Department of Precision and Microsystems Engineering

Identification of Bacteria with Artifitial Intelligence

S. Mendoza Silva

Report No : 2023.075
Coach : Aleksandre Japaridze, Irek Rosłoń
Professor : Farbod Alijani
Specialisation : Engineering Dynamics
Type of report : Master Thesis
Date : 30th August, 2023

TU Delft Delft University of Technology

Challenge the future

Delft University of Technology

Department of Precision and Microsystems Engineering

Idenfitication of Bacteria with Artifitial Intelligence

Thesis Document
Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Mechanical Engineering

Author:
Santiago Mendoza Silva
5432049

August 2023

TU Delft

Contents

1 Preface

I started this master thesis with a genuine new interest in high-tech engineering and the role that dynamics of mechanical systems could play in this area of mechanical engineering. When I was introduced to the idea of being able to implement the knowledge I acquired during my master studies in a project with the potential of creating medical breakthroughs, I felt an instant connection, as I realized it would be the perfect opportunity to implement engineering knowledge to create a positive social impact. With this idea in mind and although I started with basic knowledge about machine learning, embarking on this project was like planting a seed in fertile ground, as my enthusiasm for learning and implementing new methodologies was constantly growing since the very first day.

I wish to express my gratitude to Professor Farbod Alijani, my supervisors Aleksandre Japaridze, Irek Rosłoń, and their entire team for their constant support and invaluable feedback throughout the duration of this project. Their guidance was instrumental, and I am deeply appreciative of the exceptional opportunity they afforded me to contribute to this initiative. I am also grateful to my parents, family, and all my loved ones. Their unwavering faith and continuous encouragement, especially in moments of doubt, played a crucial role in the successful completion of this project. This thesis is dedicated to them; it stands as a testament to their support and is as much their achievement as it is mine.

In this master's thesis, you will find a comprehensive account of my discoveries throughout this profound journey. It details the complexities of the methodologies employed, the challenges encountered, and the strategies devised to address them. More than anything, as you pursue into the pages that follow, you will witness the culmination of unwavering effort and dedication over the past year, made possible by the collective endeavors of everyone involved. It is my earnest hope that this research contributes significantly to the field and serves as a catalyst for further exploration and innovation.

2

Abstract

Bacterial identification is crucial for addressing infectious diseases and enabling effective treatment strategies. Conventional bacteria identification methods like MALDI-TOF, while efficient, lack the capability for screening the effectiveness of antibiotics. On the other hand, existing antimicrobial resistance (AMR) tests, despite being reliable, suffer from time inefficiency and lack concurrent identification capabilities. In response to these challenges, the present study employs the recent advancements in single-cell nanomotion detection using graphene drums to address these limitations. We integrate nanomotion detection with Machine Learning (ML) algorithms which enables us to simultaneously identify bacteria phenotype and their resistance to antibiotics. Bacterial time signals are transformed into time-frequency spectrograms, which then serve as inputs for machine learning algorithms. Through pattern recognition, these algorithms identify features within the images, facilitating the development of robust classification models. Utilizing single-cell nanomotion signals, differentiation is achieved between the species *Escherichia coli*, *Staphylococcus aureus*, and *Klebsiella pneumoniae*, as well as in detecting antibiotic-resistant and -susceptible strains, achieving an accuracy of 98.57% in the latter. This research marks the first instance of ML integrated with nanomotion detection for bacterial species identification and antibiotic susceptibility testing. It provides a basis for advanced diagnostic tools, expediting the provision of vital data regarding bacterial identification and antibiotic susceptibility, contributing significantly to medical diagnostics.

3

Introduction

Accurate bacterial identification is crucial in clinical microbiology to direct suitable treatments and steer infection control in healthcare [1]. Efficient diagnosis is pivotal to reduce morbidity, mortality, and associated costs. While accurate identification can prevent unnecessary antibiotic use, the prevalent culture-based methods prove laborious, often requiring over two days to pinpoint pathogens and longer to generate antibiotic susceptibility profiles [2]. This latency often prompts physicians to prescribe broad-spectrum antibiotics, in conjunction with these culture-based methods that demand 48 to 96 hours until a correct prescription is found [3].

Current identification methodologies span from traditional techniques such as Polymerase Chain Reaction (PCR) with gene sequencing or Enzyme-Linked Immunosorbent Assay (ELISA) to more advanced approaches like Matrix-Assisted Laser Desorption/Ionization Time-of-Flight (MALDI-TOF) mass spectrometry. MALDI-TOF has notably revolutionized both the identification and characterization of bacterial species and the understanding of their antibiotic resistance. By measuring the mass-to-charge ratio of ionized molecules in a sample, MALDI-TOF not only enables the detection of unique bacterial signatures but also recently facilitates the identification of enzymes that bacteria produce to inactivate antibiotic molecules. However, it is sensitive to experimental conditions and its spectra can vary due to growth conditions and extraction methods, highlighting the need for alternative procedures [1, 4–6]. Moreover, while specific MALDI-TOF variations can suggest antibiotic resistance, they fail to provide direct guidance on appropriate treatment methodologies.

In contemporary medical microbiology, Antibiotic Susceptibility Testing (AST) is critical for evaluating how bacterial strains respond to various antibiotics. Methodologies like the Kirby Bauer Disk Diffraction Method, Unbiased High-Throughput Sequencing (UHTS) or Bacterial Capture Sequencing (BacCapSeq), are effective procedures that have been developed to evaluate bacterial susceptibility to antibiotics, but lack specificity in identifying bacterial species and display variable efficacy based on the bacterial concentrations in the samples. While methods such as those advocated by Cockerill et al. [7] and Lee et al. [8] necessitate substantial sample volumes for optimal sensitivity, genomic strategies like UHTS and BacCapSeq offer comprehensive genetic assessments but are resource-intensive [2, 9, 10]. Importantly, no current method offers rapid, comprehensive insights spanning phylogeny, strain identification, and antimicrobial resistance [2], highlighting the need for advanced techniques adept at both rapid bacterial identification and antibiotic susceptibility assessment.

Nanomotion detection is emerging as a potent technique for antibiotic susceptibility testing [11]. This approach, spotlighting tiny oscillations triggered by living microorganisms, has seen increased attention due to advancements in sensitivity [12]. Although nanomotion spectroscopy exhibits considerable promise, its potential to simultaneously conduct bacterial identification and AST remains uncertain. This is because existing nanomotion signals obtained from AFM cantilevers typically represent averaged motions emanating from populations of 100-1000 bacteria [13–15]. Such aggregate signals make precise bacterial identification challenging, as indi-

vidual bacterial vibrational profiles are crucial for accurate species differentiation [16]. Recognizing these challenges, recent advancements in nanomotion measurement have culminated in the use of graphene drums, further enhancing detection sensitivity [17]. This evolution in the field potentially provides a renewed opportunity to explore the capability of nanomotion spectroscopy in detailed bacterial identification and antibiotic susceptibility testing.

Addressing this concern, the present investigation explores the viability of utilizing nanomotion drum detection signals in bacterial identification and classification. In tandem, the study probes the efficacy of Machine Learning algorithms such as Convolutional Neural Networks (CNNs) and Support Vector Machines (SVM), in discriminating between different bacterial types, including *Escherichia coli*, *Staphylococcus aureus*, and *Klebsiella pneumoniae*. In addition, these algorithms were adeptly selected to distinguish not only between bacterial species but also between antibiotic-resistant and susceptible strains. Within this framework, a total of 456 data measurements were utilized for the identification study case, and a total of 347 data measurements were employed for the antibiotic susceptibility study case. Bacterial nanomotion signals were transformed into time-frequency spectrograms, serving as inputs for these advanced machine learning algorithms. With an emphasis on pattern recognition, the algorithms were designed to identify intricate features within the spectrogram images, culminating in the construction of robust classification models. In addition, the performance of these models was critically evaluated using established metrics in machine learning theory such as accuracy, sensitivity, specificity, and area under the Receiver Operating Characteristic (ROC) curve, while considering computational efficiency. This study provides the first evidence of integration of ML algorithms with single-cell detection, targeting bacterial species identification and antibiotic susceptibility testing. By assessing the viability of this approach for bacterial identification and antibiotic testing, the investigation provides crucial insights, setting the stage for subsequent research in this area.

4

Paper: Identification of Bacteria with AI

Nanomotion-Based Machine Learning for Simultaneous Bacterial Identification and Antibiotic Susceptibility Testing

Nanomotion-Based Machine Learning for Simultaneous Bacterial Identification and Antibiotic Susceptibility Testing

Santiago Mendoza Silva, Aleksandre Japaridze, Irek Rosłoń, Farbod Alijani

Precision and Microsystems Engineering, Delft University of Technology
Mekelweg 2, Delft, 2628LX, The Netherlands

Abstract

Bacterial identification is crucial for addressing infectious diseases and enabling effective treatment strategies. Conventional bacteria identification methods like MALDI-TOF, while efficient, lack the capability for screening the effectiveness of antibiotics. On the other hand, existing antimicrobial resistance (AMR) tests, despite being reliable, suffer from time inefficiency and lack concurrent identification capabilities. In response to these challenges, the present study employs the recent advancements in single-cell nanomotion detection using graphene drums to address these limitations. We integrate nanomotion detection with Machine Learning (ML) algorithms which enables us to simultaneously identify bacteria phenotype and their resistance to antibiotics. Bacterial time signals are transformed into time-frequency spectrograms, which then serve as inputs for machine learning algorithms. Through pattern recognition, these algorithms identify features within the images, facilitating the development of robust classification models. Utilizing single-cell nanomotion signals, differentiation is achieved between the species *Escherichia coli*, *Staphylococcus aureus*, and *Klebsiella pneumoniae*, as well as in detecting antibiotic-resistant and -susceptible strains, achieving an accuracy of 98.57% in the latter. This research marks the first instance of ML integrated with nanomotion detection for bacterial species identification and antibiotic susceptibility testing. It provides a basis for advanced diagnostic tools, expediting the provision of vital data regarding bacterial identification and antibiotic susceptibility, contributing significantly to medical diagnostics.

4.1. Introduction

Accurate bacterial identification is crucial in clinical microbiology to direct suitable treatments and steer infection control in healthcare [1]. Efficient diagnosis is pivotal to reduce morbidity, mortality, and associated costs. While accurate identification can prevent unnecessary antibiotic use, the prevalent culture-based methods prove laborious, often requiring over two days to pinpoint pathogens and longer to generate antibiotic susceptibility profiles [2]. This latency often prompts physicians to prescribe broad-spectrum antibiotics, in conjunction with these culture-based methods that demand 48 to 96 hours until a correct prescription is found [3].

Current identification methodologies span from traditional techniques such as Polymerase Chain Reaction (PCR) with gene sequencing or Enzyme-Linked Immunosorbent Assay (ELISA) to more advanced approaches like Matrix-Assisted Laser Desorption/Ionization Time-of-Flight (MALDI-TOF) mass spectrometry. MALDI-TOF has notably revolutionized both the identification and characterization of bacterial species and the understanding of their antibiotic resistance. By measuring the mass-to-charge ratio of ionized molecules in a sample, MALDI-TOF not only enables the detection of unique bacterial signatures but also recently facilitates the

identification of enzymes that bacteria produce to inactivate antibiotic molecules. However, it is sensitive to experimental conditions and its spectra can vary due to growth conditions and extraction methods, highlighting the need for alternative procedures [1, 4–6]. Moreover, while specific MALDI-TOF variations can suggest antibiotic resistance, they fail to provide direct guidance on appropriate treatment methodologies.

In contemporary medical microbiology, Antibiotic Susceptibility Testing (AST) is critical for evaluating how bacterial strains respond to various antibiotics. Methodologies like the Kirby Bauer Disk Diffraction Method, Unbiased High-Throughput Sequencing (UHTS) or Bacterial Capture Sequencing (BacCapSeq), are effective procedures that have been developed to evaluate bacterial susceptibility to antibiotics, but lack specificity in identifying bacterial species and display variable efficacy based on the bacterial concentrations in the samples. While methods such as those advocated by Cockerill et al. [7] and Lee et al. [8] necessitate substantial sample volumes for optimal sensitivity, genomic strategies like UHTS and BacCapSeq offer comprehensive genetic assessments but are resource-intensive [2, 9, 10]. Importantly, no current method offers rapid, comprehensive insights spanning phylogeny, strain identification, and antimicrobial resistance [2], highlighting the need for advanced techniques adept at both rapid bacterial identification and antibiotic susceptibility assessment.

Nanomotion detection is emerging as a potent technique for antibiotic susceptibility testing [11]. This approach, spotlighting tiny oscillations triggered by living microorganisms, has seen increased attention due to advancements in sensitivity [12]. Although nanomotion spectroscopy exhibits considerable promise, its potential to simultaneously conduct bacterial identification and AST remains uncertain. This is because existing nanomotion signals obtained from AFM cantilevers typically represent averaged motions emanating from populations of 100-1000 bacteria [13–15]. Such aggregate signals make precise bacterial identification challenging, as individual bacterial vibrational profiles are crucial for accurate species differentiation [16]. Recognizing these challenges, recent advancements in nanomotion measurement have culminated in the use of graphene drums, further enhancing detection sensitivity [17]. This evolution in the field potentially provides a renewed opportunity to explore the capability of nanomotion spectroscopy in detailed bacterial identification and antibiotic susceptibility testing.

Addressing this concern, the present investigation explores the viability of utilizing nanomotion drum detection signals in bacterial identification and classification. In tandem, the study probes the efficacy of Machine Learning algorithms such as Convolutional Neural Networks (CNNs) and Support Vector Machines (SVM), in discriminating between different bacterial types, including *Escherichia coli*, *Staphylococcus aureus*, and *Klebsiella pneumoniae*. In addition, these algorithms were adeptly selected to distinguish not only between bacterial species but also between antibiotic-resistant and susceptible strains. Within this framework, a total of 456 data measurements were utilized for the identification study case, and a total of 347 data measurements were employed for the antibiotic susceptibility study case. Bacterial nanomotion signals were transformed into time-frequency spectrograms, serving as inputs for these advanced machine learning algorithms. With an emphasis on pattern recognition, the algorithms were designed to identify intricate features within the spectrogram images, culminating in the construction of robust classification models. In addition, the performance of these models was critically evaluated using established metrics in machine learning theory such as accuracy, sensitivity, specificity, and area under the Receiver Operating Characteristic (ROC) curve, while considering computational efficiency. This study provides the first evidence of integration of ML algorithms with single-cell detection, targeting bacterial species identification and antibiotic susceptibility testing. By assessing the viability of this approach for bacterial identification and antibiotic testing, the investigation provides crucial insights, setting the stage for subsequent research in this area.

4.2. Measurement Setup

In our study, the experimental arrangement consisted of bi-layer CVD graphene, stretched over circular cavities with dimensions of $8\,\mu$m in diameter and $285\,$nm in depth, etched into SiO_2. The assembly integrated an extensive array of these graphene-covered cavities on a silicon substrate, which were submerged in a Luria-Bertani (LB) medium containing a single type of bacteria. The bacteria-mediated nanomotion caused perturbations in the graphene layer, the extent of which was captured using laser interferometry, as depicted in Figure 4.1a. The nanomotion induced time-sensitive deflection at the centre of the graphene drum, discerned via variations in the reflected light intensity. For this study, all measurements were acquired over a 30-second interval [17]. Ultimately, all obtained measurements are duly annotated with the corresponding bacterial type and the specific conditions under which the measurements were conducted. Thus, this annotated dataset then serves as relevant information for the subsequent training of the classification algorithms.

4.3. Signal Processing

For the successful integration of artificial intelligence in bacterial signal classification, it was essential to develop a method to input the signal data into machine learning algorithms. Given the extensive size, complexity and lack of periodicity of the time-domain signals used in our study, which comprise 60,000 data points each, feature extraction methods were employed to isolate relevant characteristics from the signals. These extracted features then provided feasible input data for machine learning algorithms. This type of approach is common in Artificial Intelligence (AI), as underscored by earlier work for detection of apneas in human via analysing cerebral blood flow signals [18].

Therefore, to distill pertinent features from the signal, the time-domain representation was transposed into a frequency-domain format. This conversion renders a structure more suitable for artificial intelligence-driven analysis, as suggested in prior research [19, 20]. Such a methodology finds precedence in biological signal classification research, including a study, wherein Electroencephalogram (EEG) time signals were recast as spectrogram images to automate the detection of Autism Spectrum Disorders [21]. Hence, a fitting approach for this study consisted of using the time-signal to be subdivided into concise segments through a window function, and each segment was subsequently transformed into frequency-domain data using a Short Time Fourier Transform (STFT), to generate a spectrogram image as seen in Figure 4.1a. This technique transmutes the challenge of signal classification into an image classification task and supposes an improved approach when compared to only using Fast Fourier Transform, as STFT is capable of providing insights into both the time and frequency domains which is crucial for examining biological signals with shifting frequency components over time.

