC	Dinucleotide-based Cross Covariance
DL	Deep Learning
DNA	Deoxyribonucleic acid
DNC	Di-Nucleotide Composition
DNN	Deep Neural Network
DT	Decision Tree
FCGR	Frequency Chaos Game Representation
FN	False Negative
FP	False Positive

G	Guanine
GC	Guanine-Cytosine
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
KNN	K-Nearest Neighbors
LgR	Logistic Regression
LR	Linear Regression
LSTM	Long Short-Term Memory
MAE	Mean absolute error
MAPE	Mean absolute percentage error
ML	Machine Learning
MLE	Maximum Likelihood Estimation
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NAC	Nucleic Acid Composition
NLP	Natural Language Processing
PCA	Principal Component Analysis
PseACC	Pseudo Amino Acid Composition
PseDNC	Pseudo Dinucleotide Composition
PseKNC	Pseudo K-Tupler Composition
PseNAC	Pseudo Nucleic Acid Composition
RCKmer	Reverse Compliment Kmer
RF	Random Forest
RNA	Ribonucleic acid
RNN	Recurrent Neural Network

SGD	Stochastic Gradient Descent
SVM	Support Vector Machines
T	Thymine
TAC	Trinucleotide-based Auto Covariance
TACC	Trinucleotide-based Auto-Cross Covariance
TCC	Trinucleotide-based Cross Covariance
TF	Transcription Factor
TN	True Negative
TNC	Tri-Nucleotide Composition
TP	True Positive

Chapter 1

Introduction

1.1 Context and Motivation

Biomedical data has grown at an exponential rate in recent years, requiring the use of a variety of machine learning approaches to handle new issues in biology and clinical research. Machine learning methods are often integrated with bioinformatics methodologies, as well as curated databases and biological networks, to improve training and validation, find the most interpretable features, and enable feature and model research [1].

In organisms, DNA is a biomacromolecule. It holds life's genetic information and controls biological growth as well as the proper functioning of life's functions. Machine learning is now frequently utilized in sequence data analysis, and it has a wide range of applications in terms of enhancing data processing capacities and providing useful biological data [2].

The assignment of a function to a sequence representing a part of a DNA molecule is a core problem in Bioinformatics, extremely important for biomedical research. Current solutions involve the use of homologies, inferred by sequence similarity, i.e., classifying new sequences based on known functions in sequences with a high degree of similarity. Deep neural networks are an alternative that can automatically learn and comprehend informative sequence representations to get a better understanding of the regulatory code that governs gene expression [3].

There is already a platform, developed within the Biosystems group at CEB/ U. Minho, devoted to the classification of peptides/proteins sequences using machine learning and deep learning called *ProPythia* [4]. One of the objectives of this thesis will be the integration of a tool to support DNA sequence classifiers on the mentioned platform.

1.2 Research Objectives

The main aim of this work is to develop an automatic classification system for DNA sequences using machine and deep learning algorithms, expecting performance gains in terms of response time to annotate large numbers of sequences (e.g., complete genomes), as well as the accuracy of the results obtained.

In detail, the work will address the following scientific/technological objectives:

- Review relevant literature and existing tools regarding deep learning methods and their applications in sequence classification.
- Develop and compare data pre-processing techniques and understand the impact of different methods on the classifying performance of machine and deep learning models.
- Develop a tool to support machine and deep learning models for DNA sequence classification to integrate in *ProPythia*.
- Develop automated machine learning (AutoML) classifiers for DNA sequences, which will be integrated into *OmniumAI* software platforms.
- Validate the developed tool and platform with case studies in the areas of biotechnology and health, e.g., transcription factor annotation and essential genes determination [5–7].
- Write the master thesis.

1.3 Document Structure

This thesis is divided into seven chapters, each of which is briefly described as follows:

- Chapter 1: Overview of the work's subject, as well as the motivation and key objectives.
- Chapter 2: Theoretical concepts of machine and deep learning, as well as their objectives, algorithms, workflows, and architectures.
- Chapter 3: Introduction of DNA and DNA sequence classification topics. Applications of machine and deep learning in DNA sequence classification, as well as relevant previous work.
- Chapter 4: Decisions and methods for implementing the proposed work, including details on the feature extraction and classification models used.
- Chapter 5: Integration of the developed tool into *ProPythia* and *OmniumAI* software platforms.

- Chapter 6: A summary of the case study datasets, how they were acquired, and the tool's effectiveness on them.

- Chapter 7: Brief summary of the dissertation, followed by a discussion of the goals, which are presented in this chapter, and the results from the previous chapter. Also provided at the end are some suggestions for future work.

Chapter 2

Machine and Deep Learning

The modern world is overflowing with data. It has reached a stage where humans can no longer regulate it since the rate of analysis is far slower than the continuous growth of data. According to Figure 1, this rapid growth is not expected to stop anytime soon, so tools and technologies are needed to make the process of making sense of data more efficient.

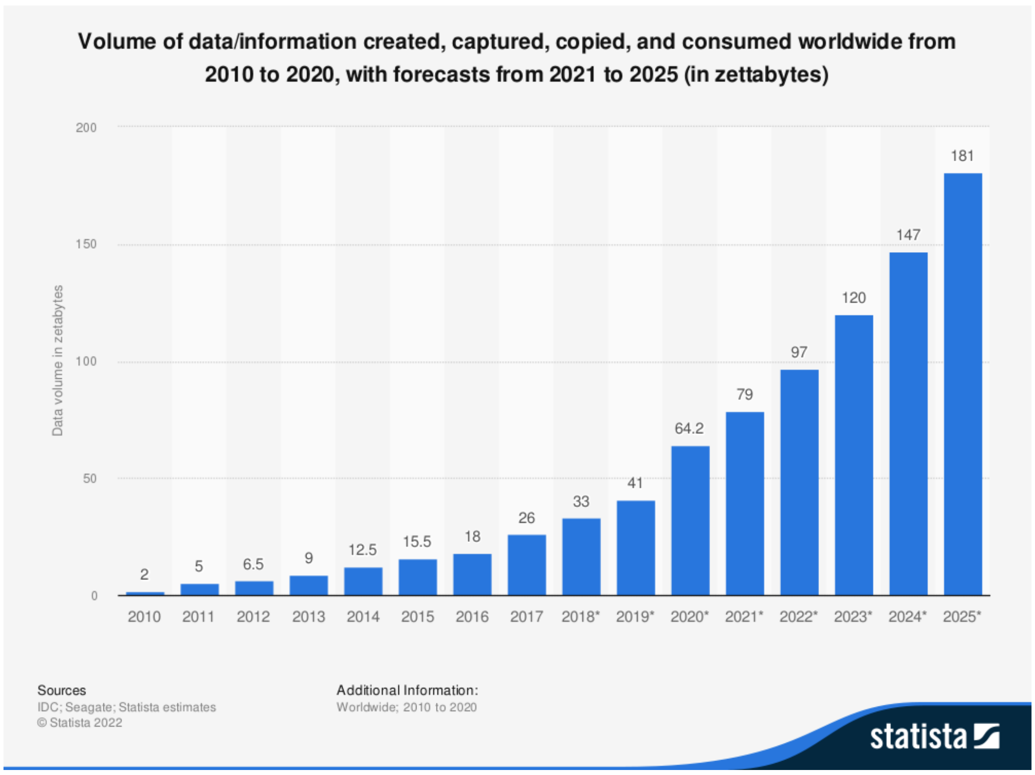


Figure 1: The volume of data created, captured, copied, and consumed from 2010 to 2025 [8]

Machine Learning (ML), which is a subfield of artificial intelligence and computer science, is a promise that humans will be able to extract useful information from all these data. It focuses on using data and algorithms to imitate the way humans learn while improving accuracy [9].

For a long time, one of the major differences between humans and computers has been that humans tend to naturally improve their approach to solving problems by learning from their mistakes and trying to fix them. Traditional computer programs are unable to improve their behavior since they do not consider the outcome of their job [10].

This topic is addressed by ML, which entails the development of computer systems that can learn and improve their performance by accumulating more data and experience. A. Samuel was the first scientist to design a self-learning program in 1952 when he developed a program that improved at playing checkers as the number of games increased [10, 11].

ML relies solely on the availability of the data and does not need any rule-based programming. There is a distinction to be made between traditional programming and ML. In traditional programming, data and programs are sent as inputs to the machine, and it produces an output, whereas in ML, data and outputs are inputs to the system, and the machine's output is the program that has been learned to make predictions on unknown examples. The primary difference between traditional programming and ML's approach is represented in Figure 2.

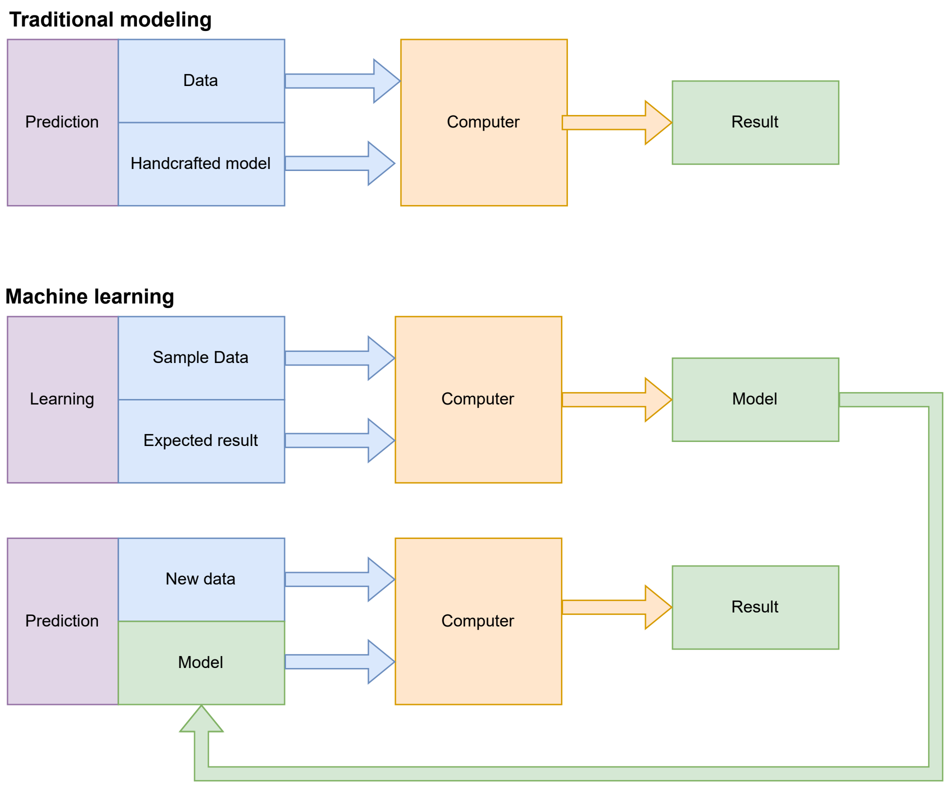


Figure 2: Difference between Traditional Programming and Machine Learning. Adapted from [12]

To understand better the concepts of ML, Table 1 provides a few important terminologies.

Table 1: Machine Learning concepts [13]

Concept	Description
Dataset	Collection of data. In tabular data, each column represents a feature and each row represents a given record of the data set in question. Instead of tables, datasets can also consist of a collection of files.
Model	Representation of a ML system after it has learnt from the training data.
Feature	Measurable characteristic of the dataset.
Feature Vector	Multiple features are used as an input to the ML model.
Training	Procedure for obtaining appropriate values for model weights and bias (parameters).
Parameter	Model variable that is self-taught by the ML system.
Prediction	Once the ML model is complete, it can be fed input data to accurately predict.
Label	Value that the ML model must predict.
Overfitting	Making a model that is so similar to the training data that it fails to generate accurate predictions on new data.
Underfitting	The model fails to detect the underlying trend in the input data.

The two primary categories of ML algorithms are supervised and unsupervised learning. Other categories include semi-supervised learning and reinforcement learning. The following sections provide an overview of the two primary categories, describing their properties and explaining their algorithms. The supervised learning section is more detailed as it is more relevant to the purpose of this thesis.

2.1 Unsupervised learning

In unsupervised learning, algorithms are used when the data used in the training process is not categorized. Although they cannot figure out the proper output, they can infer a function to identify trends or hidden structures from unlabeled data in the dataset [14]. The two most common unsupervised categories are clustering and dimensionality reduction. Clustering involves grouping input variables with similar qualities, and it is applied in targetted marketing problems and recommender systems [15]. Dimensionality reduction algorithms are techniques that reduce the number of input variables in a dataset, and they are used for big data visualization and structure discovery [16].

In unsupervised ML, several algorithms and computing approaches are utilized. The following are some of the most popular clustering and dimensionality reduction algorithms: K-means clustering and Principal Component Analysis (PCA) [18].

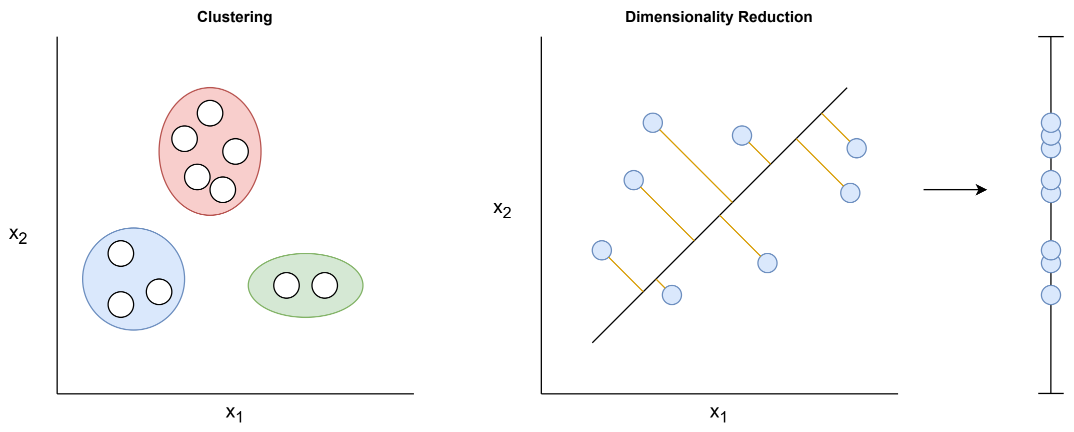


Figure 3: Clustering vs Dimensionality Reduction. Adapted from [17]

K-means clustering is a clustering ML technique in which data points are divided into K groups. The data points nearest to a certain centroid will be clustered together. Smaller groupings with more granularity are indicated by a higher K value, whereas bigger groupings with less granularity are indicated by a lower K value [19]. Figure 4 provides a visual representation of K-means clustering, with $K = 3$.

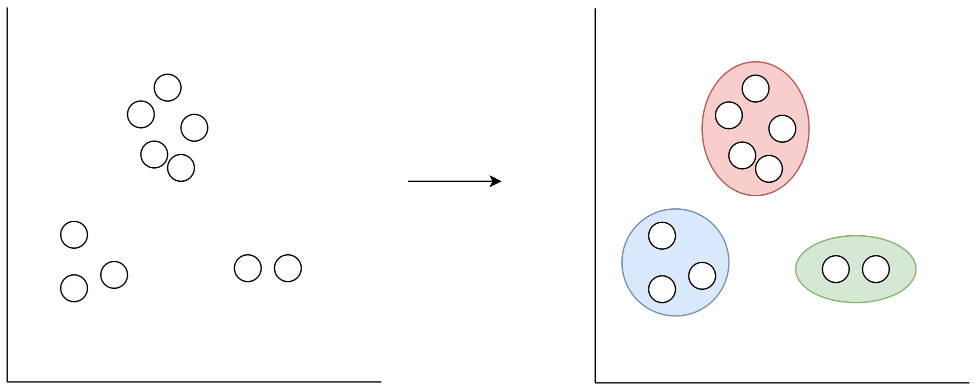


Figure 4: Visual representation of K-means clustering. Adapted from [20]

PCA is a dimensionality reduction approach that uses feature extraction to eliminate redundancies and compress datasets, while retaining as much of the information contained in the original data as possible [19]. Working with too many variables can be difficult for ML since there is a chance of overfitting, a lack of appropriate data for each variable, and a degree of correlation between each variable and the output [18]. In order to do this, PCA projects the data into a lower-dimensional subspace that mostly retains the variance between the data points.. Figure 5 depicts an example of PCA's influence on 2-dimensional space data.

The green line was created through mathematical optimization in order to maximize the variance between the data points as much as possible along that line. This line is referred to as the first principal component. Since a dimension has been lost to separate them, the points on the line are closer to each other than they were in the original 2D environment. However, in many circumstances, the simplification

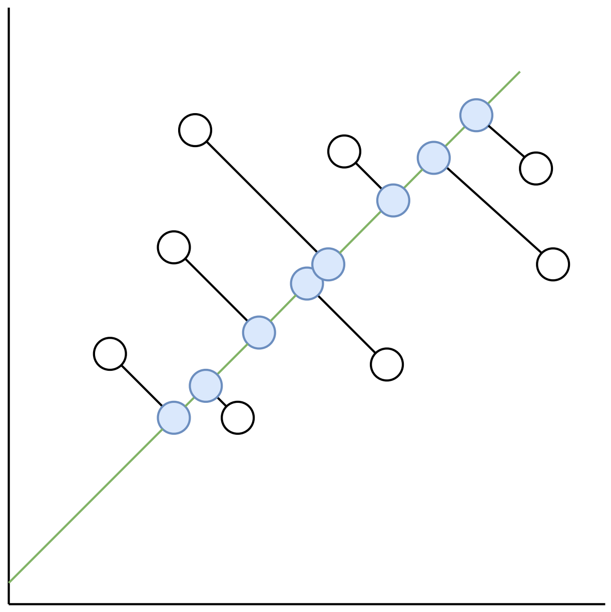


Figure 5: Principal Component Analysis application on 2D space data. Adapted from [21]

in dimensionality compensates the loss of information. When moving to higher dimensions, it will most likely be necessary to use multiple principal components since the variance described by one principle component will not be enough. Principal components are vectors that form a 90-degree angle between each other (orthogonal vectors), and are independent in a way that the second principal component does not overlap with the variance explained by the first. The first principal component will capture the majority of the variance; the second will catch the second-largest portion of the variance left unexplained by the first, and so on.

2.2 Supervised learning

Supervised learning is a ML paradigm for obtaining knowledge about a system's input-output relationship from a set of paired input-output training examples [22], with classification and regression being the two most common supervised categories. In classification, a class label is predicted for a given sample. In other words, it maps a function from input variables to output variables as target, label or categories [23]. In addition, there are multiple classification problems, such as binary classification, which refers to tasks with two class labels, such as ”true and false”, multiclass classification, which refers to classification tasks having more than two class labels, and multi-label classification when an example is associated with multiple classes or labels. On the other hand, regression contains approaches for predicting a continuous output variable based on the value of one or more predictor variables. The key difference between classification and regression is that the former predicts labels for certain classes while the latter permits the prediction of a continuous variable. A clearer distinction between classification and regression is illustrated in Figure 6.

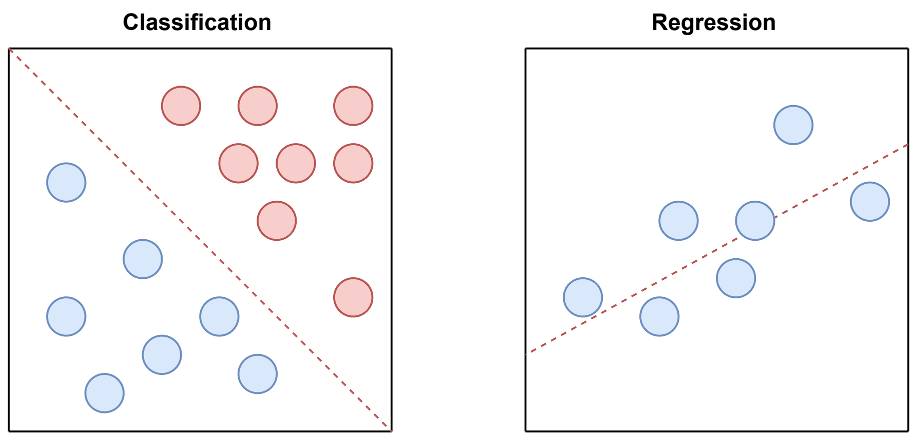


Figure 6: Classification vs Regression. Adapted from [24]

While there are several supervised learning algorithms available, most of them follow the same fundamental steps for producing a predictor model. The next section describes the general workflow process of building a supervised machine learning project.

2.2.1 Workflow

ML workflows specify which phases of a ML project are implemented. While these measures are widely acknowledged as best practices, there is still potential for improvement.

When developing a ML workflow, the first step is to define the project before determining the best working strategy or attempting to fit the model into a predetermined workflow. Instead, a flexible workflow should be created to start small and work its way up to a production-ready solution.

The steps taken during a ML implementation are defined by workflows. ML workflows differ depending on the project, but they usually consist of seven steps.

1. **Data Gathering** It is the practice of acquiring and analyzing data from a variety of sources. Data must be collected and kept in a form that makes sense for the challenge at hand in order to be used to build a viable machine learning model. This phase is crucial because the quality and quantity of data collected will directly affect how accurate the predictive model is. Publicly available datasets are frequently the best source of data, and websites like *Kaggle* provide an enormous quantity of huge datasets. Working with these selected datasets reduces the time and effort required to begin a ML project.

   Structured, semi-structured, and unstructured data are examples of different types of data, and Table 2 provides details about each one of them.

2. **Data pre-processing** Cleaning, validating, and converting data into a usable dataset, are all part of pre-processing. This may be a simple operation if the data was collected from a single source. If

Table 2: Types of data. Adapted from [23]

Data type	Description	Examples
Structured	Well-defined structure, well-organized and accessible. Often stored in a tabular manner such as relational databases.	names, addresses, credit card numbers
Unstructured	Since there is no pre-defined format or organization, it is significantly more difficult to acquire, handle, and analyze data that is largely text and multimedia.	emails, PDF files, audio files, videos, photos
Semi-structured	Contains organizational qualities that make it easier to examine.	HTML, XML, JSON documents

not, the data format must match between the different sources and be equally credible without any potential duplicates. The majority of real-world data is disorganized; examples include:

- **Missing data**: when it is not created continuously or when there are technical issues with the application.
- **Noisy data**: also known as outliers, this can be caused by human error (manually obtaining data) or a technical issue with the device at the time of data collection.
- **Inconsistent data**: This type of data may be gathered as a result of human error (mistakes in names or values) or data duplication.

And there are types of raw data too, including:

- **Numeric**: height, weight, age, IQ.
- **Categorical**: race, sex, nationality.
- **Ordinal**: low/medium/high, education level (”high school”, ”BS”, ”MS”, ”PhD”).

However, there is an important aspect of ML models as they can only handle numeric features. As a result, all types of data must be converted into numeric features. This process of transforming raw data, such as images or text, into suitable modelling features is called feature extraction.

Most of the time, datasets have an excessive number of features that are not required for the predictive model. In fact, removing irrelevant features and keeping the sufficient and essential ones can help reduce the ML model training time, as well as reduce overfit and improve accuracy. This filtering process is called feature selection and is usually performed after feature extraction.

3. **Splitting the Data** It is usual to divide a dataset into two portions for creating ML models: training and testing. The training set is used to estimate the model’s parameters. The accuracy of the model

is then tested using the test dataset. This dataset splitting process is done to prevent overfitting. If the entire dataset was used for training, then the model would overfit the data, meaning it would fail to generate accurate predictions on unseen data [25].

The simplest and most popular approach for dividing such a dataset is to randomly sample a portion of it. For example, 80% of the dataset's rows can be randomly selected for training, while the remaining 20% can be utilized for testing [25].

It's also typical to save a part of the training set for validation purposes. The validation set may be used to fine-tune the model's performance by selecting hyper-parameters (constant parameter whose value is determined before the learning process) and regularization parameters [25].

4. **Building the model** The key goal now is to choose the type of model which fits better the desired problem.

   Data can be any of the types listed above (Table 2), and they can differ from one application to the next in the real world. Therefore, different types of ML approaches can be used to evaluate a specific problem field and extract insights or usable knowledge from the data to construct real-world intelligent systems. One of the models described in Section 2.2.2 that best suits the problem's goal should be chosen.

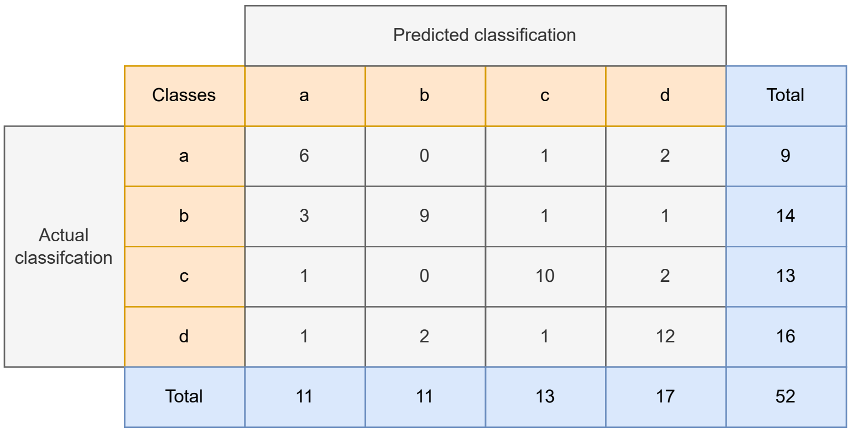
5. **Training and evaluation** The objective now is to use the pre-processed data to train the best-performing chosen model, followed by its performance evaluation. The training set will be used to train the model and then the model outputs will be compared with the unseen values. To evaluate these outputs, multiple metrics are used which can vary depending on the problem.

   For binary classification problems, the most commonly metrics include: confusion matrices, accuracy, recall, precision and f1-score [26]. These are based on True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) values, which are the four possible predictions outcomes for a classification problem. TP outcomes are correct predictions of the positive class, while TN still are correct predictions but of the negative class. FP outcomes are incorrect predictions of the positive class, while FN still are incorrect predictions but of the negative class.

   The confusion matrices are tables which rows represent the real classes and the columns represent the predicted classes. So, for each class, the table shows how many predictions were and were not correct. Figure 7 shows an example of a confusion matrix and Figure 8 illustrates how the the TP, TN, FP and FN values are perceived within the table.

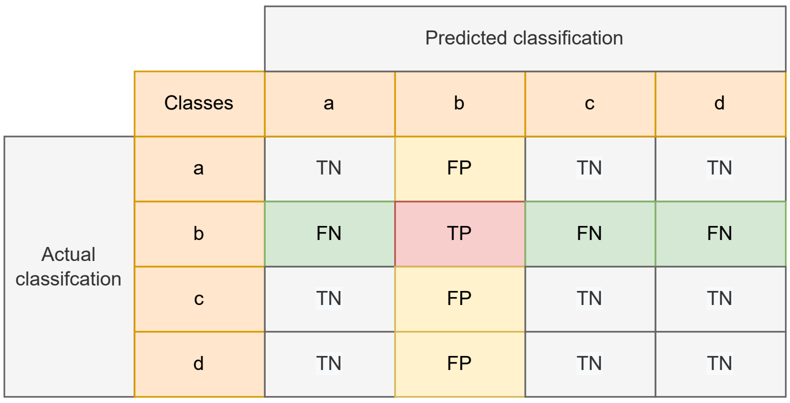
   After defining TP, TN, FP and FN values, the other classification metrics are simple to calculate. Table 3 explains each classification metric and how each one is calculated.

