Disclaimer

This document is an **unofficial English translation** of my original bachelor's thesis, which was written in Italian. The translation was generated with the assistance of artificial intelligence and has not been professionally reviewed. While I have made efforts to ensure accuracy, there may still be errors or misinterpretations.

For the **official and authoritative version** of my thesis in Italian, please refer to the original document available at the following link:

`hdl.handle.net/20.500.12608/72195`

Readers are encouraged to consult the official Italian version for precise information and context. If you have any questions or require clarification, please feel free to contact me.

Author: Alessandro Trigolo
Date: February 21, 2025

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Bachelor Degree in Computer Engineering

Convolutional Neural Networks to study non coding variants in genomic sequences

Candidate

Alessandro Trigolo

Student ID 2043049

Supervisor

Prof. Cinzia Pizzi

University of Padova

Academic Year
2023/2024

“ *If you only do what you can do,*
you will never be more than who you are. ”

Master Shifu

Abstract

Non-coding variants, genome regions that are not translated into proteins, are a major cause of genetic diseases, such as Mendelian disorders. The functional effects of these mutations remain difficult to fully comprehend. However, thanks to advances in sequencing technologies — which have greatly enriched biological data banks — and the development of sufficiently powerful hardware, it has become possible to design neural network-based tools capable of analyzing genomic sequences and providing valuable insights into the functional effects of these specific DNA regions.

This thesis aims to introduce molecular biology concepts and provide mathematical tools for understanding neural networks. Specifically, it will explore the structure and functioning of convolutional neural networks with the goal of analyzing three tools based on this technology. The thesis will focus on DeepSEA, Basset, and DeepSATA — three tools designed to enhance the understanding of the functional impact of non-coding variants.

Contents

List of Figures

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
ATP	Adenosine TriPhosphate
AUC	Area Under the Curve
AUROC	Area Under the ROC Curve
BCE	Binary Cross Entropy
CNN	Convolutional Neural Network
DHS	DNase I hypersensitive sites
DL	Deep Learning
DNA	DeoxyriboNucleic Acid
ENCODE	Encyclopedia of DNA Elements
FPR	False Positive Rate
GD	Gradient Descent

GEO	Gene Expression Omnibus
GPU	Graphics Processing Unit
GWAS	Genome-Wide Association Study
mRNA	messenger RNA
MSE	Mean Squared Errors
NCBI	National Center for Biotechnology Information
ncDNA	non-coding DNA
NLL	Negative Log Likelihood
NLP	Natural Language Processing
NN	Neural Network
OCB	Open Chromatin Bin
OCR	Open Chromatin Region
PDB	Protein Data Bank
PWM	Position Weight Matrix
ReLU	Rectified Linear Unit
RNA	RiboNucleic Acid

ROC	Receiver Operating Characteristics
SGD	Stochastic Gradient Descent
SRA	Sequence Read Archive
SVM	Support Vector Machines
TAD	Topologically Associating Domains
TF	Transcription Factors
TPR	True Positive Rate
tRNA	transfer RNA
UTR	UnTranslated Region

1 Introduction

To date, the advancement of genomics—a branch of molecular biology that studies the genome of living organisms—has proven to be highly significant in deepening our understanding of diseases related to genome mutations in individuals. It is estimated that only between 1% and 2% of DNA contains *genes*, which are specific regions holding all the necessary information for the synthesis of amino acids that will eventually form proteins [1], [2]. Nevertheless, the vast majority of genomic disorders are caused by mutations in non-coding regions [3], known as *non-coding variants*. Mutations in these regions of the genome, which seemingly play a marginal role, are responsible for the development of significant disorders such as *Mendelian diseases*[1], epilepsy, cardiovascular diseases, and particularly cancers—including colorectal cancer and breast cancer [3]–[11]. It is therefore crucial to continue studying the effects that non-coding variants in genomic sequences have on individuals.

In recent decades, advancements in *sequencing* techniques [12] have significantly boosted the development of *bioinformatics*—a discipline that combines computer science and biology. Bioinformatics focuses on organizing biological data to facilitate access and the inclusion of new information (such as PDB [13]), developing *tools* that enable data analysis, and ultimately providing meaningful interpretation of the obtained results [14]. More recently, the growing availability of biological data and the constant increase in computational power have made it possible to apply *deep learning* (DL) techniques in the field of bioinformatics. This remarkable progress allows for the discovery and refinement of computational solutions that increasingly accurately define the role of mutations in non-coding regions of DNA. Thanks to these new technologies, *functional genomics*—a branch of genomics focused on describing the relationships between components of a biological system, such as genes and proteins [15]—has seen significant advancements in studying non-coding variants. However, there remain significant gaps in understanding the relationship between genetic mutations and gene expression. The use of deep learning techniques is therefore crucial to continuing research in this area. The goal of

[1]Mendelian diseases, caused by the mutation of a single gene, include cystic fibrosis and Huntington's disease.

this thesis is to discuss and compare three tools that use *convolutional neural networks* to predict the effect of non-coding variants in genomic sequences: DeepSEA [16], Basset [17], and DeepSATA [18].

More specifically, Chapter 2 will introduce the basics of molecular biology necessary to fully understand the importance of non-coding variants. Subsequently, Chapter 3 will delve into the fundamental principles of neural networks and how convolutional networks can be used as an excellent tool to predict the effect of genomic sequences. Chapter 4, instead, will examine the implementation details of each of the three tools, focusing primarily on aspects related to sequence encoding, network architecture, and the *dataset* used to train the model. Finally, Chapter 5 will summarize the differences analyzed in the previous chapter, providing an overall view of the comparison among the three tools.

2

Biological Background

The cell is the fundamental unit of life. It is a small aqueous mixture with chemical components, enclosed within a membrane, and possesses the remarkable ability to replicate. The primary characteristic that distinguishes cells is the presence of a nucleus. Cells without a nucleus are called *prokaryotes*—these are the most widespread and make up unicellular organisms such as bacteria and archaea. In contrast, cells that contain a nucleus are called *eukaryotes*—they are generally larger and more complex, forming multicellular life forms such as animals, plants, and fungi [19].

Inside the eukaryotic cell (Figure 2.1), various *organelles* are present within the *cytoplasm*, each carrying out a specific function. The *mitochondria* are the most widespread organelles. Their role is to generate chemical energy for the cell: through the oxidation of sugars and fats,

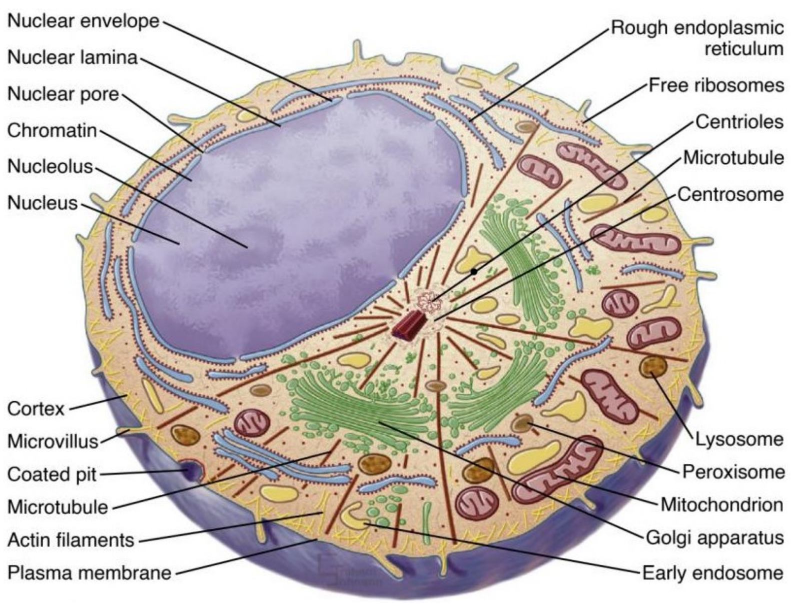

Figure 2.1: Schematic representation of the eukaryotic cell. The main organelles, including mitochondria, lysosomes, peroxisomes, the endoplasmic reticulum, and the nucleus, can be observed [2].

a substance is created that is used in most cellular activities[1]; this process is also known as *cellular respiration* because it consumes oxygen and releases carbon dioxide. In addition to being the primary energy source for the cell, mitochondria also play important roles in metabolism regulation, the cell cycle, antiviral responses, and even cell death [19]–[21].

The *endoplasmic reticulum*, on the other hand, is a very extensive organelle with multiple functions. Among these functions are protein translocation and protein folding [19], [22]. The *lysosomes* are responsible for degrading and recycling cellular waste and play a fundamental role in cellular homeostasis[2], development, and aging [23]–[25]. Finally, the *peroxisomes* are small vesicles that provide a protected environment to manage toxic molecules such as fatty acids, which are broken down through β-oxidation [19], [26].

The most important organelle in the cell remains the *nucleus*. Enclosed within the *nuclear envelope*, this organelle contains all the genetic information in the form of a long molecule of deoxyribonucleic acid (commonly known as DNA), which, once packaged, forms the *chromosome* [2], [19]. The DNA molecule is a double-helix structure made up of *nucleotides*. As shown in Figure 2.2, nucleotides consist of three fundamental elements: a *nitrogenous base*, a *sugar*, and a *phosphate group*[3]. There are four nitrogenous bases — Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) — which pair with each other through hydrogen bonds following a specific pattern: Adenine pairs only with Thymine (forming the *AT* bond), while Cytosine

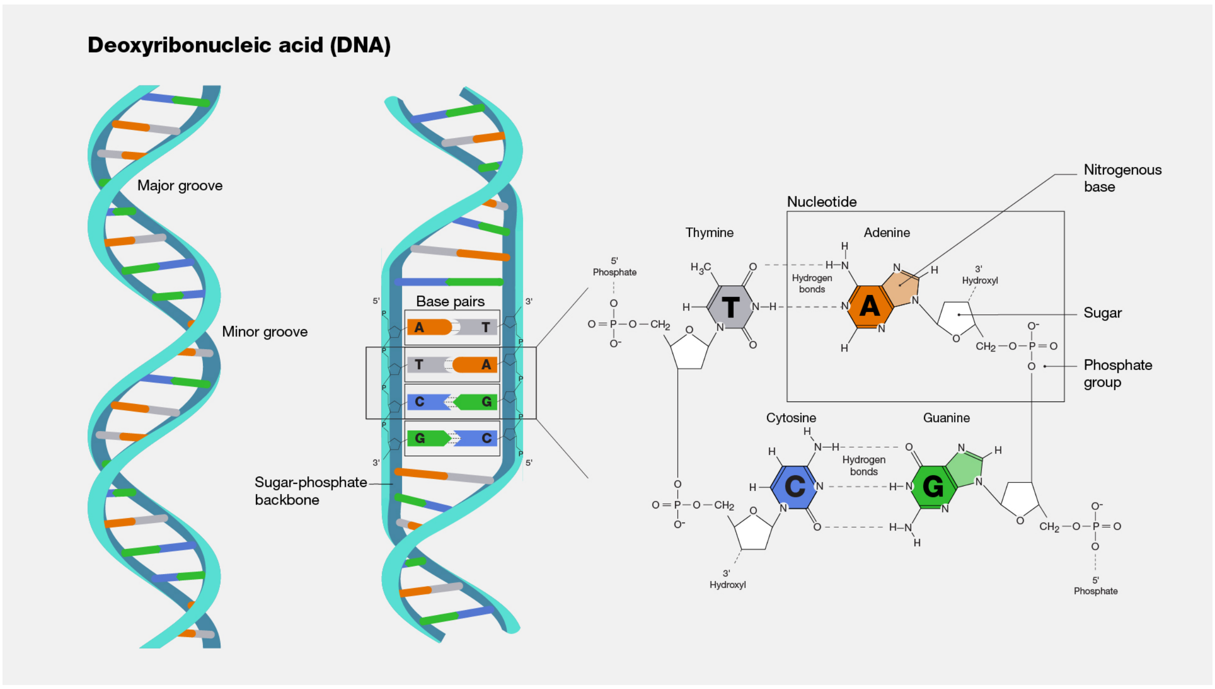

Figure 2.2: Schematic representation of DNA. The nitrogenous base pairs can be observed, bound together through sugars and phosphate groups [27].

[1]This substance is called *adenosine triphosphate* or ATP, and it has a structure similar to that of a nucleotide: it is composed of adenine, a sugar, and three phosphate groups.

[2]Cellular homeostasis refers to the set of mechanisms necessary to maintain the cell's functions at an optimal level.

[3]Phosphate groups have a negative charge and give the molecule its acidic properties.

pairs only with Guanine (creating the *CG* pair) [1], [28]. Furthermore, it is observed that the nucleotide of one pair and the next one are always linked in the same way through sugar and phosphate groups: the phosphate group of one nucleotide always binds to the sugar of the next one. Consequently, when considering a strand of the DNA double helix, the two ends are not identical, as one ends with a phosphate group (the $5'$ end), and the other ends with a sugar (the $3'$ end).

Through a series of foldings, a DNA molecule approximately two meters long can coil itself into a chromosome with a size of less than 2 microns (Figure 2.3). The process of *DNA packaging* begins with the DNA double helix wrapping around proteins called *histones*, forming *nucleosomes*. Subsequently, nucleosomes cluster together, forming a fiber known as *chromatin*, which further compacts upon itself to create the chromosome [29], [30].

2.1 Central Dogma

The relevance of DNA lies in the essential information it contains. This information resides in genes, which are genomic sequences that encode one or more functional biological products [32].*Gene expression* is the process that allows the data within the gene to be used for the creation of macromolecules, such as proteins. For example, skin cells exposed to intense sunlight may express genes that regulate skin pigmentation [33]. Gene expression is divided into two main phases: *transcription* — which is responsible for producing RNA molecules that reflect the gene to be expressed — and *translation* — which translates the information in the RNA by synthesizing the protein.