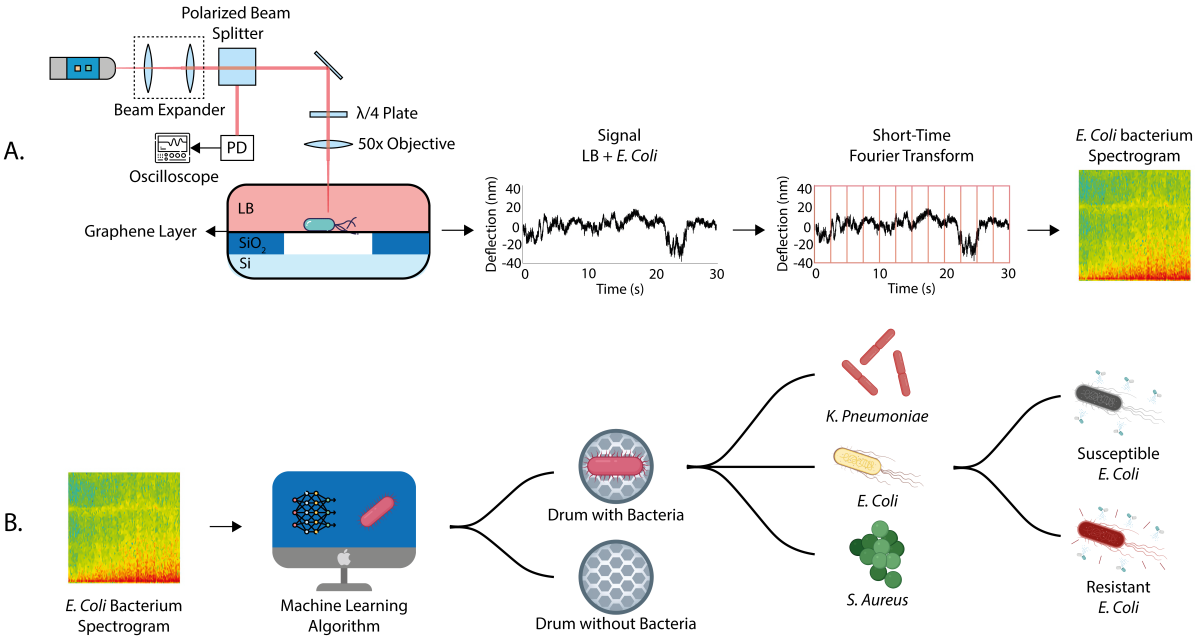

Figure 4.1: Illustration of the summary of the methodology for bacteria identification and susceptibility tests. **A)** Illustration of the procedure, from using the interferometric setup for capturing nanomotion bacterial signals, through to the employment of the Short-Time Fourier Transform for generating the spectrograms. **B)** Illustration of the remaining procedure steps, from the input of spectrograms into the AI algorithm for bacterial type identification to ultimately discern susceptibility or resistance.

For the signals utilized in this study, each with a duration of 30 seconds, the choice of window and overlap sizes was critical to effectively capture essential features and mitigate issues such as spectral leakage. Thus, a window duration of 0.256 seconds and an overlap of 0.128 seconds were selected to optimize feature representation while minimizing artifacts. The resultant spectrograms, presented in Figure 4.2, provide a continuous time-frequency depiction where each spectrogram is representative of an individual single-cell measurement. Considering the extensive dataset, it was imperative to ensure consistency across all spectrograms. This consistency was achieved not only by applying consistent window and overlap values but also by using a standardized Power Spectral Density (PSD) scale. The endpoint of this scale were determined by the average maximum and minimum PSD values across the dataset. Such standardization enabled a clearer distinction of intrinsic spectral features, with more pronounced attributes manifesting higher PSD values.

In this study, a diverse selection of bacteria, including both gram-negative and gram-positive categories, as well as motile and non-motile variants, was chosen to explore the distinctive signal behaviors of these microorganisms. To provide a representative statistical depiction of the majority of spectrograms for each case, Figure 4.2 displays a single measurement from the most important scenarios in the investigation of bacterial species differentiation. For each delineated condition, a measurement approximating the median value, that is, the second quartile Q2 of the box plot, was selected to best characterize that particular instance.

In Figure 4.2a, the spectrogram represents an empty cavity without graphene or bacteria. The absence of salient features underscores the lack of bacterial activity influencing the graphene drum. However, minor features still can be seen in this image. Due to the sensitivity of the measurement set-up, minor features, possibly arising from sources like Brownian motion, environmental noise, and instrumental or external harmonic interferences become pronounced, presenting in some cases horizontal harmonic lines in the spectrograms. In contrast, Figure 4.2b presents the spectrogram for a live *E. coli* bacterium. This visualization reveals distinct

and pronounced features, characterized by a series of brief pulses spanning a diverse frequency range in a compact time frame. These distinct features emerge, likely attributable to inherent bacterial activity or bacteria and flagella interactions with the graphene drum, underscoring the nuanced dynamics of bacterial behavior and nanoscale interactions.

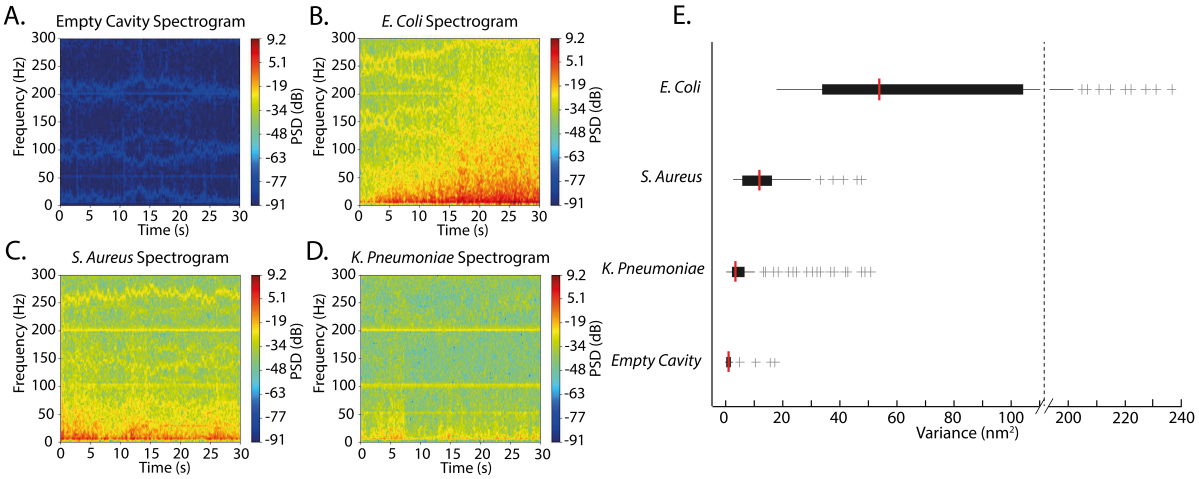

Figure 4.2: Visualization of individual bacterial type spectrograms and their respective variances. **A)** Spectrogram for a single empty cavity (with no graphene). **B)** Spectrogram for a single *E. coli* bacterium. **C)** Spectrogram for a single *S. aureus* bacterium. **D)** Spectrogram for a single *K. pneumoniae* bacterium. **E)** Box plot for all type of bacteria measurements on graphene as well as for empty control cavities without graphene in LB. The box plot delineates quartiles 25th, 50th (mean as red line) and 70th. Outliers are marked with crosses.

It is imperative to acknowledge that the *E. coli* strains utilized in this research are inherently motile, which might contribute to the pronounced intensity of the PSD observed. On the other hand, the signal derived from a *S. aureus* bacterium, as illustrated in Figure 4.2c, manifests a diminished PSD intensity and a narrower frequency range. Such observations align with the understanding that this specific bacterium is a non-motile species, leading to reduced dynamic activity on the drum. Subsequently, in Figure 4.2d, characteristics of the *K. pneumoniae* bacterium are displayed. While the broad frequency pulses remain, they show reduced intensity and frequency range compared to *E. coli* and *S. aureus*. As a non-motile bacterium without flagella, *K. pneumoniae* lacks the dynamic movements observable in motile *E. coli* or *S. aureus* non-motile counterparts. Consequently, the spectrogram captures fewer pronounced variances in frequency and intensity, reflecting the more stationary nature of the bacterium in the observed environment.

After selecting the optimal parameters for spectrogram generation, it was considered essential to ensure the quality of data for machine learning training. The measurement procedure, which could be influenced by factors such as impurities in the Luria-Bertani (LB) medium, inconsistencies in the graphene coating, and external interferences, might introduce variances in the obtained data. Such factors could alter bacterial behavior, affect graphene membrane displacement, and introduce potential artifacts in the spectrograms. Considering these complexities, a thorough analysis of signal variance was conducted. This allowed for the identification and exclusion of anomalous data measurements, as seen in the variance box-plot analysis (Figure 4.2). Removing such outliers ensured a dataset more aligned with genuine bacterial behavior, enhancing the accuracy and robustness of subsequent algorithmic predictions.

4.4. Implementation of ML Algorithms

There are several reasons why inputting signals as images may be preferable to other methods. Foremost, spectrograms provide a two-dimensional time-frequency representation of signals, which allows to encapsulate more detailed information about the signal [22]. Furthermore, using image-based data can facilitate the use of deep learning techniques popularly used in machine learning applications, as it allows for robust local pattern recognition, contributing to a higher degree of noise resilience in the classification process, particularly for cases where the amount of training available data is limited. Therefore, two different types of machine learning algorithms were widely used in this study, Support Vector Machines (SVM) and Convolutional Neural Networks (CNN) as seen in Figure 4.3. These types of deep learning models, are particularly well-suited to image-based data, as they are able to automatically identify features and patterns in the data without the need for additional manual feature engineering [23].

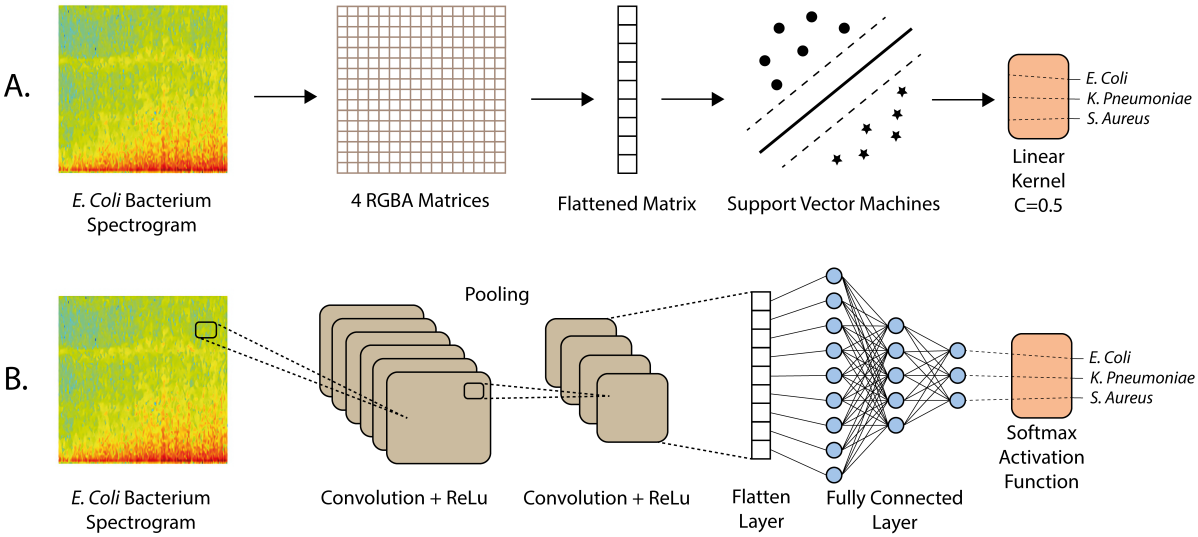

Figure 4.3: Illustration of the two machine learning algorithms employed in the bacteria species identification study case. **A)** Schematic representation of the applied procedure in the implementation of the SVM algorithm. Spectrograms are transformed into RGBA matrices that are used during training for later performing classification. **B)** Schematic representation of the applied procedure in the implementation of the CNN algorithm. Spectrograms are used in convolution operations to train the algorithm to ultimately perform the classification task.

In order to implement the spectrograms, this data was divided into training, validation, and testing datasets for both algorithms, aligning with the principles of supervised learning [24]. Figure 4.3 explains the general methodology implemented for both SVM and CNN algorithms during their training stages in the multiclass classification study case of differentiating between bacteria species.

4.4.1. Support Vector Machines Algorithm

For the implementation of SVM, the spectrograms are assessed based on their pixel intensity and color attributes. Subsequently, as presented in Figure 4.3a, the method applies this color map to the data in the image to create a new array of RGBA values. In this instance, the alpha channel A, functions as an indicator of color opacity, where a value of 0 stands for complete transparency and 1 denotes full opacity. After producing the 2D numpy array of RGBA values, the matrix is subjected to a flattening process. This transformation changes the matrix from two dimensions to a one-dimensional array, making it suitable for subsequent computational evaluations, particularly as input for the SVM algorithm.

Upon receiving the flattened RGBA array representations of the spectrograms, the SVM algorithm processes these high-dimensional data. In its training phase, the algorithm identifies an optimal hyperplane to maximize the margin between bacterial categories. This involves weight adjustments based on input features and, when necessary, the use of a chosen kernel function to refine the data representation. In this process, the solvers convergence benefited from allowing maximal iterations, a technique particularly useful with smaller datasets. This unrestricted exploration of parameter space, combined with SVM proficiency in managing dataset imbalances, ensured a robust prediction mechanism [25]. It should be emphasized that the SVM architectural design required rigorous planning. The configuration was intricately aligned with the particular scenario under study. Different architectures were utilized for classifying bacterial types and discerning between antibiotic-resistant and susceptible bacteria, such as using linear or polynomial kernels, ensuring an algorithmic approach best suited to each specific case.

4.4.2. Convolutional Neural Networks

In contrast, for the CNN approach shown in Figure 4.3b, the analysis now would not be focusing on the attributes of pixel intensity and color, but rather on the pattern recognition and the spatial relationships across the input data using convolution operations, inherent in the design of the Convolutional Neural Network algorithm. In this case, the spectrogram images would also function as the primary inputs. However, upon integrating these images into the CNN model, a sequence of convolution and pooling operations is in charge of extracting the elemental features from the spectrograms, as depicted in Figure 4.3b.

The convolution operation serves as the foundation of the CNN algorithm, processing the input image through several convolutional layers. These layers apply adaptable filters, termed kernels, to recognize local patterns while preserving spatial relationships. Iterative training refines these filters to detect features such as edges and corners [24, 26]. After convolution, Rectified Linear Unit (ReLU) activation functions were implemented in the model to introduce nonlinear elements. The rationale for this selection is that the use of ReLU addresses the vanishing gradient issue, thereby enhancing both the learning efficacy and robustness of the model. [27].

After this transformation, pooling operations condense the spatial dimensions of images while retaining critical information. In this research, max pooling was utilized, highlighting prominent features by selecting the maximum value from each feature map region. After several cycles of convolution, activation, and pooling, the high-dimensional feature maps are flattened into one-dimensional vectors. These flattened layer matched the input size of the subsequent fully connected layers that interpret the complex feature representations of images [28].

In the last stage of the learning process, a Softmax activation function was used in the final decision layer to generate a probability distribution across the multiple outcomes, to effectively represent the mutually exclusive class probabilities for all three bacteria species. However, for the classification of antibiotic resistance versus susceptibility, a binary distinction, the Sigmoid activation function was employed in the final layer. As its output range, spanning from 0 to 1, was ideally suited to yield precise probability estimates for this specific context [27].

Finally, to address potential overfitting, an image augmentation strategy was adopted for the CNN multiclass study case. Data augmentation, widely utilized in image-centric machine learning, counteracts this by introducing random transformations such as rotation, zoom, shift, and flip [29]. Given their relevance to bacterial signal spectrograms, these alterations facilitated the ability of the model to discern generalizable features. Alongside techniques like dropout layers, the approach aimed to harmonize effective training with reliable out-of-sample predictions.

4.5. Results

Utilizing all available data, two primary analytical scenarios were pursued. The first entailed differentiating among three bacterial species, *Escherichia coli*, *Staphylococcus aureus*, and *Klebsiella pneumoniae*. The second aimed to determine the susceptibility or resistance of *E. coli* to antibiotics. Both scenarios employed the nanomotion signals with CNN and SVM algorithms, each customized for the specific task. Performance metrics included accuracy, sensitivity, specificity, and the area under the ROC curve, with emphasis on computational efficiency. Variations in data volume between the cases arose from variance analysis and outlier exclusion to maintain dataset integrity. As Figure 4.2 illustrates, the aim is that this analysis reveals marked distinctions in bacterial signals, emphasizing their importance in classification and confirming genuine differences among bacterial species.

