Politecnico di torino

Master degree course in Biomedical Engineering

Master Degree Thesis in

Convolutional Networks for predicting antimicrobial resistance

Politecnico di Torino

Supervisors
Squillero Giovanni
Tonda Alberto Paolo
Barbiero Pietro
Ferrero Giulio

Candidate
Alessandri' Simone
ID: 267642

Academic year 2020/21

Chapter 1

Introduction

Antimicrobial resistance occurs when a microbe evolves in order to resist antimicrobials such as antibiotics, which are agents that kill or stop bacteria's growth. However, nowadays they are losing their efficacy becoming a global public health threat:

"Infections by multidrug-resistant bacteria are estimated to cause 25,000 deaths in the EU every year. Antimicrobial resistance also places a tremendous burden on healthcare systems and society, with an annual cost due to healthcare expenditures and productivity losses estimated at approximately €1.5 billion in the EU." [10]

The antimicrobial resistance occurs naturally when bacteria is exposed to an antibiotic, through different mechanisms:

1. **Intrinsic resistance**: Some bacteria possess innate resistance against some antibacterial action thanks to the structure of the cell envelope
2. **Mutation**: Bacteria are subjected to DNA mutations, this is fundamental for their evolution because allows them to adapt to the environment.
3. **Selective pressure**: Only the new resistant microbes will survive and replicate.
4. **Gene transfer**: Microbes can take genes from other microbes using horizontal gene transfer mechanisms(HGT).

However, human's behaviour is the cause of the emergency: behaviors such as the massive overuse of antibiotics, overprescription, drug cycles interruption, and also the usage of antibiotics in farms animals are significant boosters of antimicrobial resistance. The traditional method for assaying AMR is the Antimicrobial Susceptibility Testing (AST). It consists in placing disks of drugs in a Petri dish and then spreading in it an agar plate inoculated with the isolated pathogen. After the incubation, the disk is inspected to determine which drugs either killed or prevent the growth of the microbe. AST is widely used in hospital clinical microbiology laboratories, but it has many limitations such as:

1. It has low throughput
2. It is an in vitro technique, therefore the behavior of bacteria in vivo might be different.
3. It needs of microbiology facilities and trained clinical microbiology personal
4. It is viable only on cultivable bacteria.

To overcome these disadvantages, many alternatives are emerging every year and most of them are sequencing-based. The popularity of these techniques is due to the improvements of the sequencing technology and the consequential decrease of the cost that leads to an higher number of bacterial sequence data. In short, sequencing technology returns fragments of DNA, these fragments can be processed using a de novo assembly, which recreates the genome without using a reference, using an alignment method, which uses a reference to recreate the genome or they can be used without alignment (alignment-free method). One innovative approach to the AMR problem is Deep learning, many studies report great results in using it for different purposes, including DNA or RNA classifications. Machine learning is a technology that is transforming every aspect of our life, it is used in many sectors such as finance, health care, smart cities, and also for analyzing biological data such as DNA. In this thesis I adopted a Deep learning algorithms, a subset of machine learning, with the ability to extract features automatically. Indeed the feature extraction and selection step are essential in a machine learning algorithm and thanks to the introduction of deep learning, it is possible to automate the process of feature engineering. The elementary unit of a machine

learning model is the artificial neurons, they are defined by a mathematical function designed by inspiring biological neurons which decide the output value based on the activation function, the weight, and the input values. Weight and bias are the trainable part of a neural network and are updated using a back propagation algorithm. A machine learning model consist of different neurons, organized in input, hidden, and output layers, depending on these, various architecture were created. There are the principal components of a neural network model:

1. **Parameters**: these are the coefficients of the model and they change across the epoch based on the loss or cost function.

2. **Hyperparameters**: contrary to parameters, these are set before the training step and don't change across the learning step.

3. **Loss function**: the loss function compares the predicted output with the correct value and returns a number that is used to update machine learning parameters.

The highlights of deep learning are the scalability of neural networks, which means that the results get better with more data and using a large model and the automatic feature extraction from data, also called feature learning. Deep learning models make use of several algorithms, no one network is considered perfect but some algorithms are better suited to perform specific tasks. My thesis aims to test if a convolutional neural network, a deep learning algorithm, is able to predict whether bacteria are sensible or resistant to a specific antibiotic, by training it with DNA samples and data obtained from AST analysis. The main aspect featuring a CNN are convolution and pooling, the convolution layer performs a dot product between two matrices, where one matrix is the kernel, also called filter, that is the set of learnable parameters and the other matrix is a portion of the input data. Pooling is an operation generally performed after the convolutional layer, it is done separately upon each feature map and creates a new set of the same dimensions of pooled feature maps. The main advantages of using a CNN are:

1. CNN learns the filters automatically without mentioning it before, this helps in extracting the relevant features from the input data.

2. It can learn to recognize patterns across space.

3. It is able to extract local and position-invariant features, this is an advantage if data are random-ordered.

To implement machine learning you need a programming language, there are plenty of them and a lot of aspects to consider, for example, it must have many good deep learning libraries, but should also feature good runtime performance, a large community of programmers, and a healthy ecosystem of supporting packages. C, Java, Matlab, etc. are all good alternatives, each one with advantages and drawbacks, more interesting is Python, a high-level object-oriented programming language, that provides an easy experience for ML. Thanks to this it is possible to focus only on the machine learning task, rather than the specifics of the language. After creating the script, I performed some pre-processing on data:

- **Data conversion**: Considering that input data of a neural network must be numerical, this means that categorical data cannot be stored without changes. There are different ways to convert categorical variables to numerical values, such as ordinal encoding and one-hot-encoding.

- **Dataset division**: a first approach might be to use the entire dataset to train the neural network, because the more the sample, the more accuracy, but it is not a correct choice because without division it is impossible to know if the NN is encounter overfitting and underfitting problems. To overcome this it is essential to divide the dataset into two parts, one used to train and the other to test. After training evaluation metrics are fundamental to determine the performance of the machine learning model when used on a dataset different from the trained one, obviously, the model you have trained will perform better on a training dataset, but if it doesn't work on other data, it is useless.

To understand the validity of a model, K-fold cross-validation is widely used, it consists in the splitting of original training data set into K equal subsets, and in training k time, using each time one set as training, and the others as a test. For each one I evaluated the performance in different moments:

- **During training** evaluating the loss function is the primary parameter to understand if the neural network is learning.

- **At the end of each epoch** the accuracy calculated over the validation set is essential to notice if the model is overfitting dataset.

- **At the end of the training**, on the test set, evaluating the accuracy, the average loss (precision), and other parameter derived from confusion matrix.

And at the end of the k-test, I evaluate the average of all the scores calculated.

Contents

List of Figures

List of Tables

Chapter 2

Background Information

2.1 DNA and genomic

DNA is a molecule that contains the instructions which allow living beings to develop, live and reproduce. It is contained in each cell of the organism and is housed in the nucleus of every cell (in humans and bacteria, DNA is wound up and compacted into chromosomes) and is transferred from the parent cells to children cells. Structurally it is composed of two complementary strands of nucleotides organized in double helix structures (Figure 2.1. DNA is essential in all living organisms, indeed it coordinates its replication and the making of other molecules (proteins). For these reasons, changes in DNA produce variations in the characteristics of a species, even a slight change can have enormous positive or negative consequences.

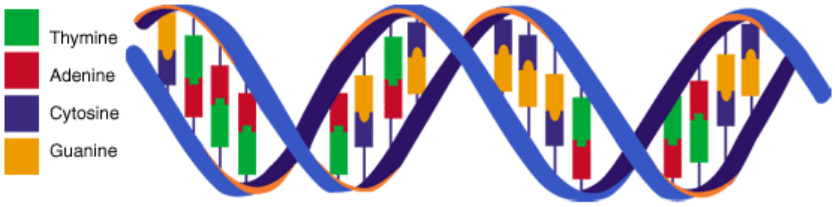


Figure 2.1: DNA double helix representation

2.1.1 How does DNA work

The individual molecules that make up DNA are called nucleotides. There are only four nucleotides (A = adenine, T = thymine, C = cytosine, and G = guanine) that define the DNA alphabet. The central dogma of life can be defined in a fairly simple way: DNA makes RNA, which in turn makes proteins, in the transcription phase DNA is converted into RNA, then it is translated into proteins. The amino acids that make up proteins are encoded by a nucleotide triplet codon: for example, the protein serine is encoded by the codons UCU, UCC, UCA, and UCG. In DNA or RNA, this relationship between amino acids and codons is described in the genetic code.

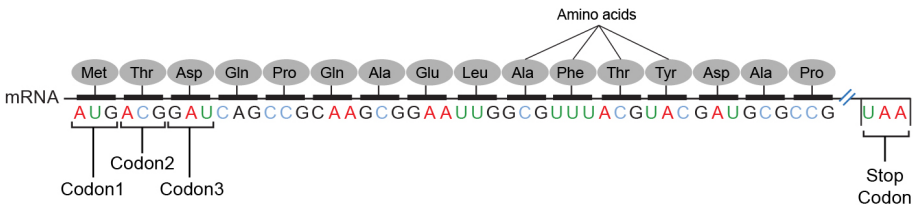


Figure 2.2: Representation of mRNA conversion to protein

Coding DNA

Coding DNA is the section of DNA that encodes a protein, surprisingly it accounts for 1% of the human genome. Coding DNA sequences are separated by long regions of DNA called introns that have no apparent function. These region is eliminated before the translation in RNA.

Non Coding DNA

Noncoding DNA instead accounts for 99% and it does not provide instructions for the synthesis of protein. Non coding DNA generally acts as regulatory elements, determining when and where genes are turned on and off.

2.1.2 DNA mutations

DNA mutations occur when there are changes in the nucleotide sequence. These changes can be caused by various factors such as mistakes in DNA replication and environmental influences. In figure 2.3 there are illustrated the principal mutation mechanism. The majority of mutations have neither

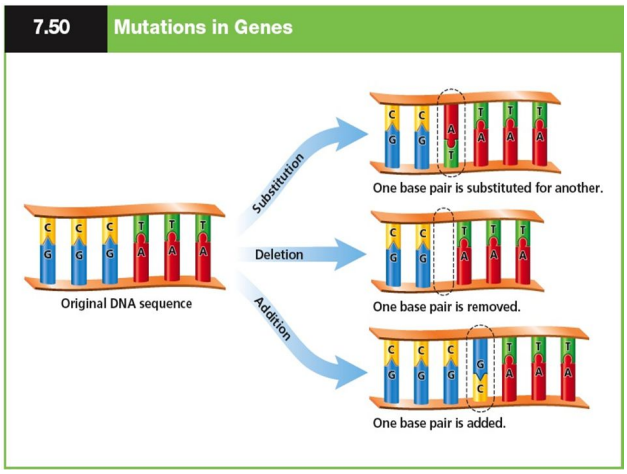


Figure 2.3: Principal mechanism of DNA mutations

negative nor positive effects on the organism in which they occur, these are called neutral mutations. Some other have a positive effect on the organism and are called beneficial mutations because they lead to new versions of proteins that help organisms adapt to changes in their environment. Anyway any random change in a gene's DNA might potentially results in a protein that does not function normally or may not function at all, these mutations are likely to be harmful causing genetic disorders or cancer.

2.1.3 Common file format for DNA storage

As we said before, DNA is composed of four nucleotides, therefore to represent DNA it might be sufficient to store the ordered sequence. The problem is that often we need to know other important elements, such as the quality of the sequencing, depending on the sequencing technique and on our uses. The most common file format that represents nucleobase (but also amino acid sequence) are FASTA and FASTQ, both text based.

FASTA

It is the simplest format. The first line of the file can be started with both "¿" or ";" and they are automatically ignored by the software during the reading, in fact, the first line is used to briefly describe the sequence. Following there are the sequences that have a description line, that begins with "¿" and is followed by the sequences. In Figure 2.4 there is an example.

Figure 2.4: FASTA format example

FASTQ

This is the most widely used in sequence analysis, the format is similar to FASTA and it was developed to overcome the problem of confidence during sequencing, in fact, it indicates the quality scores for each sequence.In Figure 2.5 there is an example. A FASTQ record has the following format:

1. A line starting with @, containing the sequence ID.
2. One or more lines that contain the sequence.
3. A new line starting with +, that can be either empty or have the sequence ID.
4. One or more lines that contain the quality scores.

2.1.4 Genome sequencing

Before introducing Genome sequencing techniques it is important to understand what are Genes and genomes. Genes are a sequence of nucleotides in DNA that encodes the synthesis of a gene product such as a protein. They are also the basic unit that is inherited. Instead, when talking about the

Figure 2.5: FASTQ format example

genome we refer to the total DNA content in a single cell in an organism. At this point, we can define genome sequencing as the process of determining the order of nucleotides bases of an organism. Several techniques were developed in the last 30 years, evolved in different technologies generation. Before talking about the technology, there are some important considerations about the difficulty of sequencing :

1. If DNA is degraded no one techniques can make accurate sequencing.
2. Bacterial genome content varies greatly, even between closely related species. Therefore each sample can differ in genomic contents.
3. It could be impossible to sequence all the genome there are parts of the genome that are impossible to sequence.
4. Low purity is also a problem: producing reliable results in sequencing requires samples free of proteins, organic solvents, etc.

These are some of the problems affecting bacterial sequencing, and it is fundamental to take in mind that there will always be some degree of error in a sequence. Let's briefly see the most common methods of DNA sequencing.

First Generation Sequencing

These were the earliest technologies developed, they are based on the chain termination method or the chemical method.

Maxam-Gilbert-Sequencing This method, also called the chemical cleavage method or chemical degradation method, is based on the chemical modification of DNA to cleavage DNA at specific bases. The result of the treatment

is to create a series of fragments, each one radiolabelled at one end. After this procedure, using gel electrophoresis it is possible to separate it by size and recreate the sequence by the visualization of the fragments.

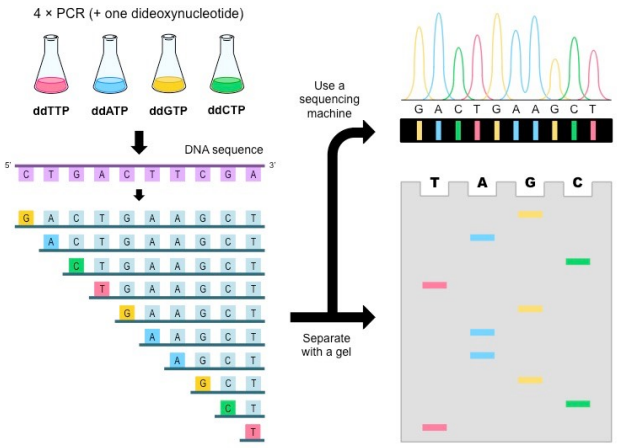


Figure 2.6: Representation of Maxam Gilbert sequencing technique [4]

Sanger Sequencing Sanger sequencing is known as the chain termination method or dideoxynucleotide method or also the sequencing by synthesis method. The idea is to use one strand of DNA as a template for sequencing using nucleotides called dNTPS, which are nucleotides chemically modified to obtain a mark for each DNA base and to remove the 3' hydroxyl groups, essential for the extension of the nucleotides. In this way, they obtain different-sized DNA fragments ending with a marked dNTP. After that, the fragments are separated by size, and by visualizing DNA fragments it is possible to reconstruct the sequence. This technique was used for three decades and nowadays it is used for single or low-throughput DNA sequencing.

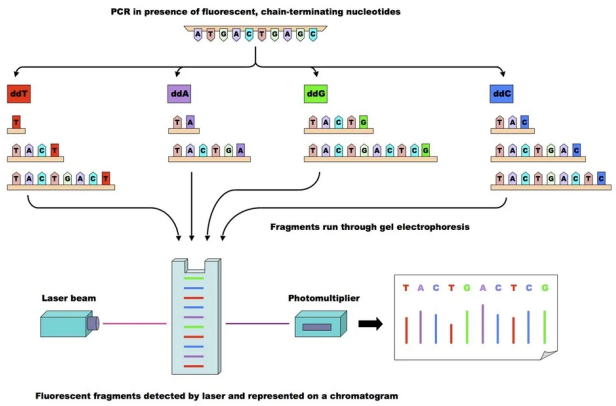


Figure 2.7: Representation of Sanger sequencing technique.[4]

Second generation of sequencing

First-generation technologies were dominant for three decades, but both cost and time were a problem. To overcome this there was developed a new technology called NGS (Next-generation sequencing) which is high throughput because they allow many fragments to be sequenced in parallel. Each technique of NGS is different, but they have all the following characteristics:

1. The parallel generation of millions of short reads
2. The low cost of sequencing
3. The improvement of sequencing speed compared to first-generation
4. They don't need to use electrophoresis

Illumina Sequencing Illumina sequencing progress consists of four steps.

- **Library preparation**: it is crucial to the success of NGS workflow. The aim of this step is to prepare DNA samples to be compatible with a sequencer. (Figure 2.8)
    1. Libraries are prepared by tagmentation of DNA into short segments.
    2. Specialized adapters are added to both ends of DNA, these adapters contain complementary sequences that allow DNA fragments to bind to the Illumina flow cell.

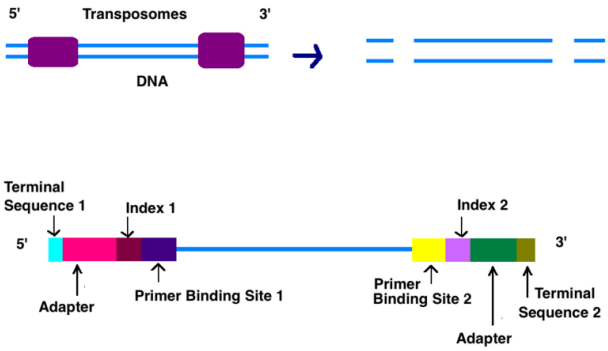


Figure 2.8: First step of Illumina sequencing[12]

- **Cluster Generation** (Figure 2.9)

  1. The prepared sequencing library is denatured and loaded into a flow cell. The flow cell is made up of glass coated with two types of oligonucleotides, one complementary to the 5' region and the other complementary to the 3' region.
  2. When single-stranded is bound, the complementary strand is generated by DNA polymerase.
  3. The strand folds over the oligonucleotide on the flow cell, then, polymerase synthesizes the double-stranded bridge. The denaturation of the bridge results in two DNA strands.
  4. Bridge amplification is repeated over and over to obtain simultaneously millions of clusters of all types of fragments in the sequencing library by clonal amplification.
  5. Then the reverse strands are washed away, retaining only the forward strands on the flow cell. In the forward strand, the 3 end is free and it is blocked in order to prevent unwanted priming.

- **Sequencing by synthesis** (Figure 2.10) The following step are performed for each fragment.

  1. Sequencing begins with the extension of the first sequencing primer. Illumina sequencing uses fluorescent dNTPs.

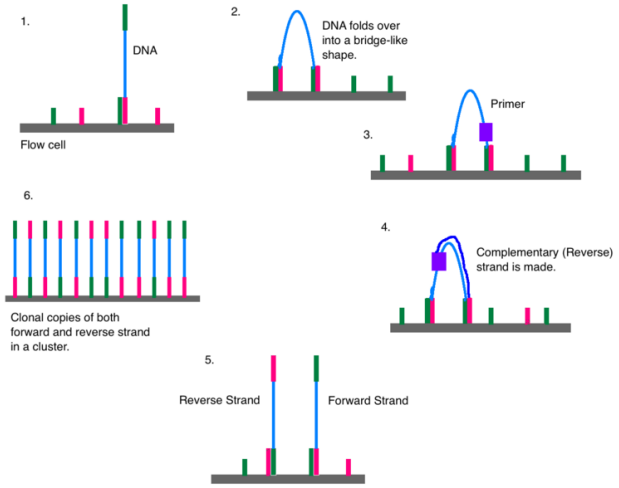


Figure 2.9: Second step of Illumina sequencing [12]

2. Next step consist in the addition of each complementary nucleotide to the fragment. When a dNTPs is added, a fluorescence is generated and captured.