In the first phase of gene expression, DNA must be transcribed into a very similar molecule,

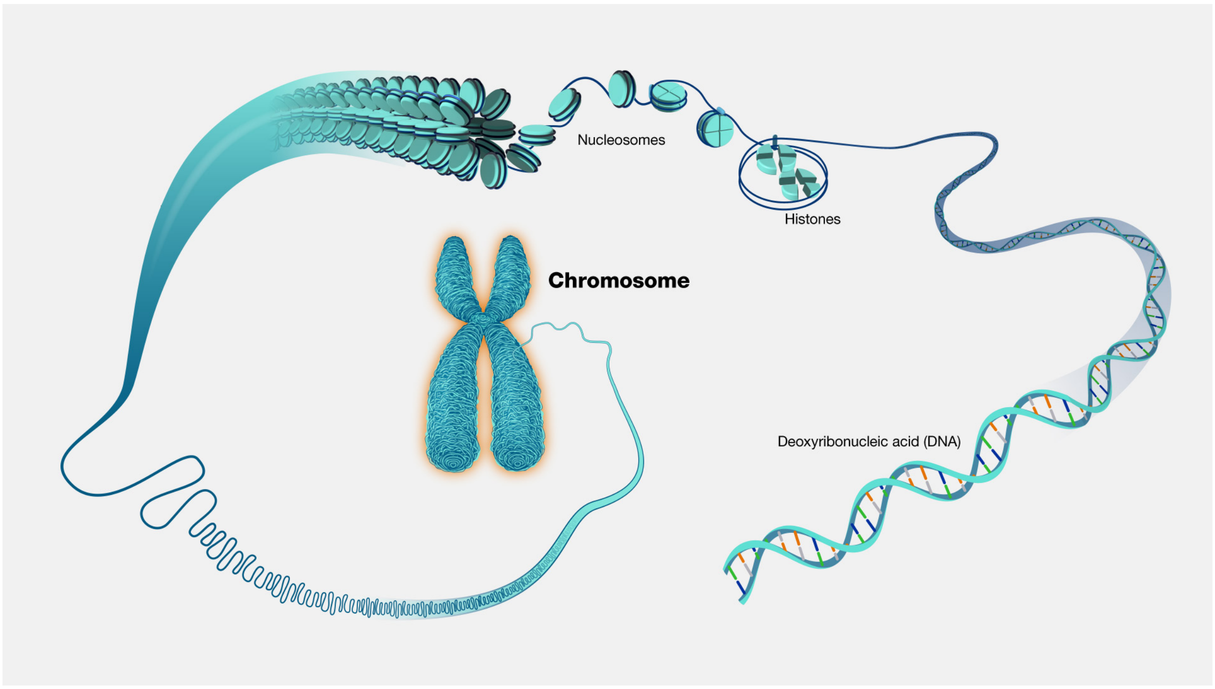

Figure 2.3: The process of DNA packaging that enables the compacting of the double-helix structure into a chromosome [31].

namely RNA — also called ribonucleic acid. This molecule differs from deoxyribonucleic acid by one nitrogenous base — instead of Thymine, Uracil (U) is present — and by the sugar — changing from deoxyribose to ribose [34]. Transcription of DNA into RNA begins when proteins, called *transcription factors* (TF), attracted to the *enhancers* of the DNA, recognize the region that marks the start of the gene molecule to be expressed, called the *promoter*. After recognizing the start of the sequence, these proteins allow an enzyme called *RNA polymerase* to attach and open the double helix of the DNA [35]. Once the double helix is opened, the actual transcription into RNA begins: the DNA strand is used as a template for the creation of the RNA; specifically, the nucleotide of the RNA will be complementary to that of the DNA (therefore, $A \rightarrow U$, $C \rightarrow G$, $G \rightarrow C$, and $T \rightarrow A$). In this way, ribonucleic acid is created one nucleotide at a time by analyzing the DNA nucleotide [34]. Transcription ends when the enzymes and proteins encounter the terminator region of the gene, which causes the separation from the strand and the termination of the *messenger* RNA (mRNA) that contains the information from the gene to be expressed. The entire transcription process is illustrated in Figure 2.4.

Before leaving the nucleus, the messenger RNA undergoes a series of modifications necessary to ensure the stored information is secure: several diseases emerge from mutations in the mRNA, including myotonic dystrophy[4] [37]. The first modification is called $5'$-*end capping*, which adds a Guanine to the $5'$ end of the mRNA via an unusual linkage, ensuring greater stability of the molecule. Secondly, *splicing* occurs, removing the non-coding regions — called *introns* — from the transcribed gene, keeping only those that will be used to synthesize proteins — the *exons* — thus facilitating the translation process. Finally, with the $3'$-*end processing*, a

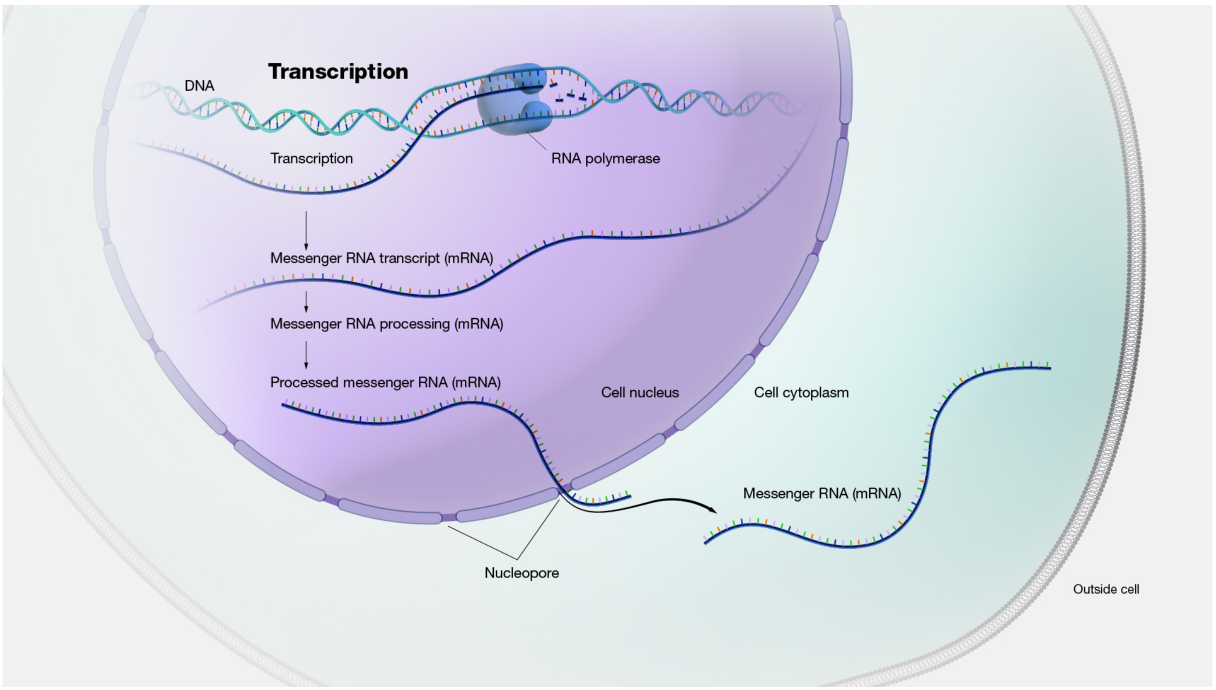

Figure 2.4: The process of DNA transcription of the gene into RNA by RNA polymerase [36].

[4]Myotonic dystrophies are diseases primarily affecting the musculoskeletal system.

tail of Adenine — known as the *polyA tail* — is added to the $3'$ end of the mRNA, which, in a manner similar to the *$5'$-end capping*, ensures stability of the ribonucleic acid strand [38], [39].

After exiting the nucleus through the *pores*, the messenger RNA reaches the cytoplasm and is ready to begin the second phase of gene expression, translation. In this phase, the mRNA is translated into a *polypeptide*, which is a sequence of amino acids that form the protein. There are more than 20 amino acids, so instead of coding for a single nucleotide of the messenger RNA, three nucleotides are coded at a time: this triplet is called a *codon*. During translation, *ribosomes* play a fundamental role as they are the organelles where translation occurs. Ribosomes are composed of two subunits, each of which has three sites for *transport* RNA (tRNA). Of the two ribosomal subunits, the smaller one binds to the mRNA and the *anticodons* (specific sequences of three bases in the tRNA) and ensures that translation occurs successfully. The larger subunit is responsible for catalyzing the peptide bond between the amino acid carried by the tRNA and the growing amino acid chain [39]–[41]. In this way, the ribosomes, analyzing codon by codon, can create the polypeptide chain through the tRNA, as shown in Figure 2.5.

Once the polypeptide sequence is created, the process of protein folding begins. Similar to how DNA is packed into chromosomes, the sequence of polypeptides initially coils into structures commonly known as *α-helix*. These then fold again, leading to the tertiary structure of the protein, which is the actual three-dimensional protein [43]. Once the protein is created, the gene has been definitively expressed. This transfer of information from DNA to protein creation is commonly referred to as the *central dogma of molecular biology*.

As mentioned at the beginning of the chapter, the cell possesses the remarkable ability to replicate. Generally, a cell duplicates during the growth and development of the organism, when it needs to be replaced or regenerated, or during the asexual reproduction of certain microor-

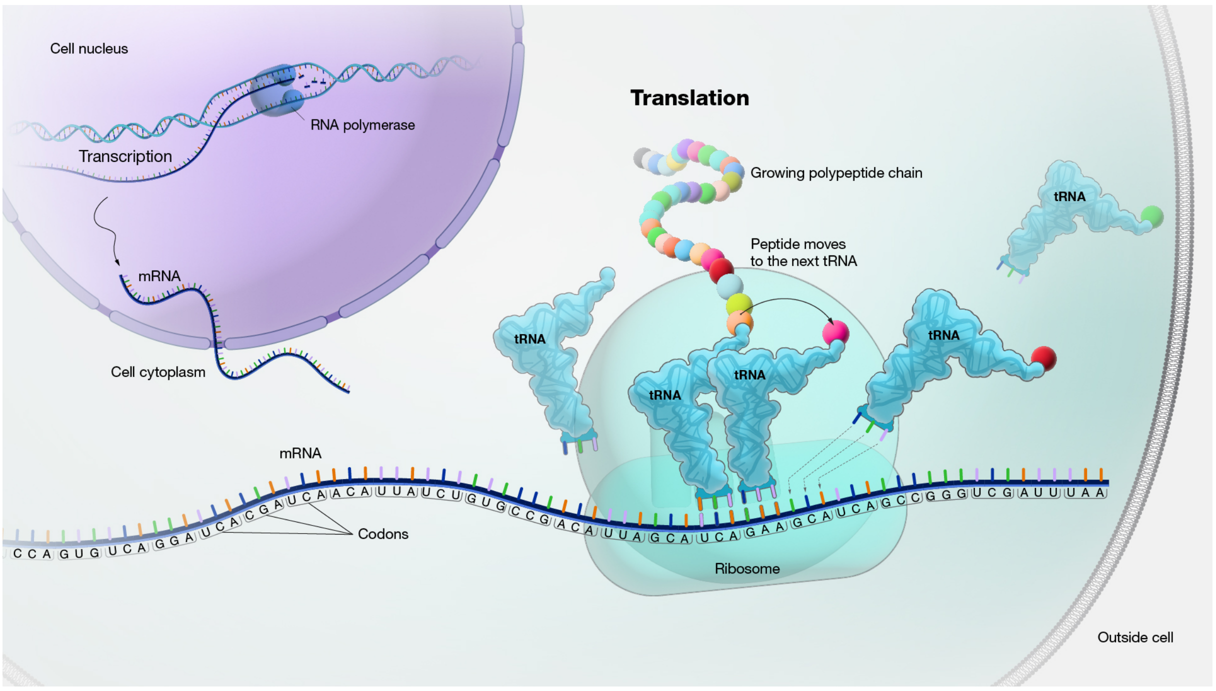
Cell nucleus
RNA polymerase
Transcription
mRNA
Cell cytoplasm
Translation
Growing polypeptide chain
Peptide moves to the next tRNA
tRNA
tRNA
tRNA
tRNA
tRNA
mRNA
Codons
CCAGUGUCAGGAUCACGAUCACAUUAUCUGUGCCGACAUUAGCAUCAGAAGCAUCAGCCGGGUCGAUUUAA
Ribosome
Outside cell

Figure 2.5: The translation process from messenger RNA to polypeptide through tRNA and ribosomes [42].

ganisms [44]. The process of cellular replication, called *mitosis*, is preceded by *interphase*, a fundamental process in which the cell grows in size and the DNA in the chromosomes duplicates, facilitating cellular replication. Mitosis can be divided into four main phases [44]–[47], which are summarized in Figure 2.6:

1. In *prophase*, the duplicated chromosomes condense in the nucleus and microtubules begin to approach the nucleus, called *centrosomes*; at the same time, the nuclear membrane starts to disappear;
2. After the microtubules have attached to the chromosomes (an intermediate phase called *prometaphase*), the cell enters *metaphase*, where all the chromosomes align along the cell's equatorial line;
3. During *anaphase*, each pair of chromosomes splits and moves toward the poles of the cell;
4. The final phase of mitosis is *telophase*, in which the two cells divide; the nuclear membranes of the two cells reform around the separated chromosomes.

For mitosis to be successful, the DNA inside the cell must first be duplicated. The DNA replication process that precedes mitosis is also known as the *S-phase* — *S-phase*.DNA duplication begins with the identification of the *replication origin*, a sequence in the DNA that specifies where the DNA should begin to replicate (there are more than one hundred thousand sites signaling a replication origin in the DNA of a cell). A *initiator protein* binds to the origin point, promoting the attachment of the *replisome* to the DNA, which consists of an enzyme called *helicase* that splits the two strands of DNA in the $5' \to 3'$ direction. At this point, the *RNA primer* begins synthesizing the DNA, promoting the attachment of the *DNA polymerase* to both strands for DNA duplication. Since the genome is complementary, one strand will have the direction $5' \to 3'$ (*leading strand*), while the other strand will be in the opposite direction, $3' \to 5'$ (*lagging strand*). Therefore, in the strand corresponding to the replisome, the polymerase will have

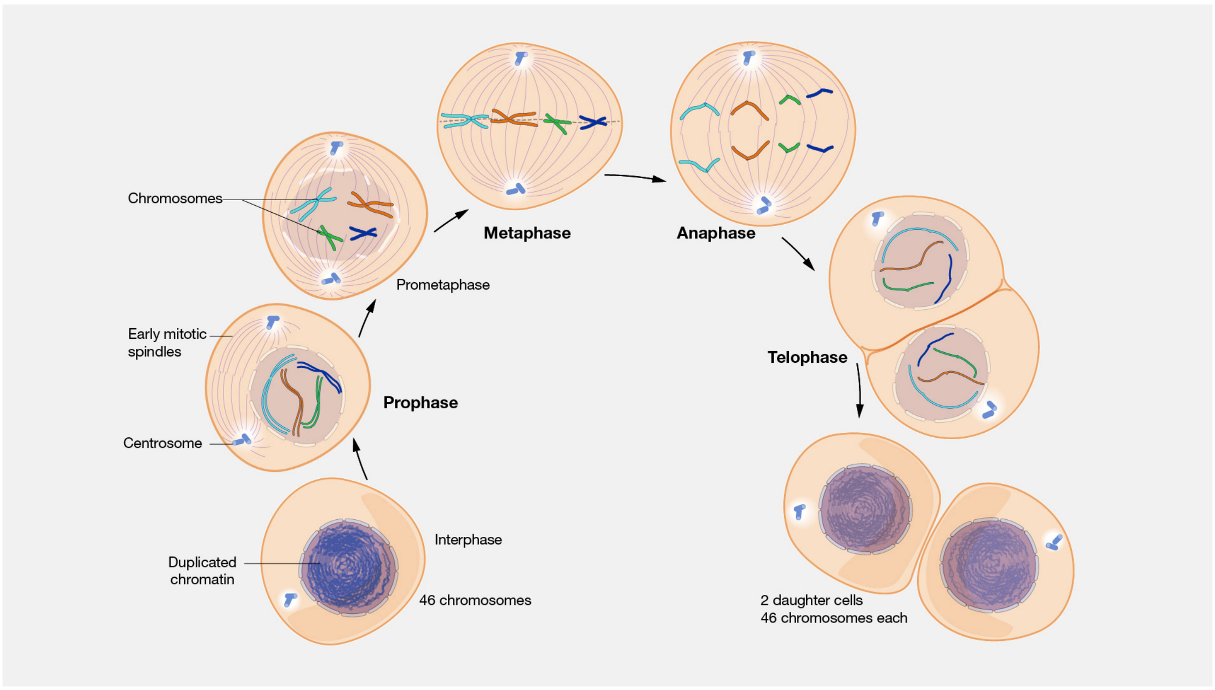

Figure 2.6: Cellular mitosis. The four phases of mitosis can be observed: prophase, metaphase, anaphase, and telophase [48].

no trouble duplicating, while in the $3' \rightarrow 5'$ strand, the DNA must be duplicated in segments, known as *Okazaki fragments*, which are joined together by *DNA ligase* [49]–[52]. Figure 2.7 summarizes the description of the synthesis phase.