4.5.1. Results for Bacteria Species Identification

CNN Implementation:

This algorithm demonstrated a robust performance, achieving a total accuracy of 88.04%. In the confusion matrix presented in Figure 4.4a, the prediction accuracy for bacterial species was observed as 73% for *E. coli*, 100% for *K. pneumoniae*, and 84% for *S. aureus*. Notably, the primary inaccuracies of the CNN algorithm occurred in misclassifying *S. aureus* as *E. coli* with an error of 20% and *E. coli* as *S. aureus* at 16%. The accuracy versus epochs graph in Figure 4.4b reveals a pronounced rise in accuracy during the early epochs, indicating efficient pattern recognition from the training data. Although there are sporadic fluctuations in accuracy, such unpredictable shifts, common in models using data augmentation, hint at the variability each epoch introduces due to augmented transformations. However, the general trend confirms an increase in accuracy until a convergence for the training and the validation curves is witnessed, most prominently by the 100th epoch. This properly reflects the improved generalization of the model and reduced risk of overfitting.

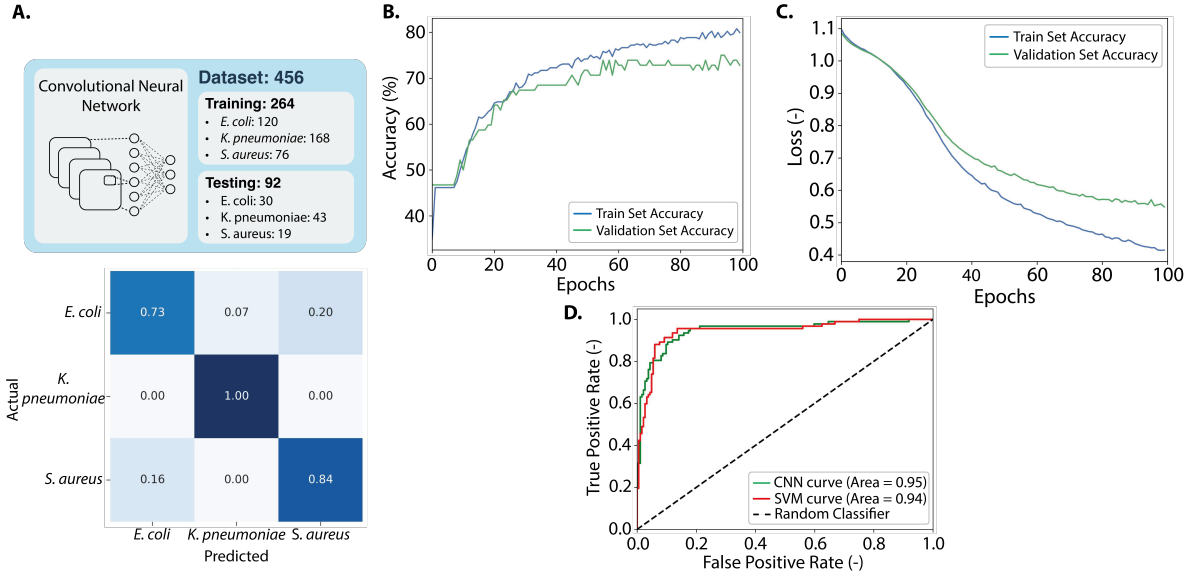

Figure 4.4: Results for the Identification of *E. coli*, *K. pneumoniae* and *S. aureus* using CNN. **A)** Table with the total size of the dataset employed in the training and testing procedures. Confusion Matrix of CNN method with prediction results using the test dataset to distinguish between the three types of bacteria species with a total accuracy of 88.04%. **B)** Accuracy of the CNN algorithm versus epochs during the training procedure. **C)** Loss value versus epochs during the training procedure of the CNN algorithm. **D)** Micro-Average Receiver operating Characteristic graph comparing the performance of all implemented algorithms.

Furthermore, Figure 4.4c provides additional information about the learning dynamics of the model. The starting loss value of 1.1, resulting from the random weight initialization, decreases steadily, signifying the proficiency of the model in learning data patterns. The smooth trajectory of the loss function underscores stable learning, suggesting that the hyperparameters, especially the learning rate, are appropriately configured and chosen. While the training loss remains slightly below the validation loss, indicating mild overfitting, their overall proximity confirms the notable generalization capability of the model, illustrating its adeptness at fitting the training data and generalizing to unseen samples.

SVM Implementation:

In parallel, the SVM algorithm exhibited a commendable performance, achieving an overall accuracy of 84.04%. As illustrated in Figure 4.5a, the species-specific accuracies were recorded as 70% for *E. coli*, 100% for *K. pneumoniae*, and 89% for *S. aureus*. Significant misclassifications included the prediction of *S. aureus* as *E. coli* with a 20% error rate, and *K. pneumoniae* as *E. coli* with a 10% error rate.

In analyzing SVM models for bacterial differentiation via spectrograms, distinct trends became evident for both linear and 4th-degree polynomial kernels. The linear kernel, as displayed in Figure 4.5b, showed peak accuracy around regularization values of 0.2 to 0.5, hinting at an optimal balance against overfitting. However, accuracy declined as values exceeded 0.5, suggesting potential overfitting beyond this point. Conversely, the 4th-degree polynomial kernel in Figure 4.5c, exhibited a steady accuracy increase from 0.1 to 2.0, followed by a plateau between values of 7 to 10. These patterns highlight the delicate balance between regularization and SVM generalization from spectrogram data, suggesting that while the linear kernel might offer marginally superior performance in certain parameter ranges, the polynomial kernel also has the capability to capture non-linear nuances on the inherent characteristics of the dataset.

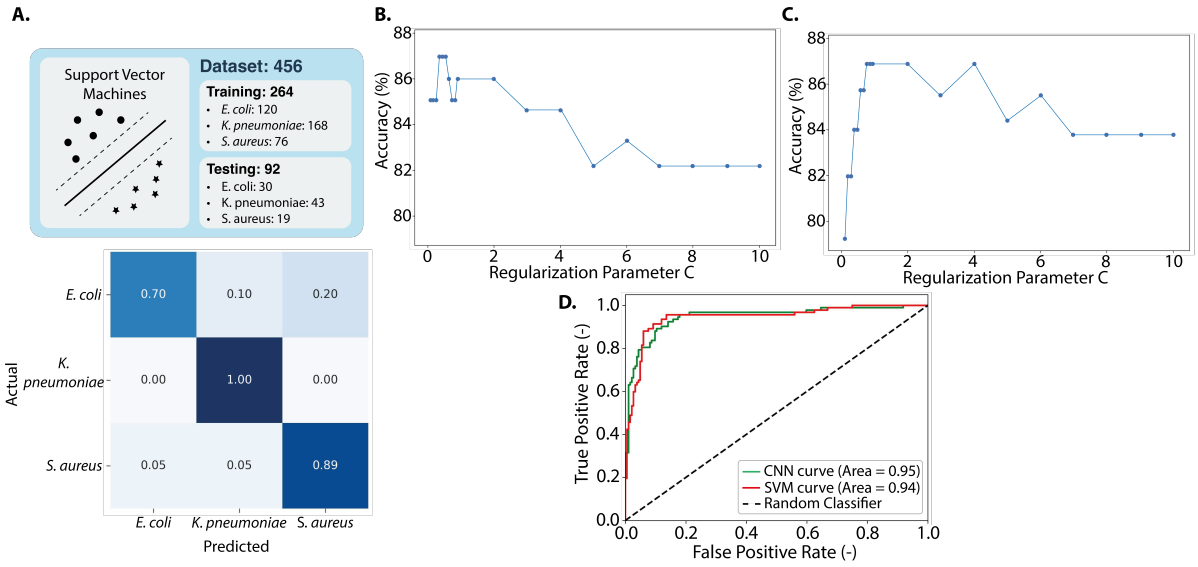

Figure 4.5: Results for the Identification of *E. coli*, *K. pneumoniae* and *S. aureus* using SVM. **A)** Table with the total size of the dataset employed in the training and testing procedures. Confusion Matrix of Linear SVM with prediction results using the test dataset to distinguish between the three types of bacteria species with a total accuracy of 87.04%. **B)** Accuracy of a Linear Kernel SVM algorithm versus the regularization parameter C during the training procedure. **C)** Accuracy of a fourth degree Polynomial Kernel SVM algorithm versus the regularization parameter C during the training procedure. **D)** Micro-Average Receiver operating Characteristic graph comparing the performance of all implemented algorithms.

Evaluating both kernels, it was noted that they achieved peak accuracies close to 87%. However, the performance of the linear kernel, especially with a smaller regularization parameter (C = 0.5), was particularly noteworthy, with results suggesting that the data is notably linearly separable in the transformed feature space. Such an observation implies that, within this high-dimensional space, the distinct bacterial categories might be delineated with relative simplicity. This linear nature not only minimizes overfitting risks but also makes the model more interpretable. While the 4th-degree polynomial kernel can discern complex patterns, its heightened complexity risks overfitting. Given their comparative performance, the linear SVM is favored for its simplicity, robustness, and better generalization potential.

4.5.2. Results for Antibiotic Susceptibility

CNN Implementation:

In the second study case, where the distinction was made between resistant and susceptible *E. coli*, the CNN achieved an overall accuracy of 98.57%. As illustrated in Figure 4.6a, the specific accuracies in the confusion matrix for correct predictions were 100% for susceptible *E. coli* and 95% for resistant *E. coli*. The most notable discrepancy was in the misclassification of susceptible *E. coli* as resistant, occurring at a rate of 5%. In the subsequent analysis, Figure 4.6b illustrates a steady upward trend in the accuracy vs. epochs graph, converging around the seventh epoch. This indicates once again a proficient pattern recognition from the training data. The absence of random oscillations, is expected as the method of data augmentation was not used in this scenario. Similarly, Figure 4.6c depicts a consistent decline in loss with the progression of epochs, reinforcing that the model has improved generalization capabilities and a mitigated risk of overfitting.

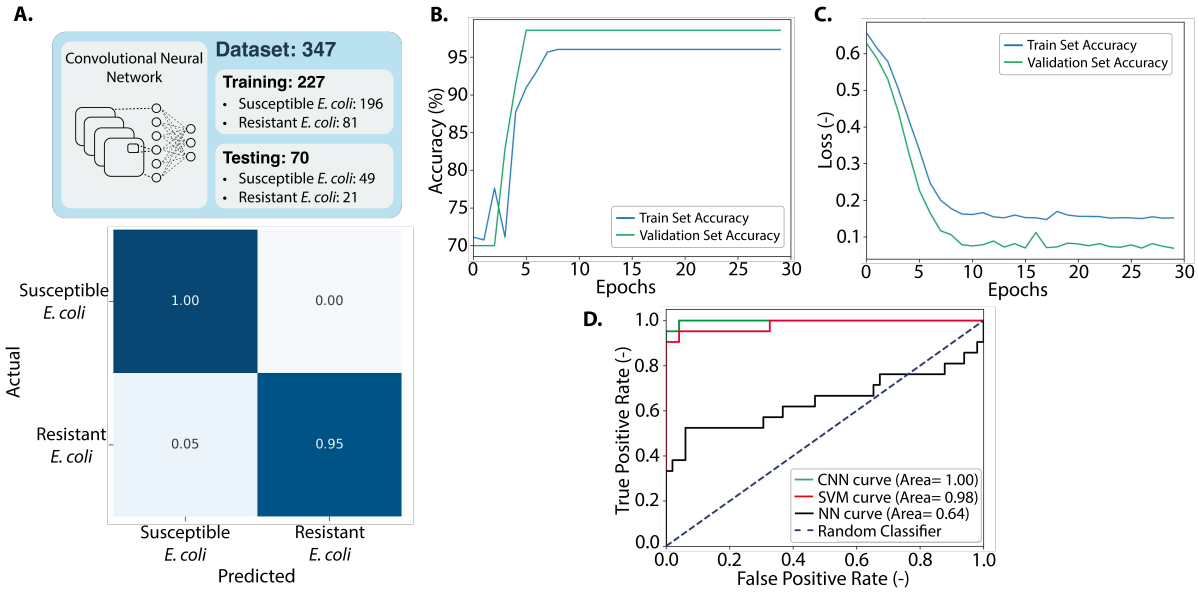

Figure 4.6: Results for the susceptibility tests using CNN. **A)** Table with the total size of the dataset employed in the training and testing procedures. Confusion Matrix for CNN method with prediction results using the test dataset to distinguish between Resistance and Susceptible *E. coli* bacteria under the influence of antibiotics with a total accuracy of 98.57%. **B)** Accuracy of the CNN algorithm versus epochs during the training procedure. **C)** Loss value versus epochs during the training procedure of the CNN algorithm. **D)** Receiver operating Characteristic graph comparing the performance of all implemented algorithms.

SVM Implementation:

Closely following the CNN results, the SVM algorithm exhibited a commendable performance, achieving an overall accuracy of 97.14%. As presented in Figure 4.7a, the algorithm accurately classified susceptible *E. coli* at a rate of 100% and resistant *E. coli* at 90%. It is pertinent to note that the primary misclassification emerged from predictions where susceptible *E. coli* was identified as resistant, marking a 10% error rate.

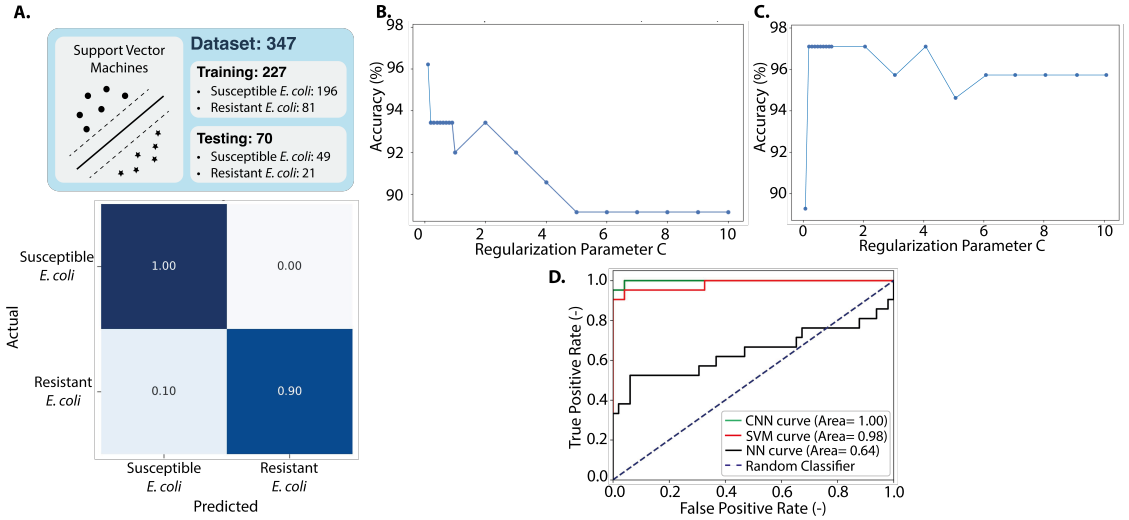

Figure 4.7: Results for the susceptibility tests using SVM. **A)** Table with the total size of the dataset employed in the training and testing procedures. Confusion Matrix of SVM-predicted resistance vs. susceptibility in *E. coli* to antibiotics using the test dataset with a total accuracy of 97.14%. **B)** Accuracy of a Linear Kernel SVM algorithm versus the regularization parameter C during the training procedure. **C)** Accuracy of a fourth degree polynomial Kernel SVM algorithm versus the regularization parameter C during the training procedure. **D)** Receiver Operating Characteristic graph comparing the performance of all implemented algorithms.

In evaluating the two SVM kernel implementations, both the linear and 4th-degree polynomial kernels achieved peak accuracies close to 96% and 97%, respectively. However, the linear kernel in Figure 4.7b exhibited greater sensitivity to alterations in the regularization parameter. As the regularization parameter increased, the accuracy significantly dropped from 96% to 89%, indicating a challenge for the model in distinguishing unseen data. This suggests that data in this study might not be linearly separable in the hyperspace when compared to the case discussed in Section 4.5.1. Contrarily, the 4th-degree polynomial kernel in Figure 4.7c, displayed steadier accuracy rates under similar conditions, reflecting the datasets inherent non-linear characteristics. Given the datasets skewed class distribution and the polynomial kernels consistent performance in capturing non-linear relations, it emerged as the preferred choice. As a result, the SVM with a 4th-degree polynomial kernel, using a regularization parameter C = 1.0, was determined to be the optimal selection within the SVM options for this particular study case.

4.6. Discussion

4.6.1. Evaluation and Selection of Optimal Algorithmic Approaches

To ascertain the optimal algorithm for the classification tasks of bacterial species identification and antibiotic susceptibility testing, both CNN and SVM were thoroughly evaluated. Emphasis was placed on metrics such as accuracy, computational efficiency, interpretability and the ROC graph which presents information about the True Positive Rate (TPR), the False Positive Rate (FPR) and the Area Under the Curve (ROC-AUC). The ROC graph reveals the performance of the model through the balance between sensitivity or TPR and the FPR, with an optimal curve nearing the top-left corner to signify high sensitivity with minimal FPR. In addition, the ROC-AUC, when ranging close to 1, signifies a strong discrimination capacity and near 0.5 represents random chance classification (as depicted by the Random Classifier dotted line in Figures 4.4d and 4.7d. Analysis of these metrics facilitates the assessment of the accuracy of the algorithm in distinguishing bacterial types, aiding in the selection of the most suitable method, particularly in imbalanced class situations.