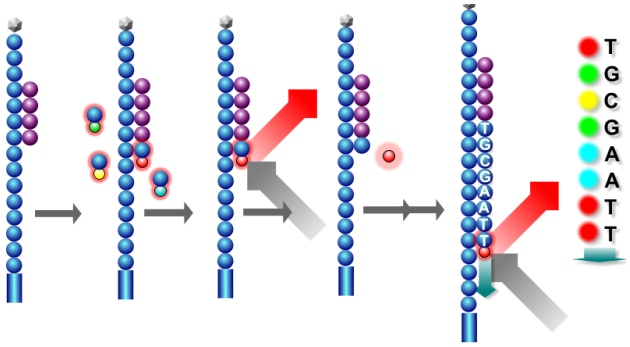


Figure 2.10: Third step of Illumina sequencing [12]

- **Data analysis** This is the last step of the process, Real-Time Analysis (RTA) software operates during cycles of sequencing chemistry and imaging, providing base calls and associated quality scores representing the primary structure of DNA or RNA strands. This built-in software performs primary data analysis on Illumina sequencing systems automatically. Short reads, generated by this technology, can either be pro-

cessed using assembly-based methods, whereby sequencing reads are first assembled into contiguous fragments (called contigs) or directly analysed.

Third generation of sequencing

The second generation has revolutionized the sequencing of DNA, but they require PCR amplification which is long and expensive procedure and also, it has a problem with sequence genomes with many repetitive areas. The main advantages provided by these techniques are:

1. Third generation is faster since it doesn't need PCR
2. It is more portable, in fact, the machine size is similar to a USB flash drive and it can be used readily by connecting a laptop
3. It produces much longer reads, solving numerous computational challenges.

2.2 AMR a global challenge accelerated by human's behaviour

Antimicrobial resistance (AMR) is a significant global challenge for food security, public health, and sustainable development. Microbes such as bacteria, viruses, and fungi are becoming resistant to drugs that previously were effective against them, this phenomenon occurs naturally when bacteria are exposed to antibiotics, but it is facilitated and accelerated by some conditions.

2.2.1 What is antimicrobial resistance and why it is a huge problem

Antimicrobial drugs such as antibiotics are medicines that fight microbial infection in both humans and animals, killing or make for it difficult to multiply. Antimicrobial resistance is a common phenomenon that happens when microbes develop the ability to survive to drugs designed to kill them, becoming no longer sensitive to it. Some infections such as tuberculosis, bacterial pneumonia, septicemia, gonorrhea, wound infections, otitis media, and many others, are now becoming hard to treat with some available antibiotics because the microbes are fast becoming resistant to some available antibiotic therapy. The fact that these microbes can't be killed with medicines makes it difficult and sometimes impossible to solve the infections: it takes longer to heal and it can get worse leading to more serious problems. In addition, the slower production of new antibiotics is complicating the situation, in fact, due to economic and regulatory obstacles, many pharmaceutical industries abandoned the antibiotic fields because antibiotic development nowadays is not considered a prudent economical investment.[5]. The emerging and steady increase in the occurrence of bacteria that are resistant to multiple antibiotics has become a global public health threat due to the lack of therapeutic options to treat certain infections in humans. Therefore, finding a solution is mandatory. The table in Figure 2.11, contains useful information which confirms the emergency of antibiotic-resistance phenomena. We can notice that microbes develop resistant genes faster on some antibiotics than others. The reasons are different, but we can simplify by saying that the most an antibiotic is used, the fastest the development of AMR is. Let us consider one example: the beginning of the antimicrobial era started with the identifi-

Antibiotic Approved or Released	Year Released	Resistant Germ Identified	Year Identified
Penicillin	1941	Penicillin-resistant *Staphylococcus aureus*	1942
		Penicillin-resistant *Streptococcus pneumoniae*	1967
		Penicillinase-producing *Neisseria gonorrhoeae*	1976
Vancomycin	1958	Plasmid-mediated vancomycin-resistant *Enterococcus faecium*	1988
		Vancomycin-resistant *Staphylococcus aureus*	2002
Amphotericin B	1959	Amphotericin B-resistant *Candida auris*	2016
Methicillin	1960	Methicillin-resistant *Staphylococcus aureus*	1960
Extended-spectrum cephalosporins	1980 (Cefotaxime)	Extended-spectrum beta-lactamase-producing *Escherichia coli*	1983
Azithromycin	1980	Azithromycin-resistant *Neisseria gonorrhoeae*	2011
Imipenem	1985	*Klebsiella pneumoniae* carbapenemase (KPC)-producing *Klebsiella pneumoniae*	1996
Ciprofloxacin	1987	Ciprofloxacin-resistant *Neisseria gonorrhoeae*	2007
Fluconazole	1990 (FDA approved)	Fluconazole-resistant *Candida*	1988
Caspofungin	2001	Caspofungin-resistant *Candida*	2004
Daptomycin	2003	Daptomycin-resistant methicillin-resistant *Staphylococcus aureus*	2004
Ceftazidime-avibactam	2015	Ceftazidime-avibactam-resistant KPC-producing *Klebsiella pneumoniae*	2015

Figure 2.11: Table containing years of resistant gene identification.[2]

cation of Penicillin which is the first compound used as an antibiotic. In 1943 Penicillin was massively produced because it was considered a miracle drug, therefore it was heavily used and in the same year, some bacteria developed resistance to it.

2.2.2 Causes of Antimicrobial resistance

Antibiotic resistance develops naturally in bacteria using different mechanism as an adaptive response to the environment (for example when they are exposed to antibiotics). In figure 2.12 is represented an example of how AMR might spread: a bacteria is consumed by the host that gets hill, in this contest is common that a provider will prescribe an antibiotic, in this way the non-resistant-bacteria get killed, leaving only the resistant fraction that can be spread globally. Let us see the resistant factors more in detail:

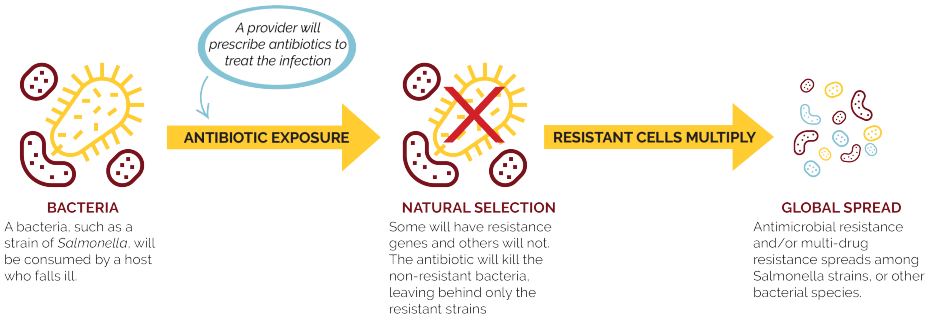


Figure 2.12: Common step of AMR development [1]

Genetic variations and selective pressure

We have previously seen what is DNA and stated that a mutation can potentially affect an organism's looks, behavior, and physiology. Bacteria are indeed subjected to DNA mutations and this is a fundamental part of their evolution because allows them to adapt to the environment to ensure their survival. Many mutations don't give any benefit or disadvantage for the survival, but some others can change drastically the situation, for example, a mutation can lead to AMR development. Let's consider a situation in which an antibiotic is assumed by a subject to heal from bacterial infection, in this contest bacteria that develop resistant genes with a mutation will survive and can reproduce. This phenomenon is called selective pressure and it is defined as any phenomenon that alters the survival or behavior of a living organism in the environment. In detail, antibiotic selective pressure is the impact of antibiotic usage over a population of bacteria, in which those who are resistant to the antibiotic will survive and replicate.

Acquired resistance from another organism

We refer to acquired resistance when a susceptible microorganism acquires ways of not being affected by the drug, indeed an insult, physical or chemical, potentially can induce changes into the microorganism. The most relevant mode of resistance emergence and spread in microbial populations is the horizontal gene transfer, which consists of the movement of genetic information among bacteria. HGT occurs by three genetic mechanism (Figure 2.13).

1. Transformation: bacteria take up DNA from environment

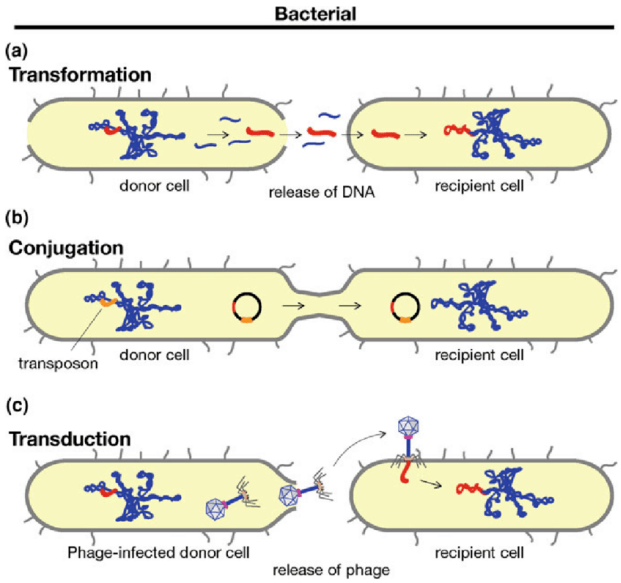


Figure 2.13: Main mechanism of HGT resistance

2. Conjugation: bacteria directly transfer genes to another cell

3. Transduction: Bacteriophages (Bacterial viruses) move genes from one cell to another.

Intrinsic resistance

Not all cases of AMR are caused by a mutation, we refer to intrinsic resistance when a bacterial species is naturally resistant to specific antibiotics, without being mutated or without gaining further genes. This type of resistance is conferred by traits that are common to an entire species of bacteria, for example having an outer membrane is an important feature for intrinsic resistance. In figure 2.14 there are represented the three principal mechanism of intrinsic resistance:

1. Reduced permeability to the outer membrane: for example gram-negative bacteria's outer membrane is very impermeable, getting hard for antibiotics to pass through.

2. Selective exclusion: which acts as pores through which only porins with a certain size or chemical properties can pass

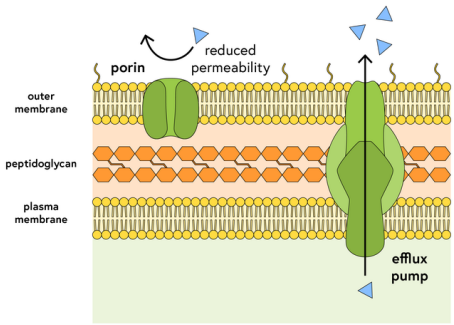


Figure 2.14: Main mechanism of intrinsic resistance

3. Efflux pumps: it is a mechanism by which antibiotics are pumped out of the cell.

Factors contributing to the AMR emergency

Beyond the natural mechanism of AMR developing, human's behavior is the main cause of the emergency. Since AMR depends on an high number of factors, it is necessary to consider the whole system: the hospital, animal production, agriculture, and the environment(Figure 2.15)

Human medicine Excessive prescription even in the absence of indications, uncertain diagnosis, self-medication are causes of the overuse of antibiotics. Also in many developing countries, this is because drugs can be purchased without a prescription. In addition, in hospitals, the massive and prolonged use of antibiotics or the presence of highly susceptible immunosuppressed patients (AIDS) also give an important impact on AMR.

Animals and agriculture Antibiotics are also used in food-producing animals and in agriculture, both for disease treatment and growth promotions.

The environment Since that most antibiotics are derived from microorganisms extracted from soil, which are intrinsically resistant to the antibiotic, the soil is considered a reservoir of antibiotic resistance genes. Also, water potentially contaminated with fecal microorganisms and organic fertilizers used on food crops may disseminate drug-resistant bacteria in the soil.

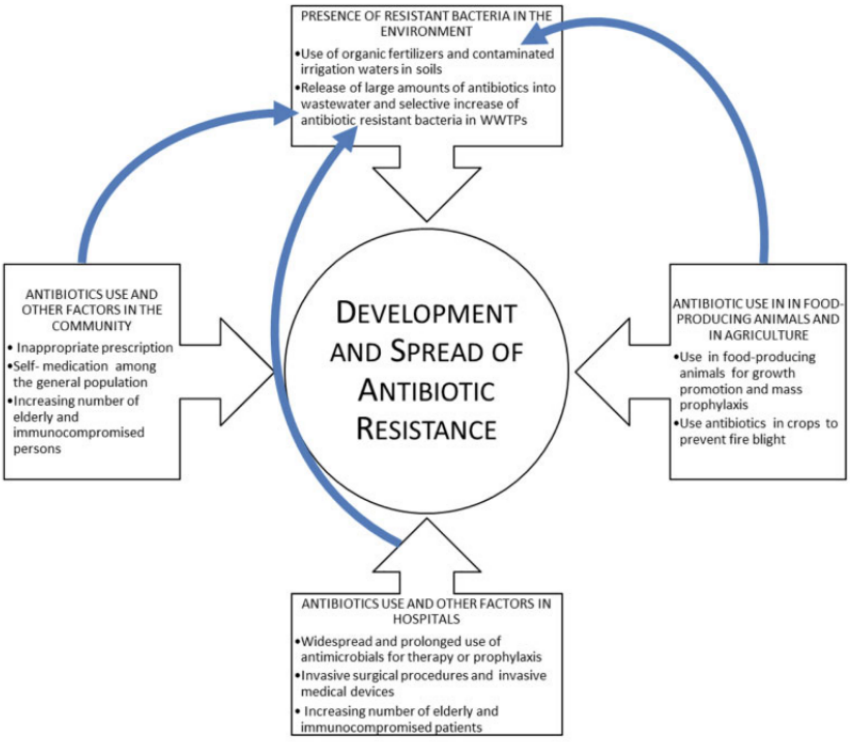


Figure 2.15: Factor involved in the spread of antibiotic resistance [20]

2.3 Common techniques for AMR study and analysis

Several methods were developed to overcome AMR problem, some of them are used to predict whether a sample is resistant or not, some others for studying the resistant mechanisms.

- Bacteria can acquire resistance to antibiotics by several mechanisms, for this reason, genotypic methods are more used when studying AMR mechanisms.
- AST techniques instead work regardless of the resistance mechanism and give answers to which antibiotic is efficient and which dose should be applied in the therapy, without giving any information about the mechanisms.

- Artificial intelligence can be considered a technique focused on prediction, but potentially it might be possible to deduce some aspect about the mechanisms.

In this chapter, I will introduce some techniques for studying or analyzing AMR.

2.3.1 AST: traditional techniques

Antimicrobial susceptibility testing is an in vitro technique used to determine the resistance or the susceptibility of bacteria to antimicrobial agents such as antibiotics. One of the main purposes of AST is to find the correct antibiotics to treat a particular infection or disease, in fact despite it is an in vitro technique, it helps in predicting the in vivo success or failure of drugs.

How does AST work

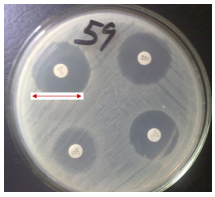
There are mainly three methods to determine antimicrobial susceptibility, the disk diffusion method, and the dilution method.

Broth dilution method It is one of the earliest AST methods: first, they prepare two-fold dilution of antibiotics in a liquid growth medium, dispensed in test tubes. Then the tubes is inoculated with a standardized bacterial suspension and incubated at 35 overnight. The lowest concentration of antibiotics able to prevent growth represented the MIC (minimal inhibitory concentration). The main drawbacks of this method are that is tedious, requires a manual task, and has a high possibility of error.

Antimicrobial gradient method It determines susceptibility based on gradient concentration in an agar medium. Thin plastic test strips are impregnated with antibiotics and radially placed on the surface of an agar plate inoculated with bacteria. Then it is incubated overnight.

The agar disk diffusion method It is one of the most common and most widely used. A disk is impregnated by an antibiotic and placed on the surface of an agar plate previously inoculated with bacteria. The antibiotics start diffuses radially through the agar generating a concentration gradient with an high concentration in the center of the plate. Then the zone without

(a)



(b)

Figure 2.16: AST gradient method (a) and agar disk diffusion method [20] (b)

bacteria is measured and compared with standards to determine whether the microbes is intermediate, resistant or susceptible to antimicrobial.

AST Disadvantages

1. These tests are performed in vitro condition (laboratory) and not in vivo (patient), therefore it cannot be inferred for example that an antimicrobial that kills in vitro will be working into the organism.

2. AST is viable only for cultivable bacteria, precluding studies on the emergence and spread of antimicrobial resistance in diverse and complex microbial communities with large fractions of currently uncultured bacteria [9].

3. Culture techniques can also fail in situations where multiple bacteria may cause symptomatic disease. In such cases, the cultured bacteria are assumed responsible, but several metagenomic studies from patient samples indicate that the bacteria detected in culture may not be responsible for disease symptoms [8]

4. Performing AST requires microbiology facilities and trained clinical microbiology personnel to obtain an accurate result, but not all health care centers have these resources[8]

5. It is low throughput [8]

6. It has low sensitivity for some bacteria such as M. tuberculosis [23].

7. These phenotypic methods have a lower resolution in examining resistance determinants and provide less information about resistance gene epidemiology than whole-genome sequencing.[8]

2.3.2 Genome sequencing-based analysis method

Nowadays several methods have been developed for detecting genetic determinants of AMR, thanks to the improvements of the sequencing technology and the consequential decrease of the cost that lead an increased number of bacterial sequence data. When studying bacteria, we differentiate between genomics and metagenomics: the difference depends on the nature of the sample, genomics explores the complete genetic information of a single organism only, whereas metagenomics explores a mixture of DNA from multiple organisms and entities, such as viruses, viroids, and free DNA. Whole-genome sequencing (WGS) of pathogens is important for epidemiological surveillance and infection control, furthermore, whole metagenome sequencing (WMS) permits culture-independent analysis of complex microbial communities, providing useful information about AMR genes occurrence. Therefore both technologies are used for tracking AMR genes and mobile genetic elements. Before analyzing sequencing data, it is important to preprocess it, depending on how data are pre-processed we divide it into two methods, assembly-based and read-based.

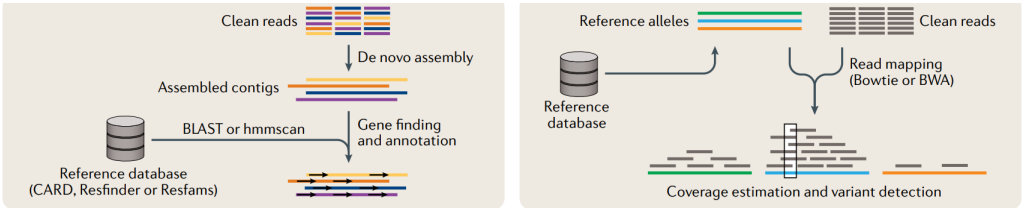


Figure 2.17: Comparison between assembly and read method [8]

Assembly based approach

Assembly is a computationally highly intense process that may take days or even weeks to produce the sequence of a more complex organism. Reconstructing a DNA sequence in the absence of a previously reconstructed reference sequence from a similar organism is called de novo assembly.

De novo assembly It consists in assembling short nucleotide sequences into longer ones without the use of reference genomes. When studying an organism without reference genomes, this approach is fundamental, because assembly-based methods can recreate whole genomes or large contigs that contain the complete genomic context. The process of de novo assembly and annotation (identifying the location of genes) is computationally expensive, time-consuming, and requires a higher coverage than reference-based assembly, which can be difficult to achieve for all samples, specifically when dealing with metagenomic samples with high microbial diversity and uneven taxonomic composition.

Read based approach

Antimicrobial resistance genes in a sample can be detected without performing de novo assembly, for example by aligning reads to the reference databases using pairwise alignment tools such as Bowtie2 or splitting reads into k-mers and mapping them to the reference databases. The read-based approach is generally fast and less computationally demanding because it bypasses de novo assembly. We can divide read based approach in alignment based and alignment free methods.