   For regression problems, the most commonly metrics include: Mean Squared Error (MSE), Mean absolute error (MAE) and Mean absolute percentage error (MAPE) [27]. They are calculated using



		Predicted classification				
	Classes	a	b	c	d	Total
Actual classifcation	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
	Total	11	11	13	17	52

Figure 7: Example of confusion matrix



		Predicted classification			
	Classes	a	b	c	d
Actual classifcation	a	TN	FP	TN	TN
	b	FN	TP	FN	FN
	c	TN	FP	TN	TN
	d	TN	FP	TN	TN

Figure 8: Example of confusion matrix with the prediction outcomes, relative to the *b* class

Table 3: Classification evaluation metrics

Metric	**Description**	**Equation**
Accuracy	Fraction of correct predictions.	$\frac{TP+TN}{TP+TN+FP+FN}$
Recall	Proportion of actual positives correctly identified	$\frac{TP}{TP+FN}$
Precision	Proportion of positive identifications actually correct	$\frac{TP}{TP+FP}$
F1-score	Harmonic mean between precision and recall.	$2 \times \frac{precision \times recall}{precision + recall}$

the difference between the predicted and the actual value. MSE measures the average squared difference between the predicted values and the real value. Over all occurrences in the test set, MAE determines the mean of the absolute values of the individual prediction errors and MAPE determines the mean of the absolute percentage errors of the individual prediction errors.

Since the objective is to build a model that can generalize the information on unseen data, it is also important to measure the generalization performance of the model. This can be achieved by applying the k-fold cross-validation method, which uses *k* different partitions of the dataset to train and test a model on different iterations. Of the *k* portions, *k-1* portions are used as training data and the remaining portion is the validation data to test the model. This process is repeated until all partitions are tested, meaning it has *k* iterations until it ends.

6. **Hyperparameter Tuning** It is the process of selecting a set of ideal hyperparameters for a learning algorithm. A hyperparameter is a model parameter whose value is determined prior to the start of the learning process, since it cannot be learned during the training process.

7. **Prediction** After obtaining an acceptable performance, guided by the evaluation phase, the next and final step is to put the developed model to work. After all this effort, the benefit of ML is recognized at this step. The benefit of ML is that it enables one to obtain an accurate prediction by feeding input data to the model rather than relying on human judgment and manual rules.

2.2.2 Models and algorithms

In supervised ML, several algorithms and computing approaches are utilized. The following are some of the most popular classification and regression algorithms: Linear Regression (LR), Logistic Regression (LgR), K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Decision Tree (DT), Random Forest (RF) and Artificial Neural Network (ANN) [18, 28].

LR is the most popular method of regression analysis, which assumes that the dependent variable (variable to be predicted) and the independent variable (base for the variable to be predicted) have a linear relationship. Linear regression creates a model for the best fit line between two variables. The outcome of interest must be a continuous variable in order to be appropriate for LR [29]. Figure 9 shows how the model (red line) is created by utilizing training data (blue points) with known labels (y axis) to fit the points as exactly as possible by minimizing the value of a given loss function (usually MSE).

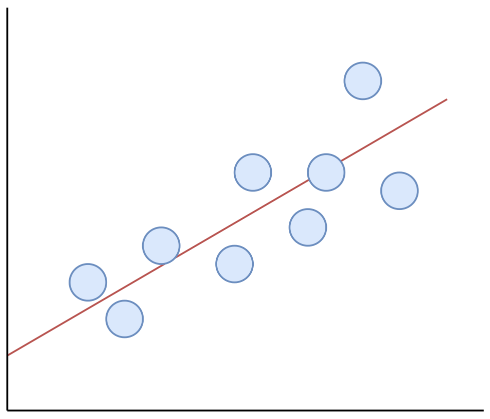


Figure 9: Visual representation of Linear Regression. Adapted from [30]

LgR is similar to LR but it is used for classification problems. To model a binary output variable, logistic regression employs the logistic function given below (Eq 1). Logistic regression's range is constrained to 0 and 1, which is the main difference between it and linear regression. In contrast to linear regression, logistic regression does not need a linear relationship between the input and output variables. Also unlike linear regression, which employs MSE as the loss function, logistic regression utilizes a conditional probability loss function called Maximum Likelihood Estimation (MLE). If the probability is greater than 0.5, the predictions will be labeled as class 0. Otherwise, you will be allocated to class 1 [31]. By default, logistic regression cannot be utilized for multi-class classification problems, which have more than two class labels. However, it is possible to adapt logistic regression to solve multi-class classification problems. One approach example is to divide the multi-class classification issue into several binary classification problems and apply a typical logistic regression model to each subproblem.

$$f(x) = \frac{1}{1+e^{-x}} \tag{1}$$

The KNN algorithm is a classification/regression technique that classifies data points based on their proximity and correlation with other data [28]. This technique assumes that data points that are comparable can be located close together. As a result, it attempts to determine the distance between data points, which is commonly done using Euclidean distance, and then assigns a category based on the most common category or average. Depending on the value of K (the number of nearest neighbors that will participate in the voting process), different results can be obtained. In Figure 10, the test sample (green circle) should fall into one of two categories: squares or triangles. If $K = 3$, then only the three nearest neighbors to the test sample will participate in the voting process. In this example, it is assigned to the triangles class since the nearest neighbors are two triangles and only one square. If $K = 5$, it is assigned to the squares class because the five nearest neighbors are three squares and only two triangles.

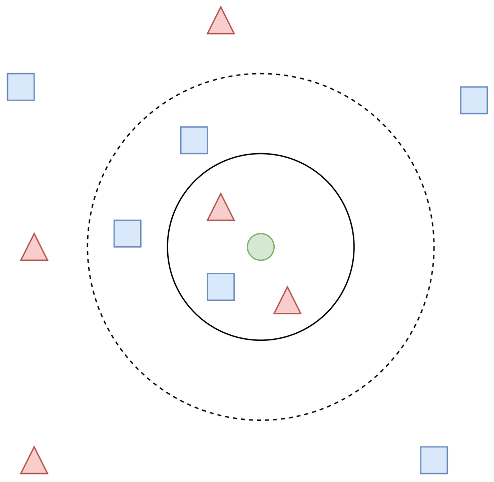


Figure 10: Visual representation of K-Nearest Neighbors. Adapted from [32]

SVMs are supervised learning models that evaluate data for classification and regression analysis. They create hyperplanes that are drawn at the maximum distance between two classes from the training data points (support vectors), since the greater the margin, the lower the classifier's generalization error. Then, new samples are predicted to belong to a category according to which side of the gap they fall [33]. Figure 11 provides an example of a linear classification perfomed by SVM. If a new sample fell to the right of the hyperplane, it would be classified as the red dot class, and it would otherwise be classified as the blue dot class if it fell to the left of the hyperplane.

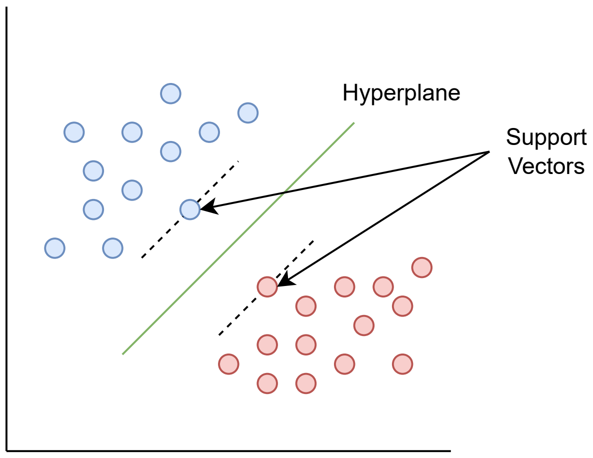


Figure 11: Visual representation of Support Vector Machines. Adapted from [34]

However, with the help of kernel functions, SVMs may do non-linear classification as well by implicitly translating their inputs into high-dimensional feature spaces. Some examples of kernel functions used in SVM classifiers are linear, polynomial and radial basis functions.

DT learning is a supervised ML technique for producing a decision tree from training data. DT builds classification or regression models in the form of a tree structure. It is a model that consists of a mapping from item observations to conclusions about its target value. In tree structures, leaves indicate labels, nonleaf nodes represent features, and branches represent combinations of features that lead to decisions on the target [35].

A RF classifier is an ensemble ML algorithm. One note should be added regarding ensemble methods before addressing RF. Ensemble learning is the process of building and combining many models to tackle a specific computational issue. Ensemble learning is generally used to improve a model's performance or reduce the risk of an unintentionally poor model selection [33]. Bagging is an ensemble learning technique for reducing variance in a noisy dataset. It consists of selecting a random sample of data from a training set with replacement (individual data points might be used multiple times). These weak models are then trained individually after multiple data samples are collected, and depending on the kind of task (for example, regression or classification), the average or majority of those predictions provides a more accurate estimate. Knowing this, RF employs parallel ensembling, in which numerous DT classifiers are fitted in parallel on

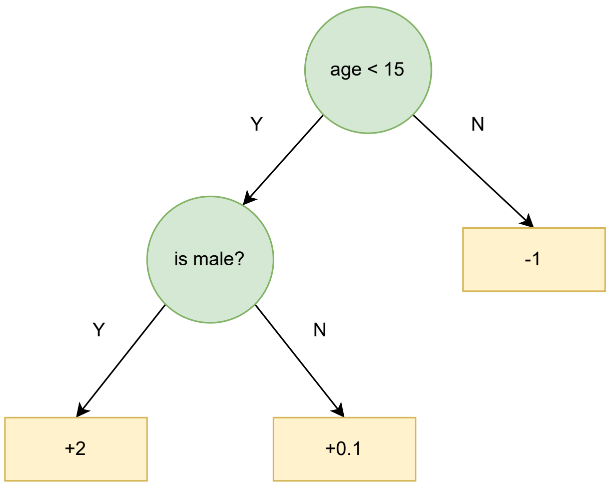


Figure 12: Visual representation of Decision Tree

distinct dataset sub-samples, as illustrated in Figure 13, and the final result is determined by majority voting or averages. Therefore, the overfitting problem is reduced, and prediction accuracy and control are improved. As a result, a RF learning model based on many DTs is usually more accurate than one based on a single DT. It combines the previously mentioned bagging technique with random feature selection to create a succession of DTs with controlled variance. It works well for both categorical and continuous variables and may be applied to both classification and regression issues [23].

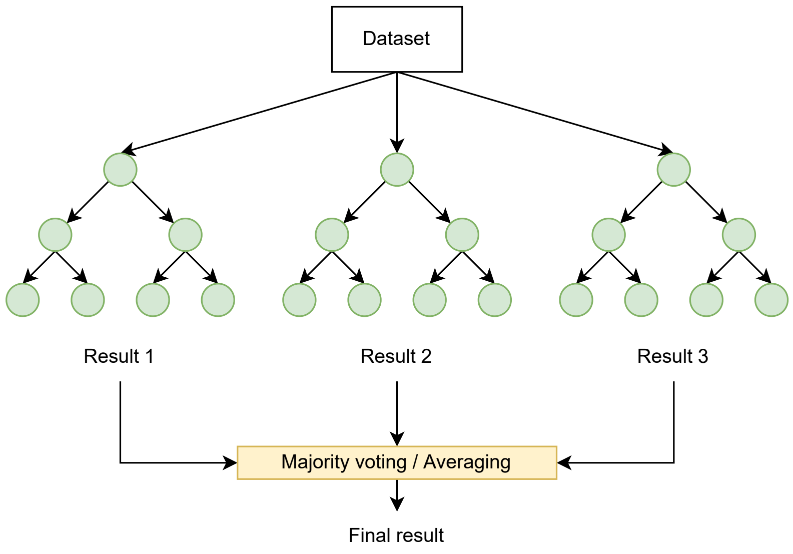


Figure 13: Random Forest structure. Adapted from [23]

ANNs-based supervised classifiers are extremely sophisticated and may be further subdivided into a number of distinct but related ideas. They are based on the neural network of the brain. These algorithms are used in most cutting-edge artificial intelligence applications and are typically used when working with very big datasets. They also serve as the foundation for deep learning approaches, as detailed below in both sections 2.2.3 and 2.3.

2.2.3 Artificial neural networks

In general, a biological neuron accepts inputs and arranges them to perform an operation, which results in the final output. When looking at biological neurons, there are four major components: dendrites, cell bodies, axons, and synapses. Dendrites are in charge of accepting incoming impulses into the cell body. The cell body subsequently processes these electrical signals and converts them to the final output. The output signal is then transferred from the cell body to the other neurons through the axon, which serves as a transmission line between neurons. Synapses are the locations placed between neurons and dendrites that are responsible for gathering input from neurons [36].

ANN is a supervised ML algorithm that is inspired by the biological structure and function of the human brain and, as seen in Figure 14, the intricacy of biological neurons in the brain can be mimicked.

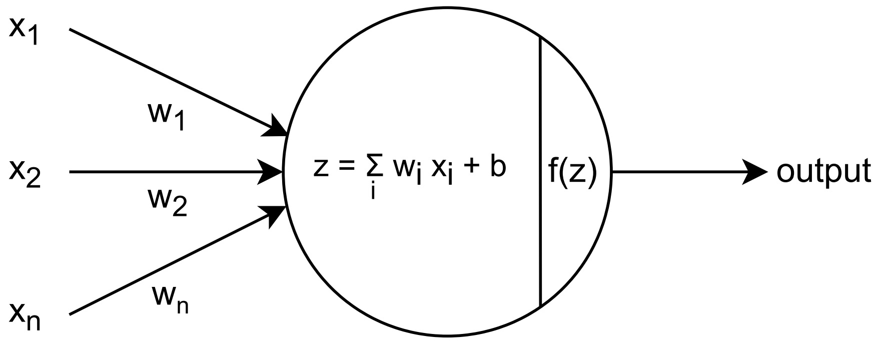


Figure 14: Artificial neuron. Adapted from [37]

In the case of ANN, inputs are directed to the body of an artificial neuron. In Figure 14, X(n) represents the inputs; each input is multiplied by its associated weight, which is a measure of the input's connection strength and is represented by W(n). The summing function is then given weighted inputs and the bias (b). The summing function's value (z) will be sent to the activation function (f), which will yield the final output [36]. Activation functions specify a range of values for the neuron's output, determining if the neuron's input to the network is essential or not [38]. Some examples of activation functions are sigmoid (Eq 2), TanH (Eq 3), and ReLU (Eq 4) [39].

$$f(x) = \frac{1}{1+e^{-x}} \tag{2}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3}$$

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{4}$$

The weights of an ANN are initially randomly assigned, but they are updated during the training process. This is possible by applying both forward and back propagation. In forward propagation, information travels in one direction only: forward. Inputs are fed into the neural network, and the produced outputs are compared to the real ones, with a loss function used to determine the difference [40]. Then, in back propagation, the internal weights are adjusted using optimization methods to reduce the loss function [41].

An ANN is composed of three layers: an input layer, a hidden layer, and an output layer. There must be a link between the nodes in the input layer and the nodes in the hidden layer, as well as between each node in the hidden layer and the nodes in the output layer [36]. In the input layer, each neuron represents an input feature, and no computation is performed. In the hidden layer the nodes are not visible. They serve as an abstraction for the neural network. The hidden layer performs all types of calculations on the features received through the input layer by using a weighted linear summation followed by an activation function, and sends the results to the output layer. Then the output layer takes the information learnt from the hidden layer and provides the final value. The number of output neurons represents the number of predictions. This means that, if it is a regression or binary classification problem, this layer will only have one neuron. If it is a multiclass classification problem, the number of neurons will be equal to the number of classes [42]. Figure 15 shows an example of an ANN's structure.

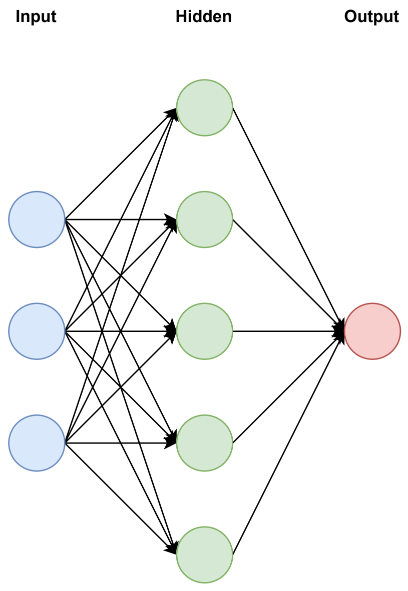


Figure 15: Artificial Neural Network

2.3 Deep Learning

Deep Learning (DL) is a subset of ML whose methods are based on multi-layered ANNs with feature learning techniques. These techniques enable a system to automatically identify the representations required for feature detection from raw data. This eliminates the need for human feature engineering by allowing a machine to learn the features and then use them to execute a specified activity. But, for this to be possible, DL algorithms demand larger amounts of data.

The simplest DL architecture is the Deep Neural Network (DNN). A DNN is an ANN with various layers between the input layer and the output layers. In other words, a DNN has multiple hidden layers [43]. Figure 16 shows a visual distinction between these two architectures.

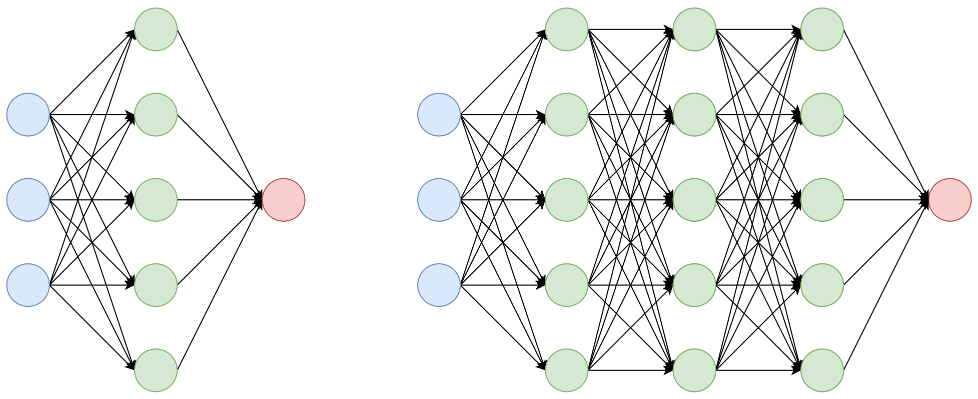


Figure 16: Artificial Neural Network vs Deep Neural Network

The inner workings of the layers and neurons are similar between ANN and DNN, meaning DNNs are feedforward networks that transfer data from the input layer to the output layer without looping back. The DNN starts by creating a map of virtual neurons and assigning random weights to their connections. The inputs and weights are multiplied, and the result is a value between 0 and 1. An algorithm would update the weights if the network did not detect a pattern correctly. As a result, the algorithm might increase the influence of specific factors until it finds the optimal mathematical manipulation to fully analyze the input.

To understand better the concepts of DL, Table 4 provides a few important terminologies.

2.3.1 Training phase

In order to start training a DL model, just as ANN, the initial model weights are chosen randomly. Then, the model weights are updated to reduce the error between the algorithm's current output and the expected output. This error is measured by the loss function and weight-finding algorithms are used to minimize this function, knows as optimization algorithms. These algorithms are specific implementations of the gradient descent algorithm, which is a technique to minimize loss by computing the gradients of loss with regard to the parameters of the model in training data.

Table 4: Deep Learning concepts [13, 44]

Concept	Description
Activation Function	Function that takes the weighted sum of all the inputs from the previous layer and then creates and passes an output value (usually nonlinear) to the following layer.
Back-propagation	Most commonly used algorithm for Gradient Descent. Begins at the output layer and traverses the network backwards.
Batch	Set of examples utilized in one model training iteration.
Batch size	The number of examples in a batch. It's an hyperparameter.
Epoch	A full training pass across the entire dataset while updating model weights. The number of epochs is a key hyperparameter.
Gradient	Partial derivatives vector with regard to all independent variables.
Gradient Descent	Technique to minimize loss by computing the gradients of loss with regard to the parameters of the model in training data.
Hyper-parameter	Constant parameter whose value is determined before the learning process.
Learning rate	Scalar that is used to train a model with gradient descent. The gradient descent technique multiplies the learning rate by the gradient at each iteration, meaning it controls the speed of the gradient update. It's also a key hyperparameter.
Loss	Function that calculates the difference between the algorithm's current output and the expected output.
Neuron	The neural network's basic unit.
Optimizer	Specific implementation of the gradient descent algorithm. Examples are SGD and Adam.
Parameter	Model variable that is self-taught by the ML system. Weights, for example, are parameters whose values the ML system learns over time through training iterations.
Training	Procedure for determining a model's optimum parameters.

Various optimizers are researched within the last few couples of years each having its advantages and disadvantages, but the most commonly used is Stochastic Gradient Descent (SGD) [45].

In SGD, the back-propagation algorithm is used to compute the gradients of the loss function, and the results are input to the SGD method to update the parameters (weights and biases) incrementally after each epoch.

Instead of computing the gradient of the error based on all training samples like GD, SGD creates an approximation of the actual gradient error based on a single training sample. As a result of the faster calculation of the approximation, DNN can train faster and generalize better with SGD [46].

The difference between the non-updated and the updated weight can be controlled using the learning rate hyperparameter and it is possible to define how to calculate the updated weight

$$W_x' = W_x - a(\frac{\partial Error}{\partial W_x}) \tag{5}$$

where W_x' is the new weight, W_x is the old weight, a is the learning rate, and $\frac{\partial Error}{\partial W_x}$ is the gradient (derivative of error with respect to weight) [47]. If the value of the learning rate hyperparameter is close to zero, the difference between the old and new weight would be minimal, meaning that the greater the learning rate, the greater the weight differential.

2.3.2 Challenges of deep neural networks

It is difficult to train a DNN that can generalize well to unknown input data. A model with insufficient capacity cannot learn the problem, while a model with excessive capacity may learn it too well and overfit the training dataset. In both circumstances, the model does not generalize effectively. However, DNNs tend to overfit because of the additional abstraction layers that enable them to model unusual relationships in the training data.

The complexity of a neural network model is determined by both its structure (number of nodes and layers) and its parameters (weights). As a result, in order to reduce overfitting, it is possible to lower a neural network's complexity by changing its structure or its parameters [48]. Instead of changing the neural network's architecture, it is more typical to limit the model's complexity by changing the model's weights, keeping them as small as possible. Small parameters imply a less complex and, as a result, more stable model that is less susceptible to statistical fluctuations in the input data.

Methods that aim to reduce overfitting by maintaining network weights minimal are referred to as regularization methods and the most common ones include early stopping, L1, L2 and dropout [48].

Early stopping takes a straightforward strategy, stopping the network's training when the model does not improve on the validation set score, or, in other words, when the error on the validation set starts to grow.

Weight decay, also known as weight regularization, is another strategy for decreasing overfitting in neural networks. Weight decay penalizes the network for having a high weight distribution by adding a cost to the training loss [48]. Examples of these methods are L1 and L2. In L1, it applies an L1 penalty equivalent to the absolute value of the magnitude of the coefficient. In L2, It applies an L2 penalty equivalent to the square of the magnitude of the coefficients [49].

In dropout, at every iteration of the training process, randomly selected nodes and their connections are dropped from the neural network. This prevents units from over-co-adapting, which would otherwise lead to overfitting problems. The derivative obtained by each parameter in a typical neural network tells it how it should change such that the resultant loss function is minimized, given what the other units are doing. As a result, units may evolve in such a way that they correct the errors of other units, which can lead to complex co-adaptations. This results in overfitting due to the fact that these co-adaptations cannot generalize to unseen data. Dropout makes the presence of additional hidden units unpredictable, preventing co-adaptation. As a result, a hidden unit cannot rely on other units to fix its errors. It must function properly in a wide range of situations created by the other hidden units [50].

In addition to the overfitting problem, DNNs also suffer from high computation times. DNNs must take into account a variety of training parameters, including the size (number of layers and units per layer), learning rate, and starting weights. Due to the time and processing resources required, sweeping across the parameter space for optimal parameters may not be practical. However, powerful processing hardware such as Graphics Processing Unit (GPU)s are ideal for training DL models as they can handle numerous computations at the same time. They feature a lot of cores, which makes it easier to run several parallel operations at the same time, resulting in a speed up in the training process.

2.3.3 Deep learning architectures

The growth in the field of DL architectures during the last two decades has provided enormous prospects for implementing it in a variety of applications. The next section introduces four popular DL architectures: DNN, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and autoencoders [51, 52].

Deep neural networks As mentioned in Section 2.3, DNNs are feed forward networks, in which data goes from the input layer to the output layer without traveling backwards. They are based on ANNs but, unlike them, DNNs contain multiple hidden layers. In spite of this, the inner workings of the layers and neurons are similar between ANN and DNN.

Convolutional neural networks A CNN is a multilayer supervised neural network that was originally inspired by the neurobiological process of animal visual cortex [51]. It is commonly used for processing data with a grid pattern, such as as images and videos. It is built to learn spatial hierarchies of features automatically, from low-level to high-level patterns.

CNNs are made up of three types of layers: convolutional, pooling, and fully connected layers. In general, the convolutional layer extracts features from the input, the pooling layer minimizes the size of the input data to the following layers, and the fully connected layers map the features into a final output, such as classification. Figure 17 provides an overview of the architecture of a CNN and how it is trained. The performance of a model with certain kernels and weights is determined using a loss function and forward propagation and they are updated using the gradient descent optimization process.

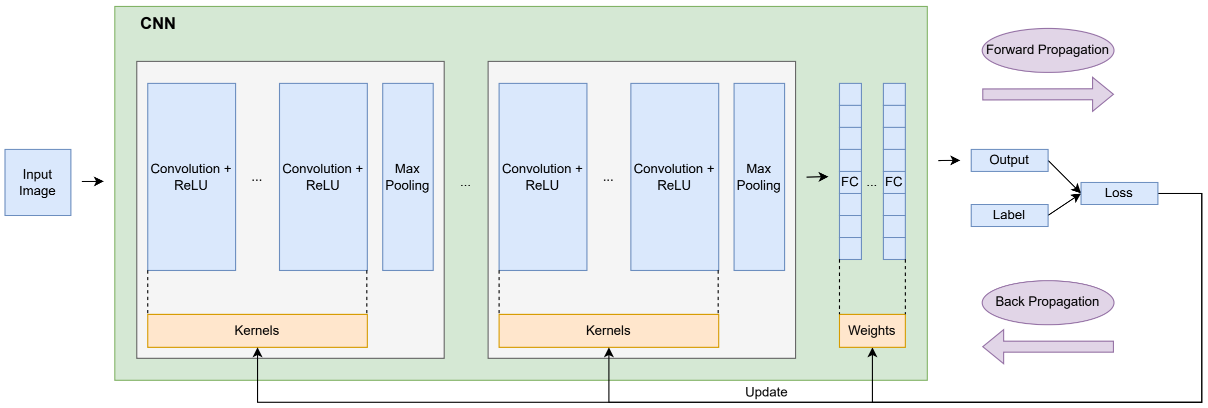


Figure 17: Convolutional Neural Network architecture. Adapted from [53]

The convolutional layer is a key component of the CNN architecture that conducts feature extraction using a mix of linear and nonlinear techniques, such as convolution and activation functions. Convolution is a form of linear operation used for feature extraction in which a tiny array of numbers, called a kernel, is applied over the tensor, which is also an array of numbers. At each point of the tensor, an element-wise product between each element of the kernel and the input tensor is computed and added to generate the output value in the corresponding place of the output tensor, referred to as a feature map (Figure 18). This technique is done with several kernels to create an arbitrary number of feature maps that reflect distinct features of the input tensors; hence, different kernels may be regarded as different feature extractors. The size and number of kernels are two fundamental hyperparameters that determine the convolution operation [53].

The stride is the distance between two consecutive kernel points, and a stride of 1 is the most popular choice. However, a stride greater than 1 is occasionally used to accomplish feature map downsampling. The pooling procedure is the alternate approach for downsampling.

It is important to emphasize that the above convolution method prevents the center of each kernel from overlapping the input tensor's outermost element, thereby reducing the output feature map's height and width in comparison to the input tensor. Padding, usually zero padding, is a strategy for dealing with this problem that involves adding rows and columns of zeros on either side of the input tensor in order to fit the center of a kernel on the outermost element while maintaining the same in-plane dimension during the convolution process. After the convolution procedure, each subsequent feature map would be smaller if there was no padding [53].