In Section 4.5.1, Figures 4.4d and 4.5d present the micro-average ROC curve for species classification using CNN and Linear SVM. Micro-average ROC is used, as a traditional ROC curve analyses binary classification, whereas the micro-average offers a comprehensive evaluation for multiclass scenarios. In this instance, while the AUC provides an overarching assessment, it may not spotlight finer intricacies within the ROC curve. In these figures, despite the CNN exhibiting an AUC of 0.9469, slightly above the SVM AUC of 0.9442, a closer inspection of the ROC curves reveals regions where the SVM surpasses the CNN. Such overlaps highlight the capacity of SVM to achieve superior performance within specific FPR intervals, illustrating its efficacy in optimizing the TPR under designated conditions. Such distinctions can be invaluable, especially in the biological context. In bacterial classification, minimizing false positives is crucial to prevent incorrect therapeutic interventions that may impact patient outcomes. Considering the appropriate balance between specificity and sensitivity provided by the SVM, the Linear SVM is recommended as the preferred classifier for bacterial species differentiation, as it enables the attainment of the maximum TPR with the minimum FPR.

In the quest to differentiate between susceptible and resistant strains of *E. coli*, both a CNN and a fourth degree polynomial SVM algorithms were rigorously selected. Examination of the confusion matrices in Figure 4.6a and Figure 4.7a indicated that the CNN outperformed the SVM approach, having only 5% of misclassification compared to the 10% of SVM. In detailed terms, the CNN accurately detected all 49 resistant cases and 20 of the 21 susceptible ones. Further clarity can be derived from the traditional ROC curves for binary cases in Figure 4.6d and Figure 4.7d. The AUC value was higher for the CNN at approximately 0.9981, in comparison to the 0.9825 obtained value for 4th degree polynomial SVM. Importantly, at the critical threshold where false positive rates initiate their rise from zero, the CNN displayed a true positive rate of 0.9524, slightly exceeding the 0.9048 value of the SVM. This nuanced but significant difference highlights the aptitude of the CNN algorithm to adeptly balance between sensitivity and specificity in this decision case. Based on these assessments, and emphasizing the critical need for accurate classification in this domain, as bigger misclassification values could produce a wrong diagnosis to a patient, it is decided that the CNN offers a more promising approach for differentiating between susceptible and resistant *E. coli* strains.

4.6.2. Error Mitigation

To reduce the risk of misclassification in these critical instances, it may be necessary to specify the selected algorithm to reject errors in a more careful way, for example, by adjusting the cost parameters or using a more complex kernel functions to better separate the involved classes. To address potential inaccuracies, several mitigation strategies can be considered. Firstly, to implement in a broader dataset to allow the model to learn even better the different features of each classification class. Another alternative is more robust data augmentation. By encompassing both bacterial dataset expansion and synthetic data generation, it offers the potential to substantially enhance model robustness by addressing data scarcity and imbalance. Algorithmic enhancements, such as ensemble learning, can increase prediction accuracy by aggregating insights from multiple models, and k-fold cross-validation could also ensure rigorous evaluation across diverse data splits.

In addition, integrating domain-specific knowledge via tailored feature engineering can capture details that might be overlooked by the spectrogram models. A detailed error analysis, including confusion matrix evaluations, could identity model weak points for improvements. By analyzing the features that lead to misclassification in these weak areas, the model could be modified to decrease errors. Ensuring model robustness against overfitting can be achieved by fine-tuning regularization and model complexity. Post-deployment monitoring with expert input for uncertain cases maintains the models relevance and accuracy. Investigating advanced neural structures and multi-modal learning can further elevate performance. By adopting these strategies, future work can substantially augment the dependability and efficacy of diagnostic algorithms in health informatics.

4.7. Conclusion

In assessing the potential of machine learning in clinical microbiology, the research sought to gauge the effectiveness of these algorithms in analyzing signals from the nanomotion drum method. Emphasis was placed on distinguishing between antibiotic-resistant and susceptible bacteria and on differentiating among three bacterial species: *E. coli*, *K. pneumoniae*, and *S. aureus*. The results highlight the capability of machine learning algorithms in classifying bacterial samples by their antibiotic resistance. In addition, these algorithms were also adept at distinguishing among the three bacterial species examined in this study. Notably, the differentiation between *E. coli* and *K. pneumoniae*, both Gram-negative bacteria with different motility patterns, and *S. aureus*, a Gram-positive bacterium, suggests the ability of the algorithms to discern variations based on species, Gram classifications, and motility using signals from the graphene drum method.

An intriguing aspect of this study was the performance difference between CNN and SVM within clinical microbiology, as a meticulous evaluation of both algorithms illuminated insights into their strengths and limitations. The CNN models, known for their deep learning architectures, showed particular efficacy in deciphering intricate data patterns, which was most evident when differentiating between the resistant and susceptible strains of *E. coli*. This effectiveness stems from the capability of the CNN model in detecting subtle variations in nanomotion drum signals when analyzing features in the spectrograms. In contrast, SVM, grounded in a mathematical framework, was more proficient in differentiating between bacterial species. The intrinsic design of the SVM model, which optimizes data classification by refining hyperplanes, proved to be appropriate in scenarios where the ramifications of false positives are dire, allowing for adjustments between sensitivity and specificity that became indispensable in species recognition.

One of the central takeaways from these findings is the importance of transcending conventional metrics such as accuracy when selecting machine learning models for healthcare. The broader clinical implications of potential outcomes must be factored in. While the pattern recognition strengths of CNNs are evident, the versatility and precision of SVMs could be indispensable in specific diagnostic contexts. This brings to the forefront the significance of juxtaposing algorithmic precision with biological relevance, given the severe consequences of misclassifications in clinical settings.

Furthermore, this research underscores the pivotal role of error mitigation in such studies. A deep dive into the decision tree structure accentuates the severity of repercussions stemming from misclassification errors at distinct stages. As decision-making becomes more nuanced and categorical, the gravity of misclassification consequences escalates, highlighting the imperative for precision and recall in diagnostic applications. Potential strategies to mitigate these risks include data augmentation, ensemble learning, and systematic error analysis. These approaches pave the way for fortifying the resilience of future machine learning applications in diagnostics.

In essence, this research marks a noteworthy stride in harnessing advanced computational strategies for bacterial identification and antibiotic resistance testing. While numerous challenges remain, the insights gleaned offer a robust scaffold for future researchers and clinicians. Succeeding endeavors in this sphere should aim to augment this foundational knowledge, delve into more sophisticated neural architectures, and consistently incorporate direct clinical feedback from patient care settings. Such a holistic approach promises to actualize the immense potential machine learning holds in transforming diagnostic paradigms.

4.8. Methods

4.8.1. Signal Processing for ML Algorithms

To extract relevant features from the signal, the time-signal was partitioned into discrete segments using a window function. Each of these segments was then translated into frequency-domain representations via the Short-Time Fourier Transform, culminating in a spectrogram image as depicted in Figure 4.1a. Notably, leveraging the STFT offers a more nuanced approach compared to solely relying on the Fast Fourier Transform, as the STFT facilitates comprehension of both time and frequency domains simultaneously. This characteristic becomes essential in the examination of biological signals, which often display frequency components that shift over time. However, it is important to note that there is an intrinsic trade-off in the STFT between time and frequency resolution. In essence, a smaller window size augments the clarity in the time domain but compromises the granularity in the frequency domain, whereas a broader window achieves the opposite effect [30].

Thus, in order to create the spectrograms, a window size needed to be determined. Selecting an adequate window size and overlap value for a STFT without preceding insights into the signal typically represents a challenging task [30]. Nonetheless, the application of certain methodologies can facilitate more informed decisions regarding these estimations. For instance, when determining the window size, a common starting point in many applications entails using a window size that accounts for 1 to 5 percent of the total signal length. This assumption is predicated on the expectation that the signal of interest will exhibit changes over the duration of the signal, rather than manifesting as a single brief event [22]. Additionally, to determine the size of the overlap, a practical initial approximation often involves adopting half the window size [22]. This approach aids in striking a balance between computational efficiency and thorough feature capture from the signal. A small overlap risks omitting short-term

features, whereas an excessively large overlap may impose unnecessary computational load [22].

Considering that the signals were measured over a duration of 30 seconds, a meticulous selection of the window and overlap sizes was crucial to effectively capture the significant features of the signal and mitigate potential issues, such as spectral leakage. After evaluating numerous values, a window size of 0.256 seconds paired with an overlap size of 0.128 seconds was selected, deemed to provide an optimal balance of feature capture and artifact minimization. Thus, the resulting spectrograms were able to present a clear representation of the signal, where continuity in the time-frequency representation can be seen, without the existence of breaks or gaps on the image as can be seen in Figure 4.2.

To accomplish this, a custom Python function was developed using existing Scipy and Matplotlib functions to process the signals. Specifically, the custom function took a single 30 seconds signal measurement at a time and inputted this data into the `spectrogram()` Scipy function to generate an image. This last function divides the input signal into windows of length *nperseg* with overlapping segments of length *noverlap*, and applies a window function to each segment before computing a Discrete Fourier Transform to that segment. The value of *nperseg* is usually inputted a power of 2 for computational efficiency, and should be chosen based on the trade-off between frequency and time resolution as mentioned before [21].

After determining the optimal parameters for spectrogram generation, careful attention was directed towards ensuring the quality of the data to be employed in the machine learning training process. It must be recognized that the measurement procedure can be influenced by numerous factors, potentially affecting the fidelity of data obtained during each individual measurement process. Given the inherent intricacy of the experimental setup and the possible impact of various unpredictable factors during the measurement process, a systematic investigation into signal variance was deemed necessary and performed accordingly. This procedure facilitated the detection and exclusion of anomalous data points, like outliers, which might otherwise introduce undesirable variance and uncertainty into the dataset. By excluding these outliers identified in the variance box-plot analysis (Figure 4.2), the resultant data set was more representative of the actual bacterial behavior, enhancing the potential for accurate and effective classification through the trained algorithms and consequently reinforcing the robustness of the future predictions.

4.8.2. Implemented ML Algorithms Architectures

For all the algorithms employed in this study, the spectrogram data was partitioned into training, validation, and testing datasets. This approach is required for the execution of supervised learning [24]. During the training phase, the machine learning algorithm was informed about the labels assigned to each measurement, thereby enhancing its ability to discern the corresponding types of bacteria. Typically, the training set, comprising input data and corresponding desired outputs, is utilized by the model to learn the input-output relationship and adjust its parameters for minimized prediction errors. The validation set, a distinct segment of the dataset, serves the purpose of parameter refinement and model performance evaluation rather than contributing to training. This set plays a pivotal role in directing adjustments to the parameters of the algorithm, without being directly involved in the training of the model. Subsequent to training and validation, the test set, which comprises the final portion of the dataset, is utilized to gauge the capacity of the model to generalize to novel, unseen data, thereby providing a measure of the comprehensive accuracy of the model [20, 31].

Support Vector Machines Algorithm

In this particular experimental configuration, the analysis of the spectrograms primarily focuses on the attributes of pixel intensity and color. Therefore, once the spectrogram image is obtained, a custom python function saves the result as an `image` object. Subsequently, a method called `.to_rgba()` is implemented. This method in the Matplotlib Python library is used to convert an image to RGBA (red, green, blue, alpha) format, which is a standard format for storing and displaying images. In more detail, when the `image.to_rgba()` method is called on an image object, it first checks the current color map of the image, which is a set of colors that are used to represent different light intensity values in an image.

Subsequently, the method applies this color map to the data in the image to create a new array of RGBA values. These values are determined by mapping the original data values to colors in the color map and then adding an alpha channel value to each color. The alpha channel specifies the opacity of the color, with 0 indicating fully transparent and 1 indicating fully opaque. Once the new array of RGBA values is created, the `image.to_rgba()` method returns this array as a 2D numpy array that can be used for further image analysis. However, before the RGBA matrix is fed as an input to the machine learning algorithm, it is necessary to flatten the 2D array. This process involves converting the two-dimensional matrix to a one-dimensional array.

Furthermore, the structural design of the Support Vector Machine algorithm required careful consideration as well, with its configuration being highly dependent on the specific scenario under investigation. Thus, distinct architectures were employed for the classification of bacterial types and the differentiation between antibiotic-resistant and susceptible bacteria. This bifurcated design strategy ensured optimal algorithm performance tailored to each unique case. Therefore, in the context of bacterial species identification, the optimal performance was achieved using a linear kernel accompanied by a regularization parameter of 0.5. Conversely, for differentiation between susceptible and resistant bacterial strains, a polynomial kernel of the fourth degree, coupled with a regularization parameter of 1.0, yielded the highest accuracy. The selected parameters, based on the information provided in Section 7.3.1, are detailed in Table 6.1.

The parameter `max_iter` in the Support Vector Machine (SVM) algorithm, which governs the maximum number of iterations allowed for the solver to converge, was intentionally set to its default value of -1, representing an unlimited number of iterations. This approach gives the model an opportunity for complete convergence, which is particularly useful when using a small size dataset. Permitting an unlimited number of iterations allows the model to fully explore the parameter space and potentially arrive at an optimal solution. These strategies, in combination with the use of SVM, which is particularly suited for cases with high unbalanced datasets, gives the algorithm robustness to perform accurately during the prediction stage [25].

Convolutional Neural Networks

In contrast, for this different experimental approach, the analysis now would not be focusing on the attributes of pixel intensity and color, but rather on the pattern recognition and the spatial relationships across the input data using convolution operations, inherent in the design of the Convolutional Neural Network algorithm. In this case, the spectrogram images would also function as the primary inputs. However, upon integrating these images into the CNN model, a sequence of convolution and pooling operations is in charge of extracting the elemental features from the spectrograms, as depicted in Figure 4.3b.

The convolution operation is the core of the CNN algorithm, as the input image undergoes multiple convolutional layers. These layers use adaptable filters, or 'kernels', across the image to discern local patterns while maintaining spatial pixel relationships. Through iterative training, these filters learn to identify features like edges and corners [24, 26]. Following each convolution operation, a Rectified Linear Unit (ReLU) activation function was implemented with the objective of introducing a nonlinear elements into the model. The incorporation of the ReLU activation function is instrumental in augmenting the effectiveness of the training process. Specifically, it mitigates the vanishing gradient issue (See Section 7.4.4), thereby enhancing the learning efficiency of the model and enhancing its robustness [27].

Following this, a pooling or subsampling operation is applied to the transformed images. The objective of this operation is to condense the spatial dimensions while preserving the essential information. In this study, max pooling methodology was employed, an operation that selects the maximum value from each region of the feature map, hence accentuating the prominent features. As the network undergoes numerous iterations of convolution, activation, and pooling operations, the resulting high-dimensional feature maps are eventually flattened into a one-dimensional vector. As a result, this final flattened array has a size equal to the number of input neurons in the subsequent fully connected layers, which interpret the found intricate feature representations of the images [28].

In the course of this research, strategic selections of activation functions were made according to the specific classification objectives. For the experiment aimed at differentiating among three types of bacteria, a Softmax activation function was employed in the final decision layer. This choice was based on the nature of the Softmax function, which effectively generates a probability distribution across multiple potential outcomes, offering an accurate representation of mutually exclusive class probabilities, ideal for multi-class classification problems [27]. Contrarily, in the study focused on classifying antibiotic resistance and susceptibility, which is a binary classification problem, a Sigmoid activation function was implemented in the final layer. The Sigmoid function confines its output to a range between 0 and 1, thereby providing a probability representation for a single class [27]. For binary classification scenarios such as this, where the objective is to decide whether a given instance belongs to a specific class or not, the Sigmoid function presents as the most suitable option.

Given the significance of correct parameter selection, hyper-parameters for the CNN model were diligently chosen with an iterative process, reflecting considerations of task complexity, data specifics, and available data. This ensured a balance between convolutional layers for feature abstraction, dropout layers for overfitting prevention, and fully connected layers for interpreting feature representations. The learning rate and batch size were optimized for efficiency and learning capacity, with iterative adjustments informed by accuracy and convergence plots.

Finally, it is important to mention that to mitigate the risk of overfitting in the model, an image augmentation strategy was implemented. Overfitting is characterized by a model's excessive reliance on intricate patterns and noise inherent in the training dataset, which hampers its ability to generalize effectively to unseen data [31]. Data augmentation, prevalent in image-based machine learning, combats overfitting by applying random transformations like rotation, zoom, shift, and flip to original images. Chosen for their relevance to bacterial signal spectrograms, these modifications prompt the model to learn generalizable features. Paired with regularization methods like dropout layers, data augmentation aimed to balance effective learning with accurate prediction on unseen data.