Alignment - based methods Alignment-based techniques are all based on the search for base-to-base matches in two or more sequences. These techniques perform a score based on the amount of mismatches and matches between the sequences from which they evaluate similarity. There are many

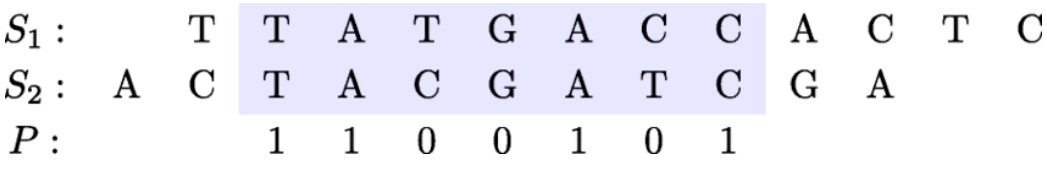


Figure 2.18: Example of alignment based method

tools developed, the most famous are BLAST, FASTA, and MUSCLE.

Alignment - free methods Alignment-free approaches to sequence comparison can be defined as any method of quantifying sequence similarity or dissimilarity that does not use or produce alignment at any step of algorithm.

These approaches can be broadly divided into two groups: methods based on the frequencies of subsequences of a defined length (word-based methods) and methods that evaluate the informational content between full-length sequences (information-theory based methods).

2.4 Artificial intelligence: an innovative technique

Artificial intelligence is a technology that is transforming every aspect of our life.

- Preventive diagnosis: machine learning is achieving good results in healthcare, in particular for heart diseases and cancer diagnosis.
- Autonomous car: in the very next years we will be relieved of the chore of driving, resulting in a significant reduction of traffic collisions and enhanced mobility for the elderly, children and disabled.
- Security: face recognition algorithms are widely used in video surveillance to find wanted criminals.
- Language translation: results provided by Google Translate are astonishing, in many languages it performs almost as well as a human translator.
- Virtual assistants: many people nowadays are used to speak to their smartphones to save appointment on the agenda or to check weather conditions.
- Fraud detection: machine learning is used in this field for analyzing mails, detecting fishing attempts and generally spam.
- Product recommendation: Amazon completely transformed online shopping by suggesting people items they may be interested in.

2.4.1 From AI to deep learning algorithm

Artificial intelligence (AI), machine learning (ML) and Deep learning are often used as interchangeably words, but they are not synonyms.(Figure 2.19) Artificial Intelligence is a field of computer science that creates intelligent systems able to simulate human intelligence, instead, machine learning is a subset of AI, that is focused on the creation of a neural network that can learn without being explicitly programmed by using data or experiences. Deep learning is in turn a subset of machine learning, with the ability to extract

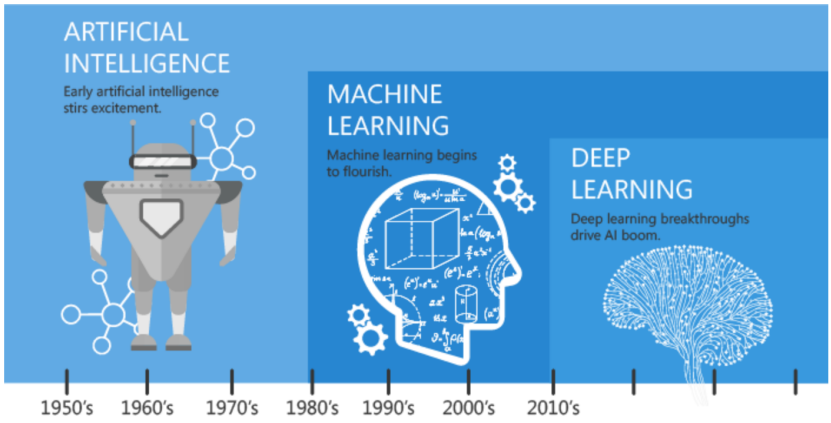


Figure 2.19: Artificial intelligence family

features automatically. Feature extraction and selection step are essential in a machine learning algorithm and thanks to the introduction of deep learning, it is possible to automate the process of feature engineering.(Figure: 2.20)

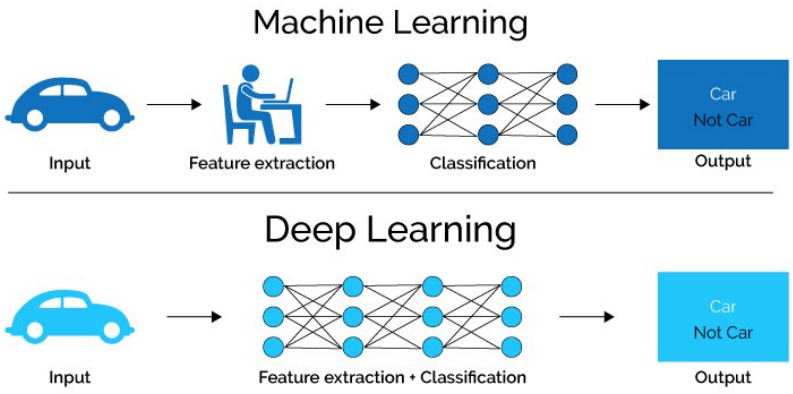


Figure 2.20: Comparison between machine deep learning.

2.4.2 Machine learning background

We can summarize and simplify the machine learning algorithm using the following equation:

$$Y = f(X)$$

This is an extreme oversimplification, but the basic concept to understand machine learning is that his main task is to estimate the target function (f(x)) to make a prediction (Y) for a given input(X).

Linear Regression

It is a linear approach that permits to model of the relation between two variables.

$$Y = a * X + b \tag{2.1}$$

Where X,Y are respectively the dependent and independent variable, instead a and b are parameters of the linear regression. Assuming that we have a

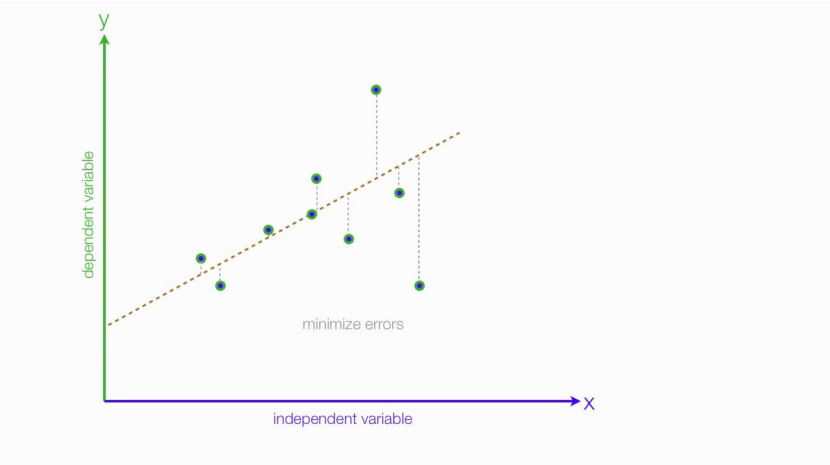


Figure 2.21: Example of Linear regression

dataset as the one in Figure 2.21, the objective of linear regression is to find a line that fits the data; for doing this, it can change a and b in such a way that minimizes the sum of error above each data.

Artificial Neurons

To understand the overall behavior of machine learning, it is important to understand what is an artificial neuron and how does it work. Artificial neurons are elementary units of a neural network and are defined by a mathematical function designed by inspiring to biological neurons. When neurons

receive input, they decide the output value based on the activation function and the input values(Figure 2.22):

$$y = f(\sum_i x_i \omega_i + b) \tag{2.2}$$

The function first compute the weighted sum of the inputs and weight (x_i

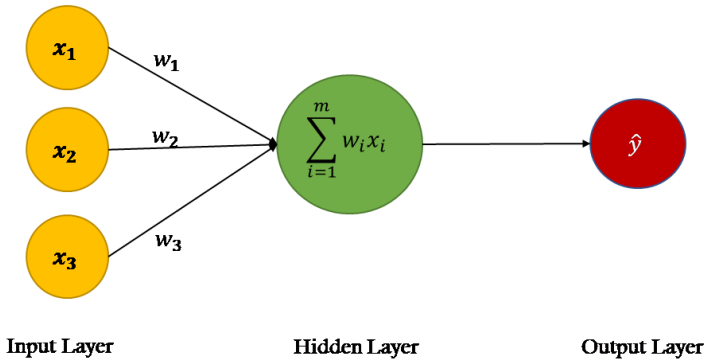


Figure 2.22: Simple model of a neuron in neural networks

and w_i), and then sum it with b, that is a special value know as bias. The results is passed as an input of the activation function (f), that is know also as transfer function, which decide the output of the neuron. Considering Equation:2.2 the equation that characterizes a neuron, if we consider just one input, the equation will be:

$$y = f(\omega_1 x_1 + b)$$

It is evident that, except for the activation function, neurons perform a linear regression (equation 2.1) for each input.

Activation function As just mentioned, the output of each neuron depends on different variables: weight, bias, and the activation function. The last one is an important parameter defined by the developer before training, and it is fundamental because if the activation function is not applied, the output signal becomes a simple linear function with limited learning power. Various types of activation functions perform this task in a different ways. In figure 2.23 are represented some basic activation function:

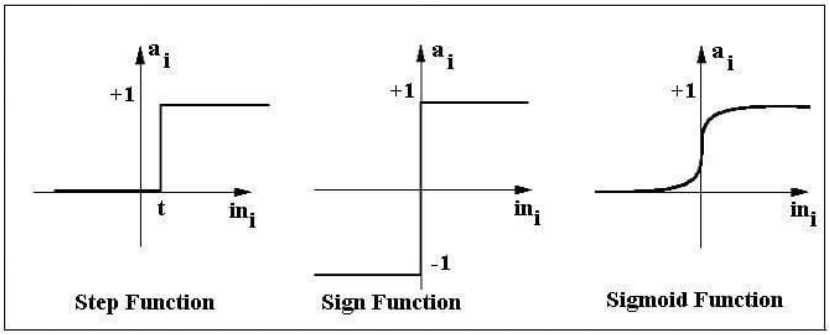


Figure 2.23: Basic activation function

- **Step function**: It gets triggered above a certain value of the neuron output, otherwise it outputs zero; the sign function outputs +1 or -1 depending if neuron output is greater than zero

- **Sigmoid**: It is a non linear activation function based on the following equation:

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

The sigmoid function is also called a squashing function as its domain is the set of all real numbers. The output value instead is always between 0 and 1. A common problem when using sigmoid as activation function, is the saturation: a neuron is said to be saturated if it reaches to its peak value either maximum or minimum:

$$if(f(x) = 0 | f(x) = 1) \Rightarrow f'(x) = f(x)(1 - f(x)) = 0$$

in this case, the derivative is 0, this means that the weight of the neuron will not update anymore. This is known as the vanishing gradient problem.

- **ReLU Function**: It is one of the most used and most simple, in fact, ReLu function returns 0 if it receives any negative input and if it is positive, it returns the same value as input.

$$f(x) = max(0, a) = max(0, \sum_{i=1}^{N} w_i x_i + b)$$

The derivative of ReLu activation function is given as

$$f(x) = \begin{cases} 0 & : if(x < 0) \\ x & : otherwise \end{cases}$$

ReLu is considered one of the best activation function for its simplicity and therefore his low computational weight but more than that, for the sparsity: a matrix in which most entries are 0 is called sparse matrix. Sparsity results in concise models that have better predictive power and less noise.

- **Leaky ReLU**: ReLU units can be fragile during training and can "die", which means that it always outputs the same value for any input. Once the gradient at 0 is also 0, gradient descent learning will not alter the weights and the condition remains.

$$f(x) = max(0.01a, a) = max(0.01a, \sum_{i=1}^{N} w_i x_i + b)$$

The derivative of Leaky ReLu activation function is given as

$$f(x) = \begin{cases} 0.01x & : if(x < 0) \\ x & : otherwise \end{cases}$$

The neurons do not die because 0.01x ensures that at least a small gradient will flow through. Although the change in weight will be small but after a few iterations it may come out from its original value. The slope coefficient is determined before training, it is not learned during training. This type of activation function is popular in tasks where it may suffer from sparse gradients

;

Back-propagation algorithm It is clear at this point that in order to calculate the output, for a given input, each neuron uses weight, bias, and input values. Weight and bias are the trainable part of a neural network, which changes across the learning steps. A common algorithm used to update weight in supervised learning is to use backpropagation, let us see briefly how does it work:

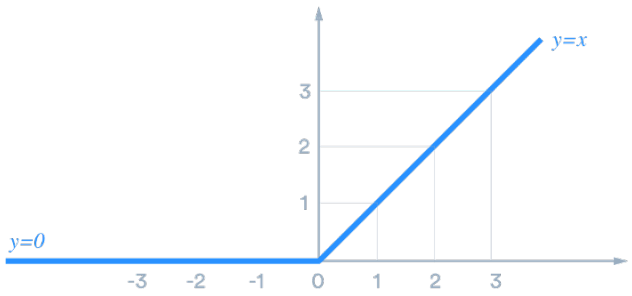


Figure 2.24: ReLu Function

1. Initialize weights of each neuron.
2. Calculate an output by propagating an input signal through the neural network.
3. Calculate the errors or cost function and its first derivatives.
4. Backpropagate through the network to determine the error derivatives.
5. Update the parameter estimates using the error derivative and the current value.

The algorithm used to backpropagate will be explained better in the next section.

2.4.3 Vanilla model: Single and multi-layer Perceptron

After understanding what a neuron is, let us consider a neural network in all its complexity, they are structures made of artificial neurons that can take in multiple-input in order to produce an output.

Single layer perceptron

It is the most simple architecture created. It was introduced by Frank Rosenblatt in 1957 as an algorithm for supervised learning in binary classifiers. Single-layer perceptron has only input and output layers, therefore it can learn only linearly separable patterns.

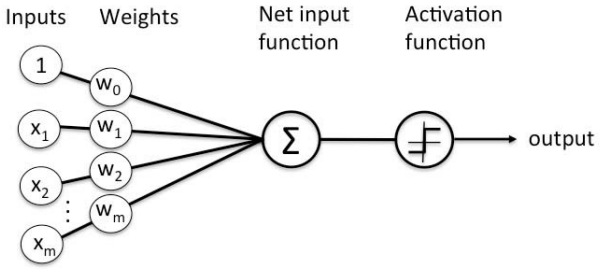


Figure 2.25: Single layer perceptron

Multiple layer perceptron

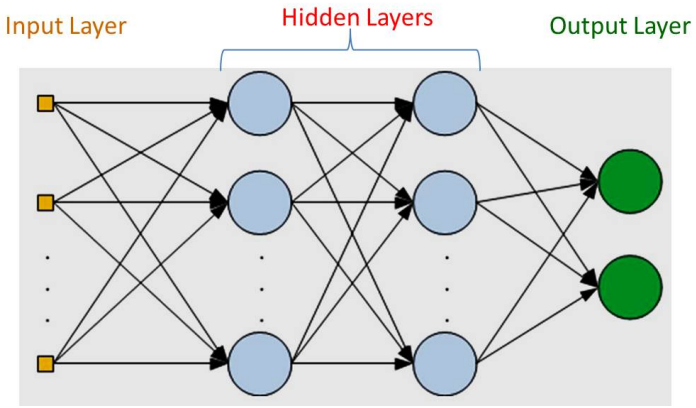


Figure 2.26: Multi layer perceptron(MLP)

By connecting multiple numbers of a single perceptron it is possible to create a more complex and efficient structure called multi-layer perceptron (MLP). Other than the input and output layers, it contains a certain number of hidden layers that are fully connected. Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

2.4.4 Backpropagation in detail

We briefly saw how backpropagation works. Considering that the understanding of this technique is extremely important for a complete comprehen-

sion of machine and deep learning, in this section I will explain more in detail his main features.

Weight initializzation

Weight initialization is a significant consideration in the design of a neural network model. The initial values of the weight define the starting point for the learning of the neural network, choosing the right initializer is important to our model's performance and training also because it might shorten the convergence time, as well as minimize the loss function by setting our initial weights. Generally, a weight initialization procedure sets them random values in a specific interval, which means that each time a neural network has a different starting point. A common approach is the xavier initialization method, it calculates the starting value as a random number with a uniform probability distribution between the range $-\frac{1}{\sqrt{n}}$ and $\frac{1}{\sqrt{n}}$, where n is the number of inputs to the node.

Cost function

Backpropagation algorithm needs to know how much the prediction is different from the true value in order to update weights. The cost function can be defined as a metric to determine the performance of a model and can be used to estimate how good the model is performing. There are different ways to calculate the error, let's have a closer look at some common types:

Distance based error Is the fundamental and most basic cost function for regression problems:

$$Error = y - y^{'}$$

where y is the actual value and y' is the predicted value. It is simple to calculate but also very limited.

Mean squared error (MSE) It is one of the most commonly used cost function in machine learning also known as the sum of squared errors:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y - y^{'})^2$$

where y is the actual value, y' the predicted value and N the total number of observation. It overcome the limitation of the previous cost function, because the squares eliminate negative values.

Root Mean Squared Error (RMSE) It is the square root of the MSE previously seen. It is calculated as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y - y')^2}$$

Mean Absolute Error (MAE) It is similar to MSE but instead of squared root, it use absolute error to avoid the possibility of negative error.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|(y - y')|$$

Categorical cross Entropy Categorical cross Entropy cost function is the most common cost function used in the classification problems where there are multiple classes and each input belong to only one class.

$$H(p, q) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \log p_{ij}$$

Where i indexes samples/observations and j indexes classes, y is the sample label and p is the prediction for a sample. It is a very good measure of how distinguishable two discrete probability distributions are from each other.

Binary cross entropy is a particular case of categorical cross-entropy. It is used when there are only 2 classes, therefore there is only one output that can assume a binary value of 0 or 1.

$$H(p, q) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

Error derivatives

After passing the input value a neural network process this value, predict the result and calculate the error between prediction and real value. In order

to update its weight, the backpropagation algorithm must know how much each node is wrong. This is achieved by calculating partial derivatives of the cost function for each weight and bias: starting from the output layer we can backpropagate the error in the neural network to the input layer. To obtain this we calculate the gradient of the last layer's weight, then, applying the chain rule it is possible to back-propagate it to calculate the gradient in the previous layer until it reaches the input.

Updating weight with gradient descent

After calculating the gradient, we must find out its minimum values, in order to minimize the cost function. Finding a global minimum analytically is often not feasible and we need to use some optimization techniques, the most common is Gradient descent. Gradient descent is an algorithm that numerically estimates where cost function (or any functions) has the lowest values. It obtain this not by setting $\nabla f = 0$ but using numbers. The idea is to use the following formula:

$$\phi_{n+1} = \phi_n - \alpha * \frac{\partial C(\phi_n)}{\partial \phi_n}$$

where ϕ is the parameter(i.e. weight) that we want to update, C is the cost function and α is the learning rate. Learning rate is a hyperparameter used in the training, with a value in the range between 0 and 1. The learning rate value can impact how fast the algorithms can learn and whether the cost function is minimized or not. It is important to set it to an optimal value because a big learning rate allows the model to learn faster, but with the risk of arriving at a sub-optimal final set, smaller learning instead can converge and trap the function cost to a local minimum.

Types of gradient descent algorithms Depending on when updating weight, there are mainly 3 types of gradient descent.

- **Stochastic Gradient descent**(SDG) calculates the error and update the model for each sample in the training dataset. The frequent updates can results in more detail and can also be helpful in escaping the local minimum but also it losses in computational efficiency if compared with batch gradient descent and can results in noisy gradients.