Identifying the kernels that perform best for a particular job based on a specific training dataset is the process of training a CNN model with relation to the convolution layer. Kernels are the only parameters in the convolution layer that are automatically learnt during the training process. The size and number of

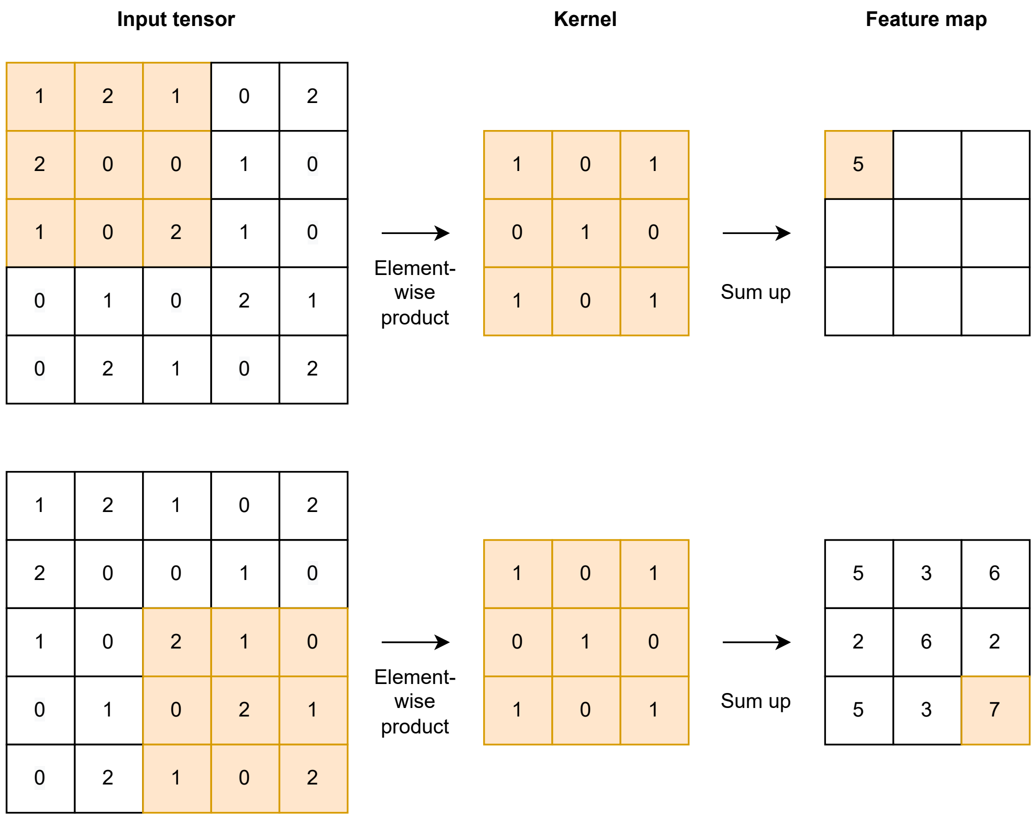


Figure 18: Convolution operation with a kernel size of 3 × 3, stride of 1, and no padding. Adapted from [53]

the kernels, stride, and padding are hyperparameters that must be defined before the training process begins [53]. Then, the output of the convolution operation is passed through a non-linear activation function, which is usually the ReLU function (Eq. 4).

After receiving the feature maps from the convolutional layer, the pooling layer will perform a standard downsampling operation on them, reducing their in-plane dimensionality in order to introduce translation invariance to tiny shifts and distortions and reducing the number of learnable parameters. Max pooling is the most popular type of pooling procedure, which selects patches from input feature maps, outputs the largest value in each patch, and discards all other values (Figure 19) [53].

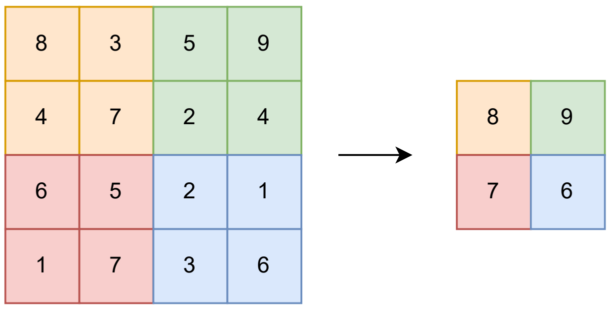


Figure 19: Max pooling with a filter size of 2 × 2, no padding, and a stride of 2. Adapted from [53]

Although max pooling is the most popular type of pooling operations, there is another type called global average pooling. It is an extreme sort of downsampling in which a feature map is downsampled into a 1 x 1 array, by simply taking the average of all the components in each feature map. Before the fully connected layers, this step is usually performed, because it can decrease the number of parameters that may be learned, and it allows the model to take inputs of varying sizes [53].

After obtaining the feature map of the last convolutional or pooling layer, it is now time to connect it to the fully connected layer. In order to do this, the output needs first to be flattened, or, in other words, needs to be transformed into a one dimensional tensor. After being flattened, the tensor is ready to be fed as input to the fully connected layer and each value of the input is connected to all neurons.

Recurrent Neural Networks In a feedforward neural network, the outputs of one layer are passed to the next layer, which is a unidirectional process. Past data cannot be stored in these feedforward networks [54].

A RNN can access previous data because of its loop-like structure, making them ideal algorithms for ML supervised challenges involving sequential data, such as speech and handwriting recognition [51]. Recurrent connections can be generated in RNN in three ways: starting in a neuron and ending in the same neuron; starting in a neuron and ending in another neuron from the same layer; or starting in a neuron and ending in another neuron from the previous layer. Only hidden and output neurons establish these recurrent connections; no input or bias neurons are involved. This design allows for the storage of historical data in order to predict current data [55], meaning that it considers both the current input and what it has learnt from previous inputs when making a decision.

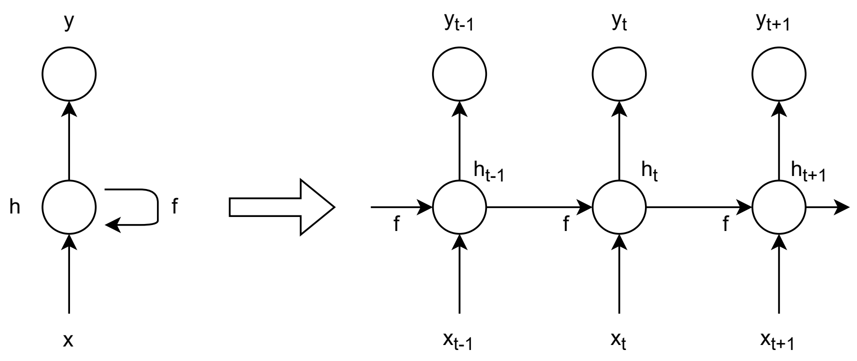


Figure 20: Recurrent Neural Network architecture. Adapted from [56]

However, RNNs suffer from the vanishing gradients problem, which makes learning large data sequences difficult. Gradients carry the information that is utilized in RNN parameter updates, and as they backpropagate through time, they become smaller and smaller. The parameter updates then become so tiny that no significant learning has taken place. Specific RNN designs, such as the Long Short-Term

Memory (LSTM) and Gated Recurrent Unit (GRU), were created to overcome the problem of vanishing gradients.

LSTM is a RNN architecture that was created to more precisely model temporal sequences and their long-range dependencies. LSTMs have a similar architecture to RNNs, with the exception that they employ a separate function to calculate hidden state in addition to the gating mechanism. To regulate the information passing through, it has three gates: an input gate, a forget gate, and an output gate. In LSTMs, there is a cell that saves past values and keeps them until a forget gate orders the cell to forget them. In another sense, it preserves earlier iterations for as long as they are required. An input gate adds a new input to the cell, while an output gate determines when the vectors from the cell should be sent through to the next hidden state [57]. Each gate is controlled by weights in the cell. The training algorithm, which is usually referred to as Backpropagation through time (BPTT), optimizes these weights depending on the network output error [52]. It is worth mentioning that LSTM only preserves information of the past because the only inputs it has seen are from the past. These type of LSTM are unidirectional LSTMs but there are also bidirectional LSTMs that run the inputs in two ways, one from past to future and one from future to past.

GRUs are a simpler RNN architecture with only two gates, the rest gate and the update gate. The rest gate is used in a model to determine how much information from the past should be forgotten or remembered. It decides which information to forget based on the information in the previous state and the next input candidate. The update gate aids the model in determining how much information from previous time steps should be passed on to future time steps.

GRU is a simple version LSTM, so it can be trained faster, and can execute tasks more efficiently. The LSTM, on the other hand, is more expressive, and with more data, it can provide better performance.

Autoencoders Autoencoder is an unsupervised neural network that learns how to compress and encode data effectively before reconstructing it back to a representation that is as similar to the original input as possible [58].

As shown in Figure 21, this type of neural network is made up of three layers: input, hidden, and output. First, a suitable encoding function is used to encode the input layer into the hidden layer. The hidden layer contains a significantly smaller number of nodes than the input layer. The compressed form of the original input is stored in this hidden layer. Using a decoder function, the output layer attempts to recreate the input layer.

As a result, autoencoders consists of 4 main parts [58, 59]:

- **Encoder**: The model learns how to compress the input data into an encoded form by reducing the input dimensions.

- **Bottleneck (latent space)**: The compressed form of the input data is stored in this layer. This is the smallest input data dimension imaginable.

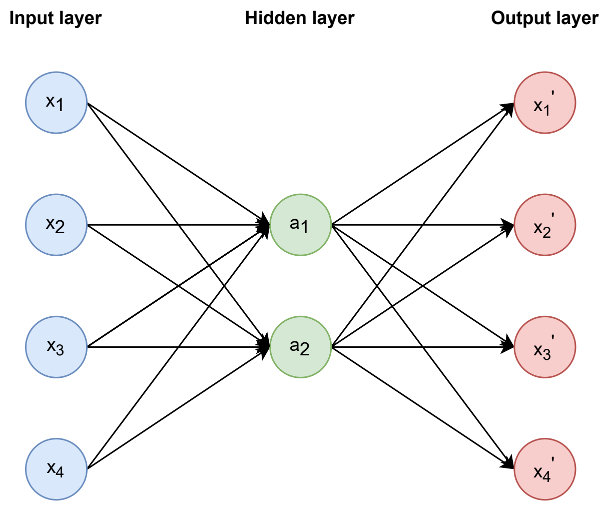


Figure 21: Autoencoders architecture. Adapted from [52]

- **Decoder**: The model learns how to reconstruct data from the encoded representation as closely as possible to the original input.
- **Reconstruction Loss**: This is a way for determining how well a decoder works and how near the output is to the original input. Back propagation is then used in the training to reduce the network's reconstruction loss.

The compressed form of the original input must only have essential information. In other words, an autoencoder can reduce data dimensionality by learning to ignore noise from the input data.

2.4 Automated machine learning

While ML has several proven benefits, its effective use needs a significant amount of work on the part of human specialists, since no algorithm can achieve good performance on all possible challenges. Despite their familiarity with data, researchers frequently lack the ML ability required to apply these approaches to large data sets. Researchers can and do collaborate with professional data scientists, but the collaborative approach involves time and effort from both parties. As a result, devising and deploying ML solutions is complex, as the process starts with a lengthy data supply procedure, continues with identifying the suitable collaborators, and requires constant back-and-forth between ML professionals and domain experts. By automating some of the components that need human skill, sectors will be able to create, verify, and deploy ML systems more quickly [60]. As a result, Automated Machine Learning (AutoML) has arisen with the goal of automatically optimizing sections of the ML pipeline, such as feature engineering, model selection, and hyperparameter optimization, as shown in Figure 22.

In recent years, multiple packages have been developed that provide AutoML. Some examples are:

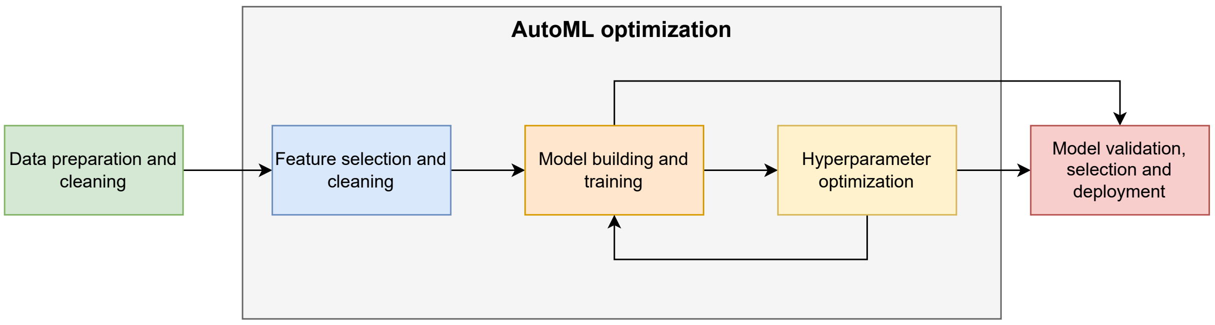

AutoML optimization
Data preparation and cleaning
Feature selection and cleaning
Model building and training
Hyperparameter optimization
Model validation, selection and deployment

Figure 22: Machine learning pipeline with AutoML. Adapted from [60]

- **AutoWEKA** [61] - package for selecting a ML algorithm and its hyperparameters at the same time; when paired with the WEKA package, it produces good models for a wide range of data sets automatically.
- **Auto-PyTorch** [62] - it is based on the PyTorch DL framework and it optimizes the network architecture and training hyperparameters together and reliably to allow fully automated DL.
- **TPOT** [63] - data science assistant that uses genetic programming to enhance ML pipelines and it is built on top of scikit-learn.
- **AutoKeras** [64] - open-source software library for AutoML. It is built in Keras and provides methods for searching for DL architectures and hyperparameters for models.

2.5 Python libraries for machine and deep learning

While there are many languages to choose from, Python[1] is one of the most developer-friendly ML and DL programming languages available, and it comes with a large library to suit any use-case or project. Most popular libraries include [65, 66]:

- **Tensorflow**[2]: high-performance numerical computing open source software framework. This library, which was created by Google researchers and engineers, has a strong support for ML and DL. It works with tensors, which are structures that imitate scalars, vectors, and matrices and allow for calculations between them. This tool's key features include straightforward numerical calculation, deployment on numerous CPUs or GPUs, and a robust data visualization interface.
- **Keras**[3]: DL framework that provides high-level building blocks for designing practically any type of DL model in a far more convenient way than constructing it from the ground up. Keras also enables users to train models on both the CPU and GPU.

[1] https://www.python.org/
[2] https://www.tensorflow.org/
[3] https://keras.io/

- **PyTorch**[4]: open-source ML/DL library created by Facebook and based on Torch. It offers a large number of tools and libraries that assist Computer Vision, Natural Language Processing (NLP), and a variety of other ML tasks. It enables developers to run Tensor computations with GPU acceleration and aids in the creation of computational graphs.

[4]https://pytorch.org/

Chapter 3

Machine and Deep Learning in DNA sequence classification

3.1 DNA sequences

Deoxyribonucleic acid (DNA) is composed of a linear string of nucleotides, or bases, which are referred to by their chemical names' first letters: Adenine (A), Thymine (T), Cytosine (C), and Guanine (G).

DNA sequencing is the technique of finding the order of the four bases. Scientists can determine the type of genetic information carried in a DNA segment by examining the sequence. Furthermore, and more significantly, sequencing data can reveal mutations in a gene that could lead to illness, by comparing a healthy and a mutated sequence [67].

The four chemical bases of the DNA double helix always connect with the same partner to produce "base pairs". A is always paired with T, while C is always paired with G. This pairing explains the technique by which DNA molecules are copied when cells divide, as well as the methods used in most DNA sequencing research. The human genome is made up of around 3 billion base pairs, which carry the instructions for creating and maintaining a human person [67].

Since the Human Genome Project's completion [68], technological advancements and automation have made it possible for individual genes to be sequenced on a regular basis, by reducing the amount of time it takes to perform the sequencing and also reducing its cost. Some labs can sequence over 100,000 billion bases per year, and a few thousand dollars is enough to sequence an entire genome [67].

3.2 DNA sequence classification - Traditional Machine Learning

Understanding the connections between protein structure and function is one of Biology's main goals. The basic amino acid sequences provide particularly helpful structural information for understanding the structure-function paradigm. The idea that sequences with similar structures have comparable functions is used to classify DNA sequences. Sequence alignment techniques such as BLAST [69] and FASTA [70] have historically been used to determine sequence similarity, and the majority of sequence classification is still done by these methods. This decision is based on two primary assumptions: the functional components have similar sequence properties, and the functional elements' relative order is preserved across sequences. These assumptions are applicable in a wide range of situations, but they are not universal [71].

Regardless, despite recent advancements, the major issue that severely restricts the use of alignment techniques remains their computational time complexity. As a result, many effective and computationally affordable approaches for analyzing sequence data have recently been presented. Since the majority of sequence analysis tasks are expressed as binary or multiclass classification tasks, ML techniques have been playing an important role [72].

In ML, the goal of classification is to use the training set to create a classification model that can predict the category of unknown incoming samples. DNA sequence classification is the process of predicting the kind of DNA sequence based on structural or functional similarities, then predicting the sequence function and relationships with other sequences, and finally assisting in the identification of genes in DNA molecules [2].

However, for a ML model to be able to predict, it first needs input features, and that is the main problem with sequences. The most difficult component is the feature extraction because the sequences only consist of a set of four letters, meaning they lack explicit features. Converting sequences into an effective numerical representation that reflects the underlying relationship with the feature to be predicted can have a big impact on the model's performance [71, 73].

It is standard practice to encode molecular information as numerical features in order to apply various ML algorithms to molecular data. Molecular descriptors are one of the most powerful tools for describing the biological, physical, and chemical features of molecules, and they have been utilized in a variety of research to better understand molecular interactions. These descriptors capture and amplify different features of molecular topology in order to better understand how molecular structure influences molecular properties [74]. However, it is worth emphasizing that these descriptors need to be manually created, meaning the feature selection is a handcrafted process. Some examples of DNA sequence descriptors are the nucleic acid composition, structure composition and sequence length [72, 73, 75]. Table 5 provides an overview of packages capable of calculating descriptors for not only DNA but also Ribonucleic acid (RNA) and protein sequences. In addition, some of these packages also perform ML functions on the calculated features.

Table 5: Overview of packages with DNA descriptors

Year	Authors	Title	Focus
2014	Liu et al.	repDNA [76]	Generate widely used features reflecting the physicochemical properties and sequence-order effects of DNAs and nucleotides
2016	Dong et al.	BioTriangle [77]	Full pipelining from getting molecular data, molecular representation to constructing ML models on DNA, RNA and protein sequences.
2017	Liu	BioSeq-Analysis [72]	Automatically completing feature extraction, predictor construction, and performance evaluation on DNA, RNA and protein sequences.
2018	Dong et al.	PyBioMed [74]	Calculate numerous features of biological molecules, with the goal of constructing integrated analytical pipelines from data gathering, data validation, and descriptor calculation to modeling.
2019	Nikam and Gromiha	Seq2Feature [78]	Extract protein and DNA sequence-based features.
2019	Muhammod et al.	PyFeat [79]	Extract features from proteins, DNAs and RNAs, with a particular emphasis on features that capture information on the interaction of nearby residues.
2019	Chen et al.	iLearn [73]	Feature extraction, clustering, normalization, selection, dimensionality reduction, predictor construction, best descriptor/model selection, ensemble learning and results visualization for DNA, RNA and protein sequences.
2021	Bonidia et al.	MathFeature [75]	Implements mathematical descriptors able to extract relevant numerical information from DNA, RNA and proteins.

Another important stage in sequence analysis is the creation of predictors. Many ML methods have been employed to predict structural and functional characteristics and to help in the annotation of genomic data. SVM, RF, ANN, KNN, and LgR are some examples of these methods [73].

3.3 DNA sequence classification - Deep Learning

A common difficulty in data mining is classifying biological sequences as a specific data type. This is a tough challenge, due to the non-numerical properties of biological sequence elements, the sequence interaction between sequence elements, and the variable sequence length [2]. ML methods for supervised classification tasks are, without a doubt, heavily reliant on the feature extraction stage, and it is required to detect and quantify relevant aspects of the objects to classify in order to develop a suitable representation. DL models have recently been shown to be capable of automatically extracting meaningful features from input patterns [71]. This is possible due to feature learning techniques that DL methods possess. These techniques allow the automatic identification of the representations required for feature detection from raw data, meaning that the human feature engineering is removed, and the machine will learn the features instead. As a result, DL models only need the sequence itself as input, unlike ML models that require the previously calculated features. However, first it is required to transform the string sequence into a numerical value in order to create an input matrix for the model [2]. Sequential encoding, one-hot encoding, and k-mer encoding are three popular methods for sequence encoding [80] and Table 6 summarizes the differences between them. The encoding technique is also significant for classification accuracy.

Table 6: DNA sequence encoding methods [2]

Encoding Method	**Features**
Sequential encoding	Each base is encoded as a number. For example, change [A,T,G,C] to [0.25,0.5,0.75,1.0], and any other character to 0.
One-hot encoding	Each base is encoded as a vector. [A,T,G,C] will become [0,0,0,1],[0,0,1,0],[0,1,0,0],[1,0,0,0]. These vectors can be combined into a 2-dimensional array.
K-mer encoding	Decomposes a sequence into k-length overlapping segments. ”ATGCATCGA”becomes ”ATGCAT”, ”TGCATG”, ”GCATGC”, ”CATGCA”when *k*=6 is used. The segments must then be converted to numerical values.

In the case of the k-mer encoding method, the encoded sequence is not yet ready to be fed into the model, as the result of the encoding method is not a numerical value. The DL model still needs this produced k-mer sentence to be transformed into a dense feature vector matrix and this can be achieved by using a word embedding layer. Word embedding is a technique for assigning each word in a sequence a continuous vector representation, and words with similar meanings are near in distance in the vector space generated. Gunasekaran et al. [81] used the k-mer encoding to generate segments of DNA sequences and then applied the word embedding layer to their model to transform the sentence into a dense feature vector matrix. It is also possible to apply an embedding layer to other encoding methods. Although there is no semantic similarity to uncover, this concept has also been applied at character-level, representing a first step toward completely automated feature learning. For example, Lo Bosco and Di Gangi [71] used CNN and RNN for the purpose of DNA sequence classification. For both models, their first layer was an embedding layer, which accepted as input 16-dimensional one-hot encoding of sequence characters and produced a 10 dimensional continuous vector. The embedding representation may be achieved using a single feed-forward layer using random weights in order to learn embedding for all of the terms in the training dataset.

In Section 3.4, relevant previous work on DNA classification will be provided, and it is possible to conclude that CNN and RNN are the most commonly used DL models.

3.4 Relevant previous work on DNA classification

DNA sequence classification is a critical problem in biomedical research, and since current solutions involve the use of homologies, which is a long and costly process, multiple ML techniques were used to successfully complete this task. In recent years, numerous articles and papers were published regarding this classification challenge with both traditional ML and DL approaches.

Nair et al. [82] proposed a unique technique for organism classification based on a combination of Frequency Chaos Game Representation (FCGR) and DNA. Chaos Game Representation (CGR) displays DNA sequences in a unique way and reveals hidden patterns in them. The frequency of sub-sequences contained in the DNA sequence is shown by FCGR, which is developed from CGR. The taxonomic distribution of Eukaryotic species is broken down into eight groups, and ANN is used to classify them.

Rizzo et al. [83] introduced a DNN based on spectral sequence representation for DNA sequence classification. The framework is evaluated on a dataset of 16S genes, and its results are compared to the General Regression Neural Network as well as Naive Bayes, RF, and SVM classifiers in terms of accuracy and F1 score. When it came to classifying short sequence fragments of 500 bp, the DL technique beat all other classifiers.

Nguyen et al. [84] developed a new method for classifying DNA sequences using a CNN while treating them as text input. Because the authors employed one-hot vectors to represent sequences as input to the model, the important position information of each nucleotide in sequences is preserved. The authors investigated the suggested model using 12 DNA sequence datasets and found substantial improvements in all of them. Out of the 12 datasets used, 10 of them include DNA sequences that wrap around histone proteins (for example, H3 and H4) and the other datasets are Splice and Promoter datasets. This finding suggests that a CNN can be used to handle additional sequence challenges in bioinformatics.

Lo Bosco and Di Gangi [71] offer two distinct DL architectures (CNN and RNN) for DNA sequence classification. For five separate classification tasks, they compare their results using a public data set of

DNA sequences. Two DL architectures were examined in this work for automated classification of bacteria species with no sequence preprocessing procedures. In comparison to a traditional CNN, the authors have presented a LSTM that utilizes the nucleotide locations in a sequence. The CNNs outperform the LSTM in the four easiest classification tasks, but their performance deteriorates in the last, when the LSTM performs better.

Abd-Alhalem et al. [85] proposed, based on a custom layer, a new technique for classification of bacterial DNA sequences. The FCGR of DNA is employed with a CNN. With a proper choice of the frequency k-lengthen words occurrence in DNA sequences, the FCGR is used as a sequence representation technique. The DNA sequence is mapped using FCGR, which generates a gene sequence image. Both local and global patterns may be seen in this sequence. For image classification, a pre-trained CNN is used.

Chen et al. [86] explain the distinction between cell-free DNA and normal DNA. The authors employed a variety of classification models to categorize normal and cfDNA, including KNN, SVM, and RF. In comparison to all other algorithms, the authors determined that the RF had the best accuracy.

Shen et al. [87] introduced a model that combines a Bidirectional GRU network with k-mer embedding to discover Transcription Factor (TF) binding sites on DNA sequences. DNA sequences are split into k-mer sequences of varying lengths and stride windows. Then, they take each k-mer as a word and train a word representation model on it using the *node2vec* technique. With this technique, the vectors are carefully designed in such a way that a simple mathematical function reflects the amount of semantic similarity between the words represented by those vectors. This means that words with common contexts are placed close to one another in the vector. Finally, for feature learning and classification, they built a deep bidirectional GRU model.

Helaly et al. [88] analyze three of the most current DL efforts for taxonomy classification using the 16S rRNA barcode dataset. Three distinct CNN architectures are examined, as well as three different feature representations: k-mer spectral representation, FCGR, and character-level integer encoding. The most fine-grained classification challenge showed that representations that hold positional information about the nucleotides in a sequence perform substantially better.

Zhang et al. [5] proposed a DL based technique (fully connected DNN) for predicting human essential genes by combining features acquired from sequencing data with a protein-protein interaction network. An embedding method (*word2vec*) is used to automatically learn the features, as well as 89 sequence features derived from DNA sequence and protein sequence for each gene. They outperform numerous traditional ML models, such as SVM, Naïve Bayes, RF, and Adaboost, using the same features. Results show that the final model can accurately predict human gene essentiality with an average performance of AUC higher than 94%.

Gunasekaran et al. [81] used CNN, CNN-LSTM, and CNN-Bidirectional LSTM architectures with Label and k-mer encoding for DNA sequence classification. Different classification metrics were used to evaluate the models. According to the findings of the experiments, the CNN and CNN-Bidirectional LSTM with k-mer encoding have good accuracy, with 93.16% and 93.13% on testing data, respectively.

Lugo and Hernández [89] presented a sequential DL approach for bacterium identification. To derive an identification model for whole-genome bacterium sequences, the proposed neural network takes advantage of the massive volumes of data supplied by Next-Generation Sequencing. The bidirectional RNN (BI-GRU) outperformed other classification algorithms (Naive Bayes, Multilayer Perceptron) after verifying the identification model. In a low-dimensional space, a distributed representation was proven as the appropriate encoding for bacterial genetic information. The distributed representation is given context by combining two or more k-mer lengths. Context makes use of positional data, which is critical in biological sequences.