Fully Connected Neural Networks

To further elucidate the capabilities of neural networks, a fully connected neural network was introduced specifically for analyzing time-domain signals derived from the graphene drums method. The underlying rationale for this approach centered on assessing whether conventional machine learning algorithms could discern subtle variances in the signals when solely relying on their time-domain data, without resorting to spectrograms. In order to implement this, the data was partitioned into training and testing datasets in equivalent proportions as previously utilized for the CNN and SVM algorithms. Yet, in this iteration, Principal Component Analysis was employed to extract salient features from the unprocessed time-domain data.

Principal Component Analysis (PCA) is a statistical procedure that employs orthogonal transformation to convert potentially correlated variables into a set of linearly uncorrelated variables known as principal components. In the context of this study, wherein signals are derived from the graphene drums method to analyze bacterial activity, PCA serves to reduce dimensionality while retaining the most significant patterns in the data. This aids in simplifying the data structure, potentially enhancing the efficiency and accuracy of subsequent analyses. In addition, PCA identifies the directions, or principal components, in the dataset where the variance is maximized. By representing the original data in terms of these components, it effectively captures the underlying patterns while often discarding noise or less informative variations. For a signal, this means that PCA can isolate the most significant trends or patterns in the data, rendering a clearer understanding of the main characteristics and potentially removing extraneous information [32].

This approach was only implemented in the differentiation between resistant and susceptible *E. coli* bacteria. Once the dataset was divided for training and testing, then the variance analysis was performed to exclude all the outliers more effectively and ensure the quality of the data used for the training of the algorithm. Subsequently, the raw input data undergoes normalization using the `StandardScaler` function from the Scikit-learn library, ensuring that each feature has a zero mean and a standard deviation of one. This normalization is important for stabilizing the numerical conditioning, facilitating faster convergence, and optimizing the performance of subsequent machine learning algorithms. To reduce the computational burden and potentially enhance the generalization capability of the model by minimizing the noise inherent in the data, the dimensionality of the normalized data is reduced using PCA, retaining 277 principal components. Following the preprocessing, a sequential fully connected neural network was constructed. The network comprised 4 hidden fully connected layers, interspersed with dropout layers for regularization to mitigate overfitting. Each dense layer employs the Rectified Linear Unit (ReLU) activation function. The culmination of the architecture is an output layer with 2 neurons, adopting a Softmax activation, yielding probabilities for the two classes: resistant and susceptible. When compiling the model, the Adam optimizer is employed with an exceptionally low learning rate of 1×10^{-6}, and Binary Cross Entropy is designated as the loss function, considering the integer nature of the multiclass labels. The model is rigorously trained on the PCA-transformed training data for an extensive 3,000 epochs and is subsequently evaluated on the transformed test set to discern its effectiveness in classifying the two *E. coli* bacterial types.

5

Conclusions and Outlook

In assessing machine learning role in clinical microbiology, this research investigated the proficiency of such algorithms in interpreting signals from the nanomotion drum method. The study aimed to distinguish bacteria based on antibiotic susceptibility and to differentiate among *E. coli*, *K. pneumoniae*, and *S. aureus*. The study findings indicate that the machine learning algorithms employed effectively sorted bacteria into resistant and susceptible categories. Additionally, they showed a commendable ability to distinguish among the three bacterial species, reinforcing the promise of machine learning in the broader spectrum of clinical microbiology. Notably, the first two species are categorized as Gram-negative bacteria; with *E. coli* being motile and *K. pneumoniae* characterized as non-motile, while *S. aureus* is recognized as a Gram-positive bacterium. Such distinctions highlight the algorithm aptitude to identify differences in species, Gram classifications, and motility features, especially within the Gram-negative category.

In this research, a notable performance difference was observed between CNN and SVM within the clinical microbiology landscape. The endeavors to ascertain an optimal algorithm for both classification tasks involved rigorous evaluations of both algorithms, resulting in valuable insights into their accuracies and limitations. The CNN approach, characterized by their deep learning architectures, displayed an adeptness in handling intricate data patterns, particularly evident when discerning between susceptible and resistant strains of *E. coli*, owing to their proficiency in detecting nuances in nanomotion drum signals. Conversely, SVM, underpinned by mathematical rigor, demonstrated their prowess in bacterial species differentiation. The architecture of SVM, which fine-tunes hyperplanes for optimal data class segregation, becomes especially important in scenarios where false positives bear significant clinical repercussions. SVM offered the ability to modulate the trade-offs between sensitivity and specificity, tailoring decision boundaries, that became crucial in species identification.

In the ever-evolving landscape of machine learning, Large Language Models (LLMs) stand out for their proficiency in processing and generating text. Beyond their role in natural language tasks, LLMs show promise in diverse applications, including the analysis of signals from the nanomotion drum method. The architectural foundation of LLMs, often rooted in recurrent neural networks or transformers, enables adept analysis of sequential data, making them suitable for time-series datasets like those from the nanomotion drum. Additionally, LLMs can extract deeper insights from data sequences, potentially interpreting bacterial behaviors or health from nanomotion patterns. As clinical microbiology pursues advanced computational methods, the incorporation of LLMs might bring enhanced depth and precision. Nevertheless, thorough exploration and validation remain crucial to ascertain the applicability and reliability of LLMs in such contexts.

In the research field of machine learning applied to clinical microbiology, the value of comprehensive and quality data is crucial. For current and emerging techniques, especially Large Language Models, robust datasets are indispensable. Although the current database has a

considerable size, further enrichment is essential to capture a comprehensive view of bacterial dynamics. Such a dataset can reveal pivotal insights into bacterial characteristics and resistance profiles. In diagnostics, where precision is critical, an enriched dataset significantly enhances algorithmic accuracy. Thus, expanding and diversifying data sources remain fundamental to future research endeavors.

Drawing upon the insights and advancements delineated in this study, there exists an undeniable momentum driving forward the fusion of clinical microbiology and computational techniques. As we stand at the cusp of a transformative era, it is a collective aspiration that this research not only sets the stage for innovative diagnostic paradigms but also kindles further inquiry, collaboration, and dedication in the scientific community. The future beckons with promise, and with continued diligence and unity, the horizon is ripe with possibilities awaiting exploration.

6

Supplementary Materials

6.1. Decision Tree for all Analyzed Cases

in the case that is desired to implement these ML algorithm as future diagnostic tools, discerning not only between different bacteria but also between non-bacterial elements such as cavities without graphene or drums without bacteria becomes crucial, as this precision ensures that the diagnostic analysis exclusively targets data reflecting bacterial nanomotion.

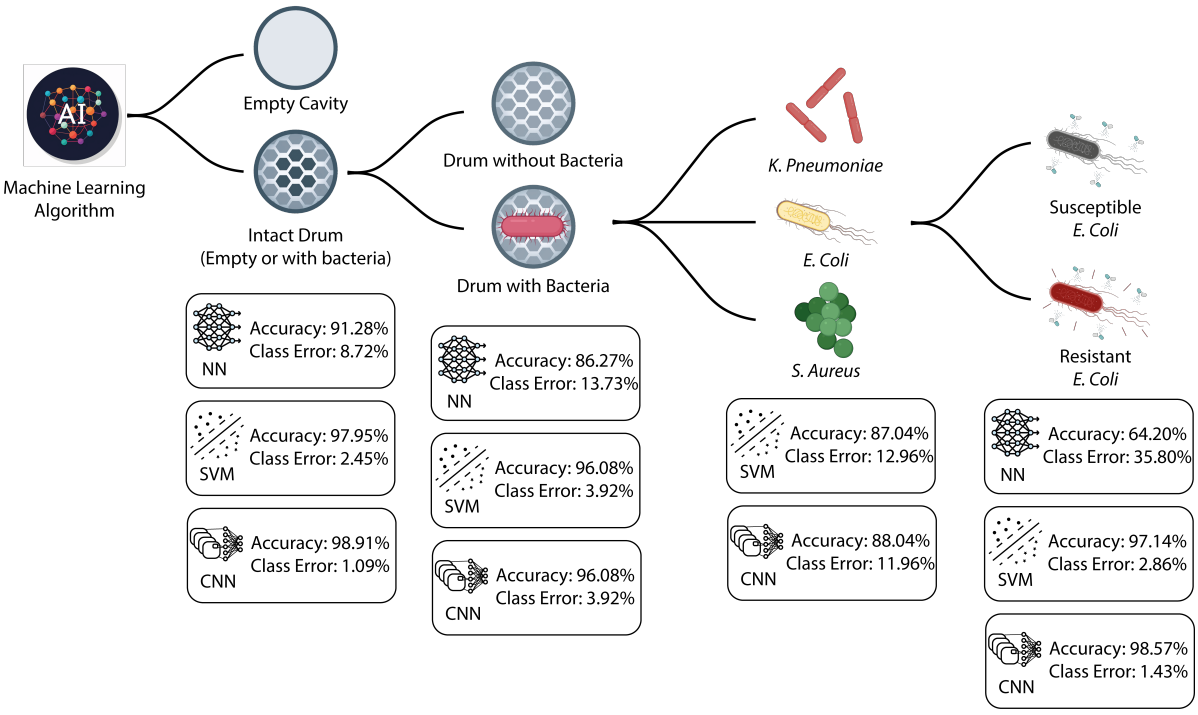

Figure 6.1: Illustration of the decision tree used for signal type determination with its different decision stages. Stage 1: Empty Cavities vs Intact Drums, Stage 2: Drums with and without Bacteria, Stage 3: Bacteria Species Identification and Stage 4: Resistant vs Susceptible *E. coli*. Each stage highlights the accuracy of the implemented algorithm and the corresponding classification error.

Therefore, the algorithms were further trained to discern among various signal types that might be encountered in a prospective diagnostic tool employing machine learning. In a real-world application, the signals must be classified without prior knowledge of whether they originate from a drum with bacteria, thereby ensuring that only signals associated with bacterial presence are utilized for accurate diagnosis. Figure 6.1 illustrates the accuracy levels of the trained algorithms at different decision stages within this automated diagnostic procedure.

Figure 6.1 delineates the classification flow to be adopted during the automated algorithmic implementation. Given the inherent challenges in ascertaining the origin of a signal from specific drum types, the algorithm first identifies whether the signal emanates from a drum with undamaged graphene and if it possesses bacteria on its surface. Only after such determinations can it advance to stages that discern bacterial type and assess their susceptibility or resistance. At every decision juncture, various algorithms were evaluated, and the optimal performer was chosen. This ensures that each stage is approached with the most effective method tailored to its specific requirements.

6.2. Consequences of Misclassification Errors

In machine learning, particularly in medical contexts, a detailed examination of algorithm limitations, including misclassifications, is required. Recognizing the severe consequences that certain errors may have, especially in patient treatment, an analysis of these errors provides information necessary to perform adjustments to the algorithms.

Decision Stage 1: Empty Cavities vs Intact Drums

1. **Empty Cavity predicted as Intact Drum:**
   In this context, errors appear minimal. If an empty cavity is misclassified initially, subsequent decision stages likely rectify this due to the signal's diminished characteristics: low intensity, variance, and Power Spectral Density compared to a drum with bacteria. Misclassifications may be corrected at secondary stages, limiting error propagation. However, if a highly noisy empty cavity persists in being misclassified, it may be wrongly identified as sensitive bacteria. This could lead to the prescription of standard-intensity antibiotics, potentially proving ineffective if the actual bacterium is resistant. Such an error may exacerbate the patient's health complications, although the likelihood of this scenario is considered low.
2. **Intact drum as predicted as Empty Cavity:**
   In this case, as there is no further continuation of the decision tree, another run would of the algorithms would need to be made to actually be able to performed a diagnosis. However, if hypothetically the tree would also ramify here to ultimately arrived to a classification and a final diagnosis, this signal then most likely would be misclassified as a drum without bacteria. However, in case is not classified there, then it would be misclassified most likely as sensitive bacteria, with the risk connotation of probably prescribing an antibiotic that would not work.

Decision Stage 2: Drums without Bacteria vs Drums with Bacteria

1. **Drum without Bacteria predicted as Drum with Bacteria:**
   In this specific scenario, the measured signal, potentially exhibiting minor variations and diminished amplitude compared to a drum containing bacteria, might be misclassified as indicative of sensitive bacteria during the final decision stage. Such misclassification can have considerable implications. If the bacterium responsible for the condition of the patient is actually resistant, this could lead to an inappropriate treatment strategy involving standard-intensity antibiotics. Consequently, this might result in an inaccurate diagnosis, potentially compromising the health outcome of patient.
2. **Drum with Bacteria as predicted as Drum without Bacteria:**
   In this scenario, given the absence of further branches in the decision tree of Figure 6.1, another iteration of the algorithms would be required to reach a definitive diagnosis.

Nevertheless, if one were to assume a continuation in this decision branch leading to a diagnosis, subsequent misclassification of the signal could hinge on its intensity. If a drum with bacteria were mistakenly classified as an empty drum, it is plausible that the signal would exhibit reduced amplitude and variance. Consequently, during the final decision stage, the signal could be misidentified as indicative of sensitive bacteria. If the actual bacterium affecting the patient is resistant, this misclassification would potentially result in the patient not receiving the correct treatment.

Decision Stage 3: Differentiating Among Bacterial Species

1. **Misclassification between *E. coli* and *K. pneumoniae*:**
   Misclassification between these two bacterial species can appear when an *E. coli* bacterium has a small signal with small variance or when a *K. pneumoniae* has a higher variance and higher intensity signal. Such a misclassification can lead to prescribe antibiotics specific to the incorrect pathogen, which may reduce treatment efficacy. However, it is noteworthy that both *E. coli* and *K. pneumoniae* are categorized as Gram-negative bacteria. Consequently, if a patient is administered a broad-spectrum antibiotic, the likelihood of an effective treatment increases due to the structural similarities in the membranes of Gram-negative bacteria [33]. Nonetheless, resistance patterns distinct to each bacterial strain, coupled with the specific site of infection, may play a pivotal role in influencing the choice of treatment. Therefore, a misclassification in this context is suboptimal, but if it occurs, it may represent a minor error.

2. **Misclassification between *E. coli* and *S. aureus*:**

   In cases where *E. coli* signals manifest with reduced intensity, characterized by notably decreased variance and amplitude, there exists a risk of erroneously classifying them as *S. aureus*. This distinction is critical given the differing characteristics of the two bacteria, as *E. coli* is Gram-negative and *S. aureus* is Gram-positive. These differences inherently dictate separate optimal treatment approaches, as *S. aureus* bacteria, especially drug-resistant strains, require a specific set of antibiotics [34]. Administering antibiotics tailored for Gram-positive bacteria like *S. aureus* when the patient actually has a Gram-negative infection, such as *E. coli*, may represent an ineffective treatment, leading to complication of the overall health status of the patient.
3. **Misclassification between *K. pneumoniae* predicted as *S. aureus*:** A misclassification of K. pneumoniae as S. aureus could be attributed to potential overlaps or similarities in the spectral signal. These organisms, although distinct in their cellular architecture and typical presentations, can occasionally present similar motion behaviour as both are non-motile specimens. A misclassification between K. pneumoniae and S. aureus is significant, as K. pneumoniae often resists many Gram-positive-targeted antibiotics, requiring distinct treatments [34]. Conversely, drugs effective against K. pneumoniae are ineffective for S. aureus. Due to their distinct antibiotic responses, incorrect treatments can lead to unwarranted antibiotic exposure and treatment failure, extending patient morbidity.

Decision Stage 4: Resistant Bacteria vs Susceptible Bacteria

1. **Sensitive Bacteria predicted as Resistant Bacteria:**
   In this situation, if the signal is misclassified as indicative of resistant bacteria, it is probable that the error arises from a particularly noisy measurement of sensitive bacteria. As a result, the medical recommendation may lean towards prescribing powerful antibiotics. Although this approach might inadvertently foster antibiotic resistance among bacteria in

the future, it prioritizes ensuring that the patient undergoes suitable treatment, thereby reducing potential health hazards.

2. **Resistant Bacteria as predicted as Sensitive Bacteria:**
   In this scenario, there is significant cause for concern. The misclassification of a resistant bacteria signal, which likely has a low amplitude and variance, as indicative of sensitive bacteria presents a critical risk. Such a misclassification can lead physicians to recommend standard-intensity antibiotics. If these antibiotics prove ineffective against the bacteria responsible for the condition of the patient, the treatment becomes inadequate, thereby elevating potential health liabilities.

Overall, misclassification in the decision tree could have significant consequences for patient outcomes and should be minimized as much as possible. Therefore, it is important to carefully evaluate the accuracy of the algorithm and take steps to improve it if necessary, such as through further training, refinement of the decision tree, or the incorporation of additional data sources. It is important to mention that, it is expected that the risk of misclassification in the proposed decision tree would be more serious as the tree progresses to more specific instances.