2.2 Non-Coding Variants

As described so far, the role of DNA is crucial as it is transmitted from cell to cell during replication and then used in gene expression to create proteins. In contrast to the regions of DNA involved in gene expression, there are regions that are not coded, such as enhancers and promoters, as well as introns of a gene. The non-coding regions of DNA represent between 98% and 99% of the entire molecule: it was once thought that these regions had no real function, and were therefore referred to as *junk DNA*, or DNA junk. Contrary to what was previously believed, a mutation in these regions can cause various genetic disorders as the organization of chromatin, the DNA replication process, and gene expression can be compromised. In understanding the association between genomic variants and the disorders they cause, the *Genome-Wide Association Study* (GWAS) has played a significant role by analyzing the genomes of numerous subjects and correlating the presence of a mutation with a disorder. Among the variants identified by the GWAS, many are non-coding mutations [3], [54], [55].

Some non-coding variants can alter the splicing of a transcribed gene. As mentioned previously, splicing is the process that allows the removal of non-coding regions of a gene (the introns) before it is translated into a protein. Splicing is part of the processes that mRNA undergoes before leaving the nucleus and beginning translation. Within the introns, there are various sequences that mark the beginning of an exon, including the *donor*, *acceptor*, *branch points*, and

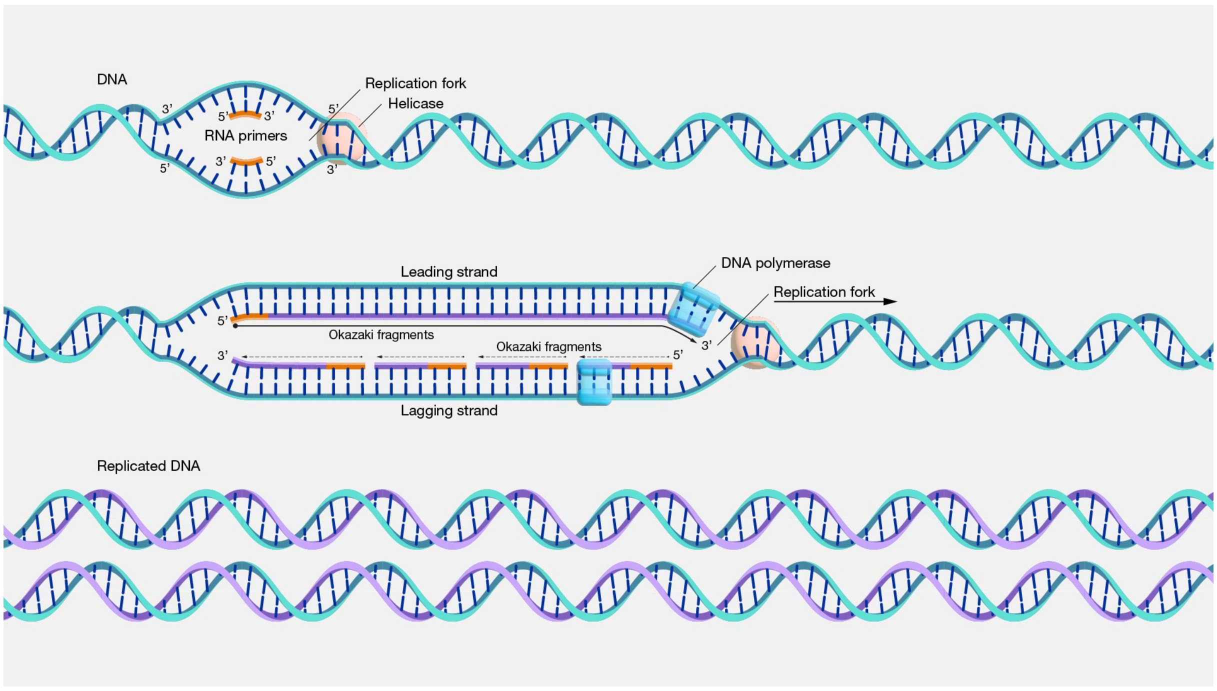

Figure 2.7: The process of DNA replication during the synthesis phase [53].

polypyrimidine stretches. Variations in these regions can lead to the failure to code for an exon or the retention of an intron: more than 15% of hereditary disorders are caused by these issues. In addition to introns, other untranslated regions (UTR) of mRNA can also lead to hereditary disorders. These regions are essential for managing the process that follows transcription: they help to make the mRNA strand more stable and robust and simplify its localization to initiate the translation process [4], [8].

Non-coding mutations, in addition to occurring in mRNA, can also occur in the non-coding regions of DNA (ncDNA), such as promoters, which attract transcription factors (TF). These proteins are responsible for binding the DNA molecule to RNA polymerase, which unwinds the double helix and begins transcription of the gene. The variation of these important regions causes incorrect transcription of the gene, leading to erroneous translation into protein. Variations in promoters are the cause of Mendelian diseases and some types of cancer. For the same reason, mutations in other *cis-regulatory* sequences of DNA — such as enhancers and *silencers*[5] — also lead to hereditary diseases. Among these disorders, in addition to the ones previously mentioned, are cardiac arrhythmia, restless leg syndrome[6], and pancreatic agenesis[7] [3], [4], [6], [8].

In addition to variations in enhancers and promoters of non-coding DNA, mutations can also cause the expansion of *tandem repeats*, both in exons and in non-coding regions of DNA. Tandem repeats are long repetitions of short DNA sequences. The consequences of variations in coding repeats are well-known (such as Huntington's disease), in contrast to the effects of the expansion of non-coding repeats. These mutations are associated with disorders in the transcription phase of DNA and the trapping of proteins involved during splicing [4].

Mutations in ncDNA can also introduce structural changes in chromatin, the fiber that makes up chromosomes. In particular, as chromatin packs to form chromosomes (Figure 2.3), it forms specific regions called TAD (*Topologically Associating Domains*), which contain groups of genes that frequently interact with each other. Consequently, even a small structural variation can lead to imperfect gene expression, caused by incorrect interactions between enhancers and promoters. It has been observed that the structural variation of TADs, in addition to causing Mendelian disorders, can lead to various neurological diseases [4], [56]. Besides TADs, other regions of chromatin play an important role during transcription, such as DNase I hypersensitive regions[8] (called DHS, *DNase I hypersensitive sites*), transcription factor (TF *binding sites*), and histone modifications. These regions are part of the more general open chromatin regions (OCR, *Open Chromatin Regions*), mutations in which can lead to inaccurate gene expression and, consequently, genetic disorders [16], [18].

In contrast to mutations in coding DNA, the effects of variations in ncDNA are still poorly

[5]Unlike enhancers, which stimulate a particular gene transcription, silencers serve to reduce or inhibit the transcription of a particular gene.

[6]This neurological syndrome causes an unbearable need for the subject to move the lower limbs.

[7]This disorder causes the absence of pancreatic tissue from birth.

[8]DNase I is an enzyme that cleaves DNA into smaller fragments. It is often used to identify chromatin regions that are more accessible to transcription.

understood. This is due to the difficulty in determining whether a non-coding variant influences the *phenotype* — that is, the set of structural and functional characteristics of an individual. Several bioinformatics tools have been developed to recognize whether a non-coding mutation has a functional effect, but since such functions also depend on the specific context of the sequence, predictions can be imprecise, thus providing unclear information about the phenotypic effect of the mutation [57], [58].

3 Neural Networks

The first artificial intelligence (AI) model dates back to 1943, when E. McCulloch and W. Pitts attempted to model a neuron as a simple predefined function. In this model, the neuron generated an output value if the processed boolean input variables exceeded a predetermined threshold [59]. A few years later, in 1950, Alan Turing published a paper defining a methodology to test the intelligence of a model [60]. This test — also known as the *imitation game* — involved evaluating whether a machine could mimic human intelligence, thus setting a goal for the field of artificial intelligence, a term that was coined for the first time during the Dartmouth conference in 1956.

Two years later, in 1958, psychologist F. Rosenblatt introduced the *perceptron* which, unlike the 1943 model, processed non-boolean inputs and had weights to balance the output [61]. Although the perceptron would form the basis of modern artificial neural networks, in the ten years following the publication of the paper, the initial expectations were not met. In 1968, a book was published analyzing the performance of the perceptron and noted its significant limitations, such as the inability to solve non-linearly separable problems [62]. Following a second paper in 1973, which highlighted the poor results obtained compared to the high expectations, the *First AI Winter* began, during which many government organizations ceased to fund research in artificial intelligence until the mid-1980s. The AI Winter ended in 1985 with the introduction of *Gradient Descent Optimization*, an algorithm that allowed the weights to be updated in such a way as to minimize the error in a network. A year later, the *back-propagation* algorithm was introduced, which was fundamental for the development of neural networks, composed of multiple layers of neurons, each of which is connected to the next layer [63].

Despite the great progress in the algorithmic part, the hardware was not computationally adequate to support the computational demands of artificial neural networks. This lack of computing power led to the *Second AI Winter*, a period in which scientific interest shifted to models requiring less computational power, such as *Support Vector Machines* (SVM) introduced in 1963. The Second AI Winter ended in the mid-1990s when hardware advancements were able to meet the computational requirements of neural network-based models. Constant development culminated in the last two decades when the GPU was introduced, which, along with the increase in available data, greatly accelerated progress in the field of AI [64], [65].

3.1 Basic Principles and Evolution

Artificial neural networks (ANN), commonly referred to as neural networks (NN), aim to represent a simplified model of the brain, treated as a structure composed of neurons. Therefore, it is essential to understand the functioning of the individual *artificial neuron* before exploring the structure of a neural network, which is a set of artificial neurons connected to each other according to specific criteria.

3.1.1 Artificial Neuron

Similar to a biological neuron, the artificial neuron (Figure 3.1) receives an arbitrary number n of input signals, which can be represented by the notation $x_1, x_2, \dots, x_n$. These values can be grouped into the input vector $\mathbf{x} = [x_1, x_2, \dots, x_n]$. To describe the result of all input signals to the neuron, a function g is introduced, which is typically a simple algebraic sum of the input signals x_i, where $i \in [1, n]$. Each input signal, before being summed, is multiplied by its respective weight (*weight*) w_i, where $i \in [1, n]$, and belongs to the weight vector $\mathbf{w} = [w_1, w_2, \dots, w_n]$. As a result, the output signal, denoted as v, will be the sum of the product of the i-th input signal (x_i) and its respective weight w_i:

$$v = g(\mathbf{w}, \mathbf{x}) = \sum_{i=1}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x} \tag{3.1}$$

To activate a neuron, the output signal v must be greater than a chosen threshold, denoted by b. This translates into subtracting this value from the signal v and checking whether the result is greater than or less than zero.

$$\sum_{i=1}^{n} w_i x_i \geq b \quad \Rightarrow \quad \sum_{i=1}^{n} w_i x_i - b \geq 0$$

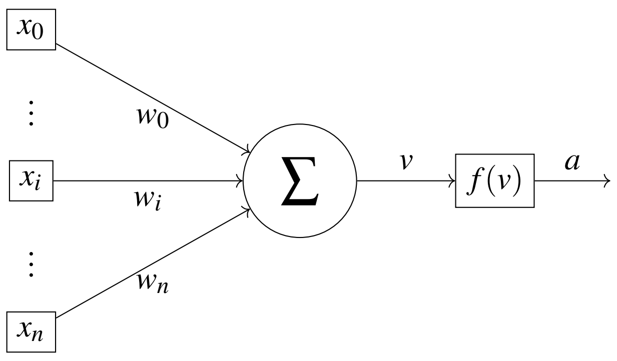

Figure 3.1: Schematic representation of the functioning of an artificial neuron. The input signals x_i are multiplied by their respective weights w_i and summed together; the result v is processed by the activation function $f(v)$ which returns the output a.

The value subtracted from the sum, called the *bias*, can be incorporated into the algebraic sum. It is sufficient to introduce an additional component $x_0 = 1$ in the input vector $\mathbf{x}$ and a component $w_0 = -b$ in the weight vector $\mathbf{w}$. In this way, equation 3.1 can be rewritten with the index i starting from 0, as follows:

$$v = g(\mathbf{w}, \mathbf{x}) = \sum_{i=0}^{n} w_i x_i = \mathbf{w} \cdot \mathbf{x}$$

The criterion that describes the activation of the artificial neuron is summarized in the *activation function*, denoted as $a = f(v)$. The simplest activation function that describes this criterion is the *step function* $\mathbf{1}(v)$, graphically depicted in Figure 3.2:

$$a(v) = \mathbf{1}(v) = \begin{cases} 1 & \text{se } v \geq 0 \\ 0 & \text{se } v < 0 \end{cases}$$

Similarly to the reactions of a biological neuron to external stimuli, an artificial neuron must also be able to respond in a specific way when receiving certain information. In order for the neuron to behave correctly in response to specific data, it is essential to train it: thus, it is necessary to use a *training dataset* that contains numerous input vectors $\mathbf{x}$, each accompanied by the correct response that the neuron should provide. When working with numerous input vectors, the notation X_j is introduced to indicate the j-th vector $\mathbf{x}$ in the dataset. Formally, we define the dataset $\mathscr{D}$ as:

$$\mathscr{D} = \left\{ \{X_1, y_1\}, \{X_2, y_2\}, \dots \{X_j, y_j\}, \dots \{X_M, y_M\} \right\}$$

The dataset[1] consists of M input vectors — X_j ($j \in [1, M]$) — and exactly M responses y_j, which represent the expected behavior of the neuron when the input vector is X_j. It is therefore

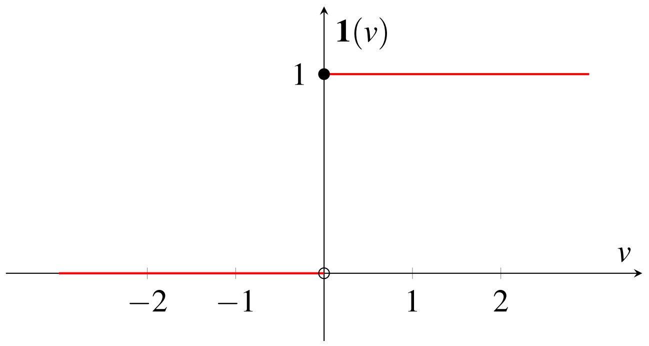

Figure 3.2: Graph of the step function $\mathbf{1}(v)$. It can be observed that it equals zero for values strictly less than zero and one for values greater than or equal to zero.