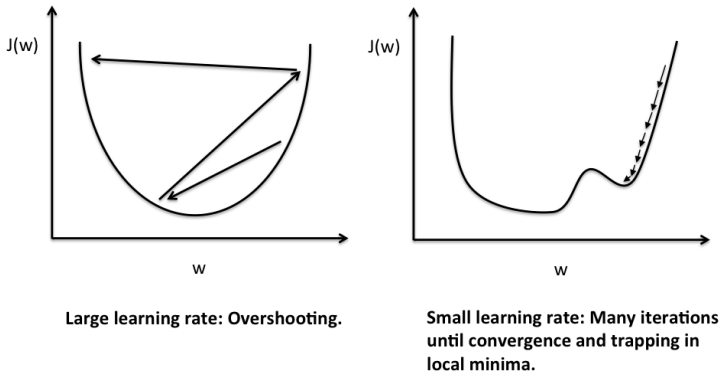


Figure 2.27: Comparison between large and small learning rate value.

- **Batch Gradient descent** calculates the error for each sample, but only updates the model after all training samples have been evaluated. A cycle of training through the entire dataset is called a training epoch. Batch gradient descent provides computation efficiency and usually produces a stable error gradient and convergence, but sometimes that convergence point is the local minimum and not the global one.
- **Mini-Batch gradient descent** splits the samples into small batches and updates the model after the evaluation of each batch. It is a compromise between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

2.4.5 Machine learning overview

After seeing how does machine learning algorithms learn, and their main features, let's have a general overview talking about some practical aspects.

Main component

There is the principal component of a neural network model:

1. **Parameters**: these are the coefficients of the model and they change across the epoch based on the loss or cost function. In other words, they are the changeable part of neural networks.

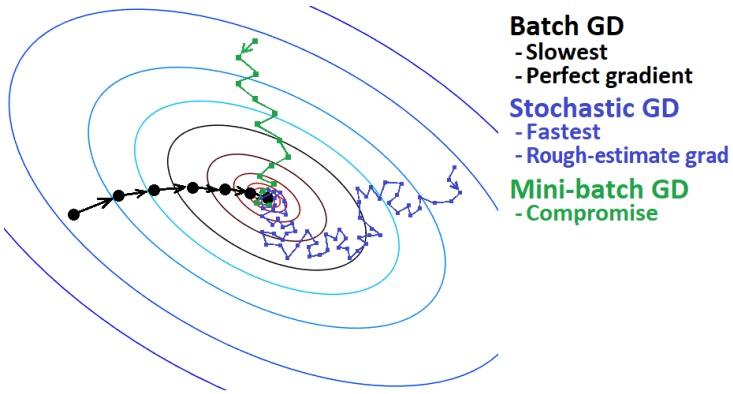


Figure 2.28: Comparison between gradient descent techniques.

2. **Hyperparameters**: contrary to parameters, these are set before the training step and don't change across the learning step.It is important to find the correct configuration of them because these are used for the prediction and for the training of the parameters.

3. **Loss function**: the loss function compares the predicted output with the correct value and returns a number that is used to update machine learning parameters. The neural network aims to minimize the loss function, in other words, it reduces the difference between the predicted and the correct values.

Classification of machine learning algorithm

Different machine learning algorithms use different strategies. They were developed depending on the nature of the problem and the type of data to analyze. Below there is a brief description of the main type of algorithms.

- **Supervised** The main feature of the supervised algorithm is that they need a dataset to train them. The goal is to approximate the mapping function so well that when you have new input data that you can predict the output variable for that data. The supervised algorithm can be divided into the classification problem category, characterized by a discrete-valued output, and into the regression problem category, characterized by a continuous-valued output.

- **Unsupervised** In this case you only have input data and no corre-

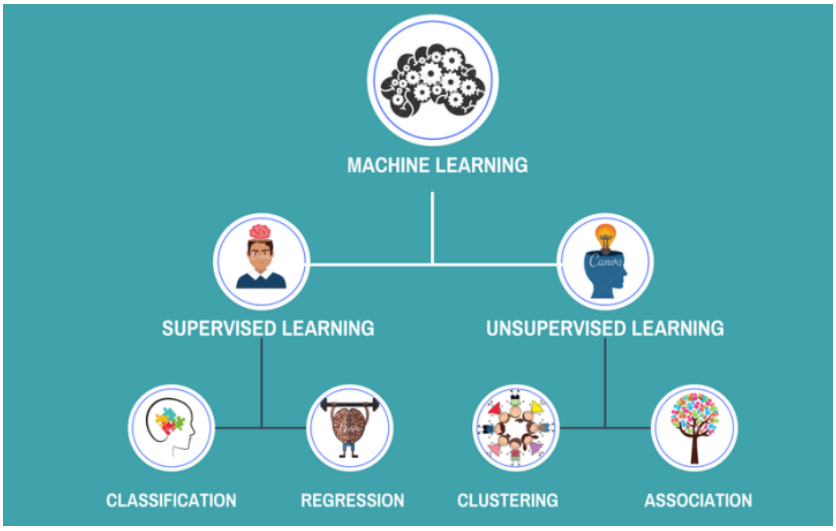


Figure 2.29: Supervised and unsupervised algorithm.

sponding output, the goal is to find out something in the data to learn more about them. Unsupervised algorithms can be further divided into clustering problems, that aim to recognize a grouping in the data, and associations problems that aim to discover rules that describe your data.

- **Parametric and not parametric** A learning model that uses a fixed set of parameters is called parametric. When you have no prior knowledge of your data, nonparametric methods are the correct choice because you don't have to decide features a priori.

2.4.6 Deep learning

As previously said, deep learning is a subset of machine learning. The highlights of deep learning are the scalability of neural networks, which means that the results get better with more data and using large model and the automatic feature extraction from data, also called feature learning. Feature engineering is a key step and it consists of two-step, feature extraction where we extract all the required features, and feature selection where we select the important features that improve the performance of the model. Deep learning models automatically learn features at multiple levels of abstraction and decide which one is the most important, this means that they don't depend completely on humans for finding features to learn function mapping the input to the output. Considering these aspects, deep learning can be a solution to the following problems:

- When human experts are not available
- When the problem solution update over time (feature importance may change)
- When the size of the problem is too large for us to solve it with reasoning abilities (sentiment analysis, matching ads, etc.)
- When is needed solutions adaptation on specific cases.

There are different types of deep learning algorithms each one is used to perform specific tasks. Let us see three common deep learning algorithms:

- Artificial neural networks (ANN) Also known as Feed-forward neural network it is the most simple (MLP is a class of ANN), but has two drawbacks, indeed the number of trainable parameters increases drastically with an increase of the dimension of input data.
- Recursive neural networks (RNN): what is featuring of RNN is the recurrent connection on the hidden state. RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.
- Convolutional neural networks (CNN): the building blocks of CNN are filters, a matrix that is used to perform a convolution all over the input. We will deepen CNN in the following section.

Convolutional neural networks (CNN)

A Convolutional Neural Network is a powerful neural network that uses filters to extract features from data, in addition, it analyzes input in such a way that position information is retained. It is used in many applications such as facial recognition, image classification and it is also used for DNA classification. The main aspect featuring a CNN are convolution and pooling layer. Therefore in order to understand how does CNN works, their comprehension is essential.

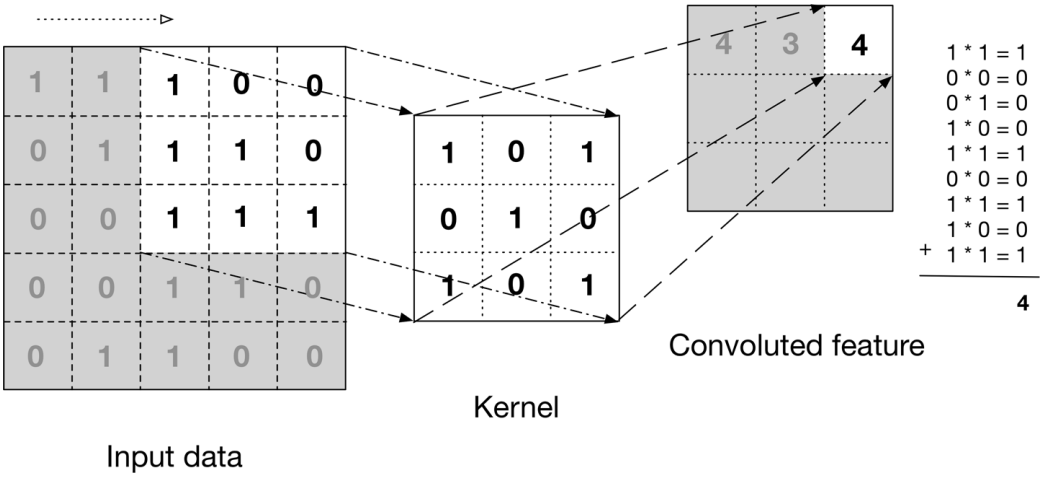


Figure 2.30: Example of convolution operation

Convolution The convolution layer performs a dot product between two matrices, where one matrix is the kernel, also called filter, that is the set of learnable parameters and the other matrix is a portion of the input data. In Figure 2.30 is represented the convolutional operation, as we can see, the kernel slides across the height and width of the input data, and for each movement, it performs a dot product. The results of this operation produce another matrix, known as an activation map.

Pooling Pooling is an operation generally performed after the convolutional one, it is done separately upon each feature map and it creates a new set of the same dimensions of pooled feature maps. Pooling can be considered as a filter applied to feature maps, and it always reduces the dimension of the

input data. It is important to underlying that, contrary to the convolutional filter, these values are not learned. There are a lot of reasons why pooling is used, the most important are listed here:

1. Downsampling: Pooling layers reduce the dimensions of the feature map, therefore they reduce the computational load, by reducing the resolution of the data, without losing important information.

2. Convolution operation retains positional information. Pooling layer permits to obtain summarised featured, getting a feature map that doesn't have this information, this makes the model more robust to variations of the position.

These are the reasons why the addition of a pooling layer after a convolutional layer is very common and this pattern can be repeated at different times in a given model. Depending on the filter operation, there are mainly 3 type of pooling:

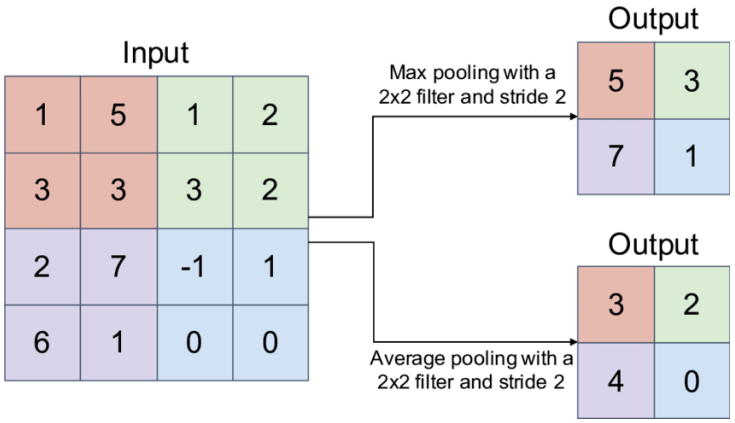


Figure 2.31: Example of average and max pooling

1. **Average pooling**: it performs the average of the elements in the region of the feature map covered by the filter. It results in considering an average of the features present.(Figure 2.31)

2. **Max pooling**: it selects the maximum element from the region of the feature map covered by the filter, therefore the output of max pooling contains the most prevailing feature. (Figure 2.31)

3. **Global pooling**: instead of down sampling patches of the input feature map, global pooling down samples the entire feature map to a single value. It can be average or max pooling, depending on the operation used. For example, global average pooling calculates the average output of each feature map in the previous layer, this reduces a large number of trainable parameters, therefore it lowers the tendency of overfitting. (Figure 2.32)

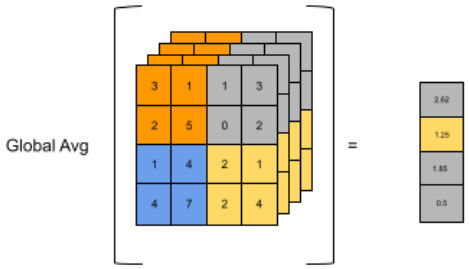


Figure 2.32: Example of average and max pooling

Below are listed some hyperparameters introduced when CNN is adopted:

- **Number of filters**: As said before the convolution is the result of a dot product between the input and the filters. The values of them are not fixed but change over the training phase. Considering that a filter aims to capture a specific pattern, the more the filter, the more will be the pattern captured but also, the more heavy will be the computational weight.
- **Kernel size**: There is not a general rule and it depends on the input and on the type of feature we want to learn.
- **Stride**: Is the number of elements shifting over the input when convolution is performed. Adding a stride to the model decreases the size of the feature map. A common value is stride = 1.

Strength and weakness of CNN

Understanding the strengths and weaknesses of machine of CNN allows me to contextualize results and it is essential for making correct decisions.

Main Advantages

1. CNN learns the filters automatically without mentioning it before, this helps in extracting the relevant features from the input data.

2. It can learn to recognize patterns across space.

3. It is able to extract local and position-invariant features, this is an advantage if data are random-ordered.

4. Parameter sharing: is sharing of weights by all neurons in a particular feature map. This helps to reduce the number of parameters in the whole system and makes the computation more efficient, in addition it enhance the generalization to avoid overfitting.

Main disadvantages

1. It needs a sizable volume of data to obtain good performance. [**?**]

2. CNN does not encode the position, they completely lose all the internal reference of input. It makes predictions by looking if a feature is present or not.

3. Vanishing gradient problem is an issue that sometimes arises when training machine learning algorithms through gradient descent. Since the gradients control how much the network learns during training, if the gradients are very small or zero, then little to no training can take place, leading to poor predictive performance.

2.4.7 Data representation in machine learning

The training of neural networks, aside from architecture, depends on the input data. Aspect such as the quality of the information or the use of an efficient representation are key factors for correct training. There are two main types of data, categorical and numerical data.

1. **Categorical** is a type of data that can be stored into groups or categories using names of labels, they can be nominal and ordinal data, with the difference that the last one can be ordered, despite it can not be measured. Some examples are the blood group of a patient or the type of water.

2. **Numerical** is a type of data that is expressed in terms of numbers and not with language description. They can be discrete and continuous data, for example, the weight of a person, the temperature, price, etc..

Before searching the best representation we must consider that input data of a neural network must be numerical, this means that categorical data cannot be stored without changes. There are three common ways to convert categorical variables to numerical values.

1. **Ordinal Encoding**: In Ordinal Encoding, each unique category is assigned to an integer value. It is the more intuitive and easy to implement encoding, taking DNA as an example, a possible approach might be the following: A=1, C=2, G=3, T=4. It is preferred to use a natural encoding for ordinal variables, in fact, it imposes an ordinal relationship.

2. **One-Hot Encoding**: it uses many binary variables for each category, it produces a vector with a length equal to the number of categories, each category is represented by a 1 and 0 for the other values. It is used for categorical variables where no such ordinal relationship exists.

3. **Dummy variable encoding** Despite one hot encoding creates one binary variables for each category, it create one less. In fact if we have for example three category, we can use [0 , 1] for class A, [1 , 0] for class B and [0 , 0] for class C.

color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0
red	1	0	0

Figure 2.33: One hot encoding for color encoding

2.4.8 AI implementation

There are quite a few programming languages for AI programming, and each of these has a framework and API with different features. None of them could be considered the best AI. Below there is a short description of the most popular programming language and libraries used for machine learning with advantages and drawbacks.

Common programming language

While choosing a programming language there are plenty of aspects to consider, for example, it must have many good deep learning libraries, but should also feature good runtime performance, a large community of programmers, and a healthy ecosystem of supporting packages, etc...

- **Python** is a high-level object-oriented programming language, that provides an easy experience for ML. Python is more intuitive than other programming languages because it has a simple syntax. Thanks to this it is possible to focus only on the machine learning task, rather than the specifics of the language. Below are listed the main advantages:

  1. Easy to learn and use
  2. Open source, you can download any source code and modify it.
  3. Massive number of libraries
  4. Easy portable over platforms.

  Disadvantages:

  1. Low speed, being interpreted the code is executed line bi line leading to slow execution.

2. Inefficient memory consumption

- **R** is both an open-source programming language and a statistical computing environment. The language provides objects, operators, and functions that make the process of exploring, modeling, and visualizing data a natural one. But compared to python it is less powerful and robust. Below are listed the main advantages:

  1. Best programming language for statistic
  2. Open source.
  3. High number of libraries
  4. Easy portable over platforms.

  Disadvantages:

  1. Low speed, less than Matlab and python
  2. Complicated language
  3. Inefficient memory consumption, less than python

- **C++** is a great option for AI, principally because of its fast calculations that solve complex computations of AI development. Moreover, it is cost-efficient as compared to other programming languages. Holding the capability of both a low-level and high-level programming language, C++ comes with a higher level of control and efficiency as compared to any other language.Below are listed the main advantages:

  1. Low-level manipulation.
  2. Memory management
  3. Large community support
  4. Fast and powerful.

  Disadvantages:

  1. Complex
  2. Less flexible, it is very strict regarding the syntax.

- **Java** is a cross-platform object-oriented programming language easy to implement on different platforms because of its Virtual Machine technology. Simply you can write the code on one machine and run it on any machine. For these reasons, Java is highly versatile and is used for robot systems, sensors, and machine learning suits. Below are listed the main advantages:

  1. Easy to learn and use.
  2. Automatic garbage collection.
  3. Platform independent.

  Disadvantages:

  1. Java needs to be interpreted during runtime and this makes it perform slower than other languages like C.
  2. High memory consumption
  3. Garbage collection cannot be controlled by a programmer.

- **Matlab** is a programming platform designed specifically for engineers and scientists to analyze and design systems and products that transform our world. Advantages:

  1. Easy to use
  2. Platform Independence
  3. Huge number of libraries
  4. GUI

  Disadvantages:

  1. Slower execution than other languages.
  2. High cost.

Useful python libraries for machine learning

As previously said, the strength of python, among other things, is due to the high number of libraries, let's consider the most used and useful for machine learning.

- **Tensorflow** is an open-source end-to-end deep learning framework focused on being flexible. It is well documented and has a lot of tutorials and trained models.
- **Keras** is a high-level neural network API written in Python. Its aim is to provide fast experimentation with deep neural network, it is focused on being modular and user-friendly, therefore it doesn't handle low-level computation. It is a good choice for starting to work with deep learning.
- **Theano** Theano is an open-source project, it was specifically designed to handle the types of computation required for large neural network algorithms used in Deep Learning. It was one of the first libraries of its kind and is considered an industry standard for Deep Learning research and development.
- **Pytorch** Is a deep learning framework based on Torch focused on simplicity, ease of use, flexibility, and efficient memory usage.

2.4.9 Training step and validation in machine learning

Training is the most important step in machine learning. In training, you pass the data to your machine learning model to make predictions. It results in the model learning from the data so that it can accomplish the task set. Over time, with training, the model gets better at predicting. To know if the training is working well there are a lot of indexes for his evaluation.

Dataset divisions

A first approach might be to use the entire dataset to train the neural network, because the more the sample, the more accuracy. But it is not a correct choice because without division it is impossible to know if the NN is encounter problems such as overfitting and underfitting. To overcome this it is essential to divide the dataset into two parts, one used to train and the other to test. Datasets are generally divided in the following parts:

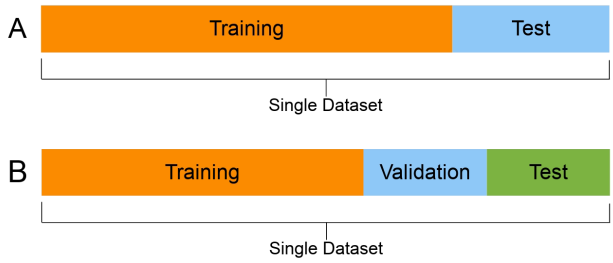


Figure 2.34: Two examples of dataset division

- **Training Set** is the part of data used to train the model during each epoch. The model will be trained only with this set and will only learn features from this.
- **Validation Set** is used to validate our model during training. The validation set is not used by the neural network to update the weight, instead, it is used by the developer to find out if the model training is working well. For example, it is important to ensure that our model is not overfitting, and therefore that it has the ability to generalize.
- **Test set** is used to test the model after the training step and provides a final check that the model is generalizing well avoiding the overfitting problem.