Table 7 provides an overview of the previous work on DNA classification described above.

Table 7: Overview of DNA classification's previous work

Year	Authors	Title	Focus	Classifier	Features / Encoding
2010	Nair et al.	"ANN based classification of unknown genome fragments using chaos game representation" [82]	Taxonomic classification of Eukaryotic species	ANN	FCGR
2015	Rizzo et al.	"A Deep Learning Approach to DNA Sequence Classification" [83]	Classify 16S bacterial genomic sequences	DNN, General Regression Neural Network, Naive Bayes, RF, SVM	k-mer encoding
2016	Nguyen et al.	"DNA Sequence Classification by Convolutional Neural Network" [84]	Solve DNA sequence classification problem in 12 datasets	CNN	one-hot vectors
2017	Lo Bosco and Di Gangi	"Deep learning architectures for DNA sequence classification" [71]	Classification of bacteria species with no steps of sequence preprocessing	CNN, RNN	Character-level one-hot encoding
2017	Chen et al.	"A Study of Cell-free DNA Fragmentation Pattern and Its Application in DNA Sample Type Classification" [86]	Cell-free DNA and normal DNA classification	KNN, SVM, RF	DNA fragmentation patterns
2018	Shen et al.	"Recurrent Neural Network for Predicting Transcription Factor Binding Sites" [88]	Identify and classify TF binding sites on DNA sequences.	BI-GRU	k-mer, word2vec

2019	Helaly et al.	"Convolutional Neural Networks for Biological Sequence Taxonomic Classification: A Comparative Study" [88]	16S rRNA barcode dataset taxonomy classification	CNN	k-mer spectral representation, FCGR, and character-level integer encoding
2020	Zhang et al.	"DeepHE: Accurately predicting human essential genes based on deep learning" [5]	Pedict human essential genes on DNA sequences	fully connected DNN	word2vec, sequence features derived from DNA sequence and protein sequence
2021	Gunasekaran et al.	"Analysis of DNA Sequence Classification Using CNN and Hybrid Models" [81]	COVID, SARS, MERS, dengue, hepatitis, and influenza classification	CNN, CNN-LSTM, CNN-Bidirectional, LSTM	Label and k-mer encoding
2021	Lugo and Hernández	"A Recurrent Neural Network approach for whole genome bacteria identification" [89]	Bacterium identification	BI-GRU	Distributed k-mer

Chapter 4

Development and Implementation

4.1 Development Strategy

The main goal of this study was to create a tool that can automatically classify DNA sequences using ML/DL models, followed by its integration into *ProPythia*. The other key objective was to integrate automated machine learning classifiers into the *OmniumAI* software platforms.

However, it is crucial to understand the steps and the technologies necessary to build such tools. These tools were built with *Python*, which is a high-level interpreted general-purpose programming language that supports vast and extensive external libraries that are constantly evolving. *PyCharm Code Editor* was used to combine *Python* code development, while also improving usability and user experience. Additionally, the *Anaconda* software application was employed to facilitate package management and distribution. Then, for building the entire classification pipeline, the *PyTorch* framework was used as it is one of the most popular free open source and powerful DL frameworks.

In an ML project, after selecting the dataset, the first step is the data processing task since, in most cases, the data is not ready to be fed into a model. The sequences need first to be converted into a numerical representation, and how this step is executed has a substantial influence on the model's ultimate performance. As mentioned in Section 2.2.1, data processing can be subdivided into feature extraction and feature selection. The first is the process of transforming raw data into suitable modelling features, and the second consists of filtering irrelevant features, keeping the essential ones. If working with a DL model, the user may not need to perform this step since these kind of models can extract the features from the training data on their own.

In this study, descriptors were chosen to be the features for shallow ML classification models, which is a manual feature extraction process. For DL models, they only need the sequences as input, but they still need to be in a numerical format. Encoders were utilized to address this problem, since they can create numerical representations of a string of letters.

4.2 Setting up the Data

It is important to note that all sequence feature vectors that will be fed into the model need to have the same input shape, regardless of whether the sequences have the same length or not. For example, a sequence with a length of 100 and another sequence with a length of 200, that belong to the same dataset, must be represented by a feature vector of the same size. When using descriptors, this can be achieved by implementing them in a way that they are not dependent on the sequence's length. However, when using encodings, the only option is to truncate or fill all sequences to a length's value. Using the example above, the two sequences could be truncated or filled to have a length of 150, and then calculate the encoding. To fill the sequence to a length's value, a letter that did not belong to the A, C, T, and G alphabet was added - the letter N.

The following chapter provides an overview of the descriptors and the encoders for DNA sequences that were developed in this study.

4.2.1 Descriptors

Table 8 provides the list of all the implemented descriptors, as well as their respective group and resulting vector size. The process of obtaining this list of descriptors was based on finding the descriptors that most DNA descriptors packages from Table 5 implemented, as well as choosing the ones that did not depend on the sequence's length, as the datasets used in the case studies did not have sequences of uniform length.

The N in the table represents the number of physicochemical indices to calculate.

Both the psychochemical properties and the nucleic acid composition are the most straight-forward approaches to represent the DNA sequences.

Length Length descriptor is a simple descriptor that calculates the length of a sequence.

GC Content Guanine-Cytosine (GC) content feature encoding represents the quantity of guanine and cytosine nucleotides in a sequence. It can be calculated as follows:

$$x = \frac{N_{(C)} + N_{(G)}}{L} \tag{6}$$

where $N_{(C)}$ donates the number of the nucleotide C in the sequence, $N_{(G)}$ the number of the nucleotide G and L the length of the sequence.

Table 8: List of implemented descriptors.

Descriptor groups	**Descriptor**	**Vector Size**
Psychochemical	Length	1
	GC Content	1
	AT Content	1
Nucleic Acid Compostion	Nucleic Acid Compostion (NAC)	4
	Dinucleotide Acid Compostion (DNC)	16
	Trinucleotide Acid Compostion (TNC)	64
	Composition of K-spaced nucleic acid pairs (CKSNAP)	16
	K-mer	4^k
	Accumulated Nucleotide Frequency (ANF)	$3 * 4$
Auto Correlation and Cross Covariance	Dinucleotide-based Auto Covariance (DAC)	$N * lag$
	Dinucleotide-based Cross Covariance (DCC)	$N * (N-1) * lag$
	Dinucleotide-based Auto-Cross Covariance (DACC)	$N * N * lag$
	Trinucleotide-based Auto Covariance (TAC)	$N * lag$
	Trinucleotide-based Cross Covariance (TCC)	$N * (N-1) * lag$
	Trinucleotide-based Auto-Cross Covariance (TACC)	$N * N * lag$
Pseudo Nucleic Acid Composition	Pseudo Dinucleotide Compostion (PseDNC)	$16 + \lambda$
	Pseudo K-tupler Compostion (PseKNC)	$4^k + \lambda$

AT Content Adenine-Thymine (AT) content feature encoding represents the quantity of adenine and thymine nucleotides in a sequence. It can be calculated as follows:

$$x = \frac{N_{(A)} + N_{(T)}}{L} \tag{7}$$

where $N_{(A)}$ donates the number of the nucleotide A in the sequence, $N_{(T)}$ the number of the nucleotide T and L the length of the sequence.

Nucleic Acid Composition The Nucleic Acid Composition (NAC) encoding [90, 91] is one of the approaches often used to represent DNA sequences; it represents the nucleotide frequencies of the sequence. The frequencies of the 4 natural nucleotides may be determined as follows:

$$f(i) = \frac{N_{(i)}}{L}, i \in \{A, C, T, G\} \tag{8}$$

where $N_{(i)}$ is the number of nucleotide type and L is the length of the DNA sequence.

Di-Nucleotide Composition Di-Nucleotide Composition (DNC) feature encoding [92, 93] is the composition of continuous dinucleotide pairs inside a DNA sequence. DNC feature encoding has 16 descriptors, which are specified as follows:

$$D(i, j) = \frac{N_{(ij)}}{L-1}, i, j \in \{A, C, T, G\} \tag{9}$$

where $N_{(ij)}$ is the number of dinucleotide represented by nucleotide types i and j, and L is the length of the DNA sequence.

Tri-Nucleotide Composition Tri-Nucleotide Composition (TNC) feature encoding [94, 95] reflects the DNA sequence's composition of continuous trinucleotide pairs. TNC feature encoding has 64 descriptors ('AAA', 'AAC', 'AAG', 'AAT',..., 'TTT'), which may be described as follows:

$$D(i, j, k) = \frac{N_{(ijk)}}{L-2}, i, j, k \in \{A, C, T, G\} \tag{10}$$

where $N_{(ijk)}$ is the number of trinucleotides represented by nucleotide types i, j and k, and L is the length of the DNA sequence.

Composition of K-spaced nucleic acid pairs Composition of K-spaced Nucleic Acid Pairs (CKSNAP) feature encoding [72, 96] denotes the composition of nucleotide pairs in a segment that are K-steps apart. Specifically, we estimated the frequency of a nucleotide pair at positions i and $i+K+1$, where i = 1,..., (l - K - 1) and l is the length of the sequence. For instance, given the sequence $ACGTACGT$ and K = 2, the nucleotide AT will appear twice, with A and T occurring at positions 1 and 4, as well as positions 5 and 8. Regardless of the number of K, there are a maximum of 16 potential nucleotide pairings. This coding method represents the close interactions between nucleic acids inside a DNA sequence segment.

K-mer The K-mer encoding [97, 98] determines the occurrence frequencies of K adjacent nucleotides in a DNA sequence, which was widely employed in enhancer discovery and regulatory sequence prediction. The K-mer (e.g. K = 4) descriptor is described as follows:

$$K(i) = \frac{N_{(i)}}{L}, i \in \{AAAA, AAAC, AAAG, ..., TTTT\} \tag{11}$$

where N_i is the number of K-tuple type and L is the length of the DNA sequence.

The K-mer descriptor implemented additionally contains the Reverse Compliment Kmer (RCKmer). The RCKmer encoding [73] is a variation of the K-mer descriptor that computes the occurrence frequencies of reverse complement k adjacent nucleotides in a DNA sequence. For example, a DNA sequence has 16 types of 2-mers. Among them, 'TT' is the opposite of 'AA'. By deleting the reverse complimentary K-mers, there are only 10 varieties of 2-mers in the RCKmer method (i.e. 'AA', 'AC', 'AG', 'AT', 'CA', 'CC', 'CG', 'GA', 'GC', and 'TA').

Accumulated Nucleotide Frequency Accumulated Nucleotide Frequency (ANF) feature encoding technique [99] represents the nucleotide density and distribution of each nucleotide inside a DNA segment. The following formula illustrates how to determine the ANF of a DNA segment of length K.

$$d_l = \frac{1}{l}\sum_{j=1}^{l} f(n_j), f(n_j) = \begin{cases} 1 & n_j = q \\ 0 & other \end{cases}, l = 1, \ldots, K \tag{12}$$

where n_j is the nucleotide at the j^{th} position and $q \in (A, C, T, G)$. When $l = 3$, the nucleotide at the l^{th} position in the sequence of 'TGCTACGC' is C, and the density of this position is calculated as $d_3 = \frac{1}{3}\sum_{j=1}^{3} f(n_j) = \frac{1}{3}[f(C) + f(A) + f(C)] = \frac{1}{3}[1 + 0 + 1] = 0.667$. All K locations' densities may be estimated identically. When computing the ANF at each place, however, there would be K values, which is not a vector of constant length. To circumvent this, the ANF was only computed for three places located at 25, 50, and 75% of the sequence's length.

Dinucleotide-based Auto-covariance The first autocorrelation descriptor is called Dinucleotide-based Auto Covariance (DAC). As one of the multivariate modeling methods, autocorrelation can turn DNA sequences of varying lengths into vectors of constant length by assessing the correlation between any two physicochemical variables. Autocorrelation generates two types of variables: autocorrelation (AC) between the same property and cross-covariance (CC) between properties with different values.

It has been proved that DNA physicochemical properties play important roles in gene expression regulation [100, 101].

There are 38 dinucleotide physicochemical properties and 12 trinucleotide physicochemical properties listed in Table 9 and Table 10, respectively, that may be utilized to construct distinct modes of dinucleotide and trinucleotide autocorrelation features. The present analysis was confined to dinucleotide and trinucleotide autocorrelation descriptors since little physicochemical property data are available for K-tuple nucleotides with $K = 4$ and higher. When the necessary physicochemical property data are available, however, the formulae described here may be simply modified to include the scenario when $K \geq 4$ [102].

Table 9: List of 38 physicochemical properties of dinucleotides in DNA. [102]

Number	**Description**
1	Base stacking
2	Protein-induced deformability
3	B-DNA twist
4	Dinucleotide GC content
5	A-philicity
6	Propeller twist
7	Duplex stability (free energy)
8	Duplex stability (disrupt energy)

9	DNA denaturation
10	Bending stiffness
11	Protein-DNA twist
12	Stabilizing energy of Z-DNA
13	Aida_BA_transition
14	Breslauer_dG
15	Breslauer_dH
16	Breslauer_dS
17	Electron interaction
18	Hartman_trans_free_energy
19	Helix-coil_transition
20	Ivanov_BA_transition
21	Lisser_BZ_transition
22	Polar_interaction
23	SantaLucia_dG
24	SantaLucia_dH
25	SantaLucia_dS
26	Sarai_flexibility
27	Stability
28	Stacking_energy
29	Sugimoto_dG
30	Sugimoto_dH
31	Sugimoto_dS
32	Watson-Crick_interaction
33	Twist
34	Tilt
35	Roll
36	Shift
37	Slide
38	Rise

Table 10: List of 12 physicochemical properties of trinucleotides in DNA. [102]

Number	Description
1	Bendability (DNase)
2	Bendability (consensus)
3	Trinucleotide GC content
4	Nucleosome positioning
5	Consensus-roll
6	Consensus-rigid
7	DNase I
8	DNase I-rigid
9	MW-daltons
10	MW-kg
11	Nucleosome
12	Nucleosome-rigid

Consider a DNA sequence D containing L nucleic acid residues, i.e.

$$D = R_1\ R_2\ R_3\ \ldots\ R_L \tag{13}$$

where R_i is the nucleic acid residue at the sequence position $i \in [1, 2, \ldots, L]$ and L is the length of the DNA sequence. The DAC evaluates the correlation of the same physicochemical index between two dinucleotides separated by a distance of lag in the sequence. This correlation can be determined as follows:

$$DAC(u, lag) = \frac{\sum_{i=1}^{L-lag-1} (P_u(R_i\ R_{i+1}) - \overline{P_u})(P_u(R_{i+lag}\ R_{i+lag+1}) - \overline{P_u})}{L - lag - 1} \tag{14}$$

where u is a physicochemical index, L is the length of the DNA sequence, $P_u(R_iR_{i+1})$ is the numerical value of the physicochemical index u for the dinucleotide R_iR_{i+1} at the i^{th} position, and $\overline{P_u}$ is the average value for physicochemical index u in the sequence [103]. The latter is calculated as follows:

$$\overline{P_u} = \frac{\sum_{j=1}^{L-1} P_u(R_jR_{j+1})}{L - 1} \tag{15}$$

The length of DAC feature vector is $N * LAG$, where N is the number of physicochemical indices and LAG is the maximum of $lag, lag \in [1, 2, \ldots, LAG]$ [103].

Dinucleotide-based Cross Covariance Given a DNA sequence D (Eq 13), the Dinucleotide-based Cross Covariance (DCC) method examines the correlation between two distinct physicochemical indices between two dinucleotides separated by lag nucleic acids in the sequence. This correlation may be determined using:

$$DCC(u_1, u_2, lag) = \frac{\sum_{i=1}^{L-lag-1}(P_{u_1}(R_i\ R_{i+1}) - \overline{P_{u_1}})(P_{u_2}(R_{i+lag}\ R_{i+lag+1}) - \overline{P_{u_2}})}{L - lag - 1} \tag{16}$$

where u_1, u_2 are two different physicochemical indices, L is the length of the DNA sequence, $P_{u_1}(R_iR_{i+1})(P_{u_2}(R_iR_{i+1}))$ is the physicochemical index value $u_1(u_2)$ for the dinucleotide R_iR_{i+1} i^{th} position, and $\overline{P_{u_1}}$ $(\overline{P_{u_2}})$ is the average value for physicochemical index value u_1, u_2 along in the sequence:

$$\overline{P_u} = \frac{\sum_{j=1}^{L-1} P_u(R_jR_{j+1})}{L-1} \tag{17}$$

The length of DCC feature vector is $N * (N-1) * LAG$, where N is the number of physicochemical indices and LAG is the maximum of $lag, lag \in [1, 2, ..., LAG]$.

Dinucleotide-based Auto-Cross Covariance Combining DAC and DCC results in Dinucleotide-based Auto-Cross Covariance (DACC), meaning the length of the DACC vector is $N * N * LAG$, where N is the number of physicochemical indices and LAG is the maximum of $lag, lag \in [1, 2, ..., LAG]$.

Trinucleotide-based Auto Covariance Given a DNA sequence D (Eq. 13), the Trinucleotide-based Cross Covariance (TCC) method examines the correlation of the same physicochemical index between two trinucleotides separated by *lag* nucleic acids in the sequence. This correlation may be determined using:

$$TAC(lag, u) = \frac{\sum_{i=1}^{L-lag-2}(P_u(R_iR_{i+1}R_{i+2}) - \overline{P_u})(P_u(R_{i+lag}R_{i+lag+1}R_{i+lag+2}) - \overline{P_u})}{L - lag - 2} \tag{18}$$

where u is a physicochemical index, L is the length of the DNA sequence, $P_u(R_iR_{i+1}R_{i+2})$ is the numerical value of the physicochemical index u for the trinucleotide $R_iR_{i+1}R_{i+2}$ at i^{th} position, $\overline{P_u}$ is the average value for physicochemical index u along the whole sequence:

$$\overline{P_u} = \frac{\sum_{j=1}^{L-2} P_u(R_jR_{j+1}R_{j+2})}{L-2} \tag{19}$$

The length of Trinucleotide-based Auto Covariance (TAC) feature vector is $N * LAG$, where N is the number of physicochemical indices and LAG is the maximum of $lag, lag \in [1, 2, ..., LAG]$.

Trinucleotide-based Cross Covariance Given a DNA sequence D (Eq. 13), the TCC method examines the correlation of two different physicochemical indices between two trinucleotides separated by *lag* nucleic acids in the sequence. This correlation may be determined using:

$$TCC(u_1, u_2, lag) = \frac{\sum_{i=1}^{L-lag-2} (P_{u_1}(R_i\ R_{i+1}\ R_{i+2}) - \overline{P_{u_1}})(P_{u_2}(R_{i+lag}\ R_{i+lag+1}\ R_{i+lag+2}) - \overline{P_{u_2}})}{L - lag - 2} \tag{20}$$

where u_1, u_2 are two different physicochemical indices, L is the length of the DNA sequence, $P_{u_1}(R_iR_{i+1}R_{i+2})(P_{u_2}(R_iR_{i+1}R_{i+2}))$ is the numerical value of the physicochemical index $u_1(u_2)$ for the trinucleotide $R_iR_{i+1}R_{i+2}$ at i^{th} position, $\overline{P_{u_1}}$ $(\overline{P_{u_2}})$ is the average value for physicochemical index value u_1, u_2 in the sequence:

$$\overline{P_u} = \frac{\sum_{j=1}^{L-2} P_u(R_jR_{j+1}R_{j+2})}{L-2} \tag{21}$$

The length of TCC feature vector is $N * (N-1) * LAG$, where N is the number of physicochemical indices and LAG is the maximum of $lag, lag \in [1, 2, ..., LAG]$.

Trinucleotide-based Auto-Cross Covariance Combining TAC and TCC results in Trinucleotide-based Auto-Cross Covariance (TACC), meaning the length of the TACC feature vector is $N*N*LAG$, where N is the number of physicochemical indices and LAG is the maximum of $lag, lag \in [1, 2, ..., LAG]$.

Pseudo Dinucleotide Composition All of the mentioned techniques relied only on the composition of nucleic acids, ignoring the influence of sequence order. Pseudo Dinucleotide Composition (PseDNC) is an approach incorporating the contiguous local sequence-order information and the global sequence-order information into the feature vector of the DNA sequence.

Given a DNA sequence D (Eq. 13), if the feature vector of D is formulated by its NAC, we have:

$$D = [f(A)\ f(C)\ f(G)\ f(T)]^T \tag{22}$$

where $f(A)$, $f(C)$, $f(G)$ and $f(T)$ are the normalized occurrence frequencies of the nucleotides in the DNA sequence and the T is the transpose operator symbol. As shown by Eq. 22, all the sequence-order information is lost when a DNA sequence is represented by NAC. If the DNC is used to represent the DNA sequence, rather than the four components provided in 22, the associated feature vector will have $4 * 4 = 16$ components, as shown in the following equation.

$$D = [f(AA)\ f(AC)\ ...\ f(TT)]^T = [f_1\ f_2\ ...\ f_{16}]^T \tag{23}$$

where each element is the normalized occurrence frequency of the dinucleotide in the DNA sequence. Eq. 23 contains the most contiguous local sequence-order information, but the formulation does not take into account any of the global sequence-order data.

The global sequence-order information may be included into the feature vector for the DNA sequence using the procedure that follows.

$$\begin{cases} \theta_1 = \frac{1}{L-2}\sum_{i=1}^{L-2}\Theta(R_iR_{i+1}, R_{i+1}R_{i+2}) \\ \theta_2 = \frac{1}{L-3}\sum_{i=1}^{L-3}\Theta(R_iR_{i+1}, R_{i+2}R_{i+3}) \\ \theta_3 = \frac{1}{L-4}\sum_{i=1}^{L-4}\Theta(R_iR_{i+1}, R_{i+3}R_{i+4}) \qquad\qquad (\lambda < L) \\ \ldots \\ \theta_\lambda = \frac{1}{L-1-\lambda}\sum_{i=1}^{L-1-\lambda}\Theta(R_iR_{i+1}, R_{i+\lambda}R_{i+\lambda+1}) \end{cases} \tag{24}$$

where θ_1 is the first-tier correlation factor that represents the sequence-order correlation between all the most contiguous dinucleotides in a sequence (Figure 23A); θ_2, the second-tier correlation factor between all the second most contiguous dinucleotides (Figure 23B); θ_3, the third-tier correlation factor between all the third most contiguous dinucleotides (Figure 23C) and so forth [91].

In Eq. 24, the parameter λ is an integer that represents the correlation along a DNA sequence that has the greatest counted tier, and the correlation function is provided by:

$$\Theta(R_iR_{i+1}, R_jR_{j+1}) = \frac{1}{\mu}\sum_{u=1}^{\mu}[P_u(R_iR_{i+1}) - P_u(R_jR_{j+1})]^2 \tag{25}$$

where μ is the number of local DNA structural features taken into account in the current investigation, which is equal to six. $P_u(R_iR_{i+1})$ represents the numerical value of the *u-th* $(u = 1, 2, ..., \mu)$ DNA local property for the dinucleotide R_iR_{i+1} at position i. And $P_u(R_jR_{j+1})$ is the corresponding value for the dinucleotide R_jR_{j+1} at position j.

Numerous lines of research have shown that certain local DNA structural properties, such as angular parameters (twist, tilt, and roll) and translational parameters (shift, slide, and rise), play significant roles in biological processes, including protein-DNA interactions, chromosome formation, and higher-order organization of the genetic material [104, 105]. Listed in Table 11 are the original numerical values for twist $P_1(R_iR_{i+1})$, tilt $P_2(R_iR_{i+1})$, roll $P_3(R_iR_{i+1})$, shift $P_4(R_iR_{i+1})$, slide $P_5(R_iR_{i+1})$, and rise $P_6(R_iR_{i+1})$, respectively, where R_iR_{i+1} represents the 16 possible dinucleotides. Only these six DNA local physical structural properties were used to calculate PseDNC, explaining why $\mu = 6$ in Eq. 25.

Prior to inserting into Eq. 25, the values listed in Table 11 were adjusted through a standard conversion [106], as shown by the equation below:

$$P_u(R_iR_{i+1}) = \frac{P_u(R_iR_{i+1}) - < P_u >}{SD(P_u)} \tag{26}$$

where the symbol <> is averaging the amount present for each of the 16 dinucleotides, (Eq. 23), and SD is the corresponding standard deviation. Table 12 provides the normalized values of $P_u(R_iR_{i+1})$ $(u =$

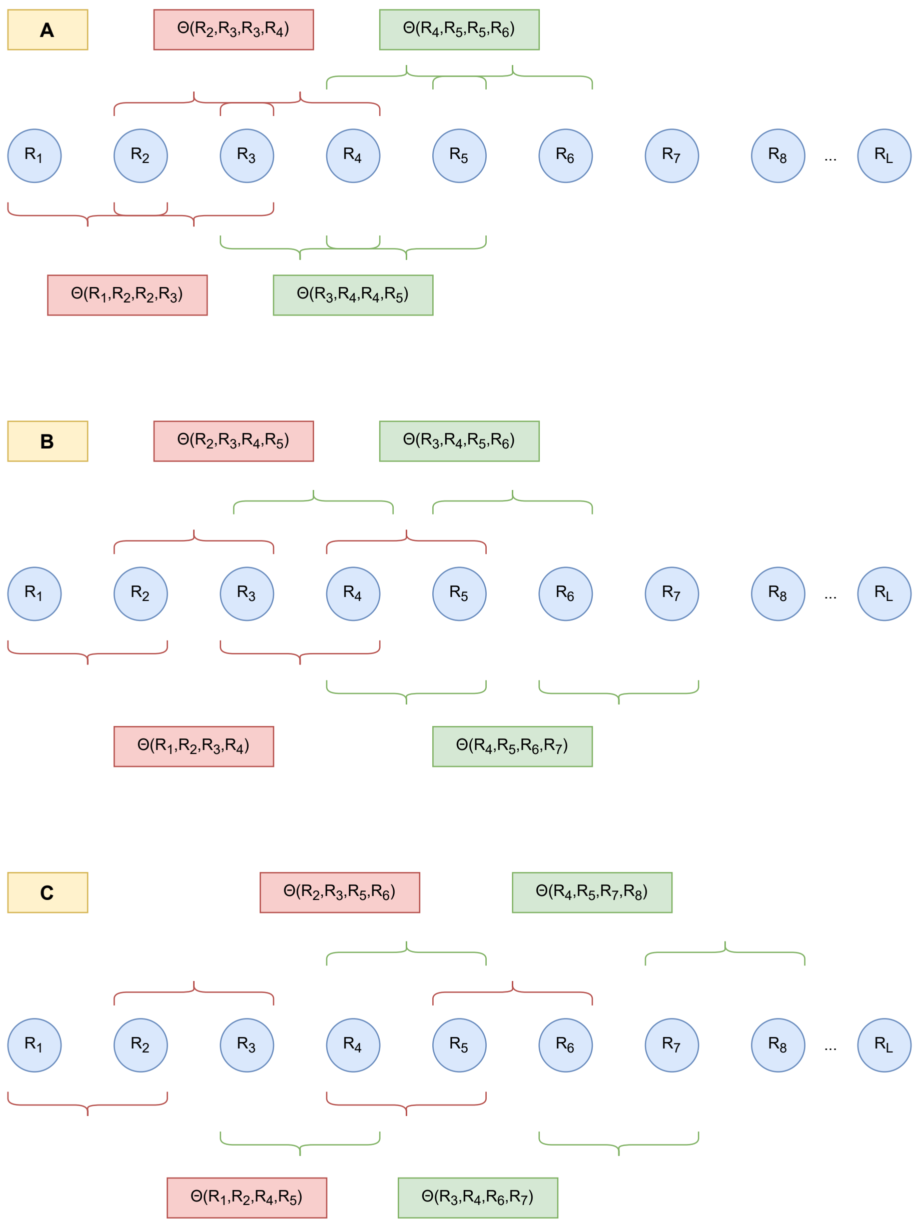


Figure 23: Correlations of dinucleotides along a DNA sequence. Adapted from [91]

Table 11: Original numerical values for the six DNA dinucleotide physical structures [91]

Dinucleotide	$P_1(R_iR_{i+1})$	$P_2(R_iR_{i+1})$	$P_3(R_iR_{i+1})$	$P_4(R_iR_{i+1})$	$P_5(R_iR_{i+1})$	$P_6(R_iR_{i+1})$
AA	0.026	0.038	0.020	1.69	2.26	7.65
AC	0.036	0.038	0.023	1.32	3.03	8.93
AG	0.031	0.037	0.019	1.46	2.03	7.08
AT	0.033	0.036	0.022	1.03	3.83	9.07
CA	0.016	0.025	0.017	1.07	1.78	6.38
CC	0.026	0.042	0.019	1.43	1.65	8.04
CG	0.014	0.026	0.016	1.08	2.00	6.23
CT	0.031	0.037	0.019	1.46	2.03	7.08
GA	0.025	0.038	0.020	1.32	1.93	8.56
GC	0.025	0.036	0.026	1.20	2.61	9.53
GG	0.026	0.042	0.019	1.43	1.65	8.04
GT	0.036	0.038	0.023	1.32	3.03	8.93
TA	0.017	0.018	0.016	0.72	1.20	6.23
TC	0.025	0.038	0.020	1.32	1.93	8.56
TG	0.016	0.025	0.017	1.07	1.78	6.38
TT	0.026	0.038	0.020	1.69	2.26	7.65

$1, 2, ..., 6)$ which were derived using the standard conversion calculation (Eq. 26) from the Table 11 original values.