[1]A dataset defined in this way is used in *supervised learning*, where both the input vectors and the expected responses are present in the dataset. In contrast, *unsupervised learning* — which will not be covered — includes only the input vectors, with the expected responses being unknown.

possible to define the matrix **X**, which contains exactly M vectors — each in a row — composed of exactly n components (or *features*), which are the components associated with the weight vector **w**, previously introduced. Associated with the matrix **X** is the vector **y**, also of dimension M, such that the component y_j is the expected response for the input vector X_j.

$$\mathbf{X} = \begin{bmatrix} X_1^0 & X_1^1 & \cdots & X_1^i & \cdots & X_1^n \\ X_2^0 & X_2^1 & \cdots & X_2^i & \cdots & X_2^n \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ X_j^0 & X_j^1 & \cdots & X_j^i & \cdots & X_j^n \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ X_M^0 & X_M^1 & \cdots & X_M^i & \cdots & X_M^n \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_j \\ \vdots \\ y_M \end{bmatrix}$$

From this, it follows that the vector X_j is dimensionally compatible with the weight vector **w** since they have exactly the same dimension.

$$X_j = \left[X_j^0, X_j^1, \ldots, X_j^n\right] \qquad \mathbf{w} = [w_0, w_1, \ldots, w_n]$$

Therefore, the dataset $\mathscr{D}$ can be rewritten using the matrix **X** and the associated expected response vector **y**.

$$\mathscr{D} = \{\mathbf{X}, \mathbf{y}\}$$

With an initial dataset, it is possible to define a generic algorithm capable of describing the learning process of an artificial neuron. As described in Algorithm 1, after initializing the weight vector **w**, for each vector X_j in the dataset $\mathscr{D}$, the signal v and the value of the activation function $a = f(v)$ are calculated. The idea behind the algorithm is to modify the weight vector **w** based on the result of the activation function in relation to the processed signal. In this way, once all

Algoritmo 1 Artificial neuron training

```
Input: Dataset $\mathscr{D}$

Initialize w with random numbers
$j \leftarrow 1$
while $j \leq M$ do
    Compute $v$ as a function of $X_j$ and w
    Determine $a$ based on $v$ and $y_j$
    Adjust w according to the result $a$
    $j \leftarrow j+1$
end while
```

the vectors in the dataset have been processed, it is possible to verify whether the neuron has learned the desired behavior based on the input vector [64].

3.1.2 Modello percettrone

One of the artificial neuron models used in neural networks is the perceptron. Given an input vector X_j[2] and a weight vector $\mathbf{w}$, the signal v is obtained by the algebraic sum of the products of each component:

$$v = g\left(\mathbf{w}, X_j\right) = \sum_{i=0}^{n} w_i X_j^i = \mathbf{w} \cdot X_j$$

The activation function $f(v)$ of the perceptron is the *"sign"* function (Figure 3.3) — also called the *bipolar step* function — defined as follows.

$$a(v) = \mathbf{sign}(v) = \mathbf{sign}(\mathbf{w} \cdot X_j) = \begin{cases} 1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$

The most important aspect of the perceptron is the *learning rule*, which defines the criterion according to which the weight vector is updated based on the model's prediction. If the prediction a_j for the vector X_j differs from the expected response y_j, then the weight vector is updated as follows [64], [66].

$$w_i = w_i + y_j X_j^i$$

In addition to the perceptron, the *sigmoid neuron* is introduced, which is very similar to the perceptron except for the activation function. Instead of the discontinuous "sign" function, the *sigmoid* function $\sigma(v)$ is introduced. This function is continuous and returns values between zero and one (Figure 3.4). The sigmoid function, defined below, eliminates the discontinuities of the perceptron's activation function and provides a range of real values rather than a binary

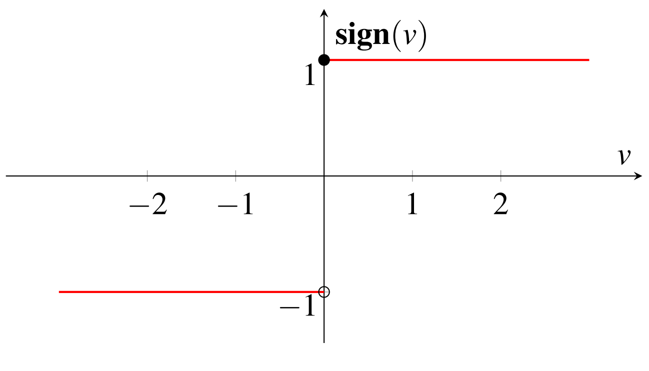

Figure 3.3: Graph of the sign function $\mathbf{sign}(v)$. It can be observed that it equals -1 for values strictly less than zero and 1 for values greater than or equal to zero.

[2]The input vector is referred to as the j-th row of the input matrix $\mathbf{X}$ from the dataset $\mathscr{D}$.

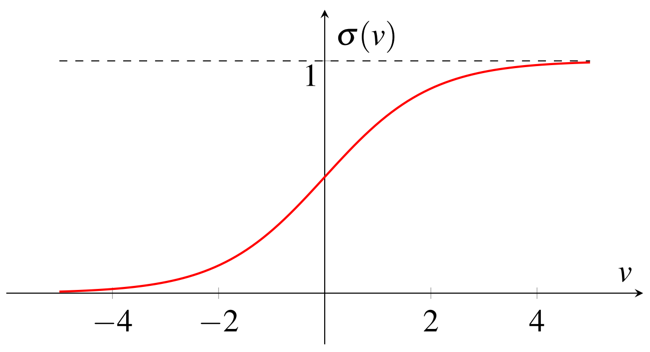

Figure 3.4: Graph of the sigmoid function $\sigma(v)$.

result.

$$\sigma(v) = \frac{1}{1+e^{-v}} = \frac{1}{1+e^{-\mathbf{w}\cdot X_j}}$$

This important difference makes the sigmoid neuron more flexible compared to the classic perceptron and better suited for use in neural networks [66].

3.1.3 Gradient descent

The learning rule introduced in the perceptron, despite its simplicity, is often replaced by the learning rule derived from *Gradient Descent* (GD, represented in Figure 3.5), an approach aimed at finding the minimum of a differentiable function, often referred to as the “cost” function. Given the weight vector $\mathbf{w} = [w_0, w_1, \dots, w_n]$, during the training of the neuron, the algorithm seeks to modify this vector by progressively minimizing the error between the value predicted by the model and the expected result. More precisely, given a cost function $C(w)$ — also called the *loss function* — the learning rule for GD is:

$$\mathbf{w} = \mathbf{w} - \eta \, \nabla C(\mathbf{w})$$

Here, η is a positive parameter, called the *learning rate*, used to control the speed at which the weight vector is updated during the algorithm’s execution. The term $\nabla C(\mathbf{w})$ represents the gradient of the loss function. The gradient of a multi-dimensional function is a vector that points in the direction of the steepest increase in the function’s value; consequently, the opposite direction is followed to reach the function’s minimum (local or global). Figure 3.5 graphically illustrates how the GD approach attempts to reach this value with each iteration. In this case, the function is represented in two dimensions, but in general, the loss function is n-dimensional. This function can be any error function: one of the most common is the function that calculates the mean of the squared errors (commonly called MSE, *Mean Squared Errors*). As a practical example, given the weight vector $\mathbf{w}$, a dataset $\mathscr{D}$ — defined as previously — and an activation function a, the MSE is defined as:

$$C(\mathbf{w}) = \frac{1}{M} \sum_{j=1}^{M} \left[y_j - a(\mathbf{w}, X_j) \right]^2$$

The approach adopted by this algorithm, while effective, is not entirely efficient. The computational complexity required is significant: at each iteration of the neuron's training, it is necessary to compute the derivative of the loss function, which involves predicting all input vectors X_j in the dataset $\mathscr{D}$ and comparing them to the expected output y_j. For medium to large datasets, this approach is quite inefficient, despite being precise. Additionally, gradient descent may lead to the weight vector being updated to a local minimum of the error function, resulting in a sub-optimal solution. Both of these issues can be addressed by a variant of this algorithm, called *Stochastic Gradient Descent* (SGD). For each iteration of the training algorithm, a number $m < M$ of input vectors with their respective expected outputs are taken, and the vector $\mathbf{w}$ is updated based on this subset of the dataset, also referred to as a *mini-batch*. In this way, the computational complexity is drastically reduced, as computing the derivative of the loss function for a smaller portion of the dataset requires fewer resources. Additionally, the randomness of the mini-batch selection can prevent convergence to a local minimum, as it makes the descent along the error function slightly more unpredictable, reducing the likelihood of sub-optimal results [66]–[68].

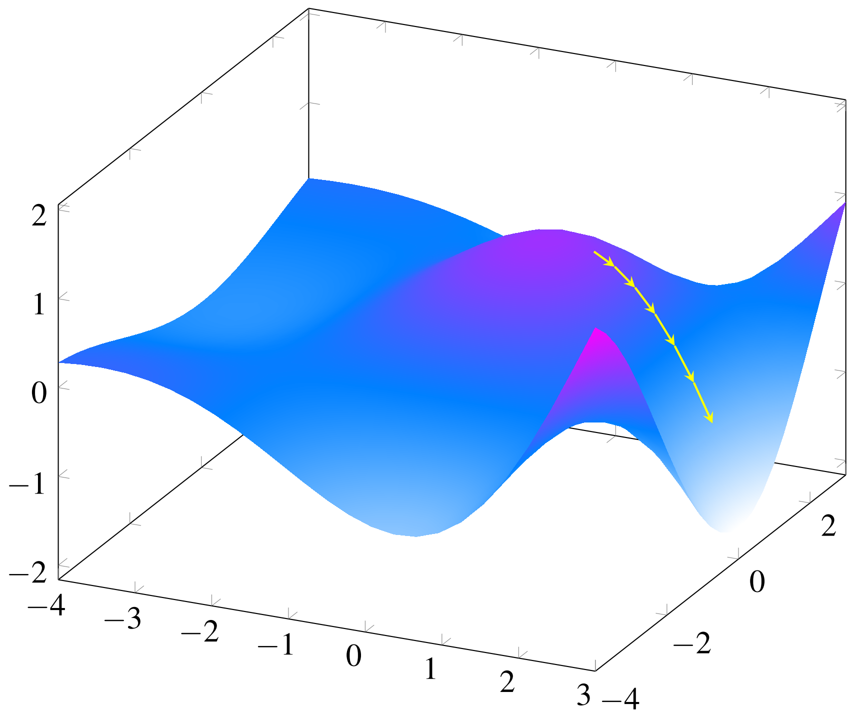

Figure 3.5: Gradient descent in a two-dimensional function. It is observed that the algorithm, through the gradient of the function, is able to move towards the minimum (local or global) of the function, similar to a glass marble rolling downhill.

3.1.4 Neural network

A neural network is a structure composed of artificial neurons that are organized into layers. In a multilayer neural network, there is always an input layer and an output layer. The layers in between are called hidden layers. The neurons in the same layer are never connected to each other but are only connected to the neurons in the previous and subsequent layers. More precisely, in a neural network, the neurons in layer ℓ receive signals from the neurons in layer $\ell-1$ and, after processing the information, send the processed signal to the neurons in layer $\ell+1$: this type of network is called *feedforward* because there are no cycles in its structure. Observing Figure 3.6, a new notation is introduced for neurons within a network. The neuron j in layer ℓ is defined as $N_j^{(\ell)}$, and its activation function output is denoted as $a_j^{(\ell)}$. The weight $w_{j,i}^{(\ell)}$ represents the weight connecting neuron $N_i^{(\ell-1)}$ to neuron $N_j^{(\ell)}$, i.e., the connection between the neuron at position i in the previous layer $\ell-1$ and the neuron $N_j^{(\ell)}$. Additionally, the biases for each neuron are made explicit: $b_j^{(\ell)}$ indicates the bias associated with the j-th neuron in layer ℓ. Assuming the use of sigmoid neurons within the NN, the output $a_j^{(\ell)}$ of neuron $N_j^{(\ell)}$, i.e., the output of its activation function, can be calculated as follows:

$$a_j^{(\ell)} = \sigma(v) = \sigma\left(\sum_i w_{j,i}^{(\ell)}\, a_i^{(\ell-1)} + b_j^{(\ell)}\right)$$

This formula indicates that the output of the activation function of neuron $N_j^{(\ell)}$ is given by the sigmoid calculated over the input to the neuron. The input is the weighted sum of the outputs of the neurons in the previous layer $(\ell-1)$, multiplied by their respective weights and added to the bias of the neuron. This notation can be rewritten in matrix form for greater elegance.

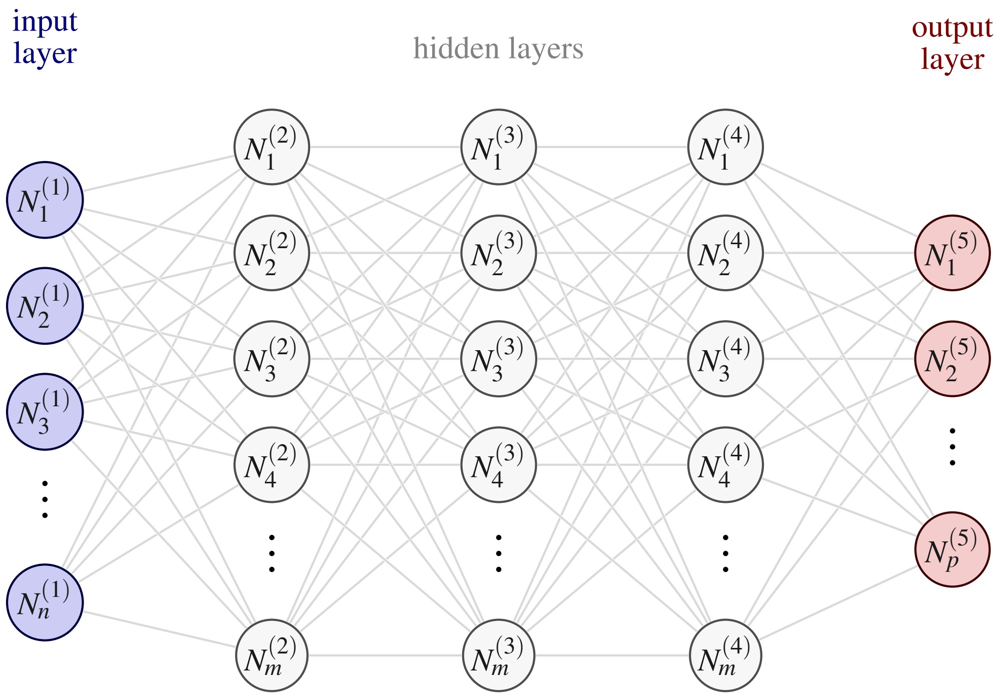

Figure 3.6: Multilayer neural network(image adapted from [69]).