Consideration on dataset splitting

There are many approaches for dataset dividing, anyway two important consideration needs to be done:

1. Training set is bigger than validation and test. Usually, 90% are used for the training and the rest is split.

2. All sets need to be representative, therefore all of them must contain samples that represent the problem space.

3. It is important to maintain a proportion between the different classes, to avoid an unbalanced dataset.

4. Validation and test set might be the same set.

Performance evaluation metrics

In machine learning, evaluation metrics are fundamental to determine the performance of the machine learning model. It is essential also to evaluate it when used on a dataset different from the trained one, obviously, the model you have trained will perform better on a training dataset, but if it doesn't work on other data, it is useless. To overcome this, machine learning developers use evaluation metrics.

Loss function Since the aim of a model is to reduce the loss function of the models using optimization techniques like Gradient Descent, this is the primary method for evaluating how well the algorithm fit the information.

Confusion matrix The confusion matrix measures how well instances from each category are predicted by the machine learning (See Fig 2.35 A confusion matrix is a matrix that contains the amount of correct and incorrect prediction. In binary classification, it shows only four values indicating true and false negative and true and false positive. From the confusion matrix we can calculate other metrics which offer more information.

1. **Accuracy** is defined as the proportion of correct prediction over all prediction made. It give an overview of neural network performance because it return incomplete information. In fact the accuracy metric is not suited for imbalanced classes, when the model predicts that each

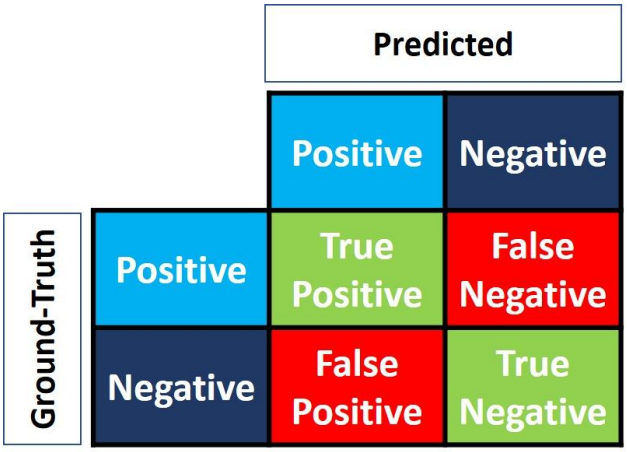


Figure 2.35: A binary confusion matrix

point belongs to the majority class label, the accuracy will be high. But, the model is not accurate.

$$Accuracy = \frac{(TruePositive + TrueNegative)}{(TruePositive + TrueNegative + FalsePositive + Falsenegative)}$$

2. **Precision** is defined as the number of true positives divided by the sum of true and false positives.The precision measures the model's accuracy in classifying a sample as positive.

$$Precision = \frac{TruePositives}{(TruePositives + FalsePositives)}$$

3. **Recall** is defined as the number of true positives divided by the sum of true positive and false negative. The recall cares only about how the machine learning classify the positive samples.

$$Recall = \frac{TruePositive}{(TruePositive + FalseNegative)}$$

4. **Specificity** is defined as the number of true negative divided by the sum of false positive and true negative. The specificity is used to calculate the fraction of negative patterns that are correctly classified.

$$Specificity = \frac{TrueNegative}{(FalsePositive + TrueNegative)}$$

When to stop training

A big challenge when training a neural network is to know when to stop training: training it for too little time will create a model that under fits the training test, on the contrary, too much training instead will return a model that overfit training test, that means performing badly with different data. The best is in the compromise, but it's hard to find it. During training there

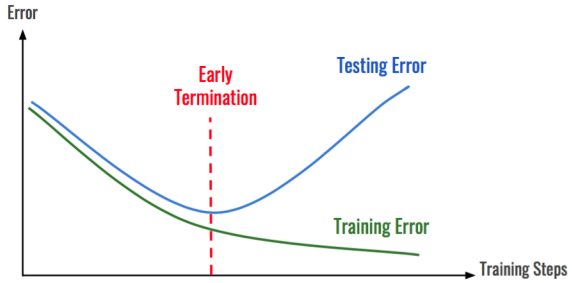


Figure 2.36: Early stopping point representation.

will be a point where the model will stop generalizing and start learning the statistical noise in the training dataset (Figure 2.36), this results in overfitting of the training, which means that the goal is to train the network up to that point. There are different approaches, for example we can track the validation test, when there is no improvement, we will stop training.

K-fold Cross Validation

A widely used technique to assess the performance of machine learning models is cross-validation. The first step consists of the split of original training data set into K equal subsets. Then for each group:

1. Take the group as a holdout of test data set
2. Take the remaining groups as a training data set
3. Train a model on the training set and test it on the test set
4. Retain the evaluation score and discard the model

Finally, you can estimate the accuracy of your machine learning model by averaging the scores. The value of k is an important choice, a common

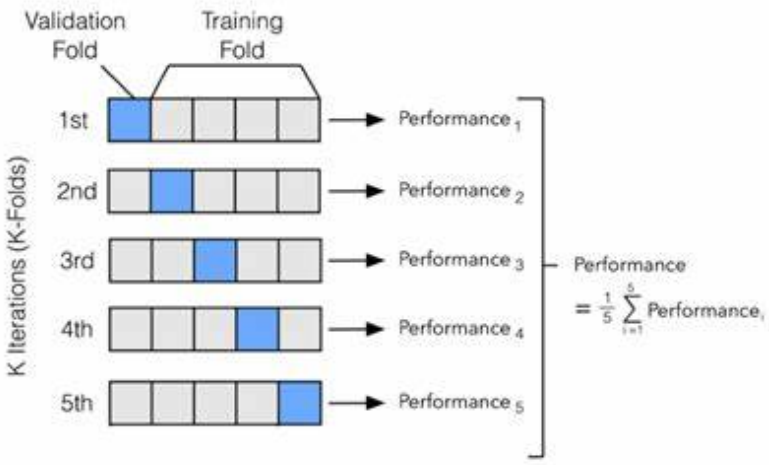


Figure 2.37: K fold validation example

problem is to create a not representative dataset that leads to invalidating the results. One common approach is to choose k in such a way that each group of data is representative of the dataset.

Handling unbalanced datasets

Most datasets are not balanced, such as biologican one. If not handled, working with this kind of datasets leads to a model that can correctly classify only the most numerous category. Let's see the most common option for dealing with this problem:

- Undersampling resamples the majority class points in the data to make them equal to the minority class points. But in this way, we do not use a significant chunk of the data, which contains some information.
- Oversampling, for example with the repetition of the minority class. In general, it increases the possibility to overfit the training data.
- In some cases it is possible to generate samples, different from the existing one, to increase the quantity of the minority class
- Use a weighted loss function, to penalize the most present category or to make minority sample more important during backpropagation.

2.4.10 State of art

Over the past couple of years many convolutional architectures have been created for different purposes, varying in many aspects such as the types of layers, hyperparameters, etc. Historically, AlexNet is one of the most popular neural network architectures, it is comprised of eight layers in total, out of which the first 5 are convolutional layers and the last 3 are fully connected. It was created to classify the 1.2 million high-resolution images into more than 1000 different categories. Another important architecture was the Inception Network (GoogleNet), its architecture consists of 22 layers (27 layers including pooling layers), and part of these layers are a total of 9 inception modules. Their architecture consisted of a 22 layer deep CNN but reduced the number of parameters from 60 million (AlexNet) to 4 million. Convolutional neural networks are really popular nowadays, and a lot of other known architectures were developed: Highway, ResNet, Xception, DenseNet, and many others.

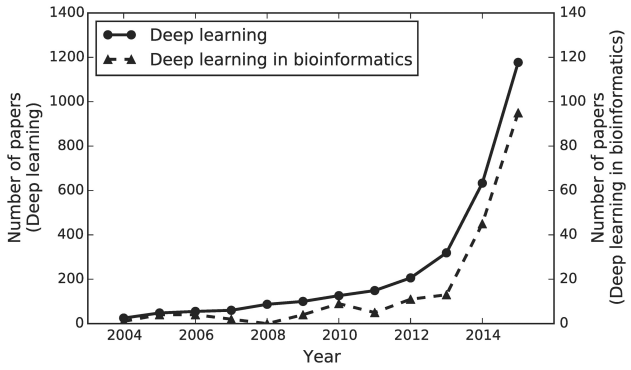


Figure 2.38: Number of published deep learning articles by year. [17]

As we can see in Figure 2.38, the number of deep learning papers in bioinformatics is increasing every year. Currently, few studies have been provided for the DNA sequence classification problem and have shown the success of using deep learning. For example, DNA classification using a CNN has been successfully applied to the classification of viruses, with a relatively simple model that was able to tackle SARS-CoV-2 [14]. We must consider that the size of a bacterial genome is much larger than that of a virus (around 10x). However other studies demonstrate this approach also works for DNA bacteria classification: for example, Rizzo et al [21] describes the first experiment

in the use of a deep neural network for the classification of DNA sequences represented as images by using the Frequency Chaos Game Representation. And also [13] present two different deep learning architectures (CNN and RNN) for the purpose of DNA sequence classification and also Ngoc Giang Nguyen [18] proposed a new approach in classifying DNA sequences using the convolutional neural network while considering these sequences as text data

Chapter 3

Proposed approach

Before taking any decision about design, I did general consideration about the study case, in particular about the nature of the input data and the resistance mechanism: a machine learning algorithm can predict the results only if the input data contains the information needed; therefore to make the model predict resistance or sensibility, this information must be contained in a DNA sequence. Since many determinants of an antibiotic-resistant phenotype are coded from the microbial DNA content (genome and plasmids), including mobile genes, single-nucleotide mutations, or wider genomic alterations, this means that a neural network could predict the resistance. Stated that the information might be present, two more considerations need to be done:

- The identification of resistant genes is not only related to the sequences of nucleotides but depends also on the position of the gene. Dataset I'm working with contains a sequence not aligned, this means that if the resistance depends also on position, the neural network might not predict it.
- The same problem is present despite the alignment of data if CNN is adopted. In fact, it can't predict resistant genes depending on position, because CNN is positional invariant.

The approach proposed here is the outcome of a study focused on obtaining the best results using the most elementary architecture, to follow this method I began with a very simple model, making it more complex only if needed. This strategy may seem a little too strict, but I think that using it is possible to find out the relationship between some hyperparameters for

future implementation, also elementary models are better than complicated one for obvious reasons, such as computational speed and ease in tuning, in addition, a simple model might make simpler understanding what the model understood. This section will describe the following steps:

1. Data preprocessing: Raw data needs to be manipulated to feed neural networks and to enhance performance.
2. Implementation: It includes the language selection and coding step.
3. Model tuning: experiment to find the best value for hyperparameters.
4. Model validation: It consists in the validation of the model selected.

3.1 Dataset and preprocessing

Dataset is a vital part of the neural network in supervised learning, without which there is no training step. Therefore studying it is essential for an AI developer, only with the full comprehension of the data it is possible to find the best way to represent it and the best way to use it for training the neural network.

3.1.1 DNA representation

Before explaining my decision, let's summarize DNA most important features:

- DNA sequences data consist of non-numeric characters (A,C,G,T).
- The length of each sequence varies greatly, from only a few dozen characters to hundred of megabytes.
- DNA information type is categorical.

When processing DNA sequences we must convert the characters into a numerical value, in this way we can use it for feeding artificial intelligence. As previously explained in Section 2.4.7 there are different ways to represent information, below are listed different ways referring to DNA:

- Sequential encoding: considering that each nucleotide is categorical information, it is better to don't use this to avoid neural networks from learning ordinal-like features.
- One hot encoding: heavier computational weight because one nucleotide is represented with 4 bit, but more suited for categorical data. One hot encoding results in a better accuracy with a categorical data, compared with others representation methods. [19]
- Dummy encoding: could be an alternative to one ho encoding to reduce by 1 the number of variables used respect one-hot-encoding, but I decided to use the all-zero vector to represent missing values.

After these considerations I decided to use One hot encoding nevertheless it has a heavier computational weight, than sequential encoding, because the difference is minimal and also I used the all-zero vectors for missing values representation.

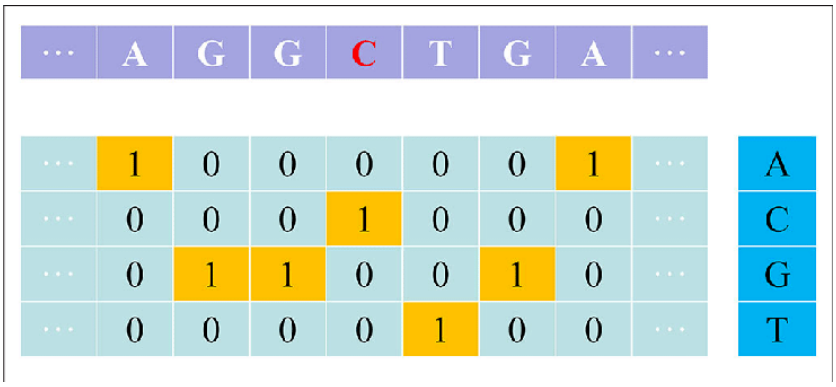


Figure 3.1: DNA representation using one hot encoding

3.1.2 Dataset

For the learning step, the model needs both sequences to study both feedback to improve his prediction. The sequences were stored in FASTA file format, where are divided into contings ordered from the longest to the shortest. The category of each sequence (i.e. resistant or sensible) was stored in a text file, containing AST analysis performed on a specific sample using specific

antibiotics. The first preprocessing step consists in creating a functional dataset from this file, obtaining for each antibiotic all the sequences tested and the associated category.For the implementation consult section 3.4.

Dataset division

As mentioned in section 2.4.9 generally developers don't use the entire dataset for training, it is usual to divide it into training and test set. Usually, developers divide it into 80-90% for the training part and the rest for testing. I use a value between 90% and 95% of the training because the number of samples for all three cases was not so high (more information in Section 3.1.2), so I preferred to give more importance to training bearing in mind the low quantity of test samples. Also, commonly developers divide the test set into two parts, one used to test during training for the validation, and one after, considering the low number of samples, I preferred to don't divide it, and to use a part of the same group for the validation.

Umbalanced datasets

Before creating the dataset, its analysis is essential to help neural network and avoid problems during his training. A common problem to check is if the dataset is unbalanced. As previously said in section 2.4.9, it is important for training purposes that each group is representative, and for a highly unbalanced dataset, this aspect might not be verified. The main motivation behind the need to manage an unbalanced dataset is that typically classifiers are more sensitive to detecting the majority class and less sensitive to the minority class and in addition, if we don't take care of the issue, the classification output will be biased, in many cases resulting in always predicting the majority class. Many methods have been proposed in the past few years to deal with imbalanced data. In the background chapter I already talked about the different techniques to manage an unbalanced dataset, below are listed some considerations about each of them, in relation to the case I'm working with:

- **Undersampling the majority class**: it is one most common and easy strategies, however, I avoided this solution because a neural network, especially a CNN, the more sample has, the best is the performance. Considering that I didn't have a lot of samples, I didn't adopt this solution.

- **Oversampling the minority class**: with this kind of solution, we duplicate or manipulate samples. Passing the same sample as differents can lead to overfitting, also in my case, the dataset is strongly unbalanced, resulting in an intensification of this problem.

- **Weighted cross-entropy function**: this is the most suitable option, easy to implement, and tunable but most important, dataset manipulation are not needed. I calculated a proportionality parameter to evaluate the amount of unbalancing, in this way I give more importance to the training of less present samples.

Pratical consideration on dataset

I worked with three drugs, each one has different features such as resistant mechanisms and a different dataset, therefore before using models for training, I collect some information for every single one:

- **Methicillin** dataset is characterized by 4021 resistant and 1466 sensible samples. The sensible, resistant ratio is 0.36, therefore it is a moderately unbalanced dataset.

- **Tetracycline** dataset is characterized by 560 Resistant and 5869 sensible samples. The resistant, sensible ratio is 0.095 resulting in a strongly unbalanced dataset.

- **Inducible Clindamycin** dataset is characterized by 1913 Sensible and 1397 resistant samples, with a sensible, resistant ratio of 0.73, resulting in a lowly unbalanced dataset.

3.2 Architecture adopted

There are many ways to solve classification problems using machine learning, considering both architecture and the hyperparameter tuning almost infinite alternatives are possible. It is not obvious to catch, before doing experiments which are the best, for these reasons, most of the time the only way to find the best architecture is to try different alternatives doing some consideration to reduce the space of possibilities.

3.2.1 Why a why not using CNN

I already wrote about the advantages of using a deep learning algorithm, here I will consider these advantages one more time, but contextualize them to DNA classification.

- **Feature extraction**: The most important reason that suits this situation is the automatic feature extraction, which permits finding if a sample is resistant including if it depends on genes that still don't discover.
- **Weight sharing**: In the convolutional layer, the weight is associated with the dimension of the kernel and not of the input. A DNA sequence is very long, this means that it reduces significantly the number of trainable parameters
- **Memory saving**: Having a reduct number of learning parameters consequentially reduces the usage of memory.
- **Independent of local variation** CNN permit finding features that are independent of the position and considering that the dataset is composed of non-aligned contig, this is an important characteristic.

Besides all these aspects, we must consider that the performance of CNN depends on the causes of the resistant mechanism, because as I said before if the resistance is provided by a particular sequence, CNN is able to identify its pattern with no problem, but if it is important also his position inside the sequence, CNN will not predict correctly.

3.2.2 A non-conventional CNN

In section 2.4.6, we seen that convolutional and pooling layer are the main element characterizing the convolutional network. Commonly developer decide to use this layer arranged as in figure 3.2. The input is followed by

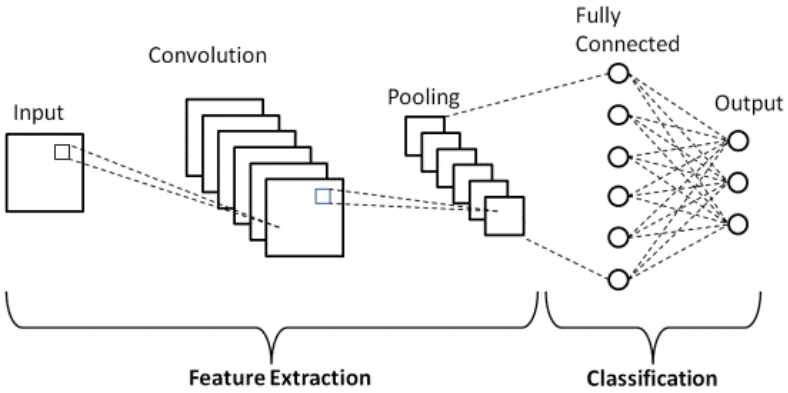


Figure 3.2: A common architecture using CNN

an arbitrary number of convolutional layers followed by a max or average pooling layer to perform the feature extraction, then there is an arbitrary number of fully connected layers followed by an output layer in order to classify. This is a widely used architecture, that works great in many situations, the problem is that if fully connected layer is adopted, the model needs to know a priori the size of the sequence, but in my case, each DNA sample has a different dimension.