6.3. Detailed Additional Results

6.3.1. Cavities with and without Graphene Results

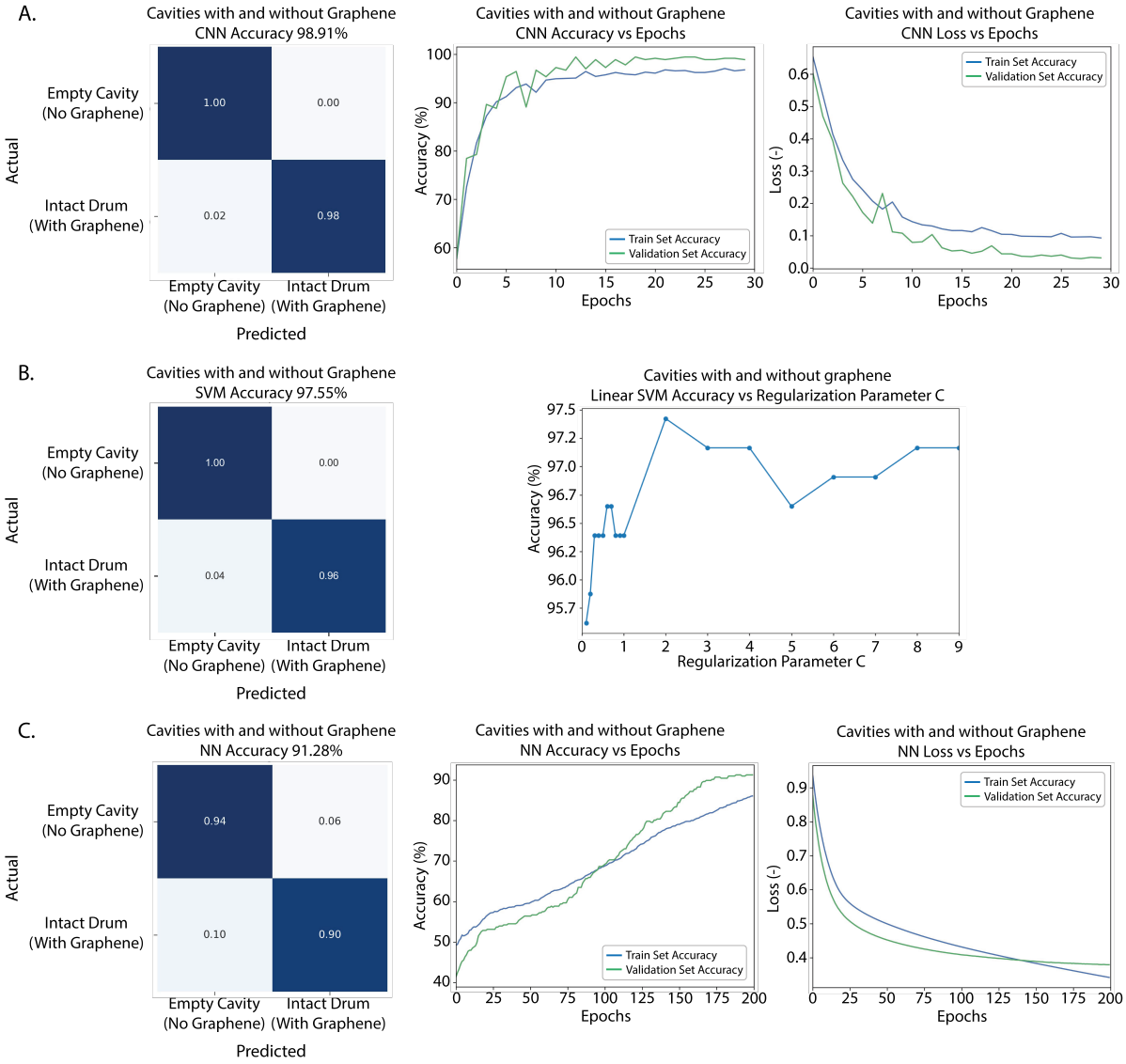

Figure 6.2: Illustration of the results obtained during the first stage of the decision tree where the signal is being classified in Empty cavities without graphene and intact drums with graphene, i.e., both with and without bacteria. **A)** Results obtained with the CNN implementation. **B)** Results obtained with the best SVM implementation. **C)** Results obtained with the Fully Connected Neural Network with Principal Component Analysis (PCA) implementation.

At this juncture in the decision tree, all algorithms demonstrated remarkable accuracy: CNN at 98.91%, SVM at 97.55%, and NN with PCA at 91.28%. Such elevated accuracy can be attributed to the pronounced signal differences between an empty cavity without graphene and a drum with intact graphene, irrespective of the drum being empty or having bacteria on it. Achieving high accuracy early in the decision tree bolsters the robustness of the classification process, minimizing errors such as misidentifying empty cavities as intact drums. It is noteworthy to mention the commendable accuracy achieved by the Fully Connected Neural Network, likely a consequence of utilizing the PCA feature extraction method. The method enhances variance differences in the signal, thereby facilitating the classification task.

6.3.2. Drums with and without Bacteria Results

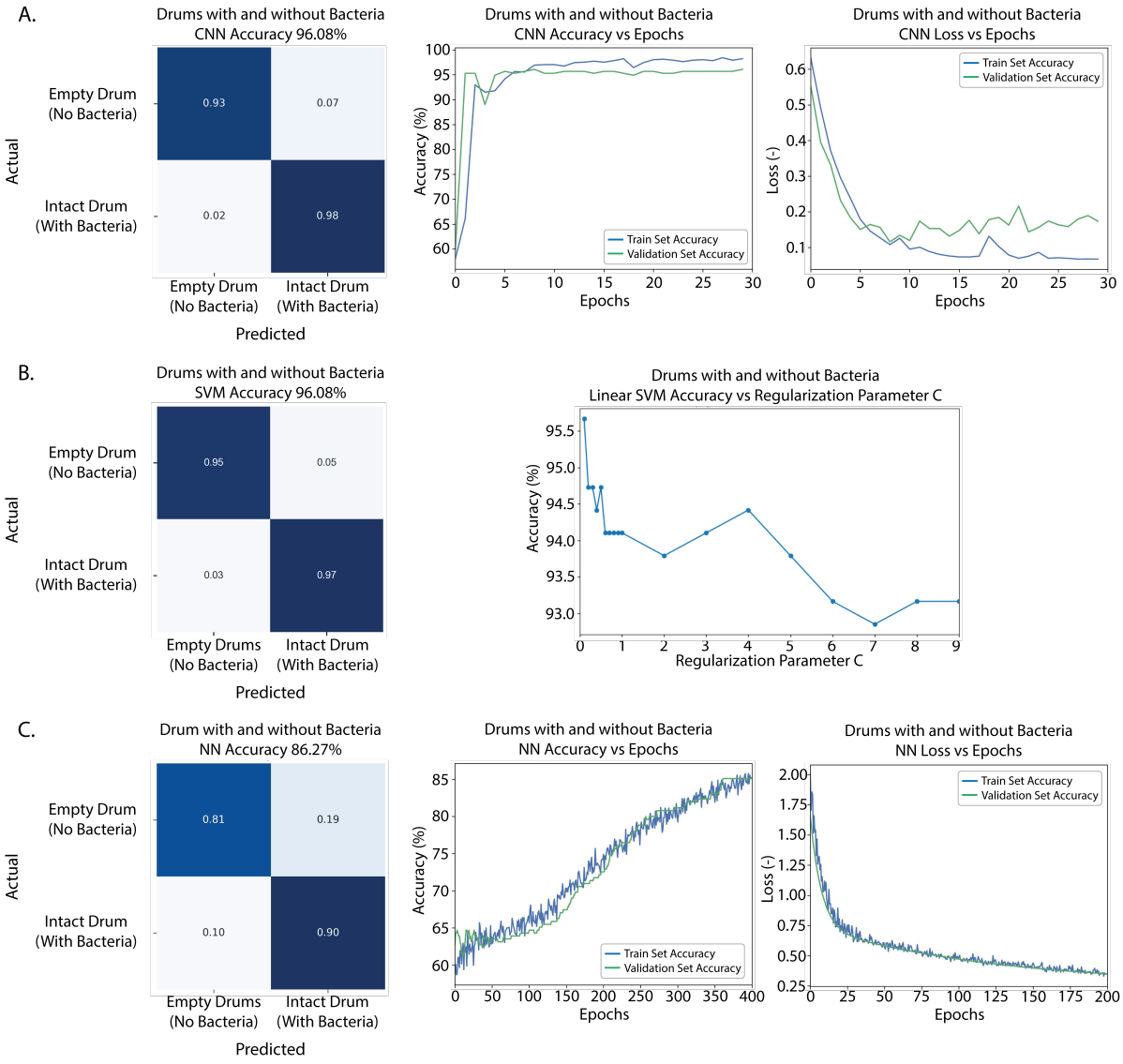

Figure 6.3: Illustration of the results obtained during the second stage of the decision tree where the signal is being classified in Empty Drums without bacteria and Drums with bacteria. **A)** Results obtained with the CNN implementation. **B)** Results obtained with the best SVM implementation. **C)** Results obtained with the Fully Connected Neural Network with Principal Component Analysis (PCA) implementation.

At this specific stage in the decision tree, each of the employed algorithms maintained commendable accuracy levels. The CNN method registered an accuracy of 96.05%, and the learning pattern observed suggests effective feature recognition via convolution. Nevertheless, the loss versus epochs graph displayed slight indications of overfitting, evidenced by a less smooth convergence compared to other examined cases. In parallel, the SVM method reported an accuracy of 96.08%. Given SVM's inherent strength in handling unbalanced class distributions, it might demonstrate superior performance on new, unseen data compared to the CNN method. Meanwhile, the Fully Connected Neural Network incorporating PCA continued to exhibit substantial accuracy. Yet, its performance lagged behind the other two spectrogram-based algorithms. In addition, in the fully connected neural network employing PCA, the non-smooth learning curve observed during training in the accuracy vs epochs graph can be at-

tributed to factors such as PCA feature selection, learning rate dynamics, batch size variability, and potential dataset inconsistencies.

6.3.3. Parameters of the Support Vector Machine Algorithm

Parameters implemented for SVM algorithm				
Parameter	Cavities with and without Graphene	Drum with and without Bacteria	Bacteria Type Classification	Resistance Susceptibility Identification
`kernel`	`linear`	`linear`	`linear`	`poly`
`C`	`2.0`	`0.1`	`0.5`	`1.0`
`max_iter`	`-1`	`-1`	`-1`	`-1`

Table 6.1: Implemented parameters used in the SVM algorithm in study cases to differentiate empty cavities vs intact drums, drums with and without bacteria, for Bacteria species identification and for susceptibility tests.

7

Appendix

7.1. Short-Time Fourier Transform

The Fourier Transform, is a fundamental tool in signal processing and image analysis, which operates by decomposing a complex signal into its constituting frequencies, essentially converting data from the time or spatial domain to the frequency domain. For a continuous function of time $f(t)$, the Fourier Transform $F(\omega)$, is obtained by calculating Equation 7.1, where ω is the angular frequency. This process enables the extraction of frequency-based information from the signal, which can reveal underlying periodic patterns or resonances that are less discernible in the original time-domain representation [22].

$$F(\omega) = \int_{-\infty}^{\infty} f(t) \cdot e^{-jwt} dt \tag{7.1}$$

Although is an invaluable tool, it has its own limitations. One primary issue arises when dealing with non-stationary signals, as the Fourier Transform assumes that signals are stationary and thus provides no temporal information, meaning it cannot discern when specific frequency components occur. In such scenarios, the Short-Time Fourier Transform (STFT) can offer a substantial advantage [22].

$$F(\tau, \omega) = \int_{-\infty}^{+\infty} f(t) w(t - \tau) e^{-i\omega t} dt \tag{7.2}$$

As seen in Equation 7.2, The STFT analyzes a small window $w(t - \tau)$ of the signal at a time, where τ is a translation parameter, thereby enabling the capture of time-varying frequencies within the signal. This operation involves performing a series of Fourier Transforms on these windows, which overlap to provide a time-continuous representation. By partitioning the signal into these windows, the STFT can give information about both frequency and time. In addition, it can highlight when certain frequencies occur in the signal, which is pivotal when analyzing biological signals, where frequency components evolve over time. Nevertheless, it is crucial to underscore that the STFT inherently involves a compromise between time and frequency resolution. Specifically, reducing the window size improves the time resolution at the cost of diminishing frequency resolution, and conversely, a larger window enhances frequency resolution while decreasing the precision in the time domain. [30].

7.2. Artificial Intelligence and Machine Learning

Artificial Intelligence

Artificial Intelligence (AI) is a field that combines computer science and physiology, aimed at making computers behave in a way that resembles human intelligence. Intelligence can be defined as the computational aspect of the ability to achieve goals in the world, which encompasses a range of cognitive abilities such as thinking, imagining, creating, memorizing, understanding, recognizing patterns, making choices, adapting to change, and learning from experience. The goal of AI is to enable computers to perform these human-like tasks in a more efficient and time-saving manner. AI technologies are used in a wide range of applications, from simple expert systems to more complex applications such as autonomous robots, natural language processing, and computer vision [35].

Machine Learning

On the other hand, Machine Learning (ML) is a subfield of AI that focuses on the development of algorithms and statistical models that enable computers to ”learn” from data and improve their performance on a specific task over time. Machine learning algorithms use statistical techniques to give computer systems the ability to learn with data, without being explicitly programmed. The goal of machine learning is to develop algorithms that can automatically improve their performance by learning from these provided data [35].

Therefore, in this context, the term ”learning” refers to the process by which a model is trained to perform a task by iteratively improving its accuracy through experience, i.e, by the input of new data. Thus, the objective of the learning stage in machine learning is to enable the model to generalize from the examples it has been trained on to new, unseen examples. This is done through adjusting the parameters of the model based on the performance on a validation set, with the goal of minimizing the error or loss between the predictions of the model and the actual outcomes. As the model continues to learn from its experiences, it can produce increasingly accurate predictions and make more informed decisions [36].

Supervised and Unsupervised Machine Learning

Unsupervised machine learning is a type of machine learning algorithm in which the system is trained on a dataset that contains only input data, and no corresponding output variables. In this case, the approach is to identify patterns or relationships within the input data. Once this relations are found, the algorithm attempts to group similar instances together and discover hidden structures in the data. Examples of unsupervised machine learning algorithms include clustering and dimensionality reduction. Clustering, for instance, is a commonly utilized technique in various applications, including Google News, where it groups news articles from the web and categorizes them into related collective stories [36].

Supervised machine learning, on the other hand, is a type of machine learning algorithm that trains on a labeled dataset, which contains both input data and corresponding output variables. To begin, a model that can accurately predict the output variables given the input data is built. The algorithm uses the labeled data to learn the relationship between the input and output variables. Once the model is trained, it can be used to make predictions on new, unseen data. This type of algorithms are typically used in problems that include regression, Decision Trees, and Support Vector Machines [36].

Training, Validation and Test Datasets

Prior to an exploration of prevalent machine learning algorithms, it is essential to comprehend a crucial procedure in regards to the utilization of datasets by these algorithms. One of the fundamental aspects of machine learning involves the utilization of a training set, which consists of a collection of examples of data and their corresponding outcomes. The objective is to develop a model that can predict outcomes based on the given data. The process of learning involves the application of algorithms that possess a set of parameters, and the task of learning requires the inference of parameters that are consistent with the training data. To achieve this, the dataset is typically divided into three distinct groups: the training set, the validation set, and the test set [20].

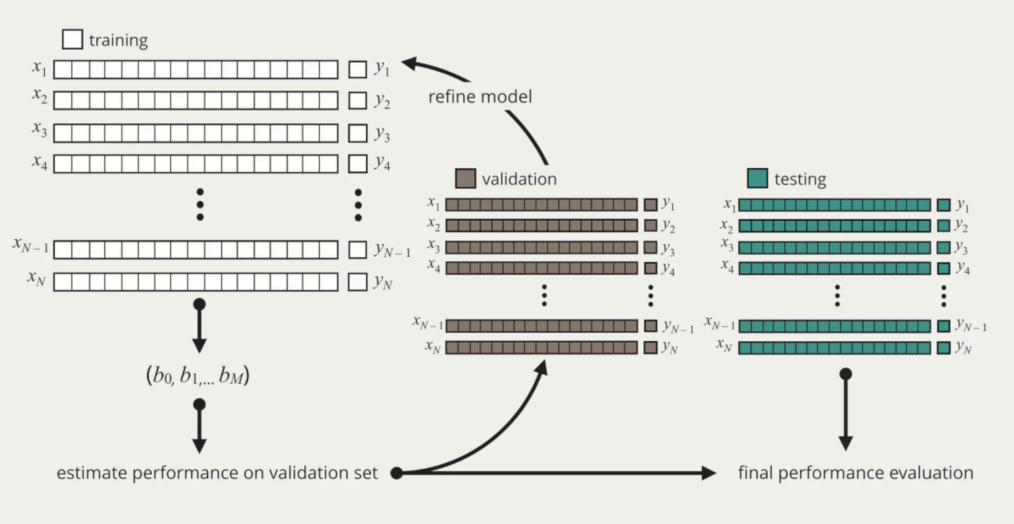

Figure 7.1: Illustration of how the dataset is divided in 3 different datasets in Machine Learning. The dataset used for training, the validation set used for efficiency validation and the test set used for evaluation of the performance of the model when using new unseen data [20].