The matrix $\mathbf{W}^{(\ell)}$ is defined to contain the weights connecting the neurons in layer $\ell-1$ to those in layer ℓ. Similarly, the vector $\mathbf{b}^{(\ell)}$ contains the biases of the neurons in layer ℓ. Finally, the vector $\mathbf{a}^{(\ell)}$ contains the output values of the activation functions of all neurons in layer ℓ. The formula can then be rewritten as follows:

$$\mathbf{a}^{(\ell)} = \sigma\left(\mathbf{W}^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right) = \sigma\left(\mathbf{v}^{(\ell)}\right)$$

This result provides an overview of the output of layer ℓ, rather than focusing on a single neuron $N_j^{(\ell)}$. Furthermore, a vector $\mathbf{v}^{(\ell)}$ is introduced, whose value is simply the input to the sigmoid function, i.e., the weighted sum of the inputs from layer $\ell-1$.

3.1.5 Backpropagation

Just as in the case of a single artificial neuron, an ANN also needs to be trained. To train a neural network, the *backpropagation* algorithm is used, introduced for the first time in 1986. Backpropagation, leveraging the principles of GD, adjusts the network's weights with the goal of minimizing a *cost function* C. This involves computing the gradient of C, specifically the partial derivative of C with respect to the weight of a neuron and its bias:

$$\frac{\partial C}{\partial w_{j,i}^{(\ell)}} \qquad\qquad \frac{\partial C}{\partial b_j^{(\ell)}}$$

Additionally, the *neuron error* $N_j^{(\ell)}$, denoted as $\delta_j^{(\ell)}$, is defined as:

$$\delta_j^{(\ell)} = \frac{\partial C}{\partial v_j^{(\ell)}}$$

This critical value indicates how much the cost function varies in response to a small change in the input of neuron $N_j^{(\ell)}$. Assuming we are at layer ℓ, modifying the vector $\delta^{(\ell)}$, and thus slightly altering the input vector $\mathbf{v}^{(\ell)}$, can lead to an overall reduction in the loss function C, propagating the input changes at layer ℓ to subsequent layers. The backpropagation algorithm provides a method for calculating the value $\delta_j^{(\ell)}$ and linking it to the partial derivatives.

Backpropagation can be described using four fundamental equations [66]. The first equation provides a method for calculating the value $\delta_j^{(L)}$, which represents the prediction error at the output layer L. As shown in Equation 3.2, this value is the product of two derivatives. The first partial derivative indicates the change in the cost function with respect to the output of neuron $N_j^{(L)}$: if the neuron has little influence on the final cost, then $\delta_j^{(L)}$ is very small. The second derivative indicates how much the sigmoid function varies at the point $v_j^{(L)}$.

$$\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}}\,\sigma'\left(v_j^{(L)}\right) \tag{3.2}$$

The second equation describing backpropagation relates $\delta^{(\ell)}$ to the error at the next layer $\delta^{(\ell+1)}$, as shown in Equation 3.3. Multiplying the transposed weight matrix $\left(\mathbf{W}^{(\ell+1)}\right)^T$ by the error $\delta^{(\ell+1)}$ is intuitively equivalent to measuring the error propagated back to layer ℓ. This result is then multiplied element-wise with $\sigma'\left(v^{(\ell)}\right)$ using the operator "$\odot$", which represents element-wise multiplication[3]. This step allows the error to be "propagated backward" through the activation function, providing the value of $\delta^{(\ell)}$. This equation offers a way to compute the weights of the previous layer by backpropagating the error through the transposed weights and the derivative of the activation function.

$$\delta^{(\ell)} = \left[\left(\mathbf{W}^{(\ell+1)}\right)^T \delta^{(\ell+1)}\right] \odot \sigma'\left(v^{(\ell)}\right) \tag{3.3}$$

The third equation (3.4) relates the derivative of the loss function with respect to the bias of a neuron ($b_j^{(\ell)}$) to its error $\delta_j^{(\ell)}$.

$$\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)} \tag{3.4}$$

Finally, the fourth equation (3.5) connects the derivative of the cost function with respect to a neuron's weight to the error of the neuron. More precisely, the error $\delta_j^{(\ell)}$ of neuron $N_j^{(\ell)}$ is linked to the derivative of the cost function with respect to the weight $w_{j,i}^{(\ell)}$ through the output value $a_i^{(\ell-1)}$ of the i-th neuron in the previous layer $\ell-1$.

$$\frac{\partial C}{\partial w_{j,i}^{(\ell)}} = a_i^{(\ell-1)}\,\delta_j^{(\ell)} \tag{3.5}$$

After defining the four fundamental equations of the algorithm, we can understand the pseudocode of backpropagation, provided in Algorithm 2. It can be observed that this algorithm calculates the error values of the neurons starting from the output layer and propagates them backward, eventually computing the gradient of the cost function C.
This fundamental algorithm, by calculating the gradient of the cost function for every iteration—or, in technical terms, *epoch*—of network training, seeks to minimize the loss function C by optimizing the weight values. In this way, the artificial neural network learns from the input data to provide valid predictions for data not present in the dataset. Running such a powerful algorithm requires significant computational effort: modern neural networks consist of thousands of neurons per layer, requiring the precise optimization of millions of weights. For this reason, only in the past decade—thanks to the development of advanced hardware—has it been possible to tackle these computational challenges [64], [66], [70].

[3]Given two vectors, $a = [a_1, a_2]$ and $b = [b_1, b_2]$, their element-wise product is $a \odot b = [a_1 b_1, a_2 b_2]$

Algoritmo 2 Backpropagation

```
Input: set of activation function outputs from the input layer, $\mathbf{a}^{(1)}$

For each $\ell \in [2, L]$, compute $\mathbf{v}^{(\ell)} = \mathbf{W}^{(\ell)}\mathbf{a}^{(\ell-1)} + \mathbf{b}^{(\ell)}$ and $\mathbf{a}^{(\ell)} = \sigma\left(\mathbf{v}^{(\ell)}\right)$

Compute the prediction error $\delta_j^{(L)} = \frac{\partial C}{\partial a_j^{(L)}}\,\sigma'\left(v_j^{(L)}\right)$

For each $\ell \in [L-1, 2]$, compute $\delta^{(\ell)} = \left[\left(\mathbf{W}^{(\ell+1)}\right)^T \delta^{(\ell+1)}\right] \odot \sigma'\left(v^{(\ell)}\right)$

Output: gradient of the cost function, provided by $\frac{\partial C}{\partial w_{j,i}^{(\ell)}} = a_i^{(\ell-1)}\,\delta_j^{(\ell)}$ and $\frac{\partial C}{\partial b_j^{(\ell)}} = \delta_j^{(\ell)}$
```

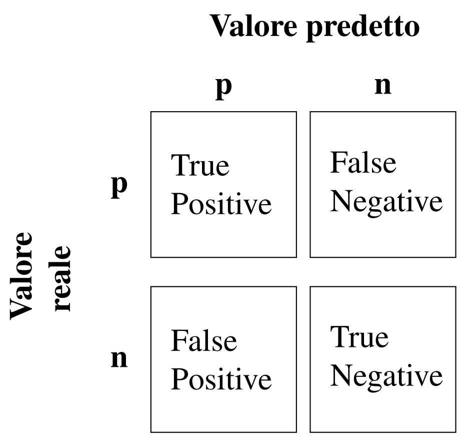

Figure 3.7: Representation of the confusion matrix.

3.1.6 Performance Metrics

Neural networks generally process a specific input and produce an output value, between 0 and 1, indicating the probability that the input belongs to a certain category, called a *class*. After training the network, it is essential to evaluate how effective the training has been by analyzing the model's performance on a dataset containing data never used during the training phase, called the *testing dataset*. Once the network processes the test dataset, predictions on the data can be categorized into four types. If the model's prediction is *positive*, meaning it correctly identifies an element as belonging to a certain category, and the element truly belongs to it, this is a true positive, or *true positive* (TP). Similarly, if an input is recognized as not belonging to a class and it indeed does not belong, this is a true negative, or *true negative* (TN). In the other two cases, the model categorizes the element as belonging to a class when it actually does not—this results in a *false positive* (FP)—or the model fails to categorize an element as belonging to a class when it actually does, resulting in a *false negative* (FN). The *confusion matrix* (Figure 3.7) summarizes this concept.

Since the output layer provides the probability that the given data belongs to a class or

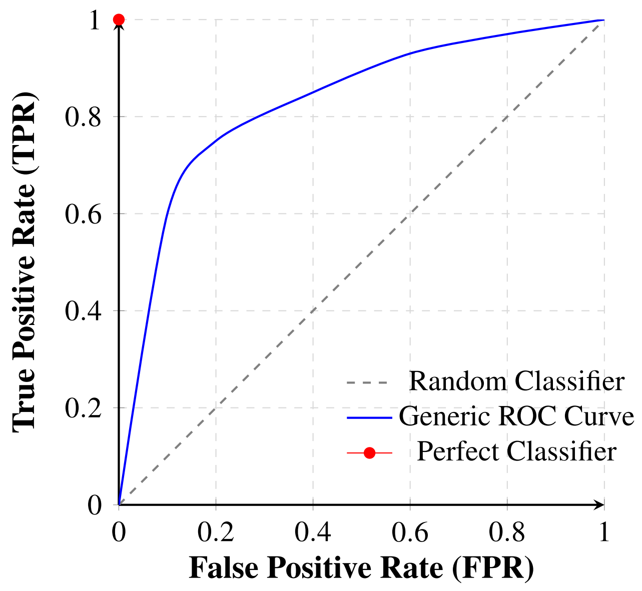

Figure 3.8: Illustration of a generic ROC curve.

not, it is necessary to set a threshold: if this value is exceeded, the data will be considered as belonging to the category; otherwise, it will be excluded. To select the optimal threshold, the ROC (*Receiver Operating Characteristics*) curve is used, which relates two metrics: the *True Positive Rate* (TPR) and the *False Positive Rate* (FPR). The TPR is the ratio of true positives (i.e., the values that are classified as positive and are truly positive) to all values that are actually positive (i.e., the true positives and false negatives):

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

A high TPR indicates that the model can correctly identify positive elements, meaning those belonging to the class under consideration. Conversely, a low TPR suggests that the model fails to recognize positive instances, thereby increasing the number of false negatives. In contrast to TPR, there is FPR, defined as the ratio of false positives (i.e., elements that are negative but classified as positive) to all negative values (i.e., false positives and true negatives):

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

This metric indicates how well the model can identify negative classes. A smaller value indicates that the model is more accurate at recognition. Consequently, a high FPR implies that the model cannot adequately distinguish between negative and positive classes [71].

The ROC curve represents, for different thresholds (ranging from 0 to 1), the TPR and FPR values of the model.

As shown in Figure 3.8, three cases are represented. The straight dashed line represents a random model, meaning one that classifies the data arbitrarily. The blue line represents a generic model. At the beginning of the curve, it can be observed that the FPR grows more slowly compared to the TPR, indicating that the model successfully identifies many true positives without classifying many false positives. In this case, the threshold is very high because an element must have a high probability of belonging to the class to be classified in the positive category. As the threshold decreases, both the TPR and FPR of the model increase, as many more elements are classified into the positive category, even if their true class is negative. The red point represents the ideal model, which has the maximum TPR value (i.e., 1) and the minimum FPR value (i.e., 0).

To simplify the interpretation of the ROC curve, the area under the curve is often used, called AUC, *Area Under the Curve* — or AUROC, *Area Under the ROC Curve*. This value greatly helps to summarize the predictive capabilities of the algorithm into a single number (Figure 3.9). It has been shown that this value provides more meaningful insights compared to other metrics, including accuracy[4] [72]. Additionally, this metric is more convenient for comparing the performance of two models rather than directly comparing their ROC curves: for this reason, it is one of the most widely used metrics today for evaluating DL models.

3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) were first introduced in 1998 with the publication of the paper "Gradient-based learning applied to document recognition" [73], which describes an artificial network called *LeNet-5*. The goal was to identify handwritten documents. This paper laid the foundation for the design of a convolutional network [74].

A convolutional network is a feedforward neural network that contains special layers aimed

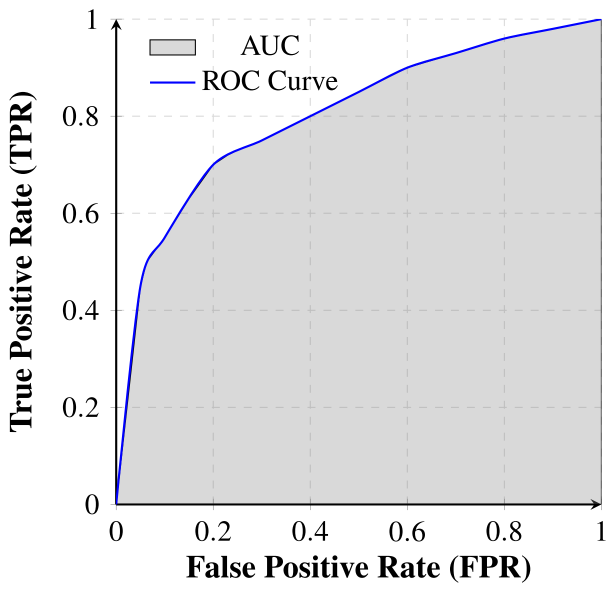

Figure 3.9: Area under the ROC curve. This value is commonly used to compare different models.

[4]The accuracy of a model is given by the ratio of the model's correct predictions (TP and TN) to the entire dataset.

at reducing the number of weights in the network. Three fundamental layers characterize a CNN: the *convolution layer*, the *padding layer*, and the *ReLU layer*. These networks are organized in such a way as to recognize the spatial characteristics of an image. For this reason, the input layer is generally represented as a two-dimensional matrix[5] of pixels: this also facilitates the understanding of the operations that process the input.