How to deal with different input size

For convolution, but also pooling layer is not a problem to deal with different input sizes, in fact, the convolution is a dot product, and it can be applied to different length matrix and the pooling layer performs only a downsizing. Indeed the problem is not associated with the convolution layer, but with the fully connected layer. A fully connected layer must know the input size, in this way they can instantiate the weight variables and this is the reason why dealing with different input sizes is a common problem also for CNN. There are many solutions to overcome this problem, for example in image classification to fix the dimensions, they just scale or crop the image, but

this kind of solution is not suitable for a DNA sequence. Another easy way to overcome this problem is by using the zero-padding techniques, which consist in the extension of all the sequences in order to have the length of the longest one, generally, they fill the empty space with 0. Despite in some cases it is a great choice, for bacterial's DNA is not a good road to follow because the difference of length of the sequences is in the order of $10^3 - 10^6$. Another approach is to use the global pooling, which i adopted in this thesis. (Section 3) As you can see in figure 3.3, starting from a shape (B, H, W, F)

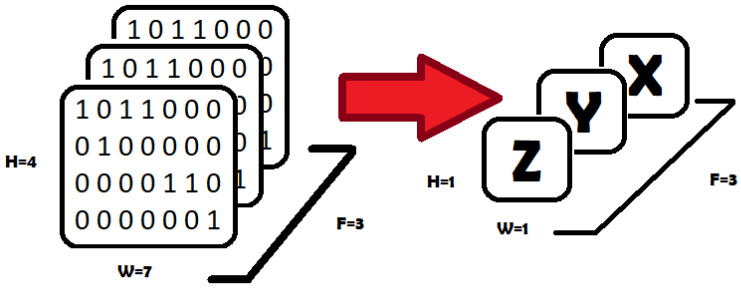


Figure 3.3: Example of global pooling operation.

the global pooling will return a shape (B,1,1,F), Where B is referred to the batch, (H,W) are the dimensions of the matrix (in the image is represented a sequenced encoded with one-hot-encoding) and F the number of the filters. In this way, the output dimensions are fixed and independent from the length of the sequence and it's now possible to use a fully connected layer, without managing the sequence length.

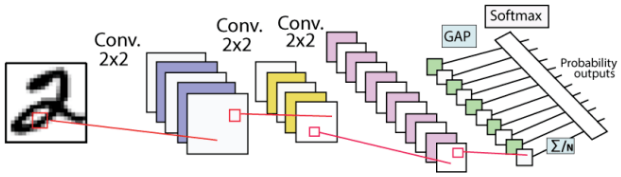


Figure 3.4: Global pooling in image classification

3.2.3 Starting configurations

The starting configuration for architecture and for hyperparameter configuration is an important step, which influences the performance and the time before finding a good solution. For the last one, trying all the combination is almost impossible, for this reason, it is important to do some consideration about each hyperparameter to reduce the combination.

Starting architecture structure

Here is a brief explanation of how my model works, below there is the sequence of the layers inside the networks.

> Convolution layer – Max pooling layer – Global pooling layer – Dense layer – Dropout – Dense layer (Output)

The input layer coincides with the convolutional layer, in this step each kernel is convoluted over the DNA sequence, with the aim of feature extraction; then it is first downsized with a max-pooling operation and finally, with global pooling procedure. The output of the feature extraction section is equal to the number of filters. This feature is then processed by a dense layer and finally passed to another dense layer for classification.

1-D convolution When talking about convolution, generally we refer to a kernel of NxN dimensions, this is because CNN was first used for image classification. An image is represented as a NxN matrix, and the convolution move across both matrix directions, this is called 2D convolution. Instead, in this case, the DNA sequence is represented with a one hot encoding, so it is 4XN length.Since that we must consider all the four values of vector at the same time, to maintain its meaning, the convolution is performed only over the second dimension. This kind of convolution is called 1D convolution and is the same used in text analysis.

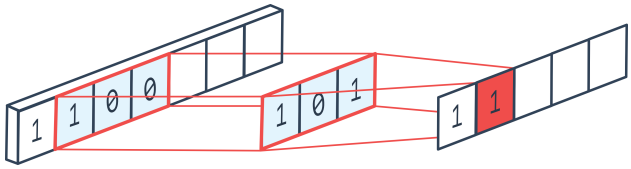


Figure 3.5: 1D convolution example.

Fixed - Hyperparameters

Each model is characterized by a huge amount of hyperparameters, trying to tune all of them at the same time is challenging, therefore I preferred to fix some hyperparameters values, using results found in literature, and tune the others.

Optimizer An optimizer is an algorithm that reduces the loss function by updating the weight in backpropagation. The most common algorithm used is stochastic gradient descent, in python, there are many alternatives for his implementation but one of the most popular and efficient is the adam optimization algorithm:

> Insofar, RMSprop, Adadelta, and Adam are very similar algorithms that do well in similar circumstances. [...] its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice.[22]

Loss function There is no universal loss function that is suitable for all machine learning models. Depending upon the type of problem statement and model, a suitable loss function needs to be selected from the set of available. I chose binary cross-entropy, the most commonly used in binary classification tasks, which is when there are only two possible categories.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i * \log(p(y_i)) + (1 - y_i) * \log(1 - p(y_i))$$

where y is the label and p(y) is the prediction, that is the probability of being a specific class. N are the number of the predictions performed by the neural network. (the probability of a items of class A to be A is 100 %)

Activation functions The choice of activation function are a critical part of the design of a neural network: in the hidden layer, the activation function will define how well the model will learn, in the output layer it will define the type of prediction done.

- For the hidden layers, I adopted ReLU activation function, for the reasons already discussed in Section 2.4.2. Leaky ReLU was also a

possible alternative, but using it means having another parameter to tune, the slope, in addition, we must consider that is true that Leaky Relu advantage is to avoid the death of the node of a neural network, but this also means adding more computational load on each epoch, where it may be not necessary. For this reason, I decide to use it only if having vanishing gradient problems.

- The output layer determines the form of the output. Classification algorithms statistically model the input data by determining the probabilities of the input belonging to different categories, for an arbitrary number of classes, normally a softmax layer is used in the output layer, it is demonstrable [7] that the sigmoid function becomes equivalent to the softmax function when dealing with two classes, this means that the choice is before using one or two neurons for classification. Since using one neuron means fewer parameters and therefore less computation are needed, i decided to use sigmoid function.

Batch size Batch size and learning rate are two fundamental hyperparameters, but considering that they may slightly interact with other hyperparameters I decided to set it initially and to re-optimize them at the end. To decide batch size I preferred to rely on the literature than on the test above then on not set neural networks. What I found is that a good batch size value is equal to 32 or smaller:

> The presented results confirm that using small batch sizes achieves the best training stability and generalization performance, for a given computational cost, across a wide range of experiments. In all cases the best results have been obtained with batch sizes m = 32 or smaller, often as small as m = 2 or m = 4. [16]

Nevertheless a lower batch size means longer training time, since they achieves the best training stability and generalization, I decided to adopt batch size equal to 2.

Learning Rate For the initial learning rate, I considered that:

> The initial learning rate [...] This is often the single most important hyperparameter and one should always make sure that it has been tuned [...] Typical values for a neural network with

> standardized inputs (or inputs mapped to the (0,1) interval) are less than 1 and greater than 10^{-6} [...] A default value of 0.01 typically works for standard multi-layer neural networks but it would be foolish to rely exclusively on this default value. [6]

In addition to this, one study demonstrate that, for deep learning:

> Learning rates 0.0005, 0.001, 0.00146 performed best and that above 0.001, increasing the learning rate increased the time to train and also increased the variance in training time (as compared to a linear function of model size).[15]

Therefore, I chose to start with a learning rate of 0.001.

3.3 Experiment approach

When finding an optimal tuning for machine learning it is essential to follow a pattern, the initial experiments may be random, but you must try to learn something from each of them.

3.3.1 Why I didn't use automatic tuning approach

There are different ways for searching the optimal tuning settings, these techniques are divisible in two categories: manual and automatic approach.

- **Automatic approach**. There are several automatic approaches such as grid search, where we test all the possible combinations, random search, Bayesian Search. In general, this kind of approach is computationally demanding or hard to implement.
- **Manual approach**. The manual approach is when the developer manually tweaks the hyperparameter combinations until the model gets the optimal performance. You have to really understand how the machine learning algorithm works when you want to go with this approach. Basically, this method works like this:

  1. Train and evaluate the model
  2. Guess a better hyperparameter combination
  3. Re-train and re-evaluate the model
  4. Continue until optimal score obtained

  This kind of approach required some experience or to have time to spend, also is hard to guess even though you really understand the algorithm and require so much time.

Considering that I had no high computational resources and considering that this approach give me more comprehension about the model I preferred the manual one.

3.3.2 Training time

I already wrote about when to stop training in Section (2.4.9), in brief, it is important to stop the training before it overfits and after underfit training

data. My approach consists in training the model for N epochs, and saving the best models, where the best is based on loss function (the error). This method is not perfect since that in an overfitting situation it is not a good evaluation parameter anymore, for this reason, each time I observe the validation test results The main problem is that the DNA sequence is long, this means that it needs a lot of times (depending also on the architecture) to perform a single epoch training, for this reason initially I used to train the model for a few epochs, then I understand that it was useless because the results were meaningful. Therefore I update the neural network class, make the training resumable, in this way I could start with a few epochs, and continue the training only if needed.

3.3.3 Results evaluation

The evaluation is performed in different moment:

- **During training** evaluating the loss function is the primary parameter to understand if the neural network is learning.
- **At the end of each epoch** the accuracy calculated over the validation set is essential to notice if the model is overfitting dataset.
- **At the end of the training**, on the test set, evaluating the accuracy, the average loss (precision), and Sensitivity(Recall) and Specificity parameter calculated on confusion matrix. Recall and specificity was defined in Section 2.4.9. Considering that dataset are unbalanced, I use this parameters to understand if the models learned to classify both classes.

After finding the correct model, for his validation, I used a kfold cross validation. Considering that

> The choice of k is usually 5 or 10, but there is no formal rule. As k gets larger, the difference in size between the training set and the resampling subsets gets smaller. As this difference decreases, the bias of the technique becomes smaller [11]

I decided to use k=5 to speed up the validation time.

3.4 Implementation

Not least important are the numerous decisions for implementing a machine learning algorithm. In this section I will explain some practical aspects of implementation, starting from the selection of programming language to code explanation. You can find the whole code on my github repository.

3.4.1 Programming language used

As seen in Section 2.4.8, the choice between all the programming language is wide, it depends on the application and needs. For example using C++ is the perfect choice when you aim a fast execution. Considering that my aim was to demonstrate that CNN is able to classify resistant samples, I preferred to use an easy programming language than a complex one with better performance, in this way I would be more focused on neural networks tuning, than on implementation itself. Python was my choice since it permits an easy implementation, thanks to the wide presence of libraries, and for his huge community.

3.4.2 Libraries used

As previously said, python is known for its simplicity granted by the huge collection of libraries. Let's see the libraries I've adopted both for machine learning implementing both for pre-processing step.

Keras and Tensorflow

To implement machine learning my first choice was to use Keras, for the same reason I chose python: easy to implement and a huge community. The problem was that Keras is easy to implement but it is also limited, therefore I decided to use both TensorFlow and Keras and since Keras is included in TensorFlow, it was easy to make it work together. Consequently, I adopted Keras due to its simplicity, and when it was too limiting, I switched to TensorFlow.

Numpy

By default python does not have a concept of array, to solve it is possible to use Numpy, a library that handles multidimensional array, rich in functions

that makes it easy to work with arrays.

Biopython Seq.IO

Biopython provides a module, Bio.SeqIO to read and write sequences from and to a file (any stream) respectively. It supports nearly all file formats available in bioinformatics, included FASTA. It was not essential considering that FASTA is not complex, but I have no requisite in computational weight during preprocessing and this library speed up the developing coding time.

3.4.3 Database extraction and pre processing

The information I need for feedineg neural network are divided into two file, therefore the first part of the algorithm developed is to create the raw dataset. Briefly, I have the FASTA file where are the sample's DNA sequences, and the text file with the results of AST techniques for different drugs. To obtain a functional dataset, first I created a python dictionary, related to only one antibiotic, with the sample name as the key, and a char as a value that can be "S" or "R", where S stands for sensible and R for resistant. After that, I cycle on all the sample's FASTA files, and if they were present inside the dictionary (so if I have the information of their sensibility or resistance) it creates a numpy file, where each of them refers to a sample. In addition, two simple text file were created to memorize which samples are resistant or sensible, in this way, when creating training and test dataset, I can simply work on a list of numbers.

Sequence storage

The sequence, stored in a numpy file, is created by joining each contigs, from the longest to the shortest, in addition I put a gap of 32 non-significative variable between each couple of contigs to correctly maintain the separation between them, in other words, to avoid that the machine learning overlaps two different contigs. Since input data consist of a sparse matrix with values 0 and 1, I used the packbits numpy function to pack the elements of a binary-valued array into bits in a uint8 array. This drastically reduced the amount of space used and speed up the transfer of files from one server to another I use numpy methods packbit packed up the sequences

Training and test dataset creation

Before starting training, I previously split the dataset into training and test, and then I split it again the training set, to create batches. The dimension of the training set is decided by the developer, in this way it may change from one drug to another depending on the dataset. Both training and test sets must be representative of the two categories, so I maintain the same proportion in both sets.

```
#Proportion between sensible and resistant samples
self.sdl = sensnum / resnum
reslen = int(trainsize / (1 + self.sdl))
senslen = trainsize - reslen
#to keep the proportion between the two categories
random.shuffle(senslist)
random.shuffle(reslist)
traininglist = [*senslist[:senslen], *reslist[:reslen]]
self.testlist = [*senslist[senslen:], *reslist[reslen:]]
random.shuffle(traininglist)
random.shuffle(self.testlist)
batchnumber = int(trainsize / batchsize)
if (trainsize / batchsize) % 1 != 0:
    rlist = traininglist[(batchnumber * batchsize):]
    traininglist = traininglist[:(batchnumber) * batchsize]
    self.batchlist = np.array(traininglist).reshape(
   batchnumber, batchsize).tolist()
    self.batchlist.append(rlist)
else:
    self.batchlist = np.array(traininglist).reshape(
   batchnumber, batchsize).tolist()
```

3.4.4 Neural network class

The aim of the neural network class is to speed up the process of training. I want to make it easier to change the hyperparameters that I need to tune. For this reason foor creating a model, two lists are needed, the first contains hyperparameters that I'm tuning, the others are hyperparameters less important to tune, such as epoch of training, batch size, etc. Beyond that, the core of the class is the epoch training function. The main idea was to train the neural network without changing the length of a sequence, so without using a technique such as zero padding. But to train neural networks you need to pass a vector containing all the items inside a batch, or more batches

and if I join the sequences I must make it the same length. For this reason, I created a personalized training loop, in which I feed the network one item at a time, and update the weight after batch size.

Personalized training loop

As we see in the simplified code below, the training cycle is split into two functions, the first looping above the epoch and batch, the second looping on the batch samples.

```
def epochtraining(self, epochnumber):
    for epoch in range(epochnumber):
        batchavgloss = []
        batchavglosso= []
        batchaccuracyavg= []
        for batch in self.batchlist:
            singlebatchavg,singlebatchavgo,accuracy = self.trainingloop(batch)
            batchavgloss.append(singlebatchavg)
            batchavglosso.append(singlebatchavgo)
            batchaccuracyavg.append(accuracy)
        epochavgloss = sum(batchavgloss) / len(batchavgloss)
        epochavglosso = sum(batchavglosso) / len(batchavglosso)
        epochaccuracyavg = sum(batchaccuracyavg) / len(batchaccuracyavg)
        self.epochavglosslist.append(epochavglosso)
        accuracy, SPV, RPV,lossavg=self.testnn(self.trainingtestqty,False) )
        if len(self.epochavglosslist)>9:
            flag=True
            for i in range(9):
                if(abs(self.epochavglosslist[epoch-i]-self.epochavglosslist[epoch-i-1])>0.001):
                    flag=False
            if flag is True:
                break
```

```
def trainingloop(self, batch):
    loss_value = tf.constant(0.)
    mini_batch_losses = []
    mini_batch_olosses = []
    correct = 0
    with tf.GradientTape() as tape:
        for seqref in batch:
```

```
            seqref = int(seqref)
            X_train, y_train = self.loadvalue(seqref)
            logits = self.model(X_train, training=True)
            originalloss = self.loss_fn(y_train, logits)
            result = round(logits.numpy()[0][0])
            if result == int(y_train.numpy()[0][0]):
                correct += 1  # to calculate accuracy
            if y_train.numpy() == 1.:
                loss_value = self.loss_fn(y_train, logits,
    sample_weight=self.firstweight)
            if y_train.numpy() == 0.:
                loss_value = self.loss_fn(y_train, logits,
    sample_weight=self.secondweight)
            mini_batch_losses.append(loss_value)
            mini_batch_olosses.append(originalloss)
        accuracy = correct / len(batch)
        loss_avg = tf.reduce_mean( mini_batch_losses)
        losso_avg = tf.reduce_mean(mini_batch_olosses)
     grads = tape.gradient(loss_avg,self.model.
    trainable_weights)
     self.optimizer.apply_gradients(grads_and_vars=zip(grads,
    self.model.trainable_weights))
     return loss_avg.numpy(), losso_avg.numpy(), accuracy
```

As you can notice I calculate two losses, this is because to overcome the unbalanced dataset problem, I use weighted loss value, but for results comparison, it is more useful to know the original value. I stored it into olosses variable. The training loop function loops over the element of the batch, then using the TensorFlow functions, I update the weight using the optimizer prevoisly defined.

Stop and resume training

Restore a neural network to resume training is not a linear task. The main idea is to save the whole neural network class, in this way all the training information and setting will be restored when loaded. The problem is that the optimizer object is not a pickable one due to lambda function, since that before saving the model I have to delete it. To restore it instead, I save his weight before deleting it and after loading I restore them:

```
opt_weights = np.load(self.savepath+'weights.npy',
    allow_pickle=True)
zero_grads = [tf.zeros_like(w) for w in self.model.
    trainable_variables]
```

```
saved_vars = [tf.identity(w) for w in self.model.
    trainable_variables]
self.optimizer.apply_gradients(zip(zero_grads, self.model.
    trainable_variables))
[x.assign(y) for x, y in zip(self.model.trainable_variables,
    saved_vars)]
self.optimizer.set_weights(opt_weights)
```

3.4.5 Overall using

This is an example of the front end usage of the class:

```
from nn import NeuralNetwork
import pickle
trainingsize=4000
epoch=100
architecture=[2,[6,6],[50,50],100,500]
hyperparameters=[trainingsize, 0.001, 2]

CNN = NeuralNetwork(architecture, hyperparameters, "pooltest"
    ,"meticillina")
CNN.epochtraining(epoch)

savepath=CNN.getsavepath()
pickle_out = open(savepath+"model", 'wb')
pickle.dump(CNN, pickle_out)
pickle_out.close()
```

where, the parameter inside the least means, respectively

- Architecture= layer deep, ksize, filtersize, poolsize, densenumber
- Hyperparameters = trainingsize, learningrate, batchsize

After training finish, the output of each layer and all the results are saved into an output file.

Chapter 4

Experiment results

4.1 Initial experiments

I started the experiment with the architecture represented below:

> Convolution (input) → Max Pooling → Global Max Pooling → Dense → Dropout → Output Dense layer

4.1.1 Experiments to find a direction

The first experiments was almost random, the idea was to explore the space of possibility to find a direction to follow. The first experiment was focused on testing different window size and different number of neurons inside the dense layer.

N	Ksize	FilterN	Pool size	Dense	Results	Epoch
1	3	50	50	100	SPE=0 SEN=1	15
2	6	=	=	=	SPE=0 SEN=1	=
3	12	=	=	=	SPE=0.77 SEN=0.73	=
4	24	=	=	=	SPE=0.79 SEN=0.72	=
5	3	=	=	500	SPE=0 SEN=1	=
6	6	=	=	=	SPE= 0.8 SEN=0.68	=
7	12	=	=	=	SPE=0.78 SEN=0.76	=
8	24	=	=	=	SPE=0.79 SEN=0.78	=

Table 4.1: Results of initial experiments results

As you can see in 4.1, the results were a bit inconclusive. On one hand because, for computational time problems, i train them for only 15 epochs, on the other hand results are almost equals, therefore it was impossible infer something. What might be useful of this first experiment is not represented in that table, in fact in cases N1 and N5, what happened is that the loss function barely change compared to the other, this means that a windows size of 3, may be to small to extract feature. After these tests, it seems obvious that 15 epochs it's too low for neural network to learn something, therefore despite increasing experiment time, I decided to test at least for 30 epoch. The second experiment was focused on finding the optimal number of filters.