The training set consists of input data and the corresponding desired outputs. The model uses the training set to learn the relationship between the inputs and outputs, and adjust its parameters to minimize the difference between the predicted outputs and the actual outputs. In addition, the validation set is another subset of the dataset that is used to tune the parameters of the model. The validation set is used to evaluate the efficiency of the model, and determine which combination of parameters results in the best performance. Here it is important to note that the model is not trained on the validation set, but its performance on the validation set is used to make decisions about how to adjust the parameters. Finally, the test set is a final subset of the dataset that is used to evaluate the performance of the model. However, the test set is used to evaluate the ability of the model to generalize to new, unseen data. This means that the model does not use the test set during the training or validation phases, and its performance on the test set is used to measure the overall accuracy of the model [20].

7.3. Machine Learning Algorithms

7.3.1. Support Vector Machines

Support Vector Machines (SVM) is a type of supervised learning algorithm that is commonly used for classification and regression analysis. It is based on the concept of finding the maximum margin hyperplane that separates the data points into different classes. The hyperplane is chosen such that it maximally separates the data points, while also ensuring that the distance between the hyperplane and the closest data points, known as support vectors, is maximized. During the training procedure, the SVM algorithm tries to find a boundary that separates the data into different classes, while also making sure that the boundary is as far away from the data points as possible. This boundary is referred to as the maximum margin hyperplane, and it is computed using mathematical optimization techniques [25].

This algorithm is particularly useful when the data is not linearly separable, and it can still perform well in these cases by using a non-linear transformation of the input data and mapping it into a higher-dimensional feature space. This is achieved through the use of a kernel function, which is a mathematical function that transforms the input data into a higher-dimensional space where a linear boundary can be found. This makes SVM a powerful tool for complex classification problems, as it can handle non-linear relationships between the features and the target. In addition, SVM can be used for both binary and multi-class classification problems and can also be used for regression problems by modifying the implemented optimization objective function. Finally, this algorithm is known for having good generalization performance, meaning that they can perform well on unseen data, and they are robust to outliers in the data [25].

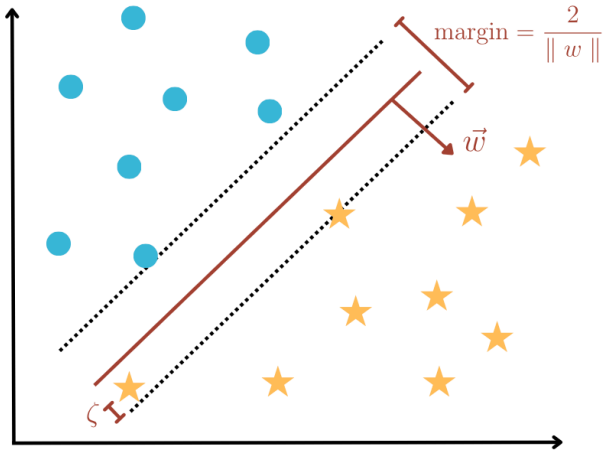

Figure 7.2: 2D Illustration of the boundary decision hyperplane in a Support Vector Machine algorithm. The red continuous line is the hyperplane, while the lack dotted lines represent the support vectors or margin size of the decision boundary.

If the equation of the continuous red hyperplane line is calculated, the following expression can be found:

$$w_1x_1 + w_2x_2 + ... + w_ix_i + b = 0 \quad \rightarrow \quad \vec{w}\vec{x} + \vec{b} = 0$$

By using this expression, it is possible to find the equation for the margin size of the support vectors. This can be done by adding to the equation of the hyperplane a step k that goes in the direction of $\vec{w}$ until one the support vectors is reached. Then an unit vector is added so that the step k is:

$$\vec{w}\vec{x} + \vec{b} = 0 \quad \rightarrow \quad \vec{w}\left(\vec{x} + k\frac{\vec{w}}{\parallel \vec{w} \parallel}\right) + \vec{b} = 0 \quad \rightarrow \quad \vec{w}\vec{x} + k\frac{\vec{w}}{\parallel \vec{w} \parallel} + b = 0 \quad \rightarrow \quad k = \frac{1}{\parallel \vec{w} \parallel}$$

However, as the margin consists of the entire area of the decision boundary, then the final equation of the margin is:

$$\text{margin} = \frac{2}{\parallel \vec{w} \parallel} \tag{7.3}$$

Therefore, the calculation becomes an optimization problem where the margin needs to be maximized. Mathematically, the optimization problem can be explain given the training vectors $x_i \in \mathbb{R}$ and the vector $y \in \{1, -1\}^n$. Here, the objective is to find $w \in \mathbb{R}$ and $b \in \mathbb{R}$ such that the prediction given by $sign(w^T\phi(x_i) + b)$ is correct for most samples. Therefore, the SVM algorithm would solve the problem stated in Equation 7.4:

$$\min_{w,b,\zeta} \ \frac{1}{2}w^T w + C\sum_{i=1}^{n}\zeta_i \tag{7.4}$$

$$\text{subject to } \ y_i(w^T\phi(x_i) + b) \geq 1 - \zeta_i$$

Where w is the vector from the decision boundary plane up to the support vector, C is the regularization term, ζ is the maximum allowed distance for a sample to be inside the decision boundary. These two last terms are added due to the fact that classification problems are not usually perfectly separable with an hyperplane under every circumstance. Therefore, the ζ term is used to allow some samples to be at a distance from the correct margin boundary or support vector while the regularization term C, or penalty term, controls the strength of this penalty, and as a result, acts as an inverse regularization parameter [37].

7.3.2. Logistic Regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable, in which there are only two possible outcomes. This algorithm is typically used to predict a binary outcome given a set of independent variables. However, implementations for multiclass classification are also possible. In this particular computational tool, the dependent variables are modeled as a function of the independent variables, and a logistic function is used to represent the relationship between them. The method is designed to map the input values to a value between 0 and 1, which can then be interpreted as the probability of the dependent variable taking a certain value [25].

The logistic regression algorithm is trained using a maximum likelihood estimation technique, which finds the coefficients that maximize the likelihood of the observed data given the model. Once the coefficients have been estimated, the logistic regression model can be used to make predictions about new observations, by plugging in the values of the independent variables and computing the predicted probability of the dependent variable with a determined value.

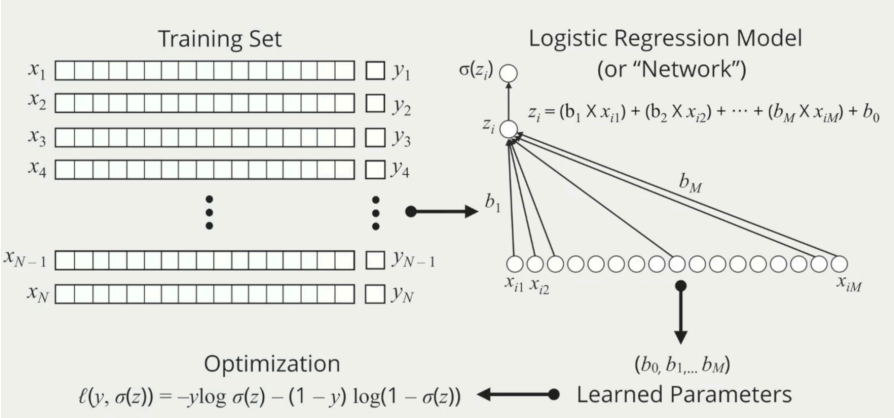

Figure 7.3: Illustration of the learning process used in Logistic Regression. This consists in using the training set in the inputs of the algorithm, so that its log-likelihood would determine how much the parameters would need to be changed in the next iteration [20].

As this algorithm only has one layer of latent processes, i.e., that there is a single underlying feature or factor that that influences the dependent variable. Its mathematical explanation can be easily explained. In the first stage, the algorithm would performed a linear combination between all the input values x_i and the automatically predefined weights b_M, which correspond to the importance of each input variable. A b_0 bias value is also added.

$$z_i = (b_1 \cdot x_i) + (b_2 \cdot x_2) + ... + (b_M \cdot x_M) + b_0$$

Once the mapping from the feature variables to the variable z_i is calculated, then the value of z_i is inputted into a logistic equation $\sigma(z_i)$, which can also be seen as a predicted probability function. This function is used, as it provides a more confident perspective on the prediction. Rather than simply assigning a prediction to a specific class, the function assigns a probability to the prediction, indicating the level of confidence with which it belongs to a determined class, which offers a more refined understanding of the prediction and its associated level of confidence [20].

$$\sigma(z_i) = \frac{L}{1 + e^{-k(z_i - z_0)}} \tag{7.5}$$

The standard logistic function is represented by Equation 7.5. Variations of this function can be derived based on the values of the parameters L, k, and z_0. For instance, when $L = 1$, $k = 1$, $z_0 = 0$, the resulting function is known as the Sigmoid function. If this function is used with the results of z_i, the outcome would be a number between 0 and 1. Therefore, when the value of z_i is significantly positive, the Sigmoid function converts it into a value that is close to one, indicating high confidence that it will belong to a class, given the value of z_i. In contrast, when z_i is highly negative, the output of the Sigmoid function is close to zero, which indicates a low probability of belonging to another class [38].

In the learning process, the objective is to determine the values of the learned parameters vector $\vec{b} = (b_1, b_2, ..., b_M)$, that result in an optimal performance of the algorithm. This is accomplished through the application of Empirical Risk Minimization, which requires the definition of a performance metric. To this end, a Loss Function is introduced into the algorithm, which measures the discrepancy between the true outcome and the prediction made by the model. Thus, the loss function is designed to penalize poor predictions and it is used as a measure of the discrepancy between the predicted and actual outcomes. An example of a loss function is

the Negative Log-Likelihood, also called Cross Entropy Loss, which is shown in equation 7.6 for a binary classification problem:

$$\mathcal{L}\{y_i, \sigma(z_i)\} = -y_i \cdot \log(\sigma(z_i)) - (1 - y_i) \cdot \log(1 - \sigma(z_i)) \tag{7.6}$$

Where y_i is the true label and $\sigma(z_i)$ is the predicted probability of belonging to a class. The idea behind the implementation of a Loss Function is that once a great prediction is done, then the algorithm would pay a small loss value. In contrast, whenever there is a poor prediction, the algorithm would pay a high loss value. Thus, a low loss value indicates an accurate prediction, while a high loss value suggests a poor prediction as seen in Figure 7.4. Ultimately, the objective is to minimize the average loss value to achieve an optimal performance. This is performed by modifying the values of learned parameters vector $\vec{b}$, so that the error function is decreased as much as possible [20]. This recalculation is achieved by using equation 7.7:

$$\vec{b} = \arg\min_{b} \frac{1}{N} \sum_{i}^{N} \mathcal{L}\{y_i, \sigma(z_i)\} \tag{7.7}$$

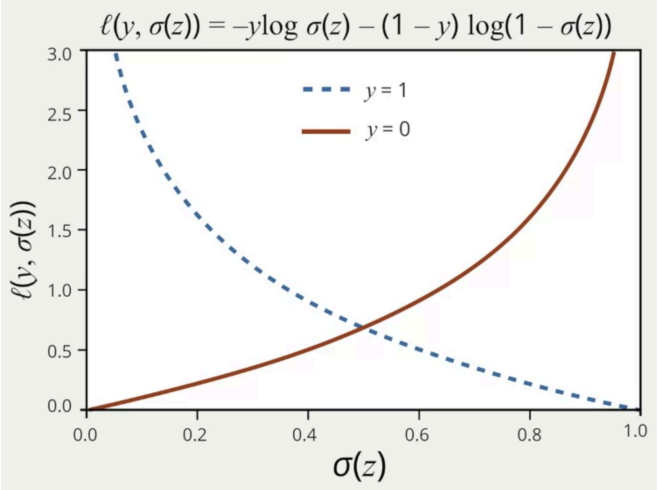

Figure 7.4: Graph of the Negative Log-Likelihood for a binary classification problem. It can be seen how an accurate prediction has a low loss value, while a poor prediction has a high loss value [20].

It can be seen from Equation 7.7 that the calculation of the optimal learned parameters $\vec{b}$ deeply depends on the minimization of the average of the implemented loss function.

7.3.3. Neural Networks

Neural networks, also called Multilayer Perceptron, are a type of machine learning model inspired by the structure and function of the human brain. They are composed of interconnected processing nodes called artificial neurons that process and transmit information. The processing nodes in a neural network are organized into multiple layers, where the input layer receives data and each subsequent layer performs progressively more complex computations. The output layer produces the final prediction based on the computations performed by the other layers [20]. The artificial neurons in a neural network are connected through directed edges, which are associated with weights that represent the strength of the connection between neurons. During the training process, the weights are updated based on the error between the

predicted output and the actual output, using an optimization algorithm such as gradient descent. This process adjusts the strengths of the connections in order to improve the overall accuracy of the predictions of the network [20].

Neural networks are a natural extension of the Logistic Regression algorithm. However, it is better suited for a wide range of more complex applications, including image recognition, speech recognition, natural language processing, and predictive analytics. The versatility of neural networks is due to their ability to learn complex relationships in data and generalize to new examples, making them a powerful tool for solving complex problems in various domains as it can also support nonlinear decision boundaries [23].

Structure of a Neuron

A neuron is a mathematical model that is inspired by the structure and function of biological neurons in the nervous system. It is composed of four key elements: an input, a linear combination, an activation function, and an output. The input to a neuron is a set of numerical values, each representing a feature or characteristic of the data being processed. These inputs are multiplied by a set of weights, representing the importance of each input, and then combined through an activation function. Therefore, the activation function is responsible for determining whether the neuron will be ”activated” and produce an output, based on the weighted inputs received. In the end, this output represents the neuron's prediction or decision about the input data. Mathematically, a neuron can be represented by Equation 7.8:

$$y = f(w_1x_1 + w_2x_2 + ... + w_nx_n + b) \tag{7.8}$$

Where y is the output of the neuron, f is the activation function, w_i is the weight associated with each feature input x_i, and b is a bias term that allows the activation function to be shifted. At the beginning of the learning process, weights can be assigned to the input features according to their importance or prominence. Additionally, the activation function can be any non-linear function, such as the Sigmoid or the Rectified Linear Unit (ReLU) function, which allows for non-linear relationships between inputs and outputs to be captured. More details about the mathematical structure of these functions would be mention on Section 7.4. The output of a neuron is then passed on as input to other neurons in the network, allowing for the processing of multiple levels of abstractions and the extraction of increasingly complex features. The connections between neurons are weighted, and the weights are adjusted during training to minimize a loss function and optimize the performance of the network [20].

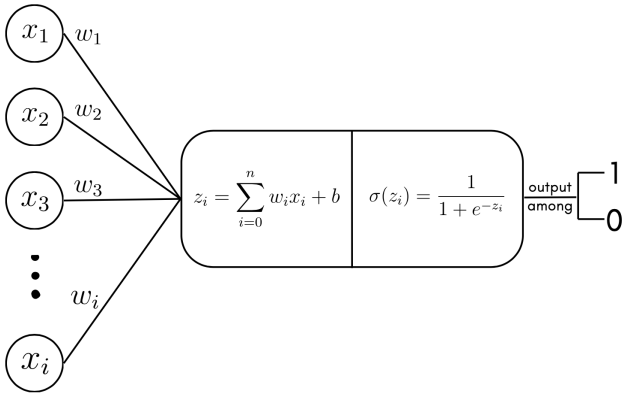

Figure 7.5: Illustration of the structure of a single neuron. Every neuron is composed by its inputs with their corresponding weights, a section that performed the linear combination, an activation function and its output.

In a neural network, instead of solely utilizing a single template vector to project the data points as it was done in logistic regression, a set of k filters or reference factors, denoted as b_1 to b_k, are considered as it can be seen in Figure 7.6. The inner product of each data point x_i with each of these filters b_1 to b_k is calculated, followed by the addition of biases consistent with the previously procedure used in logistic regression. The outputs of these operations, referred to as z_{i1} to z_{ik} in Figure 7.6, would become the k features from the first hidden layer. Afterwards, these features are then passed through a logistic function $\sigma(z_{ik})$, which maps real numbers to a range between 0 and 1, to introduce probability into the k latent features. Subsequently, these k latent probabilities are then processed through a logistic regression model once again, resulting in a single output ζ_i, calculated from the inner product of the latent features and a single template filter C_{ik}. Finally, ζ_i is then further passed through a final activation function $\sigma(\zeta_i)$, providing the probability of the data being associated with a particular binary label [28].