It is conceivable to draw the conclusion that a collection of sequence-correlation factors $\theta_1, ..., \theta_\lambda$ defined by Eq. 24 and 25, may represent the sequence-order effect of a DNA sequence. The process for augmenting the DNC of Eq. 23 to the PseDNC is similar to that described in [107] for changing the amino acid composition to the Pseudo Amino Acid Composition (PseACC).

$$D = [d_1\ d_2\ ...\ d_{16}\ d_{16+1}\ ...\ d_{16+\lambda}]^T \tag{27}$$

where

$$d_k = \begin{cases} \frac{f_k}{\sum_{i=1}^{16} f_i + w \sum_{j=1}^{\lambda} \theta_j} & 1 \le k \le 16 \\ \frac{w\theta_{k-16}}{\sum_{i=1}^{16} f_i + w \sum_{j=1}^{\lambda} \theta_j} & 17 \le k \le 16 + \lambda \end{cases} \tag{28}$$

where f_k is normalized occurrence frequency of the k-th dinucleotide in the sequence, θ_j is given by

Table 12: The normalized values for the six DNA dinucleotide physical structures [91]

Dinucleotide	$P_1(R_iR_{i+1})$	$P_2(R_iR_{i+1})$	$P_3(R_iR_{i+1})$	$P_4(R_iR_{i+1})$	$P_5(R_iR_{i+1})$	$P_6(R_iR_{i+1})$
AA	0.06	0.5	0.27	1.59	0.11	-0.11
AC	1.50	0.50	0.80	0.13	1.29	1.04
AG	0.78	0.36	0.09	0.68	-0.24	-0.62
AT	1.07	0.22	0.62	-1.02	2.51	1.17
CA	-1.38	-1.36	-0.27	-0.86	-0.62	-1.25
CC	0.06	1.08	0.09	0.56	-0.82	0.24
CG	-1.66	-1.22	-0.44	-0.82	-0.29	-1.39
CT	0.78	0.36	0.09	0.68	-0.24	-0.62
GA	-0.08	0.5	0.27	0.13	-0.39	0.71
GC	-0.08	0.22	1.33	-0.35	0.65	1.59
GG	0.06	1.08	0.09	0.56	-0.82	0.24
GT	1.50	0.50	0.80	0.13	1.29	1.04
TA	-1.23	-2.37	-0.44	-2.24	-1.51	-1.39
TC	-0.08	0.5	0.27	0.13	-0.39	0.71
TG	-1.38	-1.36	-0.27	-0.86	-0.62	-1.25
TT	0.06	0.5	0.27	1.59	0.11	-0.11

Eq. 24, λ represents the number of tiers of the correlations in the sequence and w is the weight factor. As a result, instead of a 16-dimensional vector (Eq. 23), the DNA sequence is formed by a $(16 + \lambda) - D$ vector as stated in Eq. 27. The DNA sequences with very different lengths may be turned into a collection of feature vectors with the same dimension thanks to the extra λ correlation factors, in addition to allowing for the inclusion of significant global sequence-order effects. This last requirement is crucial since many classification engines demand a collection of vectors with a certain number of components as input.

Pseudo K-tupler Composition Pseudo K-Tupler Composition (PseKNC) is the enhanced version of PseDNC by adding k-tuple nucleotide composition.

Given a DNA sequence D (Eq. 13), the feature vector of D is defined:

$$D = [d_1\ d_2\ \dots\ d_{4k}\ d_{4k+1}\ \dots\ d_{4k+\lambda}]^T \tag{29}$$

where

$$d_u = \begin{cases} \dfrac{f_u}{\sum_{i=1}^{4^k} f_i + w \sum_{j=1}^{\lambda} \theta_j} & 1 \leq u \leq 4^k \\ \dfrac{w\theta_{u-4^k}}{\sum_{i=1}^{4^k} f_i + w \sum_{j=1}^{\lambda} \theta_j} & 4^k \leq u \leq 4^k + \lambda \end{cases} \tag{30}$$

where λ is the number of tiers of the correlations in a sequence; f_u is the normalized occurrence frequency of the u-th dinucleotide in the sequence; w is a weight factor; θ_j is given by:

$$\theta_j = \frac{1}{L-j-1} \sum_{i=1}^{L-j-1} \Theta(R_i R_{i+1}, R_{i+j} R_{i+j+1}),\ j \in 1, 2, ..., \lambda; \lambda < L \tag{31}$$

which is the j-tier structural correlation factor between all the j^{th} most contiguous dinucleotides. The correlation function $\Theta(R_i R_{i+1}, R_{i+j} R_{i+j+1})$ is defined by:

$$\Theta(R_i R_{i+1}, R_{i+j} R_{i+j+1}) = \frac{1}{\mu} \sum_{v=1}^{\mu} [P_v(R_i R_{i+1}) - P_v(R_{i+j} R_{i+j+1})]^2 \tag{32}$$

where μ is the number of physicochemical indices reflecting the local DNA structural properties (Table 12); $P_v(R_i R_{i+1})$ is the value of the v^{th} physicochemical indice for the dinucleotide $R_i R_{i+1}$ at i^{th} position and $P_v(R_{i+j} R_{i+j+1})$ is the corresponding value for the dinucleotide $R_{i+j} R_{i+j+1}$ at $(i+j)^{th}$ position.

4.2.2 Encoders

DNA sequences consist of continuous sequential letters from a 4-letter alphabet, and, as mentioned in Section 3.3, it is first required to transform the string sequence into a numerical value in order to create an input matrix for the model. However, as mentioned in Section 4.2, an additional letter was included in the alphabet in order to fill sequences to a determined length - the letter N.

The encoders implemented were one-hot encoding, chemical encoding and k-mer one-hot encoding.

One-Hot Encoding One-hot encoding is extensively used in deep learning models and is well suited for most models. In addition, the performance of one-hot encoding is very stable across various data sets. However, an appropriate model is necessary to get acceptable performance. This approach preserves the positional information of each nucleotide in sequences but disregards high-order relationships between nucleotides and previous biological information [108].

As a result, A is encoded to (1, 0, 0, 0), C to (0, 1, 0, 0), G to (0, 0, 1, 0), T to (0, 0, 0, 1), and N to (0, 0, 0, 0). The final vector's dimension will be $L * 4$ where L is the length of the sequence.

Chemical Encoding The four nucleic acids each have unique chemical characteristics [109]. A and G are purines with two ring structures, whereas C and T are pyrimidines with one ring structure. C and G create strong hydrogen bonds while building secondary structures, whereas A and T form weak hydrogen bonds. In terms of their chemical functionality, A and C belong to the amino group, while G and T belong to the keto group. Accordingly, the four nucleic acids may be categorized into three separate categories (Table 13).

Table 13: Cluster of nucleotides based on chemical properties [109]

Chemical property	**Class**	**Nucleotides**
Ring structure	Purine	A,G
	Pyrimidine	C,T
Hydrogen bond	Weak	A,T
	Strong	C,G
Functional group	Amino	A,C
	Keto	G,T

Three coordinates (x, y, z) were utilized to represent the chemical characteristics of the four nucleotides, and the values 0 and 1 were given to the coordinates in order to incorporate these attributes. If x, y, and z coordinates respectively represent the ring structure, the hydrogen bond, and the chemical functionality, then each nucleotide may be encoded as (x_i, y_i, z_i), where x_i represents the ring structure, y_i represents the hydrogen bond, and z_i represents the chemical functionality.

As a result, A is encoded to (1, 1, 1), C to (0, 0, 1), G to (1, 0, 0), T to (0, 1, 0) and N to (0, 0, 0). The final vector's dimension will be $L * 3$ where L is the length of the sequence.

K-mer One-Hot Encoding As mentioned before, using one-hot encoding on DNA sequences solely preserves the positional information of each nucleotide. Recent investigations, however, have shown that including high-order dependencies among nucleotides may enhance the efficacy of DNA models [108]. To capture the dependencies, all instances are turned into image-like matrices of high-order relationships using the k-mer encoding method.

For example, in 1-mer one-hot encoding, each nucleotide is mapped into a vector of size 5 ($A = [1, 0, 0, 0, 0]^T$, $C = [0, 1, 0, 0, 0]^T$, $G = [0, 0, 1, 0, 0]^T$, $T = [0, 0, 0, 1, 0]^T$, and $N = [0, 0, 0, 0, 1]^T$). The 2-mer one hot encoding is based on the dependencies between two nearby nucleotides, and each dinucleotide is mapped to a 25-dimensional vector ($AA = [1, 0, 0, ..., 0]^T$, ..., $NN = [0, 0, 0, ..., 1]^T$). The 3-mer one hot encoding is based on the dependencies between three nearby nucleotides, and each trinucleotide is mapped to a 125-dimensional vector ($AAA = [1, 0, 0, 0, ..., 0, 0, 0, 0, 0]^T$, ... , $NNN = [0, 0, 0, 0, ..., 0, 0, 0, 0, 0, 1]^T$).

As a result, the final vector's dimension will be $(L - k + 1) * 5^k$ where L is the length of the sequence.

4.3 Classifiers Implementation

Following the selection of the dataset, the DNA sequence records were randomly rearranged and assigned to the training and test/validation sets. A suitable data preprocessing strategy was selected in order to turn the DNA sequences into a numerical representation. This format was necessary to comply with the input requirements of the classification model, which takes only numerical data.

At this stage, there will be either a collection of pre-calculated features, the descriptors, or encoded training data. It is worth noting that if the data is in descriptor form, only shallow ML models will be able to train on it, while DL models need it to be in encoding form.

4.3.1 Models

The following sections will provide an insight for each one of the implemented classification models. The process of obtaining this list of models was based on the most common choices made by the authors of the previous studies on DNA classification (Table 7).

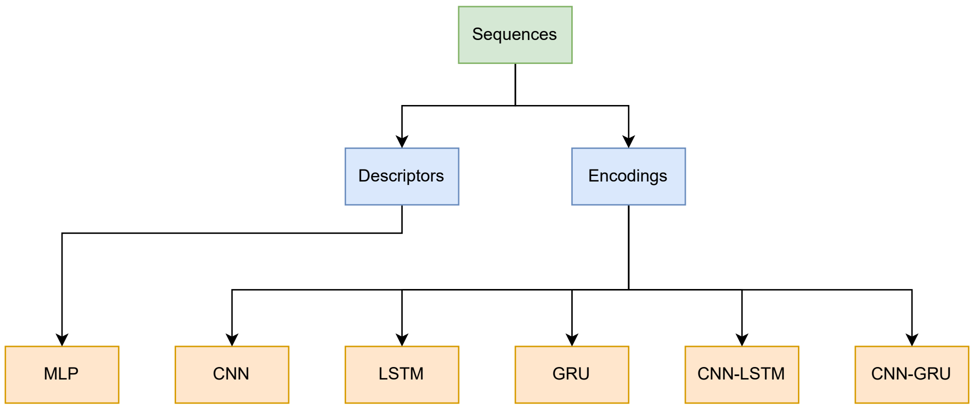


Figure 24: Models and their feature extraction methods.

MLP The first model is a feedforward artificial neural network called Multilayer Perceptron (MLP). This is the only model that can take the descriptors as input data, since it is only the shallow ML model implemented. Figure 25 shows its architecture, which was based in a Zhang et al.'s research [5], one of the cases studies of this thesis.

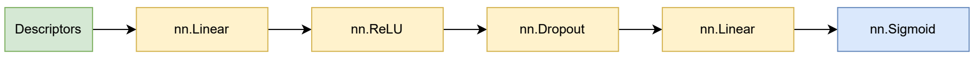


Figure 25: MLP architecture.

CNN The CNN model, which was also inspired by one of this thesis' case studies, specifically Zou et al.'s research [110] and is depicted in Figure 26.

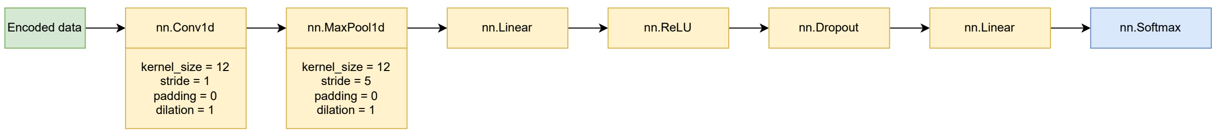


Figure 26: CNN architecture.

LSTM / BiLSTM This LSTM model is a simple model regarding the number of layers, but it is possible to pass to the *nn.LSTM* layer a parameter called num_layers that specifies the number of recurrent layers. Setting $num\_layers = 2$ would result in a stacked LSTM, which would consist of two LSTMs stacked on top of one another. The second LSTM would receive input from the first and compute the final results. Besides, this layer can also take an argument called $bidirectional$, which determines if the LSTM is bidirectional or not. This layer can also take a $dropout$ argument that introduces a dropout layer on the outputs of each LSTM layer except the last one.

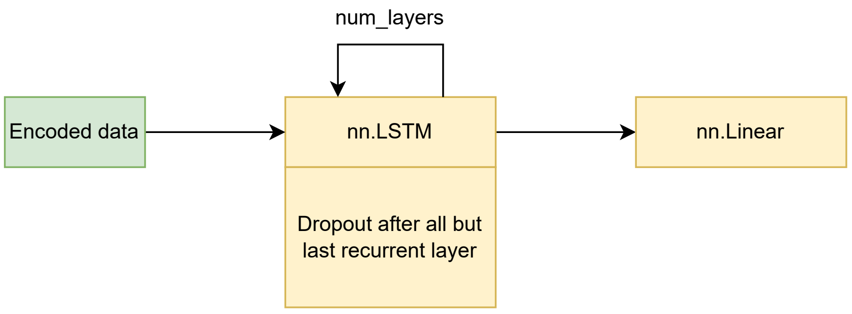


Figure 27: LSTM architecture.

GRU / BiGRU This GRU model is identical to the LSTM's, only changing from the *nn.LSTM* layer to *nn.GRU* one. This way it is possible to directly compare these two types of recurrent neural networks.

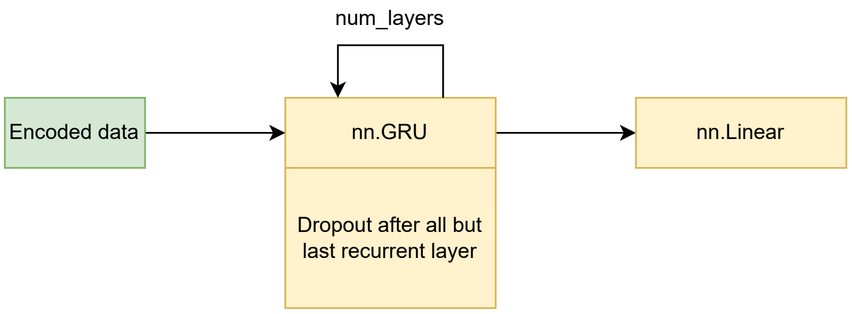


Figure 28: GRU architecture.

CNN-LSTM / CNN-BiLSTM This model is a combination of the CNN and LSTM models. When using the CNN model, the output of the CNN is fed into the LSTM model. The same previously mentioned properties of LSTM are also present (using num_layers to create a stacked LSTM and using $dropout$ to introduce dropout layers on the outputs of each LSTM layers except the last one).

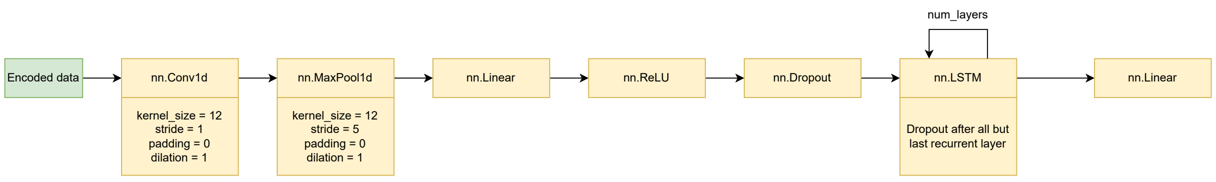


Figure 29: CNN-LSTM architecture.

CNN-GRU / CNN-BiGRU This model is a combination of the CNN and GRU models. When using the CNN model, the output of the CNN is fed into the GRU model. The same previously mentioned properties of GRU are also present (using num_layers to create a stacked GRU and using $dropout$ to introduce dropout layers on the outputs of each GRU layers except the last one).

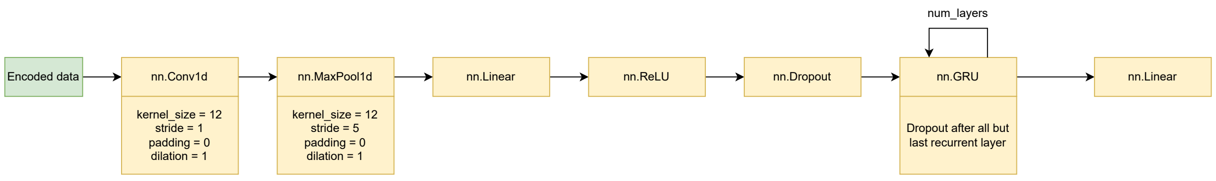


Figure 30: CNN-GRU architecture.

4.3.2 Hyperparameter Tuning

Additionally, hyperparameter optimization was considered and applied successfully. As mentioned in Section 2.2.1, the challenge of hyperparameter optimization is selecting the appropriate hyperparameters for a learning algorithm. It may distinguish an ordinary model from a very accurate one. Choosing a different learning rate or modifying the size of a network layer may have a substantial effect on the performance of the model. Ray Tune[1] is a standard tuning tool for hyperparameters [111] and it was used to complete this task. However, before finding the best combination, it is required to define the configuration of the Ray Tune's search space. The implemented one can be found in Table 14.

Ray Tune will randomly choose a combination of parameters from these search areas for each trial. It will then train many models in parallel and determine which has the highest performance, according to a defined metric. Additionally, the Ray Tune's scheduler *HyperBandScheduler* was used, which terminates poorly performing trials early, using the HyperBand optimization algorithm. This is accomplished by choosing a desired metric (loss) and measuring it at the end of each epoch. If the measure keeps worsening, reaching a specified patience value, the trial will end immediately. This strategy was also implemented in the case

[1]https://docs.ray.io/en/latest/tune/

Table 14: Ray Tune's search space.

Hyperparameter (x)	**Search Space**	**Models**
Hidden Size	$x \in \{32, 64, 128\}$	All
Batch Size	$x \in \{16, 32, 64\}$	All
Learning Rate	$x \in \{0.0001, 0.001, 0.01\}$	All
Dropout	$x \in \{0.2, 0.3, 0.4, 0.5\}$	All
Number of Layers	$x \in \{1, 2, 3\}$	All except MLP and CNN

when hyperparameter tuning is not being performed, using now the *ReduceLROnPlateau* scheduler from *PyTorch*.

Chapter 5

Software Integration

This section provides an overview on how the developed pre-processing tools and ML/DL models were integrated in *ProPythia* and *OmniumAI* software platforms.

5.1 ProPythia

As mentioned in Section 1.1, *ProPythia* [4] is a platform devoted to the classification of peptide/protein sequences using ML and DL, developed within the Biosystems group at CEB/ U. Minho. Included in *ProPythia* are modules to read and modify sequences, calculate various types of protein descriptors, pre-process datasets, execute feature selection and dimensionality reduction, visualize t-SNE and UMAP, perform clustering, train and optimize ML and DL models, and make predictions using various algorithms. *ProPythia* features an adjustable modular design that makes it a flexible and user-friendly tool for ML/DL analysis of protein sequences [4]. A schematic view of the package can be seen in Figure 31.

The first main objective of this study regarding *ProPythia* was to extend its calculation of descriptors to also include DNA descriptors. To accomplish this, a module (similar to the one for proteins) was developed that can be imported by other modules to create a *Python* dictionary containing all calculated descriptors. As the remaining ML steps are independent of one another, *ProPythia* is now capable of performing the whole ML pipeline for DNA data. The list of implemented DNA descriptors can be found later in section 4.2.1.

The other key objective was the development of a complete DL pipeline for the classification of DNA sequences. Although *ProPythia* already includes a module for DL-based classification, it was not used to complete this step since it was not built in *PyTorch*, but rather in *Tensorflow/Keras*. The implemented DL steps were encoding, data processing, model building and training, model evaluation, and, finally, hyperparameter tuning.

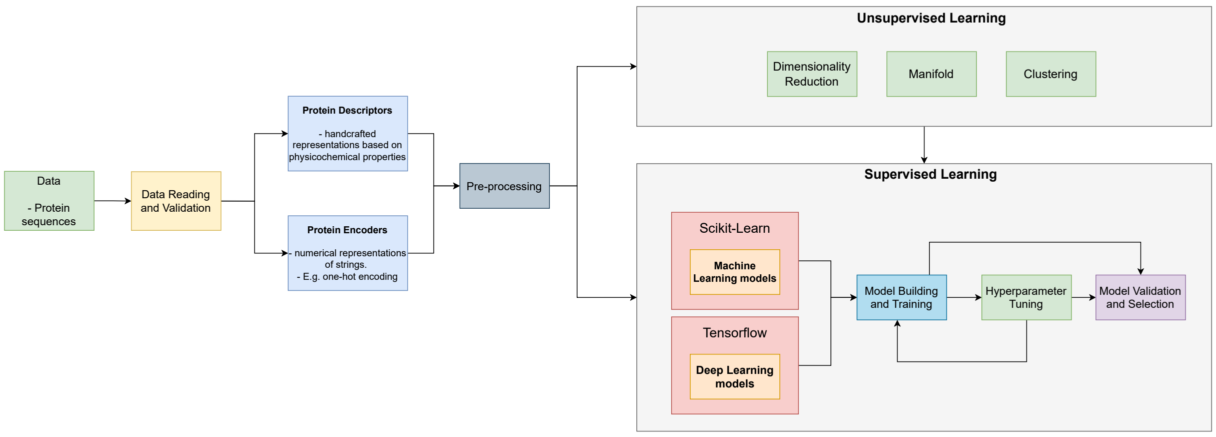


Figure 31: Schematic representation of the modules in ProPythia [4].

These DL steps, along with the calculation of descriptors, were built as separate and independent modules that can be used in combination with other modules, including those developed now and those already present in *ProPythia*. For example, it is possible to use the new DNA encodings to train a *ProPythia*'s DL model and also use *ProPythia*'s protein encoders to train a DL *PyTorch* model. Similarly, it is also possible to use the DNA descriptors to train a *ProPythia*'s shallow ML model and also use *ProPythia*'s protein descriptors to train a shallow ML *PyTorch* model.

Figure 32 provides an overview of the developed workflow.

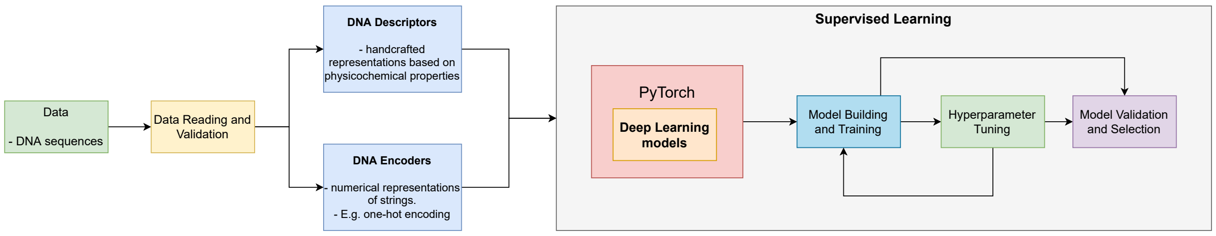

Data
- DNA sequences
Data Reading and Validation
DNA Descriptors
- handcrafted representations based on physicochemical properties
DNA Encoders
- numerical representations of strings.
- E.g. one-hot encoding
Supervised Learning
PyTorch
Deep Learning models
Model Building and Training
Hyperparameter Tuning
Model Validation and Selection

Figure 32: Implemented workflow in ProPythia for Deep Learning DNA classifications.

Figure 33 provides a visualization of how the implemented modules of DL DNA classification is integrated in *Propythia*. The coloured modules were the new ones implemented and the gray ones are from *ProPythia*.

Both implementations of DNA descriptors and the DL pipeline had a step in common: the data reading process. As a result, a data reading module was developed to read DNA sequences from *CSV* or *FASTA* files. The data is then validated, which means that every letter in the sequence has to be either an A, C, G, or T.

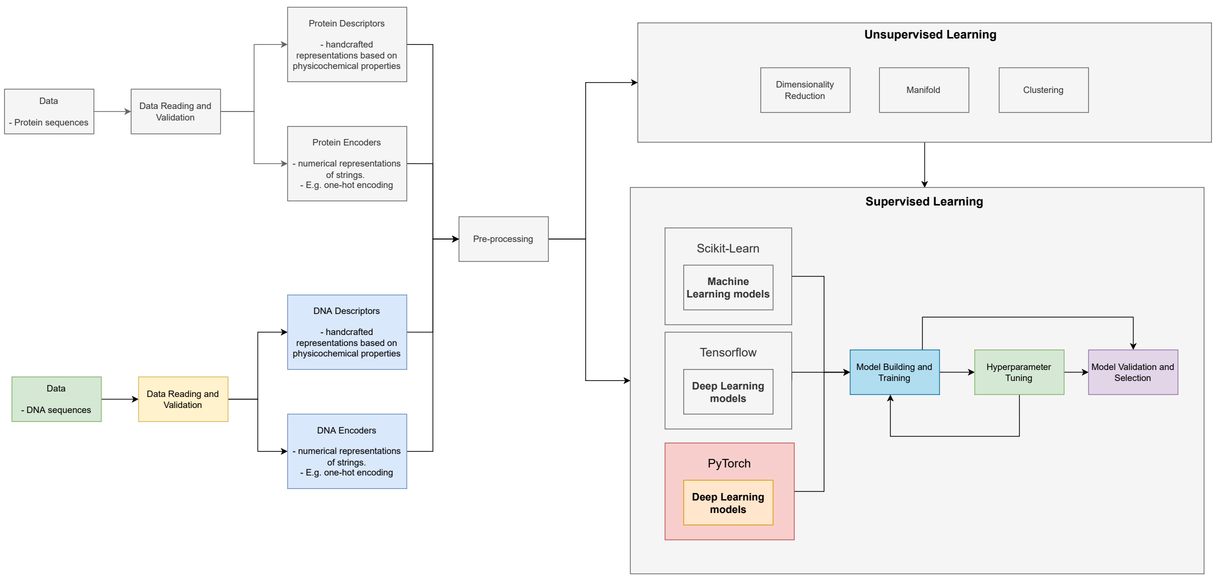


Figure 33: Integration of implemented modules in ProPythia.