3.2.1 Convolutional Layer

In convolutional neural networks, parameters are grouped into two-dimensional units called filters or *kernels*. These are matrices, generally square, that are smaller in size compared to the input. These units enable the convolution operation with the previous layer. As shown in Figure 3.10, the convolution operation slides the filter K over the matrix M and, at each step, computes the dot product between the kernel and the values of M selected at that moment. It is observed that the number of values resulting from this operation is exactly the number of neurons present in the next layer. This value can be easily calculated by knowing the dimensions of the initial matrix and the kernel. Given the number of columns in the matrix, i.e., its width w, and the number of rows, i.e., its height h, then, applying a convolution with an $n \times n$ filter, the dimensions at level $\ell + 1$ are:

$$w^{(\ell+1)} = w^{(\ell)} - n + 1 \qquad\qquad h^{(\ell+1)} = h^{(\ell)} - n + 1$$

Observing Figure 3.10, we notice that the matrix M of size 7×7 becomes a 5×5 matrix when a convolution is applied with a filter of size 3×3. Additionally, it is specified that between one layer and the next, not just a single convolution operation is performed, but multiple convolutions are carried out, each with a different filter (though of the same size). Consequently, the result of these operations will be a number of matrices d, which correspond to the number of filters used to perform the convolution on the previous layer. Applying multiple filters to the same layer makes it possible to recognize specific patterns: for example, a set of filters can be

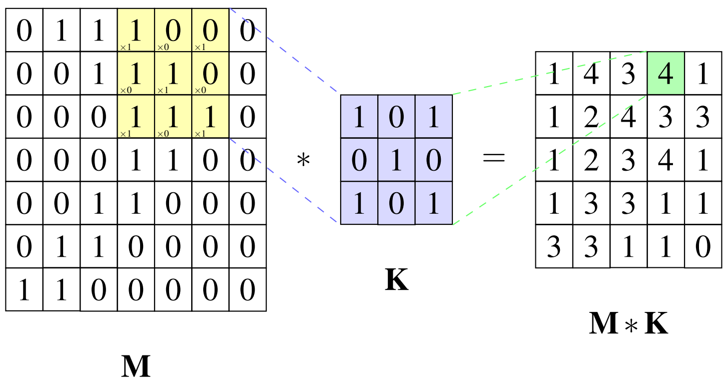

Figure 3.10: The convolution operation in a two-dimensional matrix (adapted from [75]).

[5]Assuming, for simplicity, that the image is in grayscale.

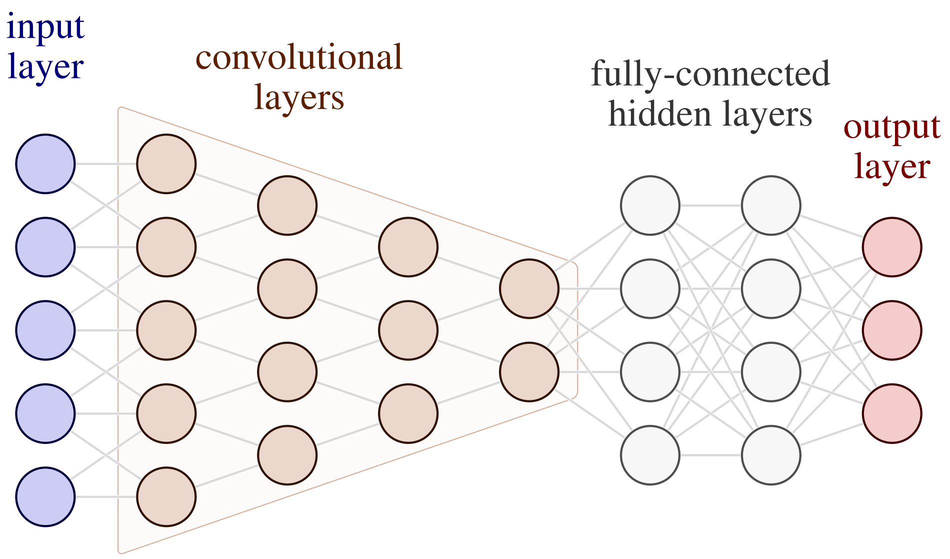

Figure 3.11: Representation of a convolutional neural network. Note the presence of multiple convolutional layers that enable the extraction of progressively more complex patterns (adapted from [69]).

used to identify a shape within an image. In this way, in a CNN, it is possible to have multiple convolutional layers, each of which recognizes an increasingly complex pattern from the initial image. Having multiple convolutional layers processing the image ensures that the last convolutional layer examines larger portions of the image, recognizing more intricate features and structures [74], [76].

As previously noted, after each convolutional layer, the image size is reduced in proportion to the filter size. Generally, dimensional reduction leads to information loss and should be avoided. For this reason, before performing the convolution operation, padding is applied. Through this operation, a series of "0"[6] values are added along the edges of the image so that, once processed, it remains the same size. On each border of the image, exactly $(n-1)/2$ pixels are added, where n is the filter size. Figure 3.12 illustrates the convolution operation presented in Figure 3.10, with the difference that the initial matrix has undergone the padding operation. Comparing the two figures, we observe that in the first case, the convolution reduced the original matrix size, while in the second case, through padding, the size remains unchanged due to the values added along the matrix borders [74].

3.2.2 ReLU Layer

Generally, each convolutional layer is followed by a ReLU layer (*Rectified Linear Unit*). This layer replaces the sigmoid activation function with a function called the rectifier, defined

[6]The value zero is chosen precisely because it has no effect and does not distort the numerical result of the convolution.

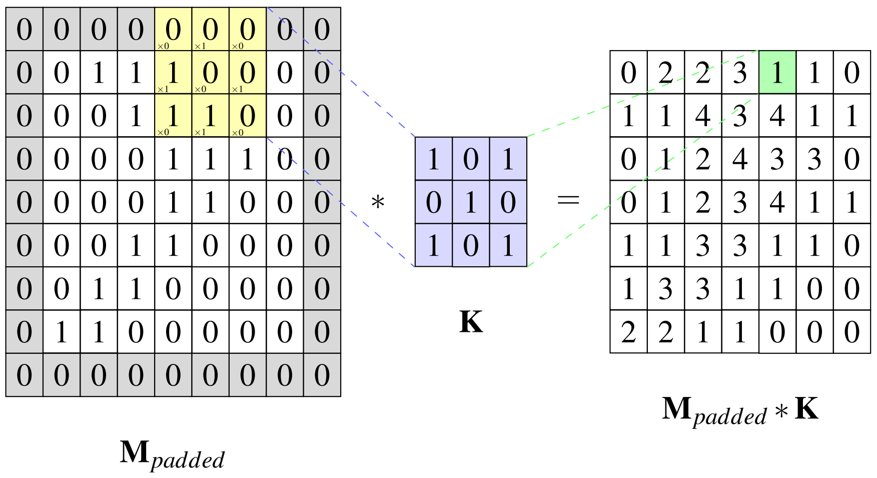

Figure 3.12: Padding of a matrix before convolution. This way, the matrix size remains unchanged after convolution (adapted from [75]).

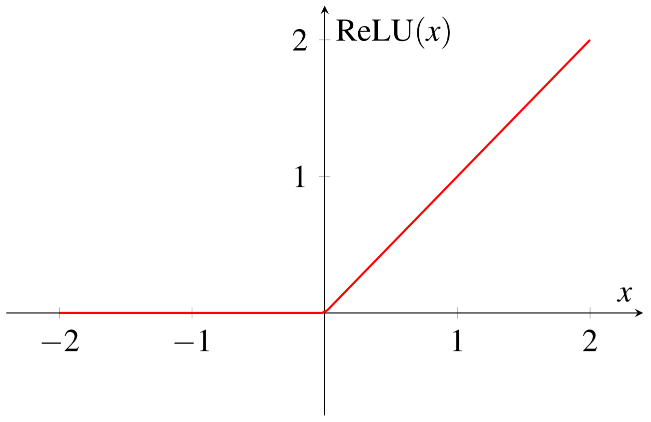

Figure 3.13: Graph of the rectifier function **ReLU**(x).

as follows:

$$\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } v > 0 \\ 0 & \text{if } v \leq 0 \end{cases}$$

It is observed that, unlike the convolutional layer, this layer does not change the matrix size as it directly maps the existing data without altering its structure. The ReLU activation function, represented in Figure 3.13, has been recently introduced in neural networks as it provides several advantages over sigmoid functions, particularly in terms of training speed[7] [74].

3.2.3 POOLING LAYER

After processing the information through convolution, in order to reduce computational load, the extracted information is concentrated using *pooling*. The most common type of pooling in NN is *max-pooling*. This operation, similar to a filter, moves along the matrix and

[7]This is because the derivative of a linear function is much simpler than the derivative of the sigmoid function.

extracts the maximum value from the selected area (Figure 3.14). The idea behind max-pooling is to retain the most prominent features from the given matrix to reduce information complexity, making the training phase less computationally expensive. In addition to max-pooling, there is also *avg-pooling*, which calculates the average of the pixels in the selected region, extracting the "average property" instead of the dominant one [74], [77].

Finally, as in all neural networks, CNNs include *fully-connected layers*. These layers, which appear only after the convolutional layers (as shown in Figure 3.11), serve the same function as a classic feedforward NN: performing a series of calculations that allow the network to correctly predict the output based on the provided input [74], [77].

3.2.4 Applications

Although CNNs have so far been described as tools that process only images, their applications are much broader. Among their numerous applications, some notable ones include *Natural Language Processing* (NLP)—a field focused on processing natural language through speech recognition and text classification—and *Computer Vision*, which includes visual recognition, *scene labeling*, and the detection of objects and human actions [78]–[80].

Beyond these applications, convolutional neural networks are widely used in bioinformatics. More specifically, one-dimensional CNNs[8] analyze sequences of DNA and RNA, learning sequential patterns—also called *motifs* in bioinformatics—that become progressively more complex through different convolutional layers. This approach enables a deeper understanding of problems related to gene expression regulatory elements, including non-coding variants [81].

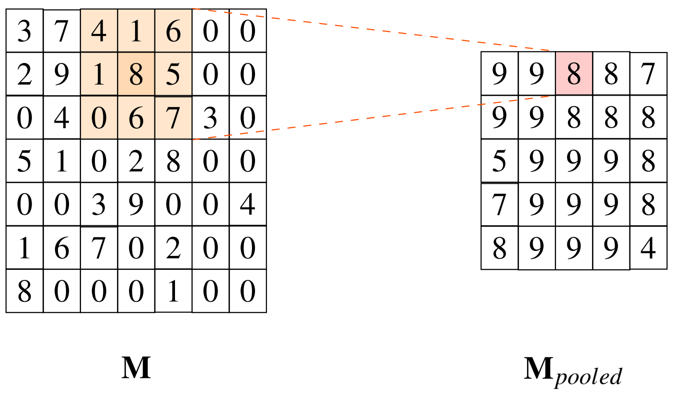

Figure 3.14: Max-pooling operation in a matrix. This operation allows extracting the dominant features after convolution (adapted from [75]).

[8] Where the input extends in one dimension, just like a sequence.

4 Convolutional Networks and Non-Coding Variants

In this chapter, three tools based on convolutional neural networks will be presented, developed with the goal of improving the functional understanding of non-coding mutations. The models discussed are *DeepSEA* [16], *Basset* [17], and *DeepSATA* [18]. Each of these models will be thoroughly analyzed to understand the main characteristics of each tool. More precisely, the dataset used to train the model, its structure—i.e., the number and type of layers present—and the applied training techniques will be discussed.

4.1 DeepSEA

The first tool to be analyzed is DeepSEA (*Deep learning-based Sequence Analyzer*), introduced in 2015 with the goal of predicting the effect of non-coding variants in chromatin. To ensure accurate prediction, this model considers long DNA sequences to better understand the context in which the mutation occurs and thus comprehend its functionalities, also thanks to the hierarchical structure of the convolutional layers, which allow the examination of local and global patterns.

Furthermore, DeepSEA is capable of learning multiple chromatin functionalities simultaneously using a multitask learning approach, which allows training the model on multiple related tasks at the same time. To gain a deeper understanding of the functional effects of non-coding mutations, it was trained to predict 919 chromatin profiles divided into three macro categories:

- 690 types of sequences associated with transcription factor (TF) binding sites, which play a fundamental role during transcription;
- 125 profiles of DNase I hypersensitive regions (DHS), indicating the presence or absence of regulatory elements in DNA, which are also important during the transcription phase;
- 104 profiles of histone modifications, i.e., mutations in histones that make DNA in chromatin less accessible, thus preventing correct transcription;

The model—implemented with the *Torch7* library—consists of exactly three convolutional layers: the first layer contains 320 filters, the second 480, and the last contains 960 kernels. The filters are *Position Weight Matrices* (PWM), i.e., matrices composed of 4 rows and M columns that, for each nucleotide base, indicate the probability of appearing in a specific position, as shown in Figure 4.1.

The filters analyze the input sequence and, through convolution, extract significant patterns by gradually shifting the window by one base in the sequence. In the first layer, the kernel has dimensions $M \times 4$, where M is the window length and 4 represents the number of nucleotide bases. The subsequent convolutional layers have dimensions $M \times k$, where k is the number of kernels used in the previous convolutional layer. After each convolutional layer, a ReLU (Figure 3.13) is applied, followed by a max-pooling layer that extracts the predominant feature from the convolution result. In this case, the pooling window step does not equal one but rather matches the window length itself. Finally, after the three convolutional layers, there is a fully connected layer—which receives the max-pooling output of the third convolution and applies a ReLU—and the output layer, which processes the information with a sigmoid function (Figure 3.4) that calculates the probability that the input sequence corresponds to one of the 919 profiles.

The 919 chromatin profiles were obtained from the *Encyclopedia of DNA Elements* (ENCODE) and *Roadmap Epigenomics* projects [82], [83]. The extracted sequences were divided into 200 bp bins[1], totaling $521,636,200$ bp. Each bin was then labeled to one of the 919 chromatin profiles if more than half of the sequence matched the theoretical profile. Each extracted bin was centered on a 1000 bp human genome sequence to provide DeepSEA with more context for deriving more meaningful information. Each input sequence was transformed into a 1000×4 matrix through One-Hot encoding, which converts the one-dimensional sequence into a two-dimensional matrix. More precisely, the nucleotide bases are encoded as follows: $A \rightarrow [1,0,0,0]$, $T \rightarrow [0,1,0,0]$, $C \rightarrow [0,0,1,0]$, and $G \rightarrow [0,0,0,1]$. Finally, each matrix is associated with the respective label vector containing the network's expected output.

To effectively train DeepSEA, the cost function was defined as the *Negative Log Likelihood*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
A	0.9	0.1	0.4	0.25	0.6	0.7	0.2	0.5	0.8	0.3
C	0.05	0.7	0.3	0.25	0.1	0.1	0.4	0.2	0.05	0.2
G	0.03	0.1	0.2	0.25	0.2	0.1	0.2	0.2	0.05	0.4
T	0.02	0.1	0.1	0.25	0.1	0.1	0.2	0.1	0.1	0.1

Figure 4.1: Representation of a PWM used as a filter within a CNN.

[1]With " bp", "base pair" is meant, indicating the sequence length in the number of base pairs.

(NLL)—also known as *Binary Cross Entropy* (BCE)—defined as follows:

$$\mathbf{NLL} = -\sum_{s}\sum_{p} \log\left[y_{s,p}\,\sigma_p(X_s) + (1-y_{s,p})(1-\sigma_p(X_s))\right]$$

In this formula, s is the index of the s-th sample (X_s) of the dataset, and p is the index of the chromatin profile. Consequently, the value $y_{s,p}$ represents the correct value of sample s with respect to chromatin profile p, while $\sigma_p(X_s)$ is DeepSEA's prediction for sample X_s concerning profile p. The actual cost function is defined as the NLL function summed with other values to prevent extitoverfitting[2]. The gradient of the loss function was computed using the back-propagation algorithm and then used for network optimization. The optimization algorithm employed was SGD with momentum, a variant used to further increase the chances of avoiding local minima [74]. Additionally, extitdropout training was applied, which involves disabling some neurons during training epochs to make the network more robust against overfitting [66].