N	Ksize	FilterN	Pool size	Dense	Results	Epoch
1	6	2	50	100	SPE=0 SEN=1	30
2	=	5	=	=	SPE=0.52 SEN=0.84	=
3	=	2=	=	500	SPE=0 SEN=1	=
4	=	5	=	=	SPE=0.44 SEN=0.87	=

Table 4.2: Results of the experiment on the correct number of filter.

As we can notice in the first two cases in 4.2 setting the filter number to a very low value (i.e. 2) make the neural network unable to extract feature and therefore to classify the two categories. Instead, a filter number of 5 seems to be enough to let the neural network learn something, but it is already too low to have meaningful results. Since the number of filters corresponds to how many features we extract from DNA sequences, and since they are the number of input neurons of the dense layer, these results are reasonable, because a low number of filters means low information for the neural network training.

4.1.2 Global max pooling before max pooling

Then I performed other experiments, but the these present in table 4.3 was the most useful. Since there is no difference in using a pooling layer ten times higher, I started to think the pooling layer before a global pooling is useless. Obviously, I need some other experiments to validate this thesis. I decided to further increase the number of epoch to 40, and I tested 2 cases with

ksize=6 and ksize=12, to find out if this aspect is maintained by changing others hyperparameters.

N	Ksize	FilterN	Pool size	Dense	Results	Epoch
1	12	50	5	500	SPE=0.74 SEN=0.79	30
2	=	=	50	=	SPE=0.73 SEN=0.81	=

Table 4.3: First evidence of max pooling uselessness before a global pooling

N	Ksize	FilterN	Pool size	Dense	Results	Epoch
1	6	50	50	500	SPE 0.74 SEN 0.76	40
2	=	=	None	=	SPE 0.77 SEN 0.78	=
3	12	=	50	=	SPE=0.81 SEN=0.74	=
4	=	=	None	=	SPE=0.81 SEN=0.70	=

Table 4.4: Validation of max pooling uselessness before global pooling

. The performance difference between the NN with and without a pooling layer (see table 4.4) was really minimal, and also training losses were really similar, since they both fluctuate around the same value and slowly get lower. Naturally, this is not a demonstration that a pooling layer before a global pooling layer is useless, but it's a hint. After making the following consideration, I decided to not use the max-pooling layer:

1. I have one hyperparameter in less to tune, and this highly simplifies each future experiment.

2. Since the max-pooling layer is not a learning layer, but an operation to simplify the input data, I believe it is not fundamental and furthermore senseless putting it before another layer with the same aim.

3. It will be possible to investigate his utility after finding a great solution.

From now on, the architecture used for the next experiment is the following:

> CNN (input) → Global Max Pooling → Dense → Dropout → Output Dense layer

4.1.3 Relation between dense layer and number of filters

Each hyperparameters is related to the other, if one hyperparameter is changed, the optimal value of the others will change. Anyway not all hyperparameters are related in the same way, the change of some of them doesn't produce big changement in the others.

Experiment to find the optimal number of neurons

The feature extracted by the convolutional layer are then elaborated by dense layer, finding the correct number of neurons means to find the number not to high, to avoid overfitting, and not to low, to permit an efficient elaboration.

N	Ksize	FilterN	Dense	Results	Best Epoch
1	6	50	100	SPE=1 SEN=0	4/50
2	=	=	200	SPE=0.77 SEN=0.73	50/50
3	=	=	500	SPE=0.70 SEN=0.80	47/50
4	=	=	1000	SPE=0.63 SEN=0.80	49/50
5	12	=	100	SPE=1 SEN=0	15/50
6	=	=	200	SPE= 0.72 SEN=0.75	49/50
7	=	=	500	SPE=0.74 SEN=0.76	50/50
8	=	=	1000	SPE=0.71 SEN=0.79	48/50

Table 4.5: Results of experiment to find best number of neuron in dense layer.

From results of experiment in table 4.5 it is noticeable that, with a fixed number of 50 filter, the optimal number of neurons in the dense layer could be more than 200 and less than 1000. The N-2-3-7-8 cases seems to produce the best results, therefore i continue their learning for more 100 epoch, to define how much better the accuracy can get with this settings. 4.6

N	Ksize	FilterN	Dense	Results	Best Epoch
1	6	50	500	SPE=0.74 SEN=0.76	76/100
2	=	=	200	SPE=0.78 SEN=0.74	74/100
3	12	=	500	SPE=0.71 SEN=0.80	99/100
4	12	=	1000	SPE=0.80 SEN=0.75	92/100

Table 4.6: Results of experiments to define how much better can get accuracy

In addition, I make two more experiments to see if there will be some noticeable improvements adding a dense layer, to improve the elaboration part. But as you can see in Tab 4.7, if compared with the same experiments without the second dense layer, there are no noticeable differences.

N	Ksize	FilterN	Dense	Results	Best Epoch
1	12	50	500	SPE=0.81 SEN=0.75	50/50
2	=	=	=	SPE=0.73 SEN=0.76	49/50
3	10	=	=	SPE=0.83 SEN=0.52	41/50

Table 4.7: Results of experiments to test the improvement when using more dense layer.

Number of filters and dense number

As said before, the number of filters can be considered the number of inputs of the dense layer, this suggests that these hyperparameters are highly related, in other words, I thought that changing the number of filters will change the optimal value of neurons number. To demonstrate this I did the same experiments already performed previously, using a different number of filters.

N	Ksize	FilterN	Dense	Results	Best Epoch
1	6	20	100	SPE=1 SEN=0	48/50
2	=	=	200	SPE=0.80 SEN=0.69	43/50
3	=	=	500	SPE=0.64 SEN=0.82	49/50
4	=	=	1000	SPE= 0.68 SEN 0.80	50/50

Table 4.8: Results of experiment to find best number of neuron in dense layer.

The results were inconclusive, because as you see in Table 4.8, there are no evident difference, then I performed the same test lowering filter number to 10, and I find out that with a limited number of filters, we must increase the number of neurons to get some results. (Tab 4.9).

N	Ksize	FilterN	Dense	Results	Best Epoch
1	6	10	100	SPE=1 SEN=0	2/50
2	=	=	200	SPE=1 SEN=0	18/50
3	=	=	500	SPE=0.77 SEN=0.75	50/50
4	=	=	1000	SPE= 0.71 SEN 0.77	45/50

Table 4.9: Results of others experiments to find best number of neuron in dense layer.

Analysis of weight learned In general, I did not get the expected results, so I try to analyze the weight of the convolution filter, to find out some useful information. In particular, I wanted to find out how many negative values each filter had learned, in fact, considering that the activation function is a Relu, a negative value means a null value. This means that the filter is useless if all the values of a filter are negative, or 0. The result of this analysis is represented in Tab 4.10, where is used the model with 500 Dense, k=6, and filter = 20 as a specimen. Then I performed the same test on a model with k equal to 8 - 10 -12 thinking that a higher number of filters might intensify this aspect, but in all cases, the number of negative values is far from being all negative. It might be because analyzing just some cases is truly limiting, or because it is a wrong path. Therefore considering also the little time available, I decided to not further investigate this aspect.

N	Negative	<0.001	<0.01	Others
1	19	0	1	4
2	20	0	1	3
3	22	0	1	1
4	20	0	0	4
5	22	0	0	2
6	22	0	0	2
7	19	0	2	3
8	18	1	0	5
9	22	0	2	0
10	18	0	1	5
11	21	0	0	3
12	18	0	0	6
13	16	0	5	3
14	23	0	0	1
15	16	0	2	6
16	18	1	0	5
17	18	0	2	4
18	23	0	1	0
19	18	0	0	6
20	22	0	0	2

Table 4.10: Table representing the weight of filters of model with 500 Dense, k=6 and filter = 20.

4.1.4 Kernel size

Window size is a fundamental and maybe the most important parameter, Therefore it was an error to test it later. Increasing the windows size, will increase the dimension of feature we are analyzing, in general there is not a rule, it is not predictable if using a bigger or smaller windows size are better. In general it depends on the kind of data. From these experiment I doesn't seem any hint on what could be the best windows size (Table 4.11)

N	Ksize	FilterN	Dense	Results	Best Epoch
1	4	50	200	SPE=0.94 SEN=0.06	49/50
2	=	=	500	SPE=0.73 SEN=0.76	49/50
3	8	=	200	SPE=0.69 SEN=0.79	49/50
4	=	=	500	SPE=0.80 SEN=0.66	36/50
5	10	=	200	SPE=0.66 SEN=0.79	42/50
6	=	=	500	SPE=0.71 SEN=0.81	47/50
7	16	20	500	SPE=0.68 SEN=0.82	35/50
8	32	20	500	SPE=0.69 SEN=0.81	35/50
9	32	50	500	SPE=0.67 SEN=0.82	47/50

Table 4.11: Results of experiment to find windows size optimal value.

At this point, I realize that the model was too simple to make neural networks to perform good. Therefore I decided to use a deeper architecture, adding another convolution layer. In this way, the model would be able to extract more features.

4.1.5 Overall conclusion

Here is a summary of the results of the initial experiments:

1. A windows size of 3 or less, is too small for feature extraction
2. Using a low value of filter, such as 2 or 10 doesn't result in a good performance. Less than 2 features make neural networks unable to train.
3. Using a max pooling layer in series of a global pooling layer is useless
4. In the dense layer, 100 neurons are a low number, more than 200 seems to have a better effect
5. To obtain better performance, it is necessary a deeper feature extraction section

4.2 A deeper architecture

As said before, using only one convolutional layer doesn't result in sufficient feature extraction. Since that, I decided to insert another convolutional layer, and therefore I reintroduce max-pooling after the first convolutional layer, which was eliminated because useless before the global one. This is the architecture tested:

> Convolution (input) → Max Pooling → Convolution → Global Max Pooling → Dense → Dropout → Output Dense layer

In this section are collected the architecture with which I obtained the best results, for each experiment there will be the graph relative to loss and accuracy, the confusion matrix, and the experiment conclusions. The first section contains all the experiments performed on Methicillin, then I used the best choice on the Inducible Clindamicine and Tetracycline.

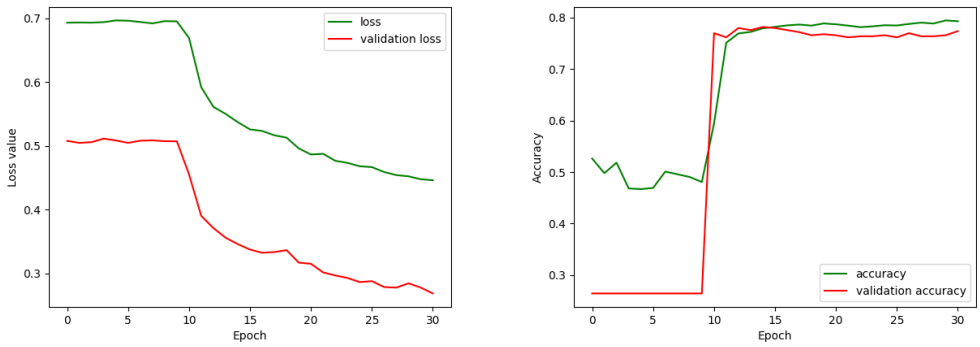
4.2.1 Metycillin

These experiments aim to find out the size of the kernel window resulting in optimal feature extraction.

Windows size 6 - 6

> Convolution (Windows size = 6, Filter size = 50) → Max Pooling(100) → Convolution (Windows size = 6, Filter size = 50) → Global Max Pooling → Dense Layer (500) → Dropout (0.5) → Output Dense layer

Training results The loss graph shows a standard trend during training both for loss and validation loss. Similar considerations can be done for accuracy and validation accuracy (Figure 4.1. There is no evident overfitting problem since over the epochs the validation accuracy increase or remain constant and the loss decrease or remain constant.



(a) (b)

Figure 4.1: Plot of loss (a) and accuracy (b) on both validation and training test with windows size 6 - 6

Test results With an overall accuracy of 0.8, the results are a better than before, but not good. In addition, considering that methicillin dataset was unbalanced, is better to evaluate resistant and sensible prediction accuracy individually: from Recall we notice that the model predict correctly only 76% from the total of resistant samples. Beside there are confusion matrix and parameters derived from it.

- Accuracy=0.799
- Precision=0.957
- Specificity=0.898
- Recall=0.766

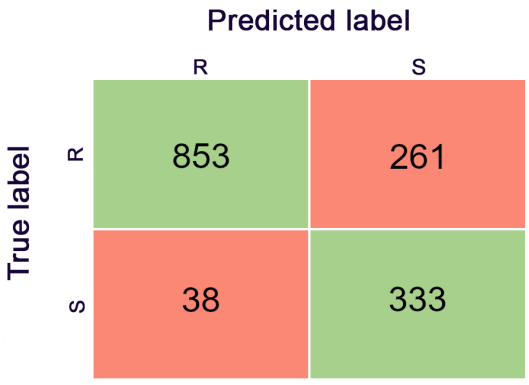
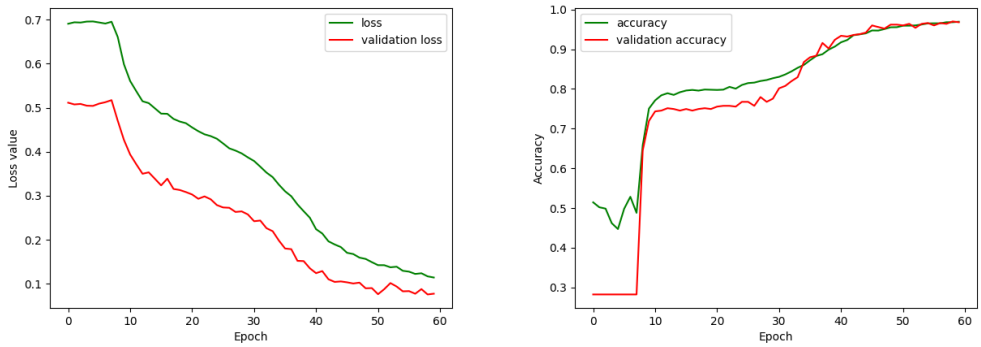


Figure 4.2: Confusion Matrix. Windows size 6 - 6

Windows size 8 - 8

> Convolution (Windows size = 8, Filter size = 50) → Max Pooling(100) → Convolution (Windows size = 8, Filter size = 50) → Global Max Pooling → Dense Layer (500) → Dropout (0.5) → Output Dense layer

Training results Almost like the previous case, the loss graph shows a normal trend during training both for loss and validation loss; similar considerations can be done for accuracy and validation accuracy (Figure 4.3. Also in this case, there is no evident overfitting problem since over the epochs the validation accuracy increase or remain constant and the loss decrease or remain constant.



(a) (b)

Figure 4.3: Plot of loss (a) and accuracy (b) on both validation and training test with windows size 8 - 8

Test results With an overall accuracy of 0.96, and a better value also for Specificity and Recall parameters, this model might be considered a great one, better than the previous case which use two convolutional layers with size six. Besides, there are confusion matrix and parameters derived from it.

- Accuracy=0.962
- Precision=0.988
- Specificity=0.967
- Recall=0.96

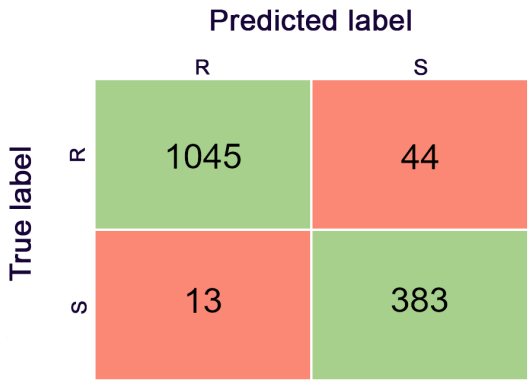
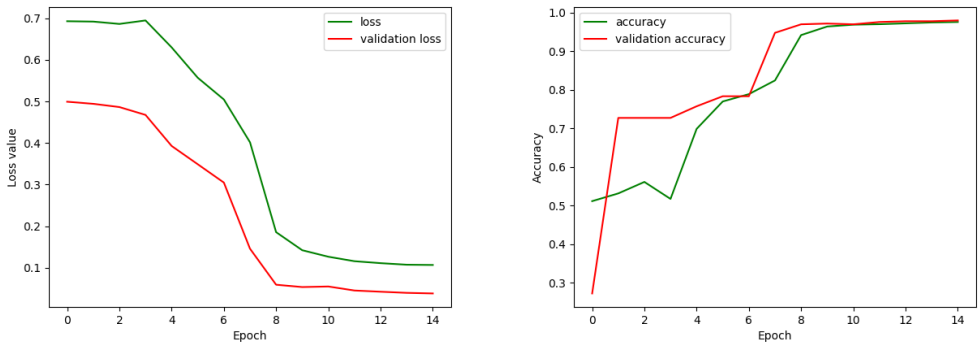


Figure 4.4: Confusion matrix using windows size 8-8

Windows size 32 - 6

Convolution (Windows size = 32, Filter size = 10) → Max Pooling(100) → Convolution (Windows size = 6, Filter size = 50) → Global Max Pooling → Dense Layer (500) → Dropout (0.5) → Output Dense layer

Training results Such as the previous cases, the loss graph shows a standard trend during training both for loss and validation loss, both for accuracy and validation accuracy (Figure 4.5). What is noticeable here is the speed of learning: the loss value is lower and accuracy is better than previously tested models, nevertheless the training epochs are about half of them.



(a) (b)

Figure 4.5: Plot of loss (a) and accuracy (b) on both validation and training test with windows size 32 - 6

Test results With an overall accuracy of 0.97, and values such as 0.96 of Specificity and 0.98 Recall, I consider this model my best result. Besides, there are confusion matrix and parameters derived from it.

- Accuracy=0.975
- Precision=0.987
- Specificity=0.965
- Recall=0.979

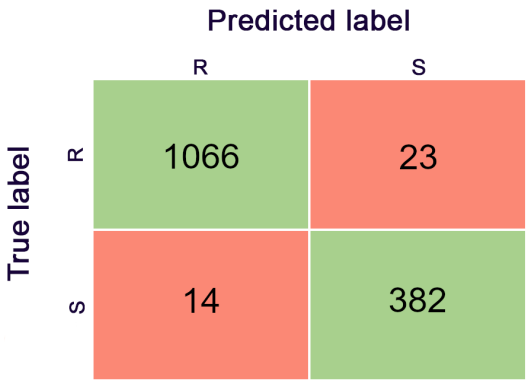
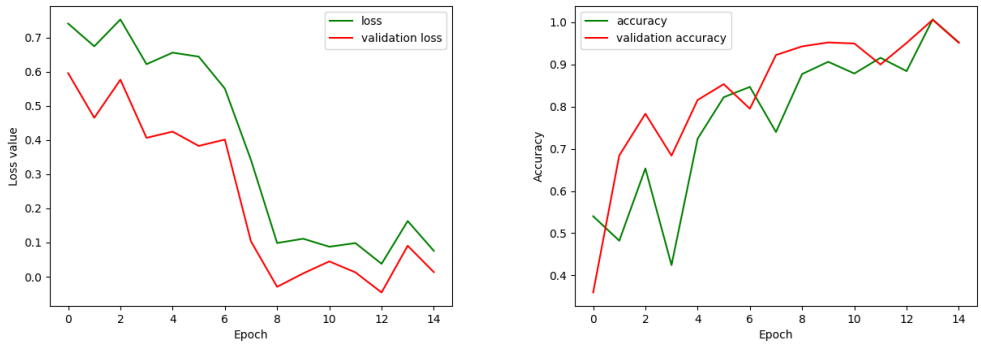


Figure 4.6: Confusion matrix using windows size 32- 6

K fold validation with the windows size 32 - 6

Since the usage of two windows size of 32 and 6 in series returns the best performance in a half the number of epoch, I decide to validate it using a 5-fold cross-validation. All the following results and graph must be considered as the average of the 5 sub experiments of k-fold validation.