5.2 OMNIA

OMNIA is the AutoML platform for bioinformatics of *OmniumAI*. It contains a set of methods for the analysis of biological data. Currently, it offers tools for analyzing the data in the following categories - compounds, proteins, metabolomics, transcriptomics, single-cell transcriptomics and text mining. An overview of this platform is depicted in Figure 34.

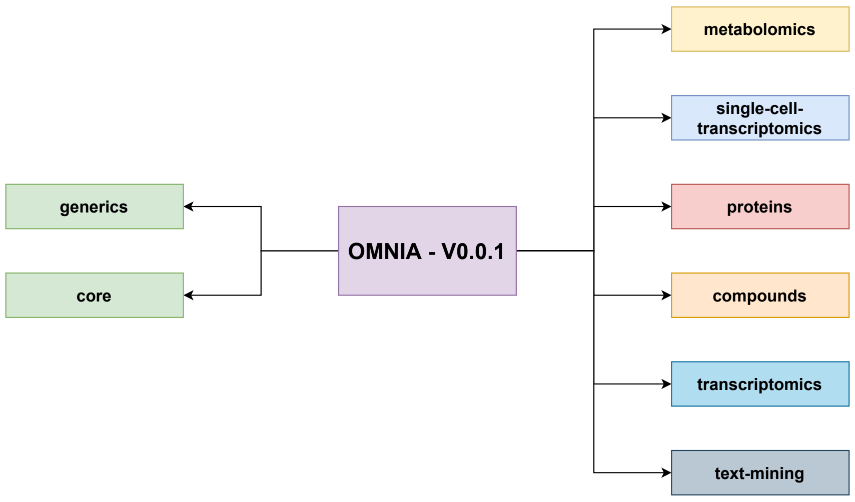


Figure 34: OMNIA overview before implementations.

As it can been observed in Figure 34, there are two modules on the left called *generics* and *core*. These are the fundamental pillars that provide structure for the whole project. The *generics* module contains generic objects, such as dataframe, estimator, and model, that may be used by other packages and are built using Sklearn[1] and Autogluon[2] libraries, which are external dependencies. Then, for the *core* module, the idea was for it to include the interfaces and some generic implementations that have no external dependencies. However, at the moment, the only feature the *core* has are the interfaces. The remaining generic implementations are in *generics* module since they almost always rely on Sklearn or Autogluon. In fact, *core* module will probably be removed from OMNIA.

The first main objective in *OMNIA* was similar to the *ProPythia*'s one, which was the creation of a module that calculates DNA descriptors. This module would serve the same purpose of the previously mentioned ones (proteins, metabolomics, etc), which is a tool to process, in this case, DNA sequences. The implementation from *ProPythia* was re-utilized, but adapted to the *OMNIA* requirements and dependencies.

Then, the other key objective was the addition of DL *PyTorch* models, as well as the functions that allow the *OMNIA* to use them. These functions include train, test, validation and data related processes, which were also implemented with *PyTorch*. Both the models and the functions would be developed in the *generics* module as it is expected that they will be used in all the other modules that handle different types of biological data, such as proteins, metabolomics, etc. Identical to the previous objective, the *PyTorch* models and functions were re-utilized from *ProPythia*, but adapted to the *OMNIA* requirements and dependencies.

Figure 35 illustrates how the *OMNIA* platform incorporates both of these goals. It is also possible to see the models that were already implemented and how the new ones integrate among them in the *generics* module.

Then, figure 36 provides the overview of the entire platform, highlighting the implemented modules and how they integrate in OMNIA. In a nutshell, the *generics* module was extended to include DL *PyTorch* classifications, and an independent biological data processing was added, the *genes* module, that provides the calculation of DNA descriptors.

Similar to *ProPythia*, it is expected that the DNA descriptors and DL models will be used in combination with other models and biological data handling modules. For example, it is possible to use the DNA descriptors to train an *OMNIA*'s shallow ML model and also use the descriptors from *proteins* module to train a shallow ML *PyTorch* model. Likewise, it is also possible to use the DNA encoders to train a *OMNIA*'s DL model and also use the encoders from the *proteins* module to train a DL *PyTorch* model.

[1]https://scikit-learn.org/stable/

[2]https://auto.gluon.ai/stable/index.html

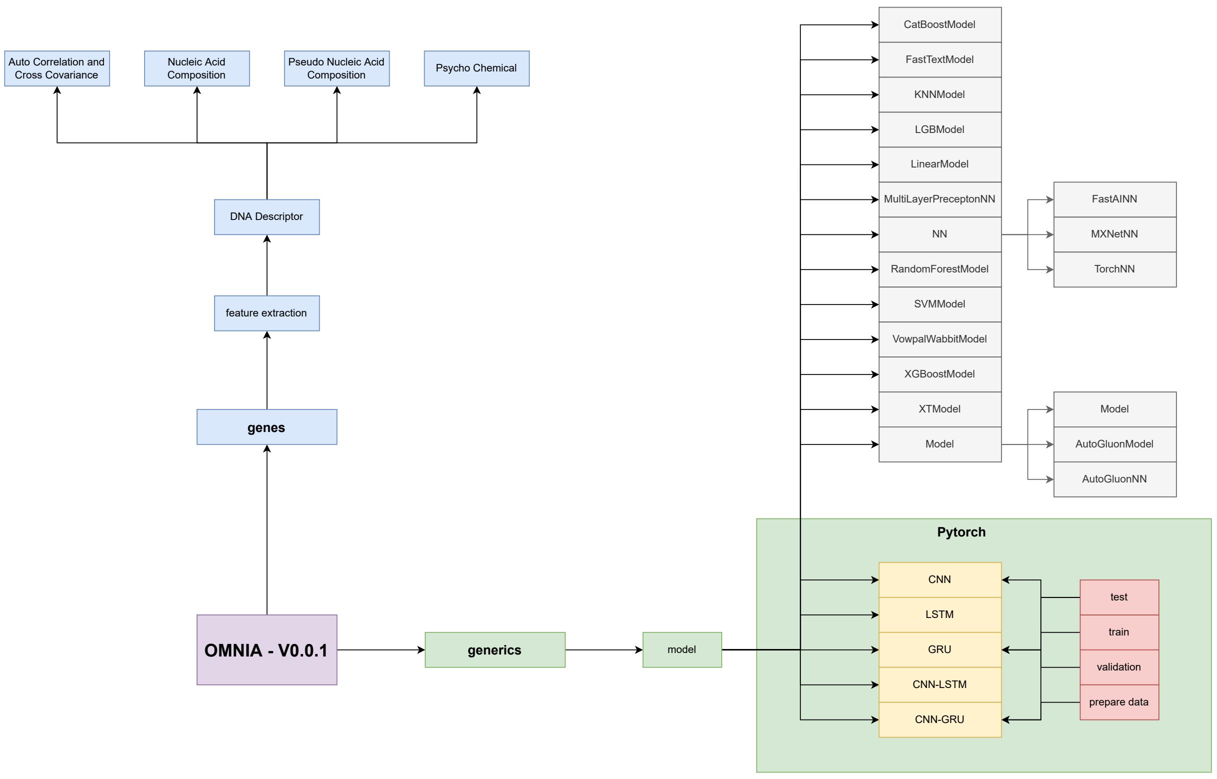


Figure 35: Modules integration in OMNIA.

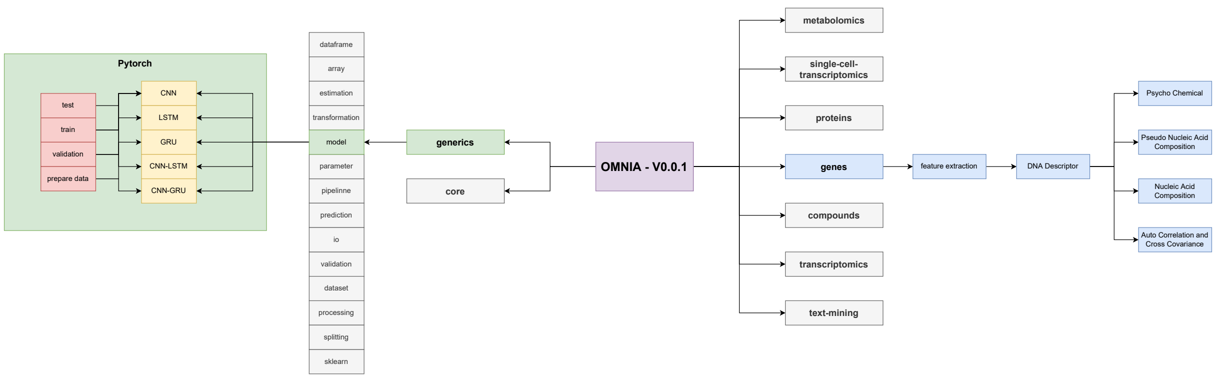


Figure 36: OMNIA overview after implementations.

Chapter 6

Validation/Case studies

The created tools and platform are now functional for testing problems, but they have not yet been tested in real-world scenarios. They have to be validated with real case studies in the areas of biotechnology and health where the application of these tools is relevant. This is the purpose of this chapter.

6.1 The Datasets

In total, 2 datasets were used as case studies and Table 15 provides an overview. The last column contains the dataset size with the number of negative and positive labels.

Table 15: Case studies.

Year	Authors	Title	Focus	Size (Negative/Positive)
2018	Zou et al.	A primer on deep learning in genomics [110]	Discovery of transcription-factor binding sites in DNA.	1013/987
2020	Zhang et al.	DeepHE [5]	Predicting human essential genes based on deep learning.	12624/2010

The first dataset used was derived from a study of Zou et al. [110]. The authors addressed an important problem in functional genomics, which is the discovery of transcription-factor binding sites in DNA. They created a neural network capable of discovering DNA binding motifs based on the results of a test that evaluates whether a longer DNA sequence binds to the protein or not. The dataset has 2000 DNA sequences of length 50, with 987 positive labels and 1013 negative labels. According to the experiments, the model achieved 98% accuracy on the testing data.

The second used in this study was used in a research paper from Zhang et al. The authors tackled the issue that the majority of essential gene prediction approaches based on ML lack the necessary ability to handle the unbalanced learning problem that is inherent in the challenge, which may be one reason impacting their performance. By combining features from sequencing data and protein-protein interaction network, the authors proposed a DL-based approach, *DeepHE*, to predict human essential genes. According to the experiment results, *DeepHE* outperformed ML models, such as SVM, Naive Bayes, RF, and Adaboost, achieving results of 90% accuracy on the testing data.

6.2 Data Collection and Transformation

The first dataset was easily accessible. The article included a link to the accompanying *CSV* file containing the sequences and their labels and no further data cleaning was required.

The second dataset was not as easily accessible. Instead of providing a *CSV* file, the authors specified the names of the databases from which they got the data. They built the essential gene dataset from the *DEG* database [112], and obtained the non-essential gene sequences from *Ensembl* database [113]. The *DEG* database contained 16 different human essential genes datasets, and 8256 human genes are identified as essential in at least one of the 16 datasets. However, the authors assumed that about 10% human genes might be essential genes, and decided to build the essential gene dataset with genes contained at least in 5 datasets, resulting in 2024 sequences.

Then, for the non-essential gene dataset, the authors only stated that they obtained the sequences from *Ensembl*. However, the *Ensembl* database contains many different queries and options, and the authors did not specify which ones they used. This lack of information made it difficult to obtain the exact same dataset as the used on the paper. The only provided information is that they downloaded the dataset from *Ensembl* and, if any of the 8256 annotated essential *DEG* genes were present in the new dataset, they were deleted, resulting in 12697 sequences.

The attempt of replicating the paper's non-essential genes dataset was the following:

1. The Human genes dataset was downloaded from the *Ensembl* 97 release, with unspliced gene and protein coding gene type filters, which resulted in 22722 entries in a *FASTA* file. Each entry is a pair of key and value, with each key being a set of 4 ids (gene stable ID, gene stable ID version, transcript stable ID, transcript stable ID version), and the value being the sequence.

2. Similarly to the study, sequences in this dataset that also appeared in the *DEG* dataset were removed, resulting in 22699 entries.

3. It was noticed that the previous step hardly affected the dataset, but the sequences from *DEG* dataset had an id (EMBL or HGNC ids) which was later found that refered to the gene stable ID of the *Ensembl* dataset. All of the ids from the *Ensembl* dataset that appeared in the *DEG* were removed, along with the sequences associated, resulting in 15888 entries.

4. Then, a cleaning step was performed, where sequences that repeated in the dataset or with invalid characters were removed, resulting in 15137 entries.

As expected, the results of attempting to replicate these steps for the positive and negative datasets were not the same to those in the study. Only 2010 sequences were retrieved for the positive dataset, while 15137 sequences were obtained for the negative dataset. The size of the positive dataset is comparable to that of the study, but the negative still had a relevant discrepancy.

However, it is important to note that the sequences did not have the same length, unlike the two first datasets. Figures 37 and 38 provide a visualization of the sequences length distribution in each dataset.

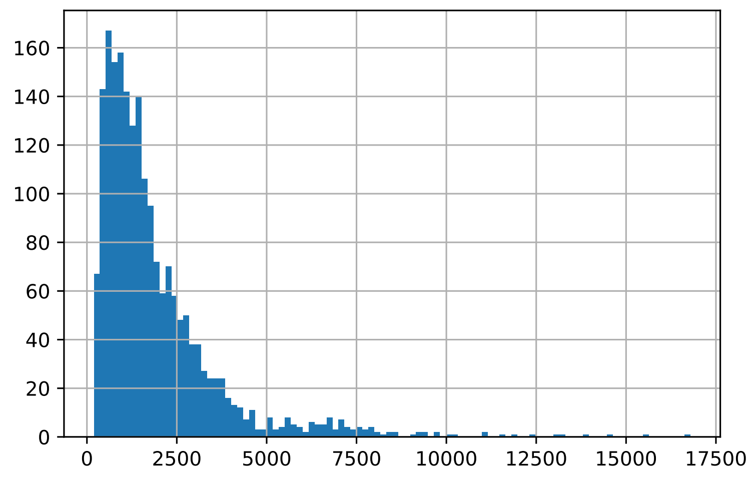


Figure 37: Sequence length and its occurrence in the positive dataset.

A significant disparity in the length of the sequences was found in both datasets, especially in the negative one. The majority of the sequences have a length between 0 and 0.1e6. To remove some data noise, to balance the sequences length distribution and to attempt to achieve a better approximation to the size of the study's negative dataset, sequences that had length bigger than 0.1e6 were deleted, resulting in 12624 sequences. Even though the lengths of the positive dataset are not balanced either, no sequences were removed because the number of sequences was already very close to the paper's.

The final version of this dataset was 2010 positives and 12624 negatives. Even though it is not exactly the same size of the study, it was considered a satisfactory approximation.

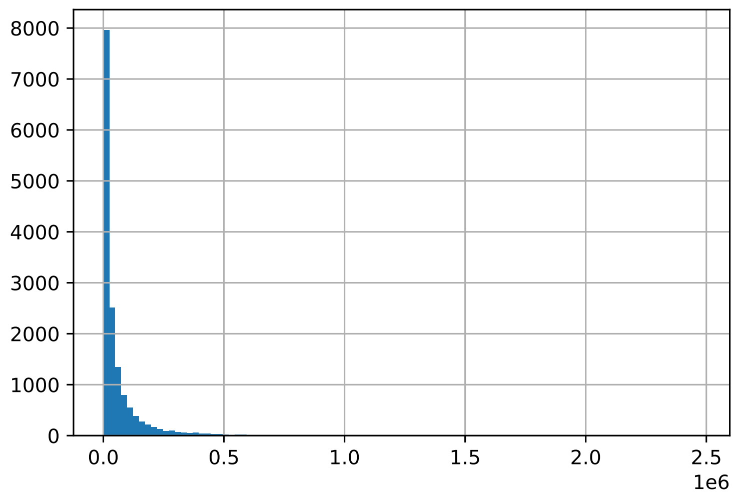


Figure 38: Sequence length and its occurrence in the negative dataset.

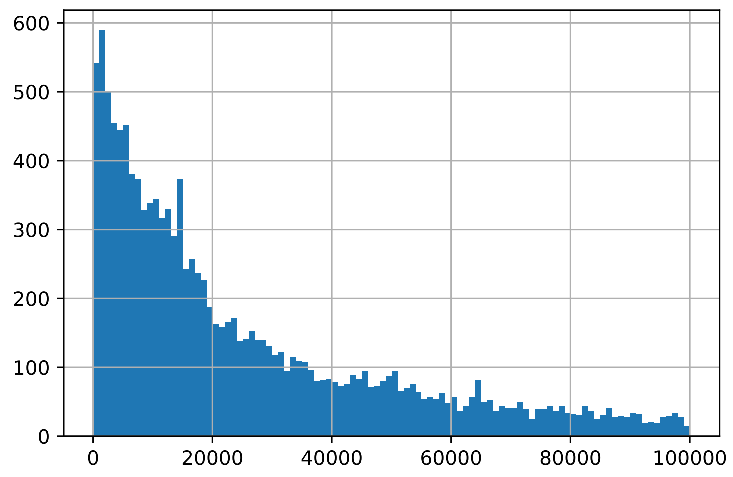


Figure 39: Sequence length and its occurrence after removing sequences bigger than 0.1e6.

As mentioned in Section 4.2, the sequence feature vectors that will be fed into the model need to have the same input shape, regardless of whether the sequences have the same length or not. Unlike the previous datasets, this one did not have sequences with uniform length. When using descriptors, this is not relevant, since they were implemented in a way that they are not dependent on the sequence's length. However, for the encodings, a length value was required, and sequences larger than that value were trimmed, while those shorter than that value were filled with letter N's. Figures 37 and 39 depict the lengths distribution in the positive and negative datasets, respectively, and it can be concluded that the lengths are significantly different between the two. Some computed statistics from the two datasets are shown below in Table 16.

Table 16: Statistics about the positive and negative datasets.

Class	**Smallest length**	**Biggest length**	**Mean length**
Negative	76	99993	25173
Positive	192	16791	1903

One approach would be setting the length to the maximum value, avoiding the cut of sequences. Table 16 reveals, however, that it would add a significant amount of data noise to the dataset, since the largest value (99993) is far from the mean length of both datasets. The length value was chosen to 2000 in order to reduce the number of N's added to sequences. This way, the positive sequences did not suffer much modifications, and most of the negative ones were trimmed. In their investigation, the authors skipped this step since they extracted features from sequences (descriptors) and did not do any encoding calculations.

6.3 Optimal Class Weight

Unlike the first dataset, the second one is greatly unbalanced, with 2010 positive labels and 12624 negative ones. This can be an obstacle for the training of the model, as it is more likely to predict the negative class. To address the uneven data distributions between the two classes, class weights were used to impose a heavier penalty when misclassifying an instance in the minority class, which is the class of essential genes. Experiments were conducted to determine the ideal weight assigned to each class, and can be found below in Table 17.

As the objective is to discover the optimal class weight to train the models for the essential genes dataset, it would be pointless to test all possible combinations of feature extractions techniques and models, as doing so would create an enormous number of possibilities. So, there had to be a selection of feature extractions methods as well as models to test the different weight classes. For feature extraction, only descriptors and one-hot encoding were selected, and for models, the MLP, CNN, LSTM, and GRU were picked, as the remaining models are built using one or more of these four.

Table 17: Results of different weights in the essential genes dataset.

Model	Feature Extraction	Weights	Accuracy	MCC	Confusion Matrix
MLP	Descriptor	1:1	0.978	0.911	$\begin{bmatrix} 2485 & 40 \\ 23 & 379 \end{bmatrix}$
MLP	Descriptor	1:2	0.979	0.915	$\begin{bmatrix} 2482 & 43 \\ 18 & 384 \end{bmatrix}$
MLP	Descriptor	**1:3**	**0.980**	**0.918**	$\begin{bmatrix} 2487 & 38 \\ 20 & 382 \end{bmatrix}$
MLP	Descriptor	1:4	0.978	0.913	$\begin{bmatrix} 2477 & 48 \\ 15 & 387 \end{bmatrix}$
CNN	One-Hot	**1:1**	**0.964**	**0.845**	$\begin{bmatrix} 2478 & 47 \\ 59 & 343 \end{bmatrix}$
CNN	One-Hot	1:2	0.956	0.815	$\begin{bmatrix} 2459 & 66 \\ 63 & 339 \end{bmatrix}$
CNN	One-Hot	1:3	0.957	0.823	$\begin{bmatrix} 2454 & 71 \\ 54 & 348 \end{bmatrix}$
CNN	One-Hot	1:4	0.945	0.785	$\begin{bmatrix} 2413 & 112 \\ 49 & 353 \end{bmatrix}$
LSTM	One-Hot	1:1	0.974	0.899	$\begin{bmatrix} 2459 & 66 \\ 10 & 392 \end{bmatrix}$
LSTM	One-Hot	**1:2**	**0.978**	**0.907**	$\begin{bmatrix} 2490 & 35 \\ 30 & 372 \end{bmatrix}$
LSTM	One-Hot	1:3	0.972	0.890	$\begin{bmatrix} 2459 & 66 \\ 16 & 386 \end{bmatrix}$
LSTM	One-Hot	1:4	0.976	0.908	$\begin{bmatrix} 2465 & 60 \\ 9 & 393 \end{bmatrix}$
GRU	One-Hot	**1:1**	**0.982**	**0.925**	$\begin{bmatrix} 2497 & 28 \\ 24 & 378 \end{bmatrix}$
GRU	One-Hot	1:2	0.979	0.913	$\begin{bmatrix} 2490 & 35 \\ 26 & 376 \end{bmatrix}$
GRU	One-Hot	1:3	0.980	0.916	$\begin{bmatrix} 2485 & 40 \\ 20 & 382 \end{bmatrix}$
GRU	One-Hot	1:4	0.981	0.920	$\begin{bmatrix} 2484 & 41 \\ 16 & 386 \end{bmatrix}$

By reviewing the data in Table 17, one could conclude that class weights have a minimal effect on the performance of each model. In fact, in certain models, the best results can be achieved without any class weight distribution at all. Due to the robustness of the models, it is reasonable to infer that the class imbalance is not a significant concern. In addition, it can be inferred that, with the exception of the CNN model, every other model achieves roughly the same level of accuracy regardless of the weight distribution, confirming the belief that the class imbalance is not a significant issue.

Therefore, no weight class distribution was selected for training the remaining feature extraction techniques and model combinations in the essential genes dataset, since it was the best performing in half of the models and also the one that obtained the highest value of MCC.

6.4 Results

To acquire the best possible results, it was necessary to test every combination of feature extraction - model - dataset. There are a total of four types of feature extraction techniques, ten distinct models, and two datasets. This means that, so far, $4 * 10 * 2 = 80$ possibilities would need testing.

However, one of the feature extraction methods, the k-mer one-hot encoding, receives a parameter k defined by the user. The number of possibilities will rapidly increase since every value of k would have to be tested, and $0 < k < L$, where L is the length of the sequence. Given a sequence with length 2000 (sequences' length in the essential genes dataset), the number of possibilities that would need testing would be $(3 + 2000) * 10 * 2 = 40060$. Due to the practical impossibility of testing this number of possibilities, a maximum value for k had to be established. It is also important to mention that the bigger the value of k, the larger the feature vector will be. With $k = 10$, every word of 10 letters (AAAAAAAAAA, AAAAAAAAAT, ..., NNNNNNNNNN) had to be calculated and assigned a unique array to represent it. This would result in 9765625 possible combinations, with each one assigned to an array of also $5^{10} = 9765625$ elements (containing exclusively zeros with the exception of a single 1 at a single position in the array). Then, the sequence had to be separated into $L - k + 1 = 2000 - 10 + 1 = 1991$ 10-mers and assign the respective array of 9765625 elements, resulting in a vector size of $1991 * 9765625 = 1.94e10$.

The time necessary to create this encoding in addition to the time required for the model to process all of this input and train with it makes large values of k impractical. This was taken into account while determining the maximum value of k, so that this value would be related with the maximum acceptable amount of time to wait for classification results. Some experiments to find the maximum value of k were performed in the essential genes dataset, which is the dataset with the largest sequences, and the CNN model, which is the DL model with the fastest train times. They can be found below in Table 18.

As anticipated, both the size of the feature vector and the training time for the model increase exponentially with the value of k. In fact, the vector size is so large that it was not possible to train the model for $k = 4$ due to limited computation resources, specifically memory. Considering the results of Table 18, it was determined that the maximum value of k would be 3.

So, at this point, there are 5 possible feature extraction methods (descriptors, one-hot, chemical, k-mer one-hot with $k = 2$, and k-mer one-hot with $k = 3$), 10 different models and 2 datasets, resulting in 100 possibilities that would need testing. However, some of these combinations are incompatible (some

Table 18: Statistics about the k-mer one-hot encoding on the essential genes dataset using the CNN model.

K	Feature Vector Size of a single sequence	Time to train (min)
1	$(2000 - 1 + 1) * 5^1 = 10\ 000$	3
2	$(2000 - 2 + 1) * 5^2 = 49\ 975$	6
3	$(2000 - 3 + 1) * 5^3 = 249\ 750$	27
4	$(2000 - 4 + 1) * 5^4 = 1\ 248\ 125$	Out of memory

models only accept encodings and vice-versa). Therefore, 74 combinations were examined, and the results are shown below in Table 19 and in Table 20. It also worth mentioning that these results were obtained using the cross entropy loss function, which is the default loss function to use for binary classification problems [114], and the Adam optimizer, which is one of the most popular optimizers and was the one used in both case studies.

Also, in unbalanced dataset problems, the accuracy is not a good indicator of the performance of the model, since it can be easily skewed by the majority class. A dataset containing, for example, 100 sequences, 90 of which are negatively labeled and 10 of which are positively labeled, a model that always predicts false would achieve an accuracy of 90%. However, this model would be useless in practice. The MCC and confusion matrices were also calculated for each combination, since they are a more accurate measure of the performance of the model.

Tables 19 and 20 only provide the accuracy metric in order to compare the results of both case studies, but the complete results for both datasets with other metrics (MCC and confusion matrix) can be found in the supplementary material in Appendix A (Table 22 and 23).

Analyzing both tables, it can be concluded that all feature extraction methods and models performed well in both datasets. The results in the Primer dataset are better than the Essential Genes ones, which is expected since the results from both studies also show that the Primer dataset is easier to classify than the Essential Genes one (the authors from the Primer study achieved 98% accuracy and the authors from the Essential Genes study achieved 90% accuracy). The results obtained outperformed the results from both studies, achieving 100% accuracy on the Primer dataset using LSTM, BiLSTM, GRU and BiGRU models, and 98% accuracy on the Essential Genes dataset using BiLSTM/3-mer one-hot combination.

The tables show that RNN (LSTM, GRU, and their variations) models are the best choice for this problem, since they achieved the best results in both datasets. RNN models are well-known for their capacity to process sequential data, which is most likely why these models produced the best results.

Table 19: Accuracy results on Primer dataset.

Model	**Descriptors**	**One-hot**	**Chemical**	**2-mer one-hot**	**3-mer one-hot**
MLP	0.958	—	—	—	—
CNN	—	0.990	0.988	0.985	0.995
LSTM	—	0.998	0.998	1	1
BiLSTM	—	1	1	1	1
GRU	—	1	1	1	0.995
BiGRU	—	0.998	1	0.993	1
CNN-LSTM	—	0.978	0.985	0.990	0.998
CNN-BiLSTM	—	0.988	0.960	0.995	0.998
CNN-GRU	—	0.985	0.995	0.995	0.985
CNN-BiGRU	—	0.993	0.990	0.995	0.998

Table 20: Accuracy results on Essential Genes dataset.