4.2 Basset

Basset[3] is a powerful tool developed in 2016, designed to analyze DNA sequences and predict the accessibility of 164 DNase I hypersensitive sites (DHS), which indicate the presence of regulatory elements. Using CNNs, this model provides valuable insights into the transcription phase from DNA to RNA by analyzing these specific chromatin sites. Like DeepSEA, multiple convolutional layers facilitate the understanding of functional aspects resulting from mutations in DHS.

Basset, like DeepSEA, was implemented using the *Torch7* library. This tool allows customization of the model, specifying the number of each type of layer, the number of filters (PWM) in convolutional layers, their size, pooling window size, the number of neurons in fully connected layers, and various hyperparameters for training and testing. Bayesian optimization was used to determine the ideal layers and hyperparameters for the architecture. The structure consists of three convolutional layers: the first contains 300 filters of length 19 bp, the second consists of 200 kernels of length 11 bp, and the third layer has 200 kernels of length 7 bp. It is important to note that after each convolutional layer, a batch normalization layer normalizes the convolution output, followed by a ReLU and a max-pooling layer. Following the three convolutional layers, there are three fully connected layers, alternating with two ReLU layers and two dropout layers (with a coefficient of 0.3) that, similar to DeepSEA, are used to prevent overfitting by disabling random neurons during training. Finally, the output layer, using the sigmoid function, calculates the probability that the input belongs to one of the 164 DHS.

[2]Overfitting occurs when the model is completely aligned with the training dataset, thus becoming less accurate in predicting data not present in the dataset.

[3]The name refers to the basset hound, known for its olfactory abilities, analogous to the model's ability to recognize patterns.

Among the DNase I hypersensitive sites, 125 were extracted from ENCODE and 39 from "*Roadmap Epigenomics*". The extracted data were processed, and all sites were isolated and enriched to form an initial dataset consisting of 600 bp sequences, totaling 2 071 886 bp. Each sequence in the dataset was then associated with a label vector indicating which of the 164 types the sequence belonged to. Among the 164 types, 17% of sites were associated with promoters, 47% were classified as intragenic sites—located within genes—and the remaining 36% were labeled as intergenic sites—located between genes. Before being used in the network, the sites were processed using One-Hot encoding, resulting in input sequences of size 600×4. From the total dataset, 71 886 bases were used for testing and another 70 000 for validation.

The model was trained using stochastic GD, aiming to optimize the BCE function whose gradient was computed via backpropagation. To prevent overfitting, the *early stopping* technique was applied, terminating training after 12 epochs when the *validation loss* remained unchanged.

4.3 DeepSATA

Published in 2023, DeepSATA is the third CNN-based tool designed to identify OCR—open chromatin regions—while also attempting to understand their function. DeepSATA focuses on non-coding mutations in TF binding sites, not only in human genomic sequences but also in other animal species such as pigs, chickens, cattle, and mice.

DeepSATA is a model based on DeepSEA. It consists of three convolutional layers, with 320, 480, and 960 kernels, respectively (also represented as PWM). Each convolutional layer is followed by a ReLU function, and then a max pooling layer extracts the most relevant features. After each convolutional layer, a dropout layer is also present, helping prevent overfitting, as in Basset. The first two dropout layers have a coefficient of 0.2, while the third has a coefficient of 0.5. Similar to Basset, the three convolutional layers are followed by a fully connected layer that prepares the data for the output layer, which, using the sigmoid function, calculates the probability that the given input belongs to one of the available OCR.

Unlike DeepSEA and Basset, the input sequence format is three-dimensional instead of two-dimensional. Specifically, the input sequence has dimensions $M \times 4 \times (N+1)$, where M is the input sequence length, 4 represents the nucleotide bases, and $N+1$—the depth of the three-dimensional matrix—indicates the specific affinities of the input sequence with the N transcription factors. An additional 1 is included to represent the One-Hot encoding layer of the sequence. This way, the TF associated with the input sequence help better understand mutation effects. The TF were obtained from the *JASPAR* database [84], and the most common motifs were identified using *FIMO*[4] [85]. The top N most frequent transcription factors were selected to be included in the initial dataset. Due to resource constraints, the authors chose $N = 10$ TF binding sites.

[4]This software identifies known motifs within a given input sequence.

The dataset containing sequences from multiple species was collected from the following databases:

- Mouse sequences were obtained from the *NCBI Sequence Read Archive* (SRA) [86];
- Human genome fragments were retrieved from ENCODE;
- Pig genomic sequences were obtained from the *Gene Expression Omnibus* (GEO) [87];
- Cattle and chicken DNA portions were downloaded from the University of California (*UC Davis*) *Farm Clusters* section [88].

After collecting all the necessary data, in a manner entirely analogous to DeepSEA, they were divided into bins of 200 bp and labeled according to the OCR they represented: if more than half of the sequence matched an OCR, the label was set to 1 for that particular chromatin region; otherwise, it was set to 0. Secondly, they were centered with sequences of 400 bp to obtain sequences of 1000 bp as input, also called *Open Chromatin Bin* (OCB). Consequently, each input bin, once the One-Hot encoding was performed, is represented by a matrix of size $1000 \times 4 \times 11$.

Since DeepSATA is a model based on DeepSEA, it was trained using stochastic GD with momentum to optimize the binary cross-entropy function. In an entirely analogous manner, the gradient of this function was calculated using the backpropagation algorithm.

5 Discussion

After having thoroughly described the characteristics of the three bioinformatics tools in Chapter 4, this chapter compares the three tools, highlighting their common properties and the aspects in which they differ, to provide an objective comparison that allows for a deeper understanding of their predictive performance.

DeepSEA, Basset, and DeepSATA share some common features. All three tools use the same sequence encoding method, One-Hot encoding, which transforms a sequence of length M into an $M \times 4$ matrix, mapping the nucleotide bases of the sequence into vectors. This way, the sequence can be properly processed by the kernels, which are PWM that assign each base in the sequence a weight (or occurrence probability) at a specific position (Figure 4.1). Finally, the three models were trained using stochastic SGD, aiming to optimize the same objective function (cost function): the binary cross-entropy (BCE).

Despite these similarities, the three tools present several differences, particularly in the network structure and the chosen training dataset. Table 5.1 summarizes the main characteristics.*DeepSEA* consists of three convolutional layers, each followed by a ReLU layer and a max-pooling layer, used to extract the dominant features processed by the convolution. The three convolutional layers contain 320, 480, and 960 filters, respectively. Subsequently, there is a fully connected layer that prepares the information to be evaluated in the output layer, which consists of a sigmoid function.

Similarly, *Basset* also consists of three convolutional layers, but in addition to the ReLU layer and the max-pooling layer, it includes a normalization layer. It also has three fully connected layers, alternating with additional ReLU layers and dropout layers to prevent overfitting. As in the previous case, the output is computed using a sigmoid function.

Like DeepSEA and Basset, *DeepSATA* also consists of three convolutional layers with 320, 480, and 960 kernels, respectively. Each convolutional layer is followed by a ReLU layer, a max-pooling layer, and a dropout layer to mitigate overfitting. After the convolutional layers, a fully connected layer processes the information for the output layer, which applies a sigmoid function. Additionally, this model, unlike the other two, processes three-dimensional input

Table 5.1: Summary of differences among the three tools.

Model	**Structure**	**Kernel**	**Input**	**Dataset**
DeepSEA	Three convolutional layers — each followed by a ReLU layer and a max-pooling layer — and a fully connected layer	320, 480, 960	1000×4	ENCODE, Roadmap Epigenomics
Basset	Three convolutional layers — each followed by a normalization layer, a ReLU layer, and a max-pooling layer — and three fully connected layers — alternating with a ReLU layer and a dropout layer	300, 200, 200	600×4	ENCODE, Roadmap Epigenomics
DeepSATA	Three convolutional layers — each followed by a ReLU layer, a max-pooling layer, and a dropout layer — and a fully connected layer	320, 480, 960	$1000 \times 4 \times 11$	ENCODE, NCBI SRA, GEO, UC Davis

data, designed to provide more context for better understanding potential mutation effects on the sequence.
The authors of the article introducing DeepSATA, after illustrating the model and its main differences from its predecessor DeepSEA, conducted an experiment comparing the predictive capabilities of DeepSEA, Basset, and DeepSATA.Specifically, after training the three models on a dataset containing sequences from various animal species, including humans, they evaluated the models' performance. From the test results (Table 5.2), it is evident that DeepSATA's predictive performance, although not by much, surpasses that of DeepSEA and Basset. In particular, a more significant difference is observed in pig genetic sequences, where the AUC/AUROC value is more than five percentage points higher. In other species, even if not by much, DeepSATA still achieves better results.

From Table 5.2, it can be seen that the AUROC values of DeepSATA and DeepSEA differ by only a few thousandths, unlike Basset's values, which are sometimes significantly lower than those of the other two models. Specifically, for human genome sequences, there is a difference of about four percentage points between Basset and DeepSEA/DeepSATA.

Consequently, the table not only indicates that DeepSATA has better predictive capabilities than the other two tools but also suggests that the DeepSEA model is better at distinguishing categories than Basset. The information in Table 5.2 is also represented in the form of ROC

Table 5.2: AUC results comparing DeepSEA, Basset, and DeepSATA.

Model	**Mice**	**Pigs**	**Cattle**	**Humans**	**Chickens**
DeepSATA	0.854	0.779	0.772	0.759	0.744
DeepSEA	0.796	0.775	0.769	0.755	0.736
Basset	0.778	0.719	0.768	0.717	0.722

curves in Figure 5.1, which shows the predictive performance of each model depending on the animal species examined. For each animal species and each of the three tools, performance graphs have been plotted. Specifically, each graph contains multiple ROC curves, each indicating the predictive capability concerning a specific OCR. The AUC value was calculated for each curve, and then the average was determined for each species, obtaining the values shown in Table 5.2.

Finally, the articles highlight some computational characteristics of the tools. Specifically, DeepSEA was trained using an GPU *NVIDIA Tesla K20m*, but the training time is not specified. However, an unofficial implementation of the tool is available in a *GitHub repository*, using the Python library TensorFlow. The repository description states that fully training the tool from scratch with a GTX 1070 took approximately 10 days [89]. The Basset article, on the other

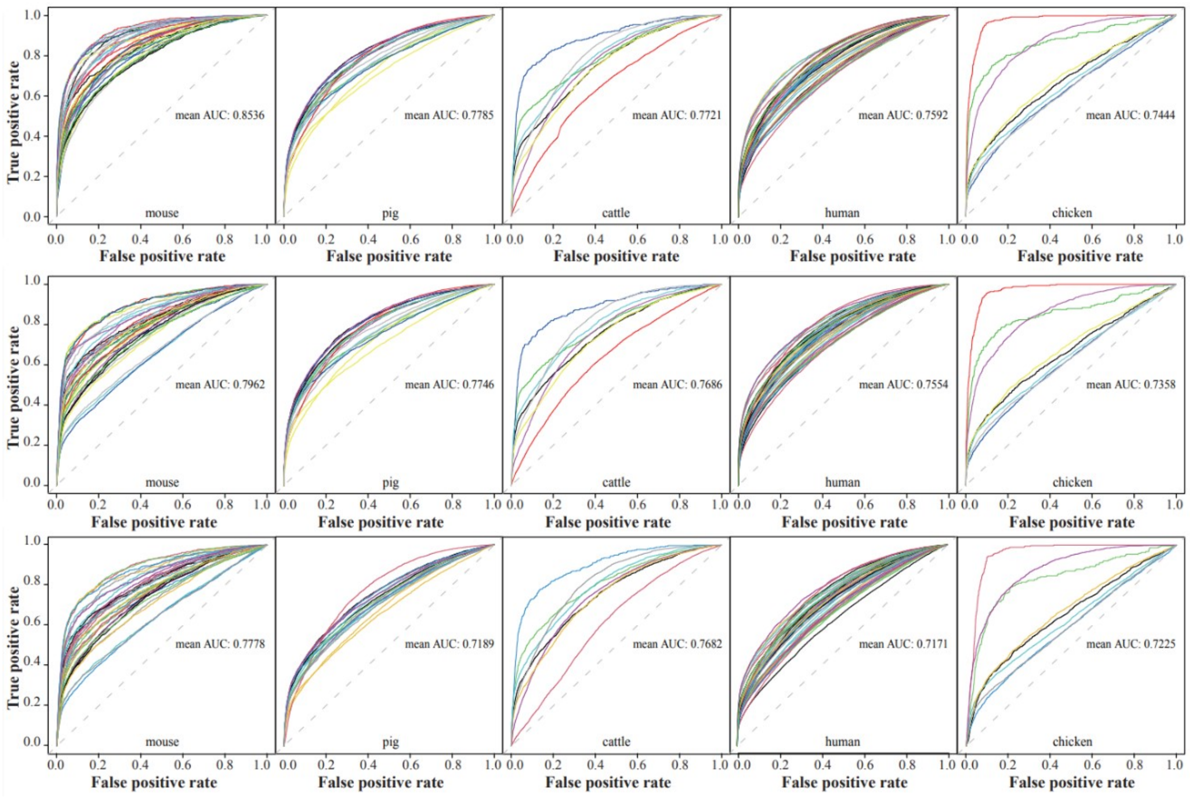

Figure 5.1: Comparison of the predictive performance of the three tools across different animal species through ROC curves, shown for each OCR, and the mean AUROC values [18].

hand, provides a more explicit description of its characteristics. Specifically, training the tool on the same GPU used for DeepSEA takes 18 minutes, while training on a *MacBook 2.8-GHz Intel Core i7* takes about six and a half hours. DeepSATA, however, does not provide any information about the machine model used for training or the time required, except that the input size had to be limited due to resource constraints.

In conclusion, given the lack of complete information from the three tools, a future experimental comparison could be conducted to obtain an impartial benchmark that provides more detailed information regarding the structure, computational requirements, and predictive performance of the tools.

6 Conclusions

The three bioinformatic tools discussed have made a significant contribution to understanding the effects of non-coding variants in the human genome. To improve predictive performance and make these tools even more precise, it is recommended to train the models with larger datasets, so as to better explore the functional consequences of non-coding mutations. Future research will focus on predicting the interactions between enhancers and promoters and their contribution to gene expression, offering new perspectives on the effects of non-coding variants.

The design of tools like these, aimed at studying non-coding variants, remains of fundamental importance. Despite the progress made thanks to these tools, the field of non-coding mutations is still underexplored by the scientific community and, therefore, requires constant and ongoing deepening and improvement. Diseases related to these mutations are widespread and still poorly understood. It is essential to continue research in this field, in the hope of making new discoveries that can enrich our understanding and lead to innovative solutions.

References

[1] S. S. Sahu and G. Panda, "Identification of protein-coding regions in dna sequences using a time-frequency filtering approach," *Genomics, Proteomics and Bioinformatics*, vol. 9, no. 1-2, pp. 45–55, 2011.