Training results The loss graph trend during training both for loss and validation loss, both for accuracy and validation accuracy (Figure 4.7) are similar to graphs previously obtained. The slight fluctuations is probably because we are considering the average of five sub experiments.



(a) (b)

Figure 4.7: Plot of loss (a) and accuracy (b) on both validation and training test. 5-fold validation on methicillin with windows size 32 - 6

Test results The average value of the test confirm that the architecture selected is able to create a great model to predict methicillin resistance and sensibility.

- Accuracy=0.974
- Precision=0.992
- Specificity=0.979
- Recall=0.973

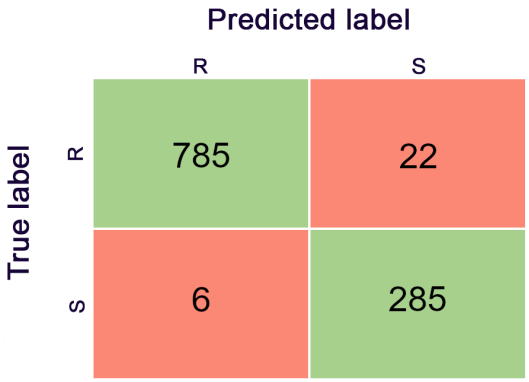


Figure 4.8: Average of confusion matrix for validation. windows size 32- 6

4.2.2 Inducible Clindamycin

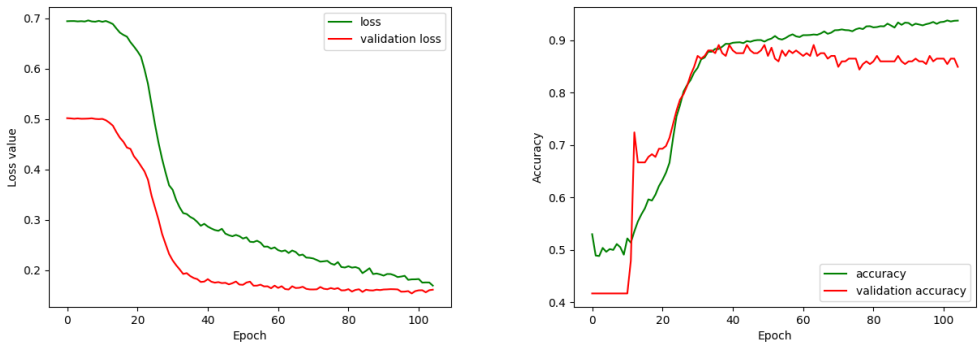
I tested the following architecture on clindamycin:

> Convolution (Windows size = 32, Filter size = 10) → Max Pooling(100) → Convolution (Windows size = 6, Filter size = 50) → Global Max Pooling → Dense Layer (500) → Dropout (0.5) → Output Dense layer

K fold validation with windows size 32

The model results are not great such in methicillin case, but it is a starting point for future works. Therefore, considering the potential, I decided to validate it.

Training results After 40 epochs, it is noticeable that meanwhile the loss slightly decreases, the validation loss remains constant. It is possible that it is the beginning of overfitting, or that it is due to the small number of samples in the validation test set. (Figure 4.9)



(a) (b)

Figure 4.9: Plot of loss (a) and accuracy (b) on both validation and training test. 5-fold validation on inducible clindamycin with windows size 32 - 6

Test results The 5-fold validation confirms the results I got, but as said before, 86% of accuracy is too low to be considered a great result, but it is a good starting point for future works.

- Accuracy=0.861
- Precision=0.952
- Specificity=0.881
- Recall=0.853

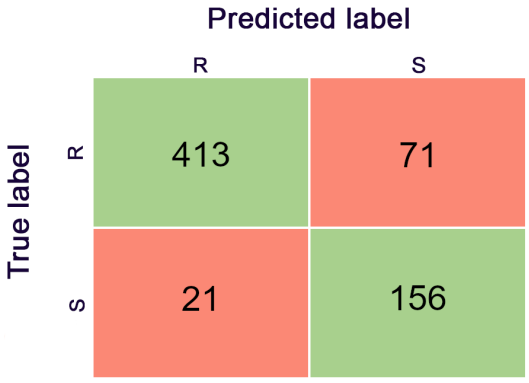


Figure 4.10: Average of confusion matrix for validation. windows size 32- 6

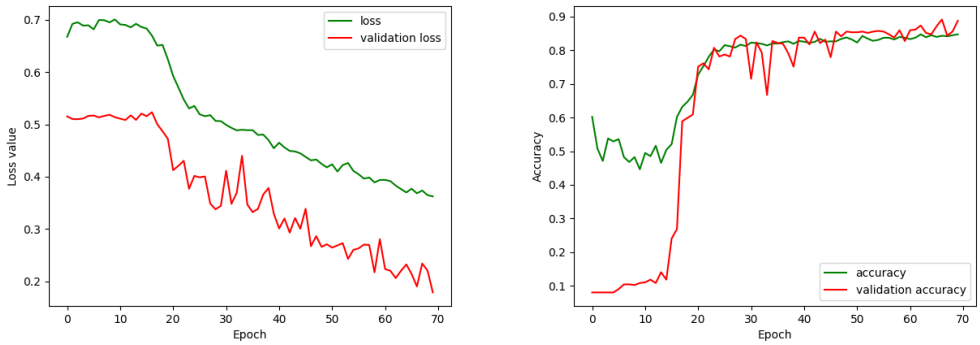
4.2.3 Tetracicline

I tested the following architecture on tetracicline:

> Convolution (Windows size = 32, Filter size = 10) → Max Pooling(100) → Convolution (Windows size = 6, Filter size = 50) → Global Max Pooling → Dense Layer (500) → Output Dense layer

Testing windows size 32 and 6

Training results The loss graph trend during training both for loss and validation loss, both for accuracy and validation accuracy (Figure 4.11) don't show particular trends.



(a) (b)

Figure 4.11: Plot of loss (a) and accuracy (b) on both validation and training test on tetracicline with windows size 32 - 6

Test results The model results in a high accuracy value, unfortunately, the tetracycline dataset is highly unbalanced and this means that accuracy is not a good parameter to evaluate performance. In fact, if we observe for example Recall, we notice that it is able to predict correctly only a half of the total resistant samples. Besides, there are confusion matrix and parameters derived from it.

- Accuracy=0.906
- Precision=0.464
- Specificity=0.944
- Recall=0.506

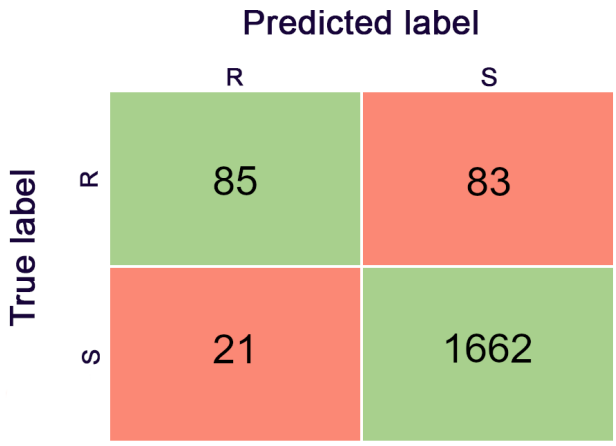


Figure 4.12: Confusion matrix using windows size 32- 6

Chapter 5

Conclusion

The approach adopted was focused on obtaining the best results using the most elementary architecture, therefore I begin with a very simple model making it more complex only if needed. Below are listed the step procedure I followed:

1. Data preprocessing: Raw data needs to be manipulated to feed neural networks and to enhance performance.

2. Implementation: It includes the language selection and coding step.

3. Model tuning: experiment to find the best value for hyperparameters.

4. Model validation: It consists in the validation of the model selected

My thesis aims to use CNN, a deep learning algorithm, to predict if a particular bacteria is sensible or resistant to specific drugs, by analyzing its DNA. Deep learning algorithms are particularly suitable for DNA classification since they automatically learn features: on one size this permits a neural network to learn the best feature during training on the other it permits a neural network to classify if a sample is resistant, despite its resistance causes is unknown. In detail, I adopted a convolutional neural network (CNN), which is a deep learning algorithm that uses convolutional operation for feature extraction. The main components of a CNN are the convolutional layer, which consists of a collection of digital filters used to perform the convolution on the input data, and the pooling layer used as a dimensionality reduction layer. A high number of decisions must be made when approaching a problem with machine learning, and they are not limited only to implementation.

Below I listed the most important consideration and decision I took before this step:

- **A machine learning algorithm can predict the results only if the input data contains the information needed.** This means that to make the model predict resistance or sensibility, this information must be contained in a DNA sequence. Since many determinants of an antibiotic-resistant phenotype are coded from the microbial DNA content, including mobile genes, single-nucleotide mutations, or wider genomic alterations, a neural network could predict the resistance.

- **DNA sequences must be convert from characters into a numerical value** to make the information compatible with the machine learning model. There are different ways to represent information, and it is essential to find the best one because depending on this you suggest the network something about data. I adopted One hot encoding nevertheless it has a heavier computational weight, than sequential encoding, because it suited better categorical data, avoiding implying a numerical dependency. One hot encoding produces a vector with a length equal to the number of categories where each one is represented by a 1 and 0 for the other values. For example, it is possible to use [1 0 0 0] to represent the A in DNA sequences.

- A common approach when dealing with CNN is to use a dense layer after the feature extraction section. The problem is that the **fully connected layer needs to know a priori the size of the sequence**. Normally this is resolved using methods such as zero padding, which is adding a non-significative value to make all sequences the same length. The problem is that a length difference between sequence of about $10\hat{3}$-$10\hat{4}$ increase significantly computational load. Therefore I adopted a global max-pooling layer after the feature extraction section, that has one output for each filter used, which is a fixed value.

- **It is essential to check if the dataset is unbalanced** because, it is important for training purposes that each group is representative, and for a highly unbalanced dataset this aspect might not be verified. Since each dataset I worked with is unbalanced, I adopted a weighted cross-entropy function to overcome this problem, since it is the most suitable option, easy to implement, tunable, with no dataset manipulation.

To implement machine learning algorithms I chose Python since is very easy to implement thanks to the many libraries dedicated to machine learning implementation. I used both Keras and TensorFlow libraries: the last one give me more freedom, the first one make it easier and faster to implement some procedures. The software I create is divided into three main parts:

- **Raw Dataset Creation** In order to learn, the machine learning algorithm needs both sequences to study and feedback on his prediction for improving. The DNA sequences are stored in FASTA file format divided into contings, instead, the category of each sequence (i.e. resistant or sensible) was stored in a text file, containing AST analysis performed on a specific sample using specific antibiotics. Therefore a section of my software consists in creating a raw dataset from these files, obtaining for each antibiotic all the sequences tested and the associated category. Each sequence is stored in a NumPy file created by joining each contigs, from the longest to the shortest. In addition, I put a gap of 32 non-significative variables between each couple of contigs to avoid that the machine learning overlapping two different contigs during analysis. To store the sensibility or resistance to a specific dataset I use two text files, one for sensible and one for resistant samples, containing just the reference(a unique number) of the sample. In this way for training, validation and test set creation and manipulation, I operate with a simple list of numbers.

- **Functional dataset creation** To properly train a model, a first approach might be to use the entire dataset, because the more the sample, the more might be the accuracy. But it is not a correct choice because, without division, it is impossible to know if the NN is encountering problems such as overfitting. To overcome this, before feeding the neural network, the software divides the dataset into two parts, one used to train and the other to test.

- **Training section** Keras library makes it easy the training step, but obviously, it comes with limitations: it doesn't permit to give one input at a time without adapting all items inside the path to the same dimension, increasing the computational load. Therefore I used TensorFlow to create a personalized training cycle, and Keras to implement other minor functionality. Another feature I created permits to stop and resume the training, it may seem banal but it is not allowed natively.

Pausing the training permits me to decide if continue or stop training without deciding a priory the number of epoch and without basing the choice only on callback algorithms.

Each model is characterized by a huge amount of hyperparameters, trying to tune all of them at the same time is challenging, therefore I preferred to fix some hyperparameters to value found in literature, to reduce the space of possibilities, and focalize the training on the following: convolutional windows and filters number, neurons number in a dense layer and pooling quantity. To evaluate the experiments I monitored the loss function and accuracy of both training and validation sets. Then after training, I evaluate the confusion matrix and parameter derived from it. The first part of the experiments aims to find out if it is possible to predict resistance using only one convolutional layer. From these tests I obtain the following results:

1. A windows size of 3 or less, is too small for feature extraction
2. Using a low value of filter, such as 2 or 10 results in bad prediction performance. Less than 2 features make neural networks unable to train.
3. Using a max-pooling layer in series of a global pooling layer is useless.
4. To obtain better performance, it is necessary a deeper feature extraction section

Since it is necessary to adopt a deeper feature extraction, I decided to insert another convolutional layer, testing the following architecture:

Convolution (input) $\rightarrow$ Max Pooling $\rightarrow$ Convolution $\rightarrow$ Global Max Pooling $\rightarrow$ Dense $\rightarrow$ Dropout$\rightarrow$ Output Dense layer

The second part of my experiment was focused on finding an optimal window size, more than on the other hyperparameters, therefore I fix them and then I start the following tests on the methicillin sample:

1. Two convolutional layers with both a window's size of 6.
2. Two convolutional layers with both a window's size of 8.
3. The first convolutional layer with a window's size of 32 and the second of 6.

Each experiment returns better results compared with models with just one convolutional layer. In particular, the third case, has the best performance, with an overall accuracy of 0.97 an Specificity(precision over sensible samples) of 0.965, and an Recall(precision over resistant samples) of 0.97; therefore I validate this result using 5-fold cross-validation. Since the third configuration returns the best model performance for methicillin, I tested it also on inducible clindamycin and tetracycline. Inducible Clindamicine accuracy was about 86%, this value is too low to be considered a great result, but it is a good starting point for future works. Instead, for tetracycline, the architecture doesn't work, probably because of the highly unbalanced dataset.

Chapter 6

Future perspectives

When dealing with machine learning, there are a lot of possible solutions considering tuning, implementation, and other practical aspects. Many decisions I made were forced by computational and timing problems, for this reason in future works many aspects don't considered or superficially explored will be deeply examined:

- **Test deeper model**: I demonstrated that using only one convolutional layer doesn't permit a sufficient feature extraction, and also that using two convolutional layers guarantees great results. Despite all, adding more convolutional layers might be speed up the training process (in terms of number of epochs) and enhance the accuracy. The same considerations are valid for the feature analysis section: using a higher number of dense layers might enhance the overal neural network performance.

- **Explore hyperparameters with automatic tool tuning**: manual tuning has its advantages, but it doesn't guarantee the finding of the best solutions, especially in lack of experience. Using automatic tools may find a better tuning for the architecture.

- **Tune all hyperparameters**: In this thesis, I was focused principally on hyperparameters relative to convolutional layers. Batch number, learning rate, and many others were fixed using the information found in scientific literature. Tuning also these values might enhance model performance.

- **Accurate filter analysis**: Filters obtained by analyzing DNA sequences, might be considered as motifs that are nucleotide sequence patterns assumed to be related to the biological function of the macromolecule.Some studies suggested that first layer convolutional filter learn representations of sequence motifs,while deeper learn combinations of these motifs [3] [24]. Learning whole motifs representations by first layer is not indicative of the performance of a model, but demonstrating that first layer filters have learned biologically representation might be a possible approach to validate a model.

- **Enhance early stopping algorithms**: The early stopping algorithms I implemented, compare only the changing of loss function across the epochs, to stop training when the networks don't train anymore. Better algorithms might be used such as the comparing of the validation loss, to avoid overfitting or a combination of different parameters.

- **Improve the handling of unbalanced datasets**: to overcome unbalanced datasets problem, I adopted the weighted cross-entropy function. Actually, some alternatives might enhance the performance, in particular for Tetracycline, which has an highly unbalanced dataset.

- **Understand genes mechanism learned by the neural network**: It would be really interesting trying to extract the feature learned by the neural network, to understand new genes mechanism.

- **Transfer-learning**: When a neural network is trained, it changes values such as weight, bias, and also the optimizer adapting to the data. These weights can be extracted and then transferred to any other neural network: instead of training from scratch, we "transfer" the learned features. This might enhance the overall accuracy.

Bibliography

[1] Center for animal health and food safety.

[2] Centers for disease control and prevention.

[3] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831–838, July 2015.

[4] Sagar Aryal. Online microbiology notes, Nov 2021.

[5] John G. Bartlett, David N. Gilbert, and Brad Spellberg. Seven Ways to Preserve the Miracle of Antibiotics. *Clinical Infectious Diseases*, 56(10):1445–1450, 02 2013.

[6] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures, 2012.

[7] Nandita Bhaskhar. Interpreting logits: Sigmoid vs softmax. *Blog: Roots of my Equation (web.stanford.edu/ nanbhas/blog/)*, 2020.

[8] Manish Boolchandani, Alaric W. D'Souza, and Gautam Dantas. Sequencing-based methods and resources to study antimicrobial resistance. March 2019.

[9] Vanessa M D'Costa, Emma Griffiths, and Gerard D Wright. Expanding the soil antibiotic resistome: exploring environmental diversity. *Current Opinion in Microbiology*, 10(5):481–489, 2007. Antimicrobials/Genomics.

[10] Ema Europe. Antimicrobial resistance.

[11] Max Kuhn. *Applied Predictive Modeling*. 2013.

[12] Lakna. How does illumina sequencing work, Apr 2018.

[13] Giosué Lo Bosco and Mattia Antonino Di Gangi. Deep learning architectures for DNA sequence classification. In *Fuzzy Logic and Soft Computing Applications*, Lecture notes in computer science, pages 162–171. Springer International Publishing, Cham, 2017.

[14] Alejandro Lopez-Rincon, Alberto Tonda, Lucero Mendoza-Maldonado, Daphne G. J. C. Mulders, Richard Molenkamp, Carmina A. Perez-Romero, Eric Claassen, Johan Garssen, and Aletta D. Kraneveld. Classification and specific primer design for accurate detection of SARS-CoV-2 using deep learning. *Scientific Reports*, 11(1), jan 2021.

[15] David Mack. How to pick the best learning rate for your machine learning project, Apr 2018.

[16] Dominic Masters and C. Luschi. Revisiting small batch training for deep neural networks. *ArXiv*, abs/1804.07612, 2018.

[17] Seonwoo Min, Byunghan Lee, and Sungroh Yoon. Deep learning in bioinformatics. *Briefings in Bioinformatics*, page bbw068, July 2016.

[18] Ngoc Giang Nguyen, Vu Anh Tran, Duc Luu Ngo, Dau Phan, Favorisen Rosyking Lumbanraja, Mohammad Reza Faisal, Bahriddin Abapihi, Mamoru Kubo, and Kenji Satou. DNA sequence classification by convolutional neural network. 09(05):280–286, 2016.

[19] Kedar Potdar, Taher Pardawala, and Chinmay Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *International Journal of Computer Applications*, 175:7–9, 10 2017.

[20] Francesca Prestinaci, Patrizio Pezzotti, and Annalisa Pantosti. Antimicrobial resistance: a global multifaceted phenomenon. 109(7):309–318, September 2015.

[21] Riccardo Rizzo, Antonino Fiannaca, Massimo La Rosa, and Alfonso Urso. Classification experiments of DNA sequences by using a deep neural network and chaos game representation. In *Proceedings of the 17th International Conference on Computer Systems and Technologies 2016*, New York, NY, USA, June 2016. ACM.

[22] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

[23] Mitsuko Seki, Hongjo Choi, Kyungjong Kim, Jake Whang, Joohon Sung, and Satoshi Mitarai. Tuberculosis: A persistent unpleasant neighbour of humans. *Journal of Infection and Public Health*, 14(4):508–513, 2021.

[24] Jian Zhou and Olga G Troyanskaya. Predicting effects of noncoding variants with deep learning–based sequence model. *Nature Methods*, 12(10):931–934, August 2015.