Model	**Descriptors**	**One-hot**	**Chemical**	**2-mer one-hot**	**3-mer one-hot**
MLP	0.978	—	—	—	—
CNN	—	0.964	0.950	0.965	0.978
LSTM	—	0.974	0.976	0.981	0.982
BiLSTM	—	0.984	0.972	0.981	**0.986**
GRU	—	0.982	0.975	0.978	0.985
BiGRU	—	0.976	0.976	0.982	0.983
CNN-LSTM	—	0.958	0.942	0.971	0.973
CNN-BiLSTM	—	0.960	0.936	0.967	0.976
CNN-GRU	—	0.963	0.943	0.970	0.978
CNN-BiGRU	—	0.962	0.946	0.971	0.977

6.5 Results Reproducibility

A machine learning pipeline has several nondeterministic stages. For instance, the training and testing sets resulting from data splitting are very likely to change across runs. This is problematic since it makes it hard to replicate results.

However, this problem is easily tackled by using the random seed that libraries like TensorFlow and PyTorch provide. By adjusting the random seed value, it is possible to assure that random numbers are created consistently across runs. This concept may seem counterintuitive at first, since one could believe that adjusting the random seed would manipulate the algorithm's randomness. This, however, is not true. The random seed is used to guarantee that the same random numbers are produced every time, and if the random seed value is altered, so will the random numbers. This is a highly helpful feature that can help in reproducing the results, and it is a common practice in machine learning.

This line of thought can raise the question of how to choose the random seed value for optimal results. The answer to this question is that there is no single answer. The random seed value must be initially set and it is common practice to experiment with multiple values. It is, however, expected that the random seed value will not have a significant impact on the results and that the results will be similar across different values.

In this study, three different random seed values were tested: 24, 42, and 2022. The results from the previous section were obtained using the random seed value of 42. To see if the results are consistent across different random seed values, the same experiment was repeated using the other two random seed values, but only using the MLP and CNN models to reduce the number of experiments as they are already a good indicator of the consistency of the results. The results in the essential genes dataset are shown in Table 21.

By analyzing Table 21, it is possible to deduce that the results obtained with the other two random seed values were quite comparable to those obtained with random seed value 42. In addition, the best results are obtained using the MLP model and the CNN/3-mer one-hot combination in all three seeds, which proves the consistency of the models as well. It is thus feasible to infer that the random seed value has no significant effect on the results.

Table 21: Results on Essential Genes dataset with different seeds.

Seed	**Model**	**Feature Extraction**	**Accuracy**	**MCC**	**Confusion Matrix**
24	MLP	Descriptor	0.980	0.917	$\begin{bmatrix}2495 & 30\\28 & 374\end{bmatrix}$
24	CNN	One-Hot	0.970	0.874	$\begin{bmatrix}2482 & 43\\44 & 358\end{bmatrix}$
24	CNN	Chemical	0.939	0.720	$\begin{bmatrix}2498 & 27\\151 & 251\end{bmatrix}$
24	CNN	2-mer one-hot	0.978	0.908	$\begin{bmatrix}2499 & 26\\37 & 365\end{bmatrix}$
24	CNN	3-mer one-hot	0.980	0.914	$\begin{bmatrix}2494 & 31\\29 & 373\end{bmatrix}$
42	MLP	Descriptor	0.978	0.911	$\begin{bmatrix}2485 & 40\\23 & 379\end{bmatrix}$
42	CNN	One-Hot	0.964	0.845	$\begin{bmatrix}2478 & 47\\59 & 343\end{bmatrix}$
42	CNN	Chemical	0.950	0.782	$\begin{bmatrix}2476 & 49\\96 & 306\end{bmatrix}$
42	CNN	2-mer one-hot	0.965	0.855	$\begin{bmatrix}2462 & 63\\40 & 362\end{bmatrix}$
42	CNN	3-mer one-hot	0.978	0.912	$\begin{bmatrix}2483 & 42\\21 & 381\end{bmatrix}$
2022	MLP	Descriptor	0.981	0.920	$\begin{bmatrix}2501 & 24\\31 & 371\end{bmatrix}$
2022	CNN	One-Hot	0.954	0.796	$\begin{bmatrix}2488 & 37\\98 & 304\end{bmatrix}$
2022	CNN	Chemical	0.942	0.742	$\begin{bmatrix}2464 & 61\\110 & 292\end{bmatrix}$
2022	CNN	2-mer one-hot	0.972	0.877	$\begin{bmatrix}2498 & 27\\56 & 346\end{bmatrix}$
2022	CNN	3-mer one-hot	0.981	0.920	$\begin{bmatrix}2500 & 25\\30 & 372\end{bmatrix}$

Chapter 7

Conclusion

7.1 Summary of the work

The main goal of this study consists in creating a solution to keep up with the exponential biomedical data growth, which current methods involve the use of homologies, a very slow and expensive process.

The solution takes the form of a tool that uses ML and DL models to automatically classify DNA sequences, improving response times and even classification accuracy. It is then integrated into *ProPythia* and *OmniumAI* software platforms.

Some previous studies have been done in this field, but the objective is for the tool to generalize well to data from different studies and sources, not just one in particular. The objectives outlined in Chapter 1 were taken into consideration when the problem's solution was being developed.

It was required to first understand the state of the art regarding the creation of ML/DL models, in a broad sense, and also specifically classifiers for DNA sequences. Then, for the DNA sequence classifiers, it was also important to find the current approaches to the data preprocessing step, which is the most crucial step of this study due to the lack of numerical properties in the sequence that the model requires.

After reviewing the state of the art, it was necessary to decide which pre-processing methods and which ML/DL models were going to be built and used. This challenge was tackled in both situations in the same manner, by picking the most common or best performing ones from previous studies in DNA classification problems. Descriptors were calculated for the shallow ML models (the MLP model), as well as encodings for the DL models (CNN, LSTM, BiLSTM, GRU, BiGRU, CNN-LSTM, CNN-BiLSTM, CNN-GRU, and CNN-BiGRU).

Following the implementation, the next step was to integrate it into *ProPythia* and into *OmniumAI* software platforms, in particular *OMNIA*. In *ProPythia*, the objectives were to extend its calculation of descriptors for proteins to also include DNA descriptors and also the development of a complete DL pipeline (data reading and validation, encodings, DL models, model training and building, hyparameter tuning and model validation and selection) for the classification of DNA sequences. Then, for *OMNIA*, the objectives were similar, which were the creation of a module that calculates DNA descriptors and the addition of DL models as well as train, test, validation and data related processes functions.

Finally, the implemented tool needed to be tested with real-world case studies. The main goal of this step was to evaluate the performance of the implemented tool and to compare it with the state of the art. The expected results are that the tool will be able generalize well to new data from different sources and that it will be able to classify DNA sequences with high accuracy. The case studies were transcription factor annotation and essential gene determination.

7.2 Discussion on the main results

The primary goal of this dissertation was to develop a tool that can automatically classify DNA sequences using ML/DL algorithms.

This objective was materialized by the tool created, which is able to replicate and even outperform results from relevant previous studies. It is worth mentioning that there were several decisions, regarding the choice of important parameters and configurations, that may have impacted the outcomes, which are listed below.

- Loss function;
- Optimizer;
- Early stopping patience;
- Sequences cutting length for the essential genes dataset;
- Number of epochs;
- Model's architecture;
- Model's parameters (strides, paddings);
- Hyperparameter search space;
- Number of samples in the Hyperparameter Tuning.

All of the experiments were performed with a fixed value for each one of the above parameters, because it was not the goal of this work to find the best possible model combination, but to compare the performance of the different pre-processing methods and models, and understand if they can generalize well. In fact, it was not feasible to experiment all the possible combinations of the above parameters, and the ones that were chosen were the ones that were considered to be the best fit for the problem at hand. For example, the early stopping patience value, which is the threshold for the number of epochs without improvement in the validation loss, was set to 2 because it was observed that the model already converged at this point.

7.3 Future Work

For the future work, one of the improvements for the developed tool would be the support for multiclass classification problems. As can be seen in Section 6.1, the datasets from the case studies are binary classification problems, which means that the model will predict if a sequence will belong to a certain class or not. Multiclass classification support was not thoroughly investigated due to the lack of accessible datasets from relevant studies. However, given the tool's level of abstraction and modularization, extending it to handle multiclass classifications would not be a particularly laborious task. Training, testing, and metric calculation methods are the only components that would need to be adapted for this to be implemented.

Another enhancement to the tool would be the development of a graphical user interface, such as a publicly accessible web application. Since the only means to engage with the system is through the command line, which needs some level of technical knowledge, it would be ideal to have an intuitive interface that would enable all users to interact, regardless of their level of technical expertise. Most of the DNA descriptors packages mentioned in Table 5 provide a web server to ease and enhance the users' experience. For instance, this web application could be implemented using the Python-based Flask[1] framework for the *backend*, and React.js[2] or Vue.js[3] for the *frontend*. The web application would be able to receive the DNA sequences from the users, allow them to choose the pre-processing methods and ML/DL models, and return the results in a user-friendly format.

The last improvement would be to this research results, and not to the developed tool itself. As mentioned in the previously section, the experiments were performed with a fixed value for a set of parameters. Among them, the hyperparameter search space and the number of samples in the hyperparameter tuning are the ones with most potential to improve even further the results. The hyperparameter search space is the set of values that the hyperparameters can take, and the number of samples is the number of combinations of hyperparameters that will be tested. The more combinations that are tested, the more likely it is that the best combination will be found. However, the more combinations that are tested, the longer the hyperparameter tuning will take. Therefore, it was important to find a balance between the number of combinations and the time it takes to perform the hyperparameter tuning. If greater computer resources were available, it would be conceivable to obtain faster training times and to test even more

[1]https://flask.palletsprojects.com
[2]https://reactjs.org/
[3]https://vuejs.org/

combinations of hyperparameters in order to boost the results even more. The lack of computational resources, particularly memory, was also the reason why the k-mer one-hot encoding method was not tested for k values greater than 3, as mentioned in Section 6.4. The k-mer one-hot encoding method is the most computationally expensive pre-processing method but also the one that obtained the best results. Although the results were already excellent and probably could not be improved much more, it still would be interesting to explore more tuning combinations and test the k-mer one-hot encoding method with k values greater than 3 to see how it would impact the results.

Bibliography

[1] N. Auslander, A. B. Gussow, and E. V. Koonin. “Incorporating Machine Learning into Established Bioinformatics Frameworks.” In: *International Journal of Molecular Sciences 2021, Vol. 22, Page 2903* 22.6 (Mar. 2021), p. 2903. issn: 14220067. doi: 10.3390/IJMS22062903. url: https://www.mdpi.com/1422-0067/22/6/2903/htmhttps://www.mdpi.com/1422-0067/22/6/2903.

[2] A. Yang, W. Zhang, J. Wang, K. Yang, Y. Han, and L. Zhang. “Review on the Application of Machine Learning Algorithms in the Sequence Data Mining of DNA.” In: *Frontiers in Bioengineering and Biotechnology* 8 (Sept. 2020), p. 1032. issn: 22964185. doi: 10.3389/FBIOE.2020.01032.

[3] J. Zrimec, F. Buric, M. Kokina, V. Garcia, and A. Zelezniak. “Learning the Regulatory Code of Gene Expression.” In: *Frontiers in Molecular Biosciences* 8 (June 2021), p. 530. issn: 2296889X. doi: 10.3389/FMOLB.2021.673363/BIBTEX.

[4] A. M. Sequeira, D. Lousa, and M. Rocha. “ProPythia: A Python Automated Platform for the Classification of Proteins Using Machine Learning.” In: *Advances in Intelligent Systems and Computing* 1240 AISC (June 2020), pp. 32–41. issn: 21945365. doi: 10.1007/978-3-030-54568-0{_}4. url: https://link.springer.com/chapter/10.1007/978-3-030-54568-0_4.

[5] X. Zhang, W. Xiao, and W. Xiao. “DeepHE: Accurately predicting human essential genes based on deep learning.” In: *PLOS Computational Biology* 16.9 (Sept. 2020), e1008229. issn: 1553-7358. doi: 10.1371/JOURNAL.PCBI.1008229. url: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1008229.

[6] D. Quang and X. Xie. “DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences.” In: *Nucleic Acids Research* 44.11 (June 2016), e107–e107. issn: 0305-1048. doi: 10.1093/NAR/GKW226. url: https://academic.oup.com/nar/article/44/11/e107/2468300.

[7] G. Novakovsky, M. Saraswat, O. Fornes, S. Mostafavi, and W. W. Wasserman. “Biologically relevant transfer learning improves transcription factor binding prediction.” In: *Genome biology* 22.1 (Dec. 2021). issn: 1474-760X. doi: 10.1186/S13059-021-02499-5. url: https://pubmed.ncbi.nlm.nih.gov/34579793/.

[8] *Total data volume worldwide 2010-2025 | Statista*. url: https://www.statista.com/statistics/871513/worldwide-data-created/.

[9] IBM Cloud Education. *What is Machine Learning?* url: https://www.ibm.com/cloud/learn/machine-learning.

[10] M. Luckert and M. Schaefer-Kehnert. "Using Machine Learning Methods for Evaluating the Quality of Technical Documents." In: (2016). url: http://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-52087.

[11] A. L. Samuel. "Some Studies in Machine Learning Using the Game of Checkers." In: *undefined* 3.3 (July 1959), pp. 210–229. issn: 0018-8646. doi: 10.1147/RD.33.0210. url: http://ieeexplore.ieee.org/document/5392560/.

[12] S. Kassel. "Predicting Building Code Compliance with Machine Learning Models | Azavea." In: (Sept. 2017). url: https://www.azavea.com/blog/2017/09/21/building-inspection-prediction/.

[13] *Machine Learning Glossary | Google Developers*. url: https://developers.google.com/machine-learning/glossary.

[14] S. M. Karazi, M. Moradi, and K. Y. Benyounis. "Statistical and Numerical Approaches for Modeling and Optimizing Laser Micromachining Process-Review." English. In: *Reference Module in Materials Science and Materials Engineering*. Reference Module in Materials Science and Materials Engineering. Netherlands: Elsevier, 2019. doi: 10.1016/B978-0-12-803581-8.11650-9. url: https://linkinghub.elsevier.com/retrieve/pii/B9780128035818116509.

[15] M. G. Omran, A. P. Engelbrecht, and A. Salman. "An overview of clustering methods." In: *Intelligent Data Analysis* 11.6 (2007), pp. 583–605. issn: 15714128. doi: 10.3233/IDA-2007-11602.

[16] L. Van Der Maaten, E. Postma, J. den Herik, and others. "Dimensionality reduction: a comparative." In: *J Mach Learn Res* 10.66-71 (2009), p. 13. url: https://members.loria.fr/moberger/Enseignement/AVR/Exposes/TR_Dimensiereductie.pdf.

[17] A. Beck and M. Kurz. *A Perspective on Machine Learning Methods in Turbulence Modelling*. Nov. 2020. doi: 10.13140/RG.2.2.17469.69608.

[18] J. Chugh. *Types of Machine Learning and Top 10 Algorithms Everyone Should Know*. Dec. 2018. url: https://blogs.oracle.com/ai-and-datascience/post/types-of-machine-learning-and-top-10-algorithms-everyone-should-know.

[19] *What is Unsupervised Learning? | IBM*. Sept. 2020. url: https://www.ibm.com/cloud/learn/unsupervised-learning?mhsrc=ibmsearch_a&mhq=unsupervised%20learning.

[20] R. Beaumont. *Image embeddings | Medium*. July 2020. url: https://rom1504.medium.com/image-embeddings-ed1b194d113e.

[21] L. Patcher. *What is principal component analysis? | Bits of DNA*. May 2014. url: https://liorpachter.wordpress.com/2014/05/26/what-is-principal-component-analysis/.

[22] Q. Liu and Y. Wu. “Supervised Learning.” In: *Encyclopedia of the Sciences of Learning* (2012), pp. 3243–3245. doi: 10.1007/978-1-4419-1428-6{_}451. url: https://link.springer.com/referenceworkentry/10.1007/978-1-4419-1428-6_451.

[23] I. H. Sarker. “Machine Learning: Algorithms, Real-World Applications and Research Directions.” In: *SN Computer Science 2021 2:3* 2.3 (Mar. 2021), pp. 1–21. issn: 2661-8907. doi: 10.1007/S42979-021-00592-X. url: https://link.springer.com/article/10.1007/s42979-021-00592-x.

[24] Y. Matanga. “Analysis of Control Attainment in Endogenous Electroencephalogram Based Brain Computer Interfaces.” Doctoral dissertation. Nov. 2017. doi: 10.13140/RG.2.2.10493.05608.

[25] V. R. Joseph and A. Vakayil. “SPlit: An Optimal Method for Data Splitting.” In: *Technometrics* (Dec. 2020). doi: 10.1080/00401706.2021.1921037. url: http://arxiv.org/abs/2012.10945http://dx.doi.org/10.1080/00401706.2021.1921037.

[26] Y. Liu, Y. Zhou, S. Wen, and C. Tang. “A Strategy on Selecting Performance Metrics for Classifier Evaluation.” In: *International Journal of Mobile Computing and Multimedia Communications* 6.4 (Oct. 2014), pp. 20–35. issn: 19379404. doi: 10.4018/IJMCMC.2014100102. url: https://www.researchgate.net/publication/291600681_A_Strategy_on_Selecting_Performance_Metrics_for_Classifier_Evaluation.

[27] A. Botchkarev. “Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology.” In: *Interdisciplinary Journal of Information, Knowledge, and Management* 14 (Sept. 2018), pp. 45–76. doi: 10.28945/4184. url: http://arxiv.org/abs/1809.03006http://dx.doi.org/10.28945/4184.

[28] *What is Supervised Learning? | IBM*. Aug. 2020. url: https://www.ibm.com/cloud/learn/supervised-learning.

[29] A. Worster, J. Fan, and A. Ismaila. “Understanding linear and logistic regression analyses.” In: *Canadian Journal of Emergency Medicine* 9.2 (2007), pp. 111–113. issn: 14818035. doi: 10.1017/S1481803500014883.

[30] V. Nasteski. “An overview of the supervised machine learning methods.” In: *HORIZONS.B* 4 (Dec. 2017), pp. 51–62. issn: 18578578. doi: 10.20544/HORIZONS.B.04.1.17.P05. url: https://www.researchgate.net/publication/328146111_An_overview_of_the_supervised_machine_learning_methods.

[31] H. Belyadi and A. Haghighat. “Supervised learning.” In: *Machine Learning Guide for Oil and Gas Using Python* (Jan. 2021), pp. 169–295. doi: 10.1016/B978-0-12-821929-4.00004-4.

[32] A. Bronshtein. *A Quick Introduction to K-Nearest Neighbors Algorithm | by Adi Bronshtein | Medium*. Apr. 2017. url: https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7.

[33] B. Mahesh. “Machine Learning Algorithms -A Review.” In: (Nov. 2019). doi: 10.21275/ART20203995. url: https://www.researchgate.net/publication/344717762_Machine_Learning_Algorithms_-A_Review.

[34] *Máquina de vetores de suporte (SVM) explicada*. url: https://ichi.pro/pt/maquina-de-vetores-de-suporte-svm-explicada-97743104690915.

[35] L. Tan. “Code Comment Analysis for Improving Software Quality.” In: *The Art and Science of Analyzing Software Data* (Jan. 2015), pp. 493–517. doi: 10.1016/B978-0-12-411519-4.00017-3.

[36] M. Imran and S. A. Alsuhaibani. “A Neuro-Fuzzy Inference Model for Diabetic Retinopathy Classification.” In: *Intelligent Data Analysis for Biomedical Applications*. Elsevier, Jan. 2019, pp. 147–172. doi: 10.1016/B978-0-12-815553-0.00007-0. url: https://linkinghub.elsevier.com/retrieve/pii/B9780128155530000070.

[37] P. Baheti. *12 Types of Neural Networks Activation Functions: How to Choose?* url: https://www.v7labs.com/blog/neural-networks-activation-functions.

[38] “Artificial neural networks.” In: *Neural Networks Modeling and Control* (Jan. 2020), pp. 117–124. doi: 10.1016/B978-0-12-817078-6.00016-7. url: https://linkinghub.elsevier.com/retrieve/pii/B9780128170786000167.

[39] C. Enyinna Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning.” In: (Nov. 2018). issn: 2331-8422. url: https://arxiv.org/abs/1811.03378v1.

[40] A. G. Farizawani, M. Puteh, Y. Marina, and A. Rivaie. “A review of artificial neural network learning rule based on multiple variant of conjugate gradient approaches.” In: *Journal of Physics: Conference Series* 1529.2 (Apr. 2020), p. 022040. issn: 1742-6596. doi: 10.1088/1742-6596/1529/2/022040. url: https://iopscience.iop.org/article/10.1088/1742-

6596/1529/2/022040https://iopscience.iop.org/article/10.1088/1742-6596/1529/2/022040/meta.

[41] G. Kim and D. S. Jeong. "CBP: Backpropagation with constraint on weight precision using a pseudo-Lagrange multiplier method." In: (Oct. 2021). url: https://arxiv.org/abs/2110.02550v2.

[42] W. S. Alaloul and A. H. Qureshi. "Data Processing Using Artificial Neural Networks." In: *Dynamic Data Assimilation - Beating the Uncertainties* (May 2020). doi: 10.5772/INTECHOPEN.91935. url: https://www.intechopen.com/chapters/71673.

[43] J. Schmidhuber. "Deep learning in neural networks: An overview." In: *Neural Networks* 61 (Jan. 2015), pp. 85–117. issn: 0893-6080. doi: 10.1016/J.NEUNET.2014.09.003.

[44] I. Shafkat. *Intuitively Understanding Convolutions for Deep Learning*. June 2018. url: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1.

[45] N. Chauhan. "Optimization Algorithms in Neural Networks." In: (Dec. 2020). url: https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html.

[46] I. Amir, T. Koren, and R. Livni. "SGD Generalizes Better Than GD (And Regularization Doesn't Help)." In: (Feb. 2021). url: https://arxiv.org/abs/2102.01117v2.

[47] H. Kim, K. Nonlaopon, and J. Rho. "Easy Access to the Update of Weight in Backpropagation Algorithm." In: 16.2 (2021), pp. 801–804. url: https://www.ripublication.com/adsa21/v16n2p30.pdf.

[48] J. Brownlee. *How to Avoid Overfitting in Deep Learning Neural Networks*. Aug. 2019. url: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/.

[49] A. Y. Ng. "Feature selection, L 1 vs. L 2 regularization, and rotational invariance." In: (). url: https://icml.cc/Conferences/2004/proceedings/papers/354.pdf.

[50] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. issn: 1533-7928. url: http://jmlr.org/papers/v15/srivastava14a.html.

[51] N. Ganatra and A. Patel. "A Comprehensive Study of Deep Learning Architectures, Applications and Tools." In: *International Journal of Computer Sciences and Engineering* 6 (Dec. 2018), pp. 701–705. doi: 10.26438/ijcse/v6i12.701705.

[52] S. Madhavan and M. T. Jones. *Deep learning architectures – IBM Developer*. Jan. 2021. url: https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/.

[53] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi. "Convolutional neural networks: an overview and application in radiology." In: *Insights into Imaging 2018 9:4* 9.4 (June 2018), pp. 611–629. issn: 1869-4101. doi: 10.1007/S13244-018-0639-9. url: https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9.

[54] A. Shewalkar, D. Nyavanandi, and S. A. Ludwig. "Performance Evaluation Of Deep Neural Networks Applied To Speech Recognition: RNN, LSTM And GRU." In: *JAISCR* 9.4 (2019), p. 235. doi: 10.2478/jaiscr-2019-0006.

[55] A. N. Shewalkar. "Comparison Of RNN, LSTM And GRU On Speech Recognition Data." In: ().

[56] D. Gupta. *Recurrent Neural Network | Fundamentals Of Deep Learning*. Dec. 2017. url: https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/.

[57] A. Khan and A. Sarfaraz. "RNN-LSTM-GRU based language transformation." In: *Soft Computing* 23.24 (Dec. 2019), pp. 13007–13024. issn: 14337479. doi: 10.1007/S00500-019-04281-Z/FIGURES/15. url: https://link.springer.com/article/10.1007/s00500-019-04281-z.

[58] W. H. Lopez Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli. "Autoencoders." In: *Machine Learning: Methods and Applications to Brain Disorders* (Mar. 2020), pp. 193–208. doi: 10.1016/B978-0-12-815739-8.00011-0. url: https://arxiv.org/abs/2003.05991v2.

[59] S. Abirami and P. Chitra. "Energy-efficient edge based real-time healthcare support system." In: *Advances in Computers* 117.1 (Jan. 2020), pp. 339–368. issn: 0065-2458. doi: 10.1016/BS.ADCOM.2019.09.007.

[60] J. Waring, C. Lindvall, and R. Umeton. "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare." In: *Artificial Intelligence in Medicine* 104 (Apr. 2020), p. 101822. issn: 0933-3657. doi: 10.1016/J.ARTMED.2020.101822.

[61] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms." In: ().

[62] L. Zimmer, M. Lindauer, and F. Hutter. "Auto-PyTorch Tabular: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.9 (June 2020), pp. 3079–3090. issn: 19393539. doi: 10.48550/arxiv.2006.13799. url: https://arxiv.org/abs/2006.13799v3.

[63] R. S. Olson, O. Edu, and J. H. Moore. “TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning.” In: 64 (2016), pp. 66–74. url: https://github.com/rhiever/tpot.

[64] H. Jin, Q. Song, and X. Hu. “Auto-Keras: An Efficient Neural Architecture Search System.” In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (June 2018), pp. 1946–1956. doi: 10.48550/arxiv.1806.10282. url: https://arxiv.org/abs/1806.10282v3.

[65] N. Jonsson. “Ways to use Machine Learning approaches for software development.” In: ().

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library.” In: *Advances in Neural Information Processing Systems* 32 (Dec. 2019). issn: 10495258. url: https://arxiv.org/abs/1912.01703v1.

[67] *DNA Sequencing Fact Sheet*. Aug. 2020. url: https://www.genome.gov/about-genomics/fact-sheets/DNA-Sequencing-Fact-Sheet.

[68] *The Human Genome Project*. url: https://www.genome.gov/human-genome-project.

[69] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. “Basic local alignment search tool.” In: *Journal of molecular biology* 215.3 (1990), pp. 403–410. issn: 0022-2836. doi: 10.1016/S0022-2836(05)80360-2. url: https://pubmed.ncbi.nlm.nih.gov/2231712/.

[70] W. R. Pearson and D. J. Lipman. “Improved tools for biological sequence comparison.” In: *Proceedings of the National Academy of Sciences of the United States of America* 85.8 (1988), pp. 2444–2448. issn: 0027-8424. doi: 10.1073/PNAS.85.8.2444. url: https://pubmed.ncbi.nlm.nih.gov/3162770/.

[71] G. Lo Bosco and M. A. Di Gangi. “Deep learning architectures for DNA sequence classification.” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10147 LNAI. 2017. doi: 10.1007/978-3-319-52962-2{_}14. url: https://link.springer.com/chapter/10.1007%2F978-3-319-52962-2_14.

[72] B. Liu. “BioSeq-Analysis: a platform for DNA, RNA and protein sequence analysis based on machine learning approaches.” In: *Briefings in bioinformatics* 20.4 (Mar. 2017), pp. 1280–1294. issn: 1477-4054. doi: 10.1093/BIB/BBX165. url: https://pubmed.ncbi.nlm.nih.gov/29272359/.