[2] T. D. Pollard, W. C. Earnshaw, J. Lippincott-Schwartz, and G. Johnson, *Cell Biology E-Book: Cell Biology E-Book*. Elsevier Health Sciences, 2022.

[3] F. Zhang and J. R. Lupski, "Non-coding genetic variants in human disease," *Human molecular genetics*, vol. 24, no. R1, R102–R110, 2015.

[4] J. French and S. Edwards, "The role of noncoding variants in heritable disease," *Trends in Genetics*, vol. 36, no. 11, pp. 880–891, 2020.

[5] H. Chial, "Mendelian genetics: Patterns of inheritance and single-gene disorders," *Nature Education*, vol. 1, no. 1, p. 63, 2008.

[6] S. Pagni, J. D. Mills, A. Frankish, J. M. Mudge, and S. M. Sisodiya, "Non-coding regulatory elements: Potential roles in disease and the case of epilepsy," *Neuropathology and Applied Neurobiology*, vol. 48, no. 3, e12775, 2022.

[7] A. Kapoor *et al.*, "An enhancer polymorphism at the cardiomyocyte intercalated disc protein nos1ap locus is a major regulator of the qt interval," *The American Journal of Human Genetics*, vol. 94, no. 6, pp. 854–869, 2014.

[8] E. Khurana, Y. Fu, D. Chakravarty, F. Demichelis, M. A. Rubin, and M. Gerstein, "Role of non-coding sequence variants in cancer," *Nature Reviews Genetics*, vol. 17, no. 2, pp. 93–108, 2016.

[9] J. Tian *et al.*, "Systematic functional interrogation of genes in gwas loci identified atf1 as a key driver in colorectal cancer modulated by a promoter-enhancer interaction," *The American Journal of Human Genetics*, vol. 105, no. 1, pp. 29–47, 2019.

[10] S. E. Bojesen *et al.*, "Multiple independent variants at the tert locus are associated with telomere length and risks of breast and ovarian cancer," *Nature genetics*, vol. 45, no. 4, pp. 371–384, 2013.

[11] K. Michailidou *et al.*, "Association analysis identifies 65 new breast cancer risk loci," *Nature*, vol. 551, no. 7678, pp. 92–94, 2017.

[12] C. S. Pareek, R. Smoczynski, and A. Tretyn, "Sequencing technologies and genome sequencing," *Journal of applied genetics*, vol. 52, pp. 413–435, 2011.

[13] S. K. Burley, H. M. Berman, G. J. Kleywegt, J. L. Markley, H. Nakamura, and S. Velankar, "Protein data bank (pdb): The single global macromolecular structure archive," *Protein crystallography: methods and protocols*, pp. 627–641, 2017.

[14] N. M. Luscombe, D. Greenbaum, and M. Gerstein, "What is bioinformatics? a proposed definition and overview of the field," *Methods of information in medicine*, vol. 40, no. 04, pp. 346–358, 2001.

[15] C. Caudai *et al.*, "Ai applications in functional genomics," *Computational and Structural Biotechnology Journal*, vol. 19, pp. 5762–5790, 2021.

[16] J. Zhou and O. G. Troyanskaya, "Predicting effects of noncoding variants with deep learning–based sequence model," *Nature methods*, vol. 12, no. 10, pp. 931–934, 2015.

[17] D. R. Kelley, J. Snoek, and J. L. Rinn, "Basset: Learning the regulatory code of the accessible genome with deep convolutional neural networks," *Genome research*, vol. 26, no. 7, pp. 990–999, 2016.

[18] W. Ma *et al.*, "Deepsata: A deep learning-based sequence analyzer incorporating the transcription factor binding affinity to dissect the effects of non-coding genetic variants," *International Journal of Molecular Sciences*, vol. 24, no. 15, p. 12 023, 2023.

[19] B. Alberts *et al.*, *Essential cell biology*. Garland Science, 2015.

[20] P. F. Chinnery and E. A. Schon, "Mitochondria," *Journal of Neurology, Neurosurgery & Psychiatry*, vol. 74, no. 9, pp. 1188–1199, 2003.

[21] H. M. McBride, M. Neuspiel, and S. Wasiak, "Mitochondria: More than just a powerhouse," *Current biology*, vol. 16, no. 14, R551–R560, 2006.

[22] G. K. Voeltz, M. M. Rolls, and T. A. Rapoport, "Structural organization of the endoplasmic reticulum," *EMBO reports*, vol. 3, no. 10, pp. 944–950, 2002.

[23] A. Ballabio, "The awesome lysosome," *EMBO molecular medicine*, vol. 8, no. 2, pp. 73–76, 2016.

[24] C. Yang and X. Wang, "Lysosome biogenesis: Regulation and functions," *The Journal of cell biology*, vol. 220, no. 6, 2021.

[25] E. C. Dell'Angelica, C. Mullins, S. Caplan, and J. S. Bonifacino, "Lysosome-related organelles," *The FASEB Journal*, vol. 14, no. 10, pp. 1265–1278, 2000.

[26] M. Islinger, S. Grille, H. D. Fahimi, and M. Schrader, "The peroxisome: An update on mysteries," *Histochemistry and cell biology*, vol. 137, pp. 547–574, 2012.

[27] National Human Genome Research Institute, *Deoxyribonucleic acid (dna) image*, `https://www.genome.gov/genetics-glossary/Deoxyribonucleic-Acid`, 2024.

[28] C. Fonseca Guerra, F. M. Bickelhaupt, J. G. Snijders, and E. J. Baerends, "Hydrogen bonding in dna base pairs: Reconciliation of theory and experiment," *Journal of the American Chemical Society*, vol. 122, no. 17, pp. 4117–4128, 2000.

[29] A. Jansen and K. J. Verstrepen, “Nucleosome positioning in saccharomyces cerevisiae,” *Microbiology and molecular biology reviews*, vol. 75, no. 2, pp. 301–320, 2011.

[30] G. Zheng, *The packaging of DNA in chromatin*. Rutgers The State University of New Jersey, School of Graduate Studies, 2010.

[31] National Human Genome Research Institute, *Chromosome image*, `https://www.genome.gov/genetics-glossary/Chromosome`, 2024.

[32] M. B. Gerstein *et al.*, “What is a gene, post-encode? history and updated definition,” *Genome research*, vol. 17, no. 6, pp. 669–681, 2007.

[33] R. J. White, *Gene transcription: mechanisms and control*. John Wiley & Sons, 2009.

[34] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter, “From dna to rna,” in *Molecular Biology of the Cell. 4th edition*, Garland Science, 2002.

[35] P. Cramer, “Organization and regulation of gene transcription,” *Nature*, vol. 573, no. 7772, pp. 45–54, 2019.

[36] National Human Genome Research Institute, *Transcription image*, `https://www.genome.gov/genetics-glossary/Transcription`, 2024.

[37] A. Philips and T. Cooper*, “Rna processing and human disease,” *Cellular and Molecular Life Sciences CMLS*, vol. 57, pp. 235–249, 2000.

[38] S. Hocine, R. H. Singer, and D. Grünwald, “Rna processing and export,” *Cold Spring Harbor perspectives in biology*, vol. 2, no. 12, a000752, 2010.

[39] M. Livingstone, E. Atas, A. Meller, and N. Sonenberg, “Mechanisms governing the control of mrna translation,” *Physical biology*, vol. 7, no. 2, p. 021 001, 2010.

[40] V. Ramakrishnan, “Ribosome structure and the mechanism of translation,” *Cell*, vol. 108, no. 4, pp. 557–572, 2002.

[41] J. Lemonnier, N. Lemonnier, S. Pascolo, and C. Pichon, “The marathon of the messenger,”

[42] National Human Genome Research Institute, *Translation image*, `https://www.genome.gov/genetics-glossary/Translation`, 2024.

[43] G. E. Schulz and R. H. Schirmer, *Principles of protein structure*. Springer Science & Business Media, 2013.

[44] R. M. Bavle, “Mitosis at a glance,” *Journal of Oral and Maxillofacial Pathology*, vol. 18, no. Suppl 1, S2–S5, 2014.

[45] C. E. Walczak, S. Cai, and A. Khodjakov, “Mechanisms of chromosome behaviour during mitosis,” *Nature reviews Molecular cell biology*, vol. 11, no. 2, pp. 91–102, 2010.

[46] X. Li, F. Yang, and B. Rubinsky, “A theoretical study on the biophysical mechanisms by which tumor treating fields affect tumor cells during mitosis,” *IEEE Transactions on Biomedical Engineering*, vol. 67, no. 9, pp. 2594–2602, 2020.

[47] M. Sullivan and D. O. Morgan, “Finishing mitosis, one step at a time,” *Nature reviews Molecular cell biology*, vol. 8, no. 11, pp. 894–903, 2007.

[48] National Human Genome Research Institute, *Mitosis image*, `https://www.genome.gov/genetics-glossary/Mitosis`, 2024.

[49] R. A. Laskey, M. P. Fairman, and J. J. Blow, “S phase of the cell cycle,” *Science*, vol. 246, no. 4930, pp. 609–614, 1989.

[50] S. P. Bell and A. Dutta, “Dna replication in eukaryotic cells,” *Annual review of biochemistry*, vol. 71, no. 1, pp. 333–374, 2002.

[51] A. Dutta and S. P. Bell, “Initiation of dna replication in eukaryotic cells,” *Annual review of cell and developmental biology*, vol. 13, no. 1, pp. 293–332, 1997.

[52] “Chapter 42 - s phase and dna replication,” in *Cell Biology (Third Edition)*, T. D. Pollard, W. C. Earnshaw, J. Lippincott-Schwartz, and G. T. Johnson, Eds., Third Edition, Elsevier, 2017, pp. 727–741, ISBN: 978-0-323-34126-4. DOI: `https://doi.org/10.1016/B978-0-323-34126-4.00042-6`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/B9780323341264000426`.

[53] National Human Genome Research Institute, *Dna replication image*, `https://www.genome.gov/genetics-glossary/DNA-Replication`, 2024.

[54] P. M. Visscher, M. A. Brown, M. I. McCarthy, and J. Yang, “Five years of gwas discovery,” *The American Journal of Human Genetics*, vol. 90, no. 1, pp. 7–24, 2012.

[55] M. Z. Ludwig, “Functional evolution of noncoding dna,” *Current opinion in genetics & development*, vol. 12, no. 6, pp. 634–639, 2002.

[56] V. B. Kaiser and C. A. Semple, “When tads go bad: Chromatin structure and nuclear organisation in human disease,” *F1000Research*, vol. 6, 2017.

[57] M. Schipper and D. Posthuma, “Demystifying non-coding gwas variants: An overview of computational tools and methods,” *Human molecular genetics*, vol. 31, no. R1, R73–R83, 2022.

[58] E. G. Peña-Martínez and J. A. Rodríguez-Martínez, “Decoding non-coding variants: Recent approaches to studying their role in gene regulation and human diseases,” *Frontiers in bioscience (Scholar edition)*, vol. 16, no. 1, p. 4, 2024.

[59] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[60] A. M. Turing, *Computing machinery and intelligence*. Springer, 2009.

[61] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain.,” *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[62] M. Minsky and S. A. Papert, *Perceptrons, reissue of the 1988 expanded edition with a new foreword by Léon Bottou: an introduction to computational geometry*. MIT press, 2017.

[63] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[64] M. Flasiński, *Introduction to artificial intelligence*. Springer, 2016.

[65] N. Muthukrishnan, F. Maleki, K. Ovens, C. Reinhold, B. Forghani, R. Forghani, *et al.*, “Brief history of artificial intelligence,” *Neuroimaging Clinics of North America*, vol. 30, no. 4, pp. 393–399, 2020.

[66] M. A. Nielsen, *Neural networks and deep learning*. Determination press San Francisco, CA, USA, 2015, vol. 25.

[67] J. Lu, “Gradient descent, stochastic optimization, and other tales,” *arXiv preprint arXiv:2205.00832*, 2022.

[68] M. Andrychowicz *et al.*, “Learning to learn by gradient descent by gradient descent,” *Advances in neural information processing systems*, vol. 29, 2016.

[69] I. Neutelings, *Neural netowrk with coefficients, shifted*, Licensed under Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0)., 2022. [Online]. Available: `https://tikz.net/neural_networks/`.

[70] R. Rojas and R. Rojas, “The backpropagation algorithm,” *Neural networks: a systematic introduction*, pp. 149–182, 1996.

[71] S. Narkhede, “Understanding auc-roc curve,” *Towards data science*, vol. 26, no. 1, pp. 220–227, 2018.

[72] J. Huang and C. X. Ling, “Using auc and accuracy in evaluating learning algorithms,” *IEEE Transactions on knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[73] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[74] C. C. Aggarwal *et al.*, *Neural networks and deep learning*. Springer, 2018, vol. 10.

[75] J. Riebesell, *Convolution operator*, Licensed under Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0)., 2022. [Online]. Available: `https://tikz.net/conv2d/`.

[76] J. Wu, “Introduction to convolutional neural networks,” *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[77] K. O’Shea, “An introduction to convolutional neural networks,” *arXiv preprint arXiv:1511.08458*, 2015.

[78] A. S. Shamsaldin, P. Fattah, T. A. Rashid, and N. K. Al-Salihi, “A study of the applications of convolutional neural networks,” *J. Sci. Eng*, vol. 3, pp. 31–39, 2019.

[79] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, “A survey of convolutional neural networks: Analysis, applications, and prospects,” *IEEE transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999–7019, 2021.

[80] D. Bhatt *et al.*, "Cnn variants for computer vision: History, architecture, application, challenges and future scope," *Electronics*, vol. 10, no. 20, p. 2470, 2021.

[81] S. Min, B. Lee, and S. Yoon, "Deep learning in bioinformatics," *Briefings in bioinformatics*, vol. 18, no. 5, pp. 851–869, 2017.

[82] *Encode project*, `https://www.encodeproject.org`.

[83] *Roadmap epigenomics*, `http://www.roadmapepigenomics.org`.

[84] *Jaspar*, `https://jaspar.elixir.no`.

[85] C. E. Grant, T. L. Bailey, and W. S. Noble, "Fimo: Scanning for occurrences of a given motif," *Bioinformatics*, vol. 27, no. 7, pp. 1017–1018, 2011.

[86] *Ncbi sequence read archive*, `https://www.ncbi.nlm.nih.gov/sra`.

[87] *Ncbi gene expression omnibus*, `https://www.ncbi.nlm.nih.gov/geo/`.

[88] *Uc davis — farm clusters*, `https://farm.cse.ucdavis.edu/~ckern/Nature_Communications_2020/Peak_Calls/`.

[89] J. Wu, *Deepsea in tensorflow*, `https://github.com/jimmyyhwu/deepsea`.

Acknowledgments

With these lines, I would like to express my gratitude to my supervisor, Professor Pizzi, for her availability in assisting me during the writing of my thesis and for her prompt and comprehensive responses.

I would also like to thank the University of Padua for offering me the opportunity to participate in the Erasmus+ program, an experience that has significantly enriched my academic and personal journey.

Finally, I would like to express my gratitude to my family and to all the people who have supported and accompanied me on this extraordinary journey.