[73] Z. Chen, P. Zhao, F. Li, T. T. Marquez-Lago, A. Leier, J. Revote, Y. Zhu, D. R. Powell, T. Akutsu, G. I. Webb, K. C. Chou, A. I. Smith, R. J. Daly, J. Li, and J. Song. “iLearn: an integrated platform and meta-learner for feature engineering, machine-learning analysis and modeling of DNA, RNA and protein sequence data.” In: *Briefings in bioinformatics* 21.3 (May 2019), pp. 1047–1057. issn: 1477-4054. doi: 10.1093/BIB/BBZ041. url: https://pubmed.ncbi.nlm.nih.gov/31067315/.

[74] J. Dong, Z. J. Yao, L. Zhang, F. Luo, Q. Lin, A. P. Lu, A. F. Chen, and D. S. Cao. “PyBioMed: a python library for various molecular representations of chemicals, proteins and DNAs and their interactions.” In: *Journal of Cheminformatics* 10.1 (Dec. 2018), pp. 1–11. issn: 17582946. doi: 10.1186/S13321-018-0270-2/TABLES/5. url: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-018-0270-2.

[75] R. P. Bonidia, D. S. Domingues, D. S. Sanches, and A. C. P. L. F. de Carvalho. “MathFeature: feature extraction package for DNA, RNA and protein sequences based on mathematical descriptors.” In: *Briefings in Bioinformatics* 2021.0 (Nov. 2021), pp. 1–10. issn: 1467-5463. doi: 10.1093/BIB/BBAB434. url: https://academic.oup.com/bib/advance-article/doi/10.1093/bib/bbab434/6423525.

[76] B. Liu, F. Liu, L. Fang, X. Wang, and K. C. Chou. “repDNA: a Python package to generate various modes of feature vectors for DNA sequences by incorporating user-defined physicochemical properties and sequence-order effects.” In: *Bioinformatics* 31.8 (Apr. 2014), pp. 1307–1309. issn: 1367-4803. doi: 10.1093/BIOINFORMATICS/BTU820. url: https://academic.oup.com/bioinformatics/article/31/8/1307/213091.

[77] J. Dong, Z. J. Yao, M. Wen, M. F. Zhu, N. N. Wang, H. Y. Miao, A. P. Lu, W. B. Zeng, and D. S. Cao. “BioTriangle: A web-accessible platform for generating various molecular representations for chemicals, proteins, DNAs/RNAs and their interactions.” In: *Journal of Cheminformatics* 8.1 (June 2016), pp. 1–13. issn: 17582946. doi: 10.1186/S13321-016-0146-2/FIGURES/5. url: https://jcheminf.biomedcentral.com/articles/10.1186/s13321-016-0146-2.

[78] R. Nikam and M. M. Gromiha. “Seq2Feature: a comprehensive web-based feature extraction tool.” In: *Bioinformatics* 35.22 (Nov. 2019), pp. 4797–4799. issn: 1367-4803. doi: 10.1093/BIOINFORMATICS/BTZ432. url: https://academic.oup.com/bioinformatics/article/35/22/4797/5499130.

[79] R. Muhammod, S. Ahmed, D. M. Farid, S. Shatabda, A. Sharma, and A. Dehzangi. “PyFeat: a Python-based effective feature generation tool for DNA, RNA and protein sequences.” In: *Bioinformatics* 35.19 (Oct. 2019), pp. 3831–3833. issn: 1367-4803. doi: 10.1093/BIOINFORMATICS/BTZ165. url: https://academic.oup.com/bioinformatics/article/35/19/3831/5372339.

[80] A. C. H. Choong and N. K. Lee. “Evaluation of convolutionary neural networks modeling of DNA sequences using ordinal versus one-hot encoding method.” In: *1st International Conference on Computer and Drone Applications: Ethical Integration of Computer and Drone Technology for Humanity Sustainability, IConDA 2017* 2018-January (July 2017), pp. 60–65. doi: 10.1109/ICONDA.2017.8270400.

[81] H. Gunasekaran, K. Ramalakshmi, A. Rex Macedo Arokiaraj, S. D. Kanmani, C. Venkatesan, and C. S. G. Dhas. “Analysis of DNA Sequence Classification Using CNN and Hybrid Models.” In: *Computational and Mathematical Methods in Medicine* 2021 (2021). issn: 17486718. doi: 10.1155/2021/1835056. url: https://www.hindawi.com/journals/cmmm/2021/1835056/.

[82] V. V. Nair, K. Vijayan, D. P. Gopinath, and A. S. Nair. “ANN based classification of unknown genome fragments using chaos game representation.” In: *ICMLC 2010 - The 2nd International Conference on Machine Learning and Computing* (2010), pp. 81–85. doi: 10.1109/ICMLC.2010.56.

[83] R. Rizzo, A. Fiannaca, M. La Rosa, and A. Urso. “A Deep Learning Approach to DNA Sequence Classification.” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9874 LNCS (2015), pp. 129–140. issn: 16113349. doi: 10.1007/978-3-319-44332-4{_}10. url: https://link.springer.com/chapter/10.1007/978-3-319-44332-4_10.

[84] N. G. Nguyen, V. A. Tran, D. L. Ngo, D. Phan, F. R. Lumbanraja, M. R. Faisal, B. Abapihi, M. Kubo, K. Satou, N. G. Nguyen, V. A. Tran, D. L. Ngo, D. Phan, F. R. Lumbanraja, M. R. Faisal, B. Abapihi, M. Kubo, and K. Satou. “DNA Sequence Classification by Convolutional Neural Network.” In: *Journal of Biomedical Science and Engineering* 9.5 (Apr. 2016), pp. 280–286. issn: 1937-6871. doi: 10.4236/JBISE.2016.95021. url: http://www.scirp.org/journal/PaperInformation.aspx?PaperID=65923http://www.scirp.org/Journal/Paperabs.aspx?paperid=65923.

[85] S. M. Abd-Alhalem, N. F. Soliman, S. Eldin, S. E. Abd Elrahman, N. A. Ismail, E. S. M. El-Rabaie, and F. E. El-Samie. “Bacterial classification with convolutional neural networks based on different data reduction layers.” In: *Nucleosides, nucleotides & nucleic acids* 39.4 (Apr. 2020), pp. 493–503. issn: 1532-2335. doi: 10.1080/15257770.2019.1645851. url: https://pubmed.ncbi.nlm.nih.gov/31418627/.

[86] S. Chen, M. Liu, X. Zhang, R. Long, Y. Wang, Y. Han, S. Zhang, M. Xu, and J. Gu. “A Study of Cell-free DNA Fragmentation Pattern and Its Application in DNA Sample Type Classification.” In: *IEEE/ACM transactions on computational biology and bioinformatics* 15.5 (Sept. 2017), pp. 1718–1722. issn: 1557-9964. doi: 10.1109/TCBB.2017.2723388. url: https://pubmed.ncbi.nlm.nih.gov/28692984/.

[87] Z. Shen, W. Bao, and D. S. Huang. “Recurrent Neural Network for Predicting Transcription Factor Binding Sites.” In: *Scientific Reports 2018 8:1* 8.1 (Oct. 2018), pp. 1–10. issn: 2045-2322. doi: 10.1038/s41598-018-33321-1. url: https://www.nature.com/articles/s41598-018-33321-1.

[88] M. A. Helaly, S. Rady, and M. M. Aref. “Convolutional Neural Networks for Biological Sequence Taxonomic Classification: A Comparative Study.” In: *Advances in Intelligent Systems and Computing* 1058 (Oct. 2019), pp. 523–533. issn: 21945365. doi: 10.1007/978-3-030-31129-2{_}48. url: https://link.springer.com/chapter/10.1007/978-3-030-31129-2_48.

[89] L. Lugo and E. B. Hernández. “A Recurrent Neural Network approach for whole genome bacteria identification.” In: *https://doi.org/10.1080/08839514.2021.1922842* 35.9 (2021), pp. 642–656. issn: 10876545. doi: 10.1080/08839514.2021.1922842. url: https://www.tandfonline.com/doi/abs/10.1080/08839514.2021.1922842.

[90] W. Chen, P. Feng, H. Ding, H. Lin, and K. C. Chou. “iRNA-Methyl: Identifying N6-methyladenosine sites using pseudo nucleotide composition.” In: *Analytical Biochemistry* 490 (Dec. 2015), pp. 26–33. issn: 0003-2697. doi: 10.1016/J.AB.2015.08.021.

[91] W. Chen, P. M. Feng, H. Lin, and K. C. Chou. “iRSpot-PseDNC: identify recombination spots with pseudo dinucleotide composition.” In: *Nucleic Acids Research* 41.6 (Apr. 2013), e68–e68. issn: 0305-1048. doi: 10.1093/NAR/GKS1450. url: https://academic.oup.com/nar/article/41/6/e68/2902382.

[92] M. G. Grabherr, J. Pontiller, E. Mauceli, W. Ernst, M. Baumann, T. Biagi, R. Swofford, P. Russell, M. C. Zody, F. Palma, K. Lindblad-Toh, and R. M. Grabherr. “Exploiting Nucleotide Composition to Engineer Promoters.” In: *PLOS ONE* 6.5 (2011), e20136. issn: 1932-6203. doi: 10.1371/JOURNAL.PONE.0020136. url: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0020136.

[93] B. Panwar and G. P. Raghava. “Identification of protein-interacting nucleotides in a RNA sequence using composition profile of tri-nucleotides.” In: *Genomics* 105.4 (Apr. 2015), pp. 197–203. issn: 0888-7543. doi: 10.1016/J.YGENO.2015.01.005.

[94] W. R. Qiu, X. Xiao, and K. C. Chou. “iRSpot-TNCPseAAC: Identify Recombination Spots with Trinucleotide Composition and Pseudo Amino Acid Components.” In: *International Journal of Molecular Sciences 2014, Vol. 15, Pages 1746-1766* 15.2 (Jan. 2014), pp. 1746–1766. issn: 1422-0067. doi: 10.3390/IJMS15021746. url: https://www.mdpi.com/1422-0067/15/2/1746/htmhttps://www.mdpi.com/1422-0067/15/2/1746.

[95] B. Panwar, A. Arora, and G. P. Raghava. "Prediction and classification of ncRNAs using structural information." In: *BMC Genomics* 15.1 (Feb. 2014), pp. 1–13. issn: 14712164. doi: 10.1186/1471-2164-15-127/FIGURES/5. url: https://bmcgenomics.biomedcentral.com/articles/10.1186/1471-2164-15-127.

[96] W. Zhang, X. Xu, M. Yin, N. Luo, J. Zhang, and J. Wang. "Prediction of methylation sites using the composition of K-spaced amino acid pairs." In: *Protein and peptide letters* 20.8 (June 2013), pp. 911–917. issn: 1875-5305. doi: 10.2174/0929866511320080008. url: https://pubmed.ncbi.nlm.nih.gov/23276225/.

[97] B. Manavalan, S. Basith, T. H. Shin, D. Y. Lee, L. Wei, and G. Lee. "4mCpred-EL: An Ensemble Learning Framework for Identification of DNA N4-methylcytosine Sites in the Mouse Genome." In: *Cells* 8.11 (Nov. 2019). issn: 2073-4409. doi: 10.3390/CELLS8111332. url: https://pubmed.ncbi.nlm.nih.gov/31661923/.

[98] Y. Huang, N. He, Y. Chen, Z. Chen, and L. Li. "BERMP: a cross-species classifier for predicting m 6 A sites by integrating a deep learning algorithm and a random forest approach." In: *International journal of biological sciences* 14.12 (Sept. 2018), pp. 1669–1677. issn: 1449-2288. doi: 10.7150/IJBS.27819. url: https://pubmed.ncbi.nlm.nih.gov/30416381/.

[99] P. Feng, H. Yang, H. Ding, H. Lin, W. Chen, and K. C. Chou. "iDNA6mA-PseKNC: Identifying DNA N 6-methyladenosine sites by incorporating nucleotide physicochemical properties into PseKNC." In: *Genomics* 111.1 (Jan. 2019), pp. 96–102. issn: 1089-8646. doi: 10.1016/J.YGENO.2018.01.005. url: https://pubmed.ncbi.nlm.nih.gov/29360500/.

[100] I. Brukner, R. Sánchez, D. Suck, and S. Pongor. "Sequence-dependent bending propensity of DNA as revealed by DNase I: parameters for trinucleotides." In: *The EMBO Journal* 14.8 (Apr. 1995), pp. 1812–1818. issn: 1460-2075. doi: 10.1002/J.1460-2075.1995.TB07169.X. url: https://onlinelibrary.wiley.com/doi/full/10.1002/j.1460-2075.1995.tb07169.xhttps://onlinelibrary.wiley.com/doi/abs/10.1002/j.1460-2075.1995.tb07169.xhttps://www.embopress.org/doi/10.1002/j.1460-2075.1995.tb07169.x.

[101] Y. Fukue, N. Sumida, J. i. Tanase, and T. Ohyama. "A highly distinctive mechanical property found in the majority of human promoters and its transcriptional relevance." In: *Nucleic Acids Research* 33.12 (July 2005), pp. 3821–3827. issn: 0305-1048. doi: 10.1093/NAR/GKI700. url: https://academic.oup.com/nar/article/33/12/3821/2400984.

[102] W. Chen, T. Y. Lei, D. C. Jin, H. Lin, and K. C. Chou. "PseKNC: A flexible web server for generating pseudo K-tuple nucleotide composition." In: *Analytical Biochemistry* 456.1 (July 2014), pp. 53–60. issn: 0003-2697. doi: 10.1016/J.AB.2014.04.001.

[103] M.-F. Zhu, J. Dong, and D.-S. Cao. “rDNAse: R package for generating various numerical representation schemes of DNA sequences COMPUTATIONAL BIOLOGY & DRUG DESIGN GROUP! CENTRAL SOUTH UNIV., CHINA.” In: (2016).

[104] J. R. Goñi, C. Fenollosa, A. Pérez, D. Torrents, and M. Orozco. “DNAlive: a tool for the physical analysis of DNA at the genomic scale.” In: *Bioinformatics* 24.15 (Aug. 2008), pp. 1731–1732. issn: 1367-4803. doi: 10.1093/BIOINFORMATICS/BTN259. url: https://academic.oup.com/bioinformatics/article/24/15/1731/264860.

[105] J. R. Goñi, A. Pérez, D. Torrents, and M. Orozco. “Determining promoter location based on DNA structure first-principles calculations.” In: *Genome Biology* 8.12 (Dec. 2007), pp. 1–10. issn: 1474760X. doi: 10.1186/GB-2007-8-12-R263/FIGURES/3. url: https://link.springer.com/articles/10.1186/gb-2007-8-12-r263https://link.springer.com/article/10.1186/gb-2007-8-12-r263.

[106] K. C. Chou. “Using amphiphilic pseudo amino acid composition to predict enzyme subfamily classes.” In: *Bioinformatics* 21.1 (Jan. 2005), pp. 10–19. issn: 1367-4803. doi: 10.1093/BIOINFORMATICS/BTH466. url: https://academic.oup.com/bioinformatics/article/21/1/10/212492.

[107] K. C. Chou. “Prediction of protein cellular attributes using pseudo-amino acid composition.” In: *Proteins: Structure, Function, and Bioinformatics* 43.3 (May 2001), pp. 246–255. issn: 1097-0134. doi: 10.1002/PROT.1035. url: https://onlinelibrary.wiley.com/doi/full/10.1002/prot.1035https://onlinelibrary.wiley.com/doi/abs/10.1002/prot.1035https://onlinelibrary.wiley.com/doi/10.1002/prot.1035.

[108] Q. Zhang, Z. Shen, and D. S. Huang. “Modeling in-vivo protein-DNA binding by combining multiple-instance learning with a hybrid deep neural network.” In: *Scientific Reports* 9.1 (Dec. 2019). issn: 20452322. doi: 10.1038/S41598-019-44966-X. url: /pmc/articles/PMC6559991//pmc/articles/PMC6559991/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC6559991/.

[109] A. T. Golam Bari, M. R. Reaz, H. J. Choi, and B. S. Jeong. “DNA encoding for splice site prediction in large DNA sequence.” In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7827 LNCS (2013), pp. 46–58. issn: 03029743. doi: 10.1007/978-3-642-40270-8{_}4/COVER. url: https://link.springer.com/chapter/10.1007/978-3-642-40270-8_4.

[110] J. Zou, M. Huss, A. Abid, P. Mohammadi, A. Torkamani, and A. Telenti. “A primer on deep learning in genomics.” In: *Nature Genetics 2018 51:1* 51.1 (Nov. 2018), pp. 12–18. issn: 1546-1718.

doi: 10.1038/s41588-018-0295-5. url: https://www.nature.com/articles/s41588-018-0295-5.

[111] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. “Tune: A Research Platform for Distributed Model Selection and Training.” In: (July 2018). url: http://arxiv.org/abs/1807.05118.

[112] H. Luo, Y. Lin, F. Gao, C. T. Zhang, and R. Zhang. “DEG 10, an update of the database of essential genes that includes both protein-coding genes and noncoding genomic elements.” In: *Nucleic acids research* 42.Database issue (Jan. 2014). issn: 1362-4962. doi: 10.1093/NAR/GKT1131. url: https://pubmed.ncbi.nlm.nih.gov/24243843/.

[113] M. Ruffier, A. Kähäri, M. Komorowska, S. Keenan, M. Laird, I. Longden, G. Proctor, S. Searle, D. Staines, K. Taylor, A. Vullo, A. Yates, D. Zerbino, and P. Flicek. “Ensembl core software resources: storage and programmatic access for DNA sequence and genome annotation.” In: *Database : the journal of biological databases and curation* 2017.1 (Jan. 2017). issn: 1758-0463. doi: 10.1093/DATABASE/BAX020. url: https://pubmed.ncbi.nlm.nih.gov/28365736/.

[114] J. Brownlee. *How to Choose Loss Functions When Training Deep Learning Neural Networks*. url: https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

Appendix A
Detailed Results

Table 22: Results on Primer dataset.

Model	Feature Extraction	Accuracy	MCC	Confusion Matrix
MLP	Descriptor	0.958	0.917	$\begin{bmatrix} 188 & 15 \\ 2 & 195 \end{bmatrix}$
CNN	One-Hot	0.990	0.980	$\begin{bmatrix} 199 & 4 \\ 0 & 197 \end{bmatrix}$
CNN	Chemical	0.988	0.975	$\begin{bmatrix} 199 & 4 \\ 1 & 196 \end{bmatrix}$
CNN	2-mer one-hot	0.985	0.970	$\begin{bmatrix} 197 & 6 \\ 0 & 197 \end{bmatrix}$
CNN	3-mer one-hot	0.995	0.990	$\begin{bmatrix} 201 & 2 \\ 0 & 197 \end{bmatrix}$
LSTM	One-hot	0.998	0.995	$\begin{bmatrix} 202 & 1 \\ 0 & 197 \end{bmatrix}$
LSTM	Chemical	0.998	0.995	$\begin{bmatrix} 202 & 1 \\ 0 & 197 \end{bmatrix}$
LSTM	2-mer one-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
LSTM	3-mer one-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$

BiLSTM	One-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
BiLSTM	Chemical	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
BiLSTM	2-mer one-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
BiLSTM	3-mer one-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
GRU	One-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
GRU	Chemical	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
GRU	2-mer one-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
GRU	3-mer one-hot	0.995	0.990	$\begin{bmatrix} 201 & 2 \\ 0 & 197 \end{bmatrix}$
Bi-GRU	One-hot	0.998	0.995	$\begin{bmatrix} 202 & 1 \\ 0 & 197 \end{bmatrix}$
Bi-GRU	Chemical	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
Bi-GRU	2-mer one-hot	0.993	0.985	$\begin{bmatrix} 200 & 3 \\ 0 & 197 \end{bmatrix}$
Bi-GRU	3-mer one-hot	1	1	$\begin{bmatrix} 203 & 0 \\ 0 & 197 \end{bmatrix}$
CNN-LSTM	One-hot	0.978	0.956	$\begin{bmatrix} 194 & 9 \\ 0 & 197 \end{bmatrix}$
CNN-LSTM	Chemical	0.985	0.970	$\begin{bmatrix} 197 & 6 \\ 0 & 197 \end{bmatrix}$
CNN-LSTM	2-mer one-hot	0.990	0.980	$\begin{bmatrix} 199 & 4 \\ 0 & 197 \end{bmatrix}$
CNN-LSTM	3-mer one-hot	0.998	0.995	$\begin{bmatrix} 202 & 1 \\ 0 & 197 \end{bmatrix}$
CNN-BiLSTM	One-hot	0.988	0.975	$\begin{bmatrix} 199 & 4 \\ 1 & 196 \end{bmatrix}$

CNN-BiLSTM	Chemical	0.960	0.923	$\begin{bmatrix} 187 & 16 \\ 0 & 197 \end{bmatrix}$
CNN-BiLSTM	2-mer one-hot	0.995	0.990	$\begin{bmatrix} 201 & 2 \\ 0 & 197 \end{bmatrix}$
CNN-BiLSTM	3-mer one-hot	0.998	0.995	$\begin{bmatrix} 202 & 1 \\ 0 & 197 \end{bmatrix}$
CNN-GRU	One-hot	0.985	0.970	$\begin{bmatrix} 197 & 6 \\ 0 & 197 \end{bmatrix}$
CNN-GRU	Chemical	0.995	0.990	$\begin{bmatrix} 201 & 2 \\ 0 & 197 \end{bmatrix}$
CNN-GRU	2-mer one-hot	0.995	0.990	$\begin{bmatrix} 201 & 2 \\ 0 & 197 \end{bmatrix}$
CNN-GRU	3-mer one-hot	0.985	0.970	$\begin{bmatrix} 202 & 1 \\ 5 & 192 \end{bmatrix}$
CNN-BiGRU	One-hot	0.993	0.985	$\begin{bmatrix} 200 & 3 \\ 0 & 197 \end{bmatrix}$
CNN-BiGRU	Chemical	0.990	0.980	$\begin{bmatrix} 199 & 4 \\ 0 & 197 \end{bmatrix}$
CNN-BiGRU	2-mer one-hot	0.995	0.990	$\begin{bmatrix} 201 & 2 \\ 0 & 197 \end{bmatrix}$
CNN-BiGRU	3-mer one-hot	0.998	0.995	$\begin{bmatrix} 202 & 1 \\ 0 & 197 \end{bmatrix}$

Table 23: Results on Essential Genes dataset.

Model	**Feature Extraction**	**Accuracy**	**MCC**	**Confusion Matrix**
MLP	Descriptor	0.978	0.911	$\begin{bmatrix} 2485 & 40 \\ 23 & 379 \end{bmatrix}$
CNN	One-Hot	0.964	0.845	$\begin{bmatrix} 2478 & 47 \\ 59 & 343 \end{bmatrix}$
CNN	Chemical	0.950	0.782	$\begin{bmatrix} 2476 & 49 \\ 96 & 306 \end{bmatrix}$
CNN	2-mer one-hot	0.965	0.855	$\begin{bmatrix} 2462 & 63 \\ 40 & 362 \end{bmatrix}$
CNN	3-mer one-hot	0.978	0.912	$\begin{bmatrix} 2483 & 42 \\ 21 & 381 \end{bmatrix}$
LSTM	One-hot	0.974	0.899	$\begin{bmatrix} 2459 & 66 \\ 10 & 392 \end{bmatrix}$
LSTM	Chemical	0.976	0.903	$\begin{bmatrix} 2470 & 55 \\ 16 & 386 \end{bmatrix}$
LSTM	2-mer one-hot	0.981	0.922	$\begin{bmatrix} 2478 & 47 \\ 10 & 392 \end{bmatrix}$
LSTM	3-mer one-hot	0.982	0.925	$\begin{bmatrix} 2492 & 33 \\ 20 & 382 \end{bmatrix}$
BiLSTM	One-hot	0.984	0.933	$\begin{bmatrix} 2490 & 35 \\ 13 & 389 \end{bmatrix}$
BiLSTM	Chemical	0.972	0.880	$\begin{bmatrix} 2493 & 32 \\ 50 & 352 \end{bmatrix}$
BiLSTM	2-mer one-hot	0.981	0.924	$\begin{bmatrix} 2484 & 41 \\ 14 & 388 \end{bmatrix}$
BiLSTM	3-mer one-hot	0.986	0.942	$\begin{bmatrix} 2505 & 20 \\ 20 & 382 \end{bmatrix}$
GRU	One-hot	0.982	0.925	$\begin{bmatrix} 2497 & 28 \\ 24 & 378 \end{bmatrix}$
GRU	Chemical	0.975	0.897	$\begin{bmatrix} 2474 & 51 \\ 23 & 379 \end{bmatrix}$
GRU	2-mer one-hot	0.978	0.909	$\begin{bmatrix} 2481 & 44 \\ 21 & 381 \end{bmatrix}$

GRU	3-mer one-hot	0.985	0.936	$\begin{bmatrix} 2509 & 16 \\ 28 & 374 \end{bmatrix}$
Bi-GRU	One-hot	0.976	0.898	$\begin{bmatrix} 2495 & 30 \\ 40 & 362 \end{bmatrix}$
Bi-GRU	Chemical	0.976	0.902	$\begin{bmatrix} 2472 & 53 \\ 18 & 384 \end{bmatrix}$
Bi-GRU	2-mer one-hot	0.982	0.925	$\begin{bmatrix} 2502 & 23 \\ 29 & 373 \end{bmatrix}$
Bi-GRU	3-mer one-hot	0.983	0.929	$\begin{bmatrix} 2505 & 20 \\ 29 & 373 \end{bmatrix}$
CNN-LSTM	One-hot	0.958	0.821	$\begin{bmatrix} 2468 & 57 \\ 66 & 336 \end{bmatrix}$
CNN-LSTM	Chemical	0.942	0.742	$\begin{bmatrix} 2465 & 60 \\ 111 & 291 \end{bmatrix}$
CNN-LSTM	2-mer one-hot	0.971	0.878	$\begin{bmatrix} 2482 & 43 \\ 42 & 360 \end{bmatrix}$
CNN-LSTM	3-mer one-hot	0.973	0.885	$\begin{bmatrix} 2489 & 36 \\ 43 & 359 \end{bmatrix}$
CNN-BiLSTM	One-hot	0.960	0.836	$\begin{bmatrix} 2454 & 71 \\ 46 & 356 \end{bmatrix}$
CNN-BiLSTM	Chemical	0.936	0.713	$\begin{bmatrix} 2473 & 52 \\ 134 & 268 \end{bmatrix}$
CNN-BiLSTM	2-mer one-hot	0.967	0.869	$\begin{bmatrix} 2456 & 69 \\ 27 & 375 \end{bmatrix}$
CNN-BiLSTM	3-mer one-hot	0.976	0.904	$\begin{bmatrix} 2476 & 49 \\ 20 & 382 \end{bmatrix}$
CNN-GRU	One-hot	0.963	0.842	$\begin{bmatrix} 2474 & 51 \\ 58 & 344 \end{bmatrix}$
CNN-GRU	Chemical	0.943	0.744	$\begin{bmatrix} 2473 & 52 \\ 116 & 286 \end{bmatrix}$
CNN-GRU	2-mer one-hot	0.970	0.874	$\begin{bmatrix} 2484 & 41 \\ 46 & 356 \end{bmatrix}$
CNN-GRU	3-mer one-hot	0.978	0.911	$\begin{bmatrix} 2486 & 39 \\ 24 & 378 \end{bmatrix}$

CNN-BiGRU	One-hot	0.962	0.837	$\begin{bmatrix} 2475 & 50 \\ 62 & 340 \end{bmatrix}$
CNN-BiGRU	Chemical	0.946	0.778	$\begin{bmatrix} 2434 & 91 \\ 67 & 335 \end{bmatrix}$
CNN-BiGRU	2-mer one-hot	0.971	0.880	$\begin{bmatrix} 2475 & 50 \\ 35 & 367 \end{bmatrix}$
CNN-BiGRU	3-mer one-hot	0.977	0.908	$\begin{bmatrix} 2480 & 45 \\ 21 & 381 \end{bmatrix}$