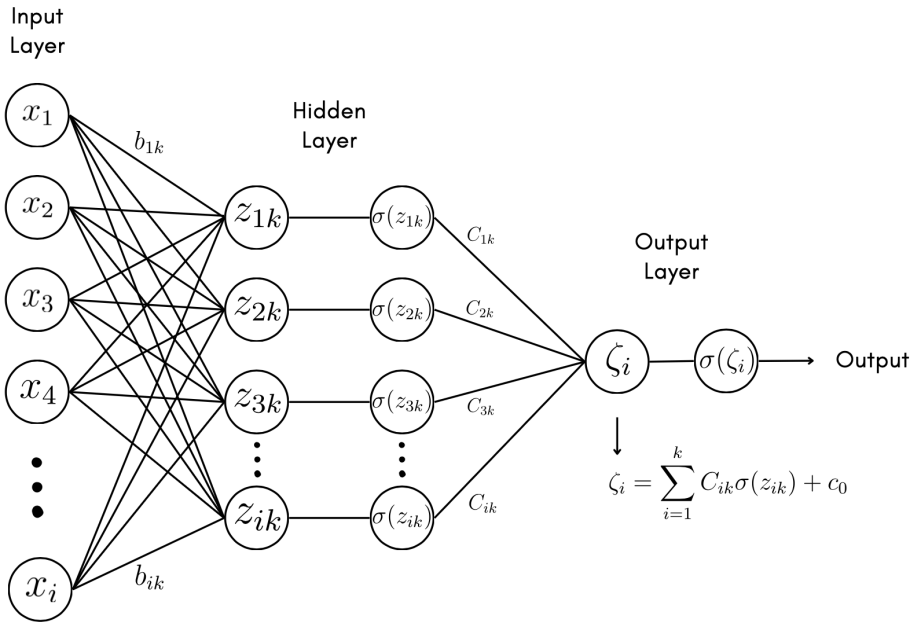

Figure 7.6: Structure of a simple neural Network with one single hidden layer. It can be seen how the hidden layer is represented by two continuous operations which correspond to the linear combination with its corresponding activation function calculation.

Once this procedure is finished, then the calculation of the error with a loss function is performed, to determine how accurate was the first prediction of the algorithm. Subsequently, this loss value is calculated and the algorithm starts an optimization procedure call Backpropagation (see section 7.6), where all the weights in the networks are modified in order to reduce the loss value in the next learning iteration. Upon completion of the learning process, the neural network is prepared to undergo testing with previously unseen data in order to generate predictions [20].

7.3.4. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of deep learning neural network that are commonly used for data that have spatial structures and correlations, this is why some popular applications for this model involve image classification and object recognition tasks. The fundamental idea behind CNNs is to extract meaningful features from the input image data through a series of convolutional, activation, and pooling operations. It is important to note that a CNN does not need to be entirely composed of convolutional layers, in fact, many popular CNN architectures are composed of different type of layers and typically end in a fully connected layer structure [23].

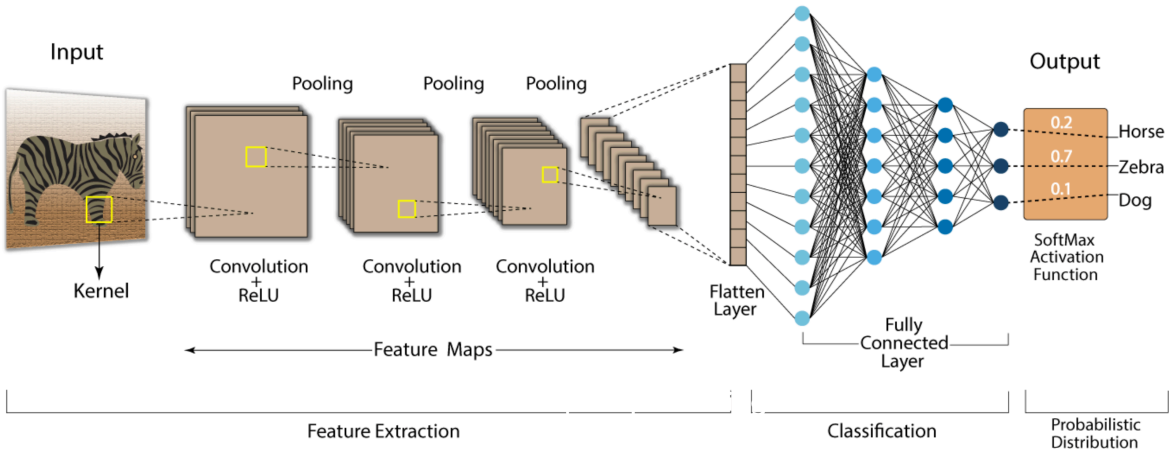

Figure 7.7: Structure of a Convolutional Neural Network. All basic structural components are being shown in the order in which they are typically implemented. That is, a Convolution layer, Pooling layers, a Flatten layer and a section with fully connected layers [26].

The architecture of a CNN typically consists of several distinct layers. The convolutional, activation, and pooling operations are repeated multiple times to create multiple feature maps, which are then fed into a fully connected layer to produce the final output. The weights of the filters and the biases in the fully connected layer are learned through the training process, where the objective is to minimize a loss function between the predicted output and the true output. In the following discussion, each of these structures will be explained in further detail.

Convolutional Layers

The convolution operation involves the extraction of essential features from an input image through the use of filters. The filters, also referred to as kernels or templates, are small matrices that scan through the input image and perform element-wise multiplications and accumulations to generate a set of intermediate feature maps. These feature maps are then passed through a activation function to introduce non-linearities into the model.

In image processing and computer vision, a motif is a recurring pattern or structure in an image. Motifs represent the basic building blocks of an image, and they can be used to extract important information and identify objects or structures in an image. In machine learning, a motif is a feature that is learned by the network from the training data and is used to identify objects and patterns in the input image. These motifs can be combined to form more complex features, which help the network to learn how recognize increasingly complex patterns in the input image [23].

During the convolution, each motif would be used as an elemental filter of size k by k. This operation starts with the motif filter positioned at the top-left corner of the input image of size N by N. Then, the filter is moved one step at a time, either vertically or horizontally, performing element-wise multiplications and accumulations with the elements of the image that it covers. The result of this operation is stored in a single entry of a feature map of size $(N-k+1)\times(N-k+1)$. This process is repeated until the filter has scanned all regions of the input image. Finally, the outcome is a first layer feature map that capture the presence of certain features, such as edges, corners, or textures, in the input image through high-correlation regions. Subsequently, to generate higher-layer feature maps, the method aggregates the elemental motif filters to form more intricate patterns, resulting in the generation of higher-order features which would act as final filters in the later classification process. Convolutional Layers are typically formed by the following elements:

- Filter Size: Is the size of the filter utilized to obtain the feature map which typically ranges from (3x3) to (7x7). It is recommended to choose a size that is large enough to capture small local features, such as edges and spaces.
- Stride: This represents the amount of pixels that one filter would move at a time. A stride of one would make the filter move along the image on row of pixels at a time whereas a stride of 2 would make the filter move two rows of the image at a time. This element can help to reduce the computational load by down sampling the input image. Common values would be around 1 to 2.
- Filter Number: The amount of filters in the neural network. These represents the amount of unique features that one could be looked for in the input image so that every unique feature has its one independent feature map.
- Padding: The result of convolution operations often yield outputs larger than the input size, which is due to the kernel extending beyond the edges of the input. This issue can be resolved by adjusting the padding of the input. Usually, padding is employed to maintain the output's spatial dimensions equal to the input's, when the stride is set to 1. By using this, it is easier to keep track of the size of the model.

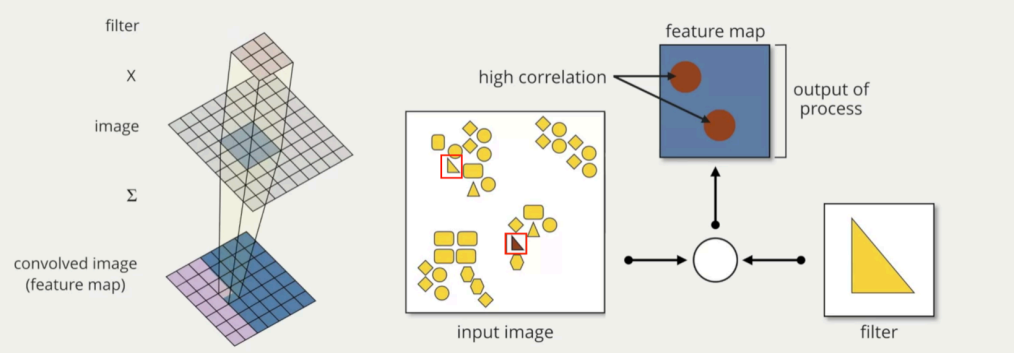

Figure 7.8: Illustration of the convolution algorithm. The triangle represents the motifs that are in every type of image. Once an elemental motif is found in the image while being scanned, a high correlation area is added to the current Feature Map [20].

Pooling layers

The pooling operation is then performed on the feature map to reduce its spatial dimension, while retaining the most important features. This process typically occurs in CNN architectures after the activation function in the convolutional layer. This operation is typically performed by taking the maximum value in a local neighborhood of the feature map, or by taking the average of the values in that neighborhood [20].

The pooling process is performed by the pooling layer, which utilizes a pooling filter. This filter functions similarly to a convolutional filter, but instead of conducting a convolution, it compresses all the pixel information within its window to a single value. This compression is achieved by taking either the maximum or average pixel value within the window of the pooling filter. For example, as seen in Figure 7.9, if the input is represented as a 4x4 matrix and a pooling layer employs a 2x2 max pooling filter with a stride of 2, the output will be a 2x2 matrix containing the maximum pixel values from the pooling filter's window. In this context, the stride refers to the step size with which the pooling filter moves over the input matrix. It determines the spatial resolution of the output produced by the pooling layer, as a larger stride results in a smaller output size. The stride affects the size of the output by skipping over some of the pixels in the input matrix, effectively reducing the size of the matrix [23].

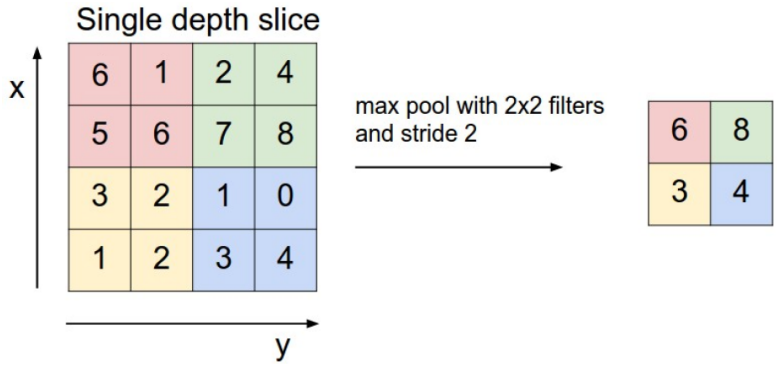

Figure 7.9: Example of a (2x2) Max Pooling Filter with a stride of 2

The use of pooling layers in a neural network presents several benefits, including reduction of computational complexity and facilitation of the training process. I addition, pooling layers also help to prevent overfitting during training while also encouraging the existence of translational invariance. In the context of machine learning, translational invariance refers to the ability of a neural network to still classify the identity of a particular feature of an image independently of its spacial location. This means that no matter where a particular feature is located inside an image, this feature is still being translated or transferred and the neural network can still determine its identity. This prevents to reduce the potential loose of information in the down sampling process of the image [20].

Fully Connected layers

In a typical convolutional neural network there are convolution operations in the convolutional layer to build feature maps. Then there are activation functions to introduce nonlinearities to the network. Subsequently, there is a pooling layer with pooling filters to down sample the feature maps. However, the network can have more subsequent convolutions and pooling layers. At the end of this process, the stacked results would create a high level representations of the features within the image. In order to process these high level features before the final

classification, a fully connected multilayer perceptron is used, with the implementation of a final set of the fully connected layers as seen in Figure 7.10 [20].

The process of inputting the high level features of the image into the dense layers involves flattening or vectorizing the feature matrices produced in the final pooling layer. An illustration of this process is shown in 7.10 with the presence of an intermediate latent or hidden layer of neurons that are connected to the elements in the pooling layer. Furthermore, a fully connected readout layer, consisting of class readout neurons, is connected to the latent hidden layer. This structure enables the algorithm to provide meta feature representations that can effectively address classification problems.

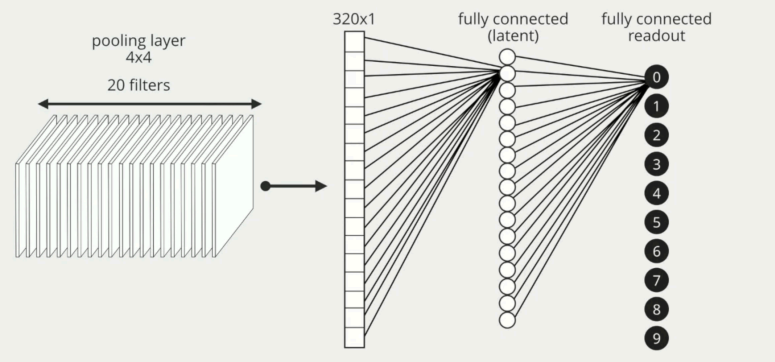

Figure 7.10: Example of a set of fully connected layers after a pooling layer in a Convolutional Neural Network. For simplicity, in the image only the connections for the first neuron of each hidden and output layer is being shown. However, every neuron is fully interconnected to its neighbor neurons in other layers.

Transfer Learning

It is important to note the significance of transfer learning. It involves utilizing the already learned parameters of a convolutional neural network that has been trained on a broad image dataset, in a more specialized model. This way, the need for learning all parameters from scratch is eliminated, and only the parameters specific to the desired classification task need to be learned. This is possible because elementary features present in an image of a cat, for instance, are similar to those present in an image of a person. These elemental low-level features are universal across all images.

The power of transfer learning lies in the ability of a convolutional neural network, once trained to identify these elementary low-level features in a large data set, to apply the learned features to quickly learn the higher-level and more specific features in a new desired classification task. This results in a significant increase in the speed of the entire training process.

7.4. Activation functions

Activation functions are used to regulate the output of neurons in a neural network. Inside a neuron, they are applied after the linear combination of features and weights, and provide a nonlinear transformation to the input. This is necessary because linear functions alone are insufficient to solve complex problems, and adding nonlinearities via activation functions helps the neural network better capture complex relationships. The use of activation functions allows for the creation of nonlinear relationships between layers, preventing the collapse of multiple linear transformations into a single linear one and enabling the neural network to approximate more complex functions and solve more challenging problems [27].

When evaluating an activation function, the most significant aspect to consider is its threshold-based classification, which determines whether a neuron is activated or deactivated based on whether the input to the activation function surpasses a specified threshold value. If the input does not exceed the threshold, the neuron is deactivated, and the output is not passed on as input to the next layer [20]. In the rest of this section some activation functions would be explained.

7.4.1. Binary Step Function

The Binary Step Function is the simplest activation function that can be implemented with simple if-else statements in Python. It is commonly utilized in the creation of binary classifiers. However, it is not suitable for use in the context of multiclass classification, as it cannot accurately represent a target variable with multiple classes. Additionally, the gradient of the Binary Step Function is zero, which can lead to hindrance in the backpropagation step as the derivative of the function with respect to x is equal to zero [27]. Mathematically, this activation function can be express as shown in Equation 7.9:

$$\sigma(z_i) = \begin{cases} 1, & z_i \geq 0 \\ 0, & z_i < 0 \end{cases} \tag{7.9}$$

7.4.2. Sigmoid Function

It is one of the most widely used activation functions due to its nonlinear properties. This function has the advantage of providing a smooth and continuous output, which makes it continuously differentiable. This enables an easier to calculation of gradients during the backpropagation phase of training a neural network. Moreover, the Sigmoid function can provide a clear interpretation of the output as a probability between 0 and 1, making it easy to interpret the results of binary classification problems, while facilitating the creation of a threshold line suitable for a classification problem. Mathematically, the Sigmoid function can be written as in Equation 7.10:

$$\sigma(z_i) = \frac{1}{1 + e^{-z_i}} \tag{7.10}$$

However, the Sigmoid function also has some disadvantages. One of the main drawbacks of the Sigmoid function is that it has a saturation effect, meaning that the output of the function becomes close to either 0 or 1, causing the gradients to become very small, this issue is called Vanishing Gradients. This can cause slow convergence, which can result in poor training performance of the network. Additionally, the sigmoid function is not zero-centered, meaning that the mean of the activation function is not zero, which can lead to asymmetrical behavior in the network. Another disadvantage is that the Sigmoid function is not well suited to model multiclass problems, as it only outputs a single probability value. In those cases, other activation functions such as the Softmax function are typically used [27].

7.4.3. Softmax Function

This function is a commonly used activation function, specifically in the output layer of a multiclass classification problem. Its main purpose is to convert a vector of arbitrary real values into a probability distribution over several classes. Thus, the function operates by squashing the input values into the range between 0 and 1, while ensuring that the sum of the values is equal to 1. Mathematically, the Softmax function is defined as:

$$\sigma(z_i) = \text{Softmax}(y_i) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_j}} \tag{7.11}$$

where z_i is a vector of real values, i is the current index, and j is over all other indices. The exponential operator (e^{z_i}) is applied to each input value, and then the resulting values are normalized by dividing by the sum of all exponentials. The output of the function will be a set of values that add up to 1 and can be interpreted as probabilities of each class. One important property of the Softmax function is that it produces outputs that are proportional to the relative probabilities of each class. This means that if two classes have a large difference in their input values, the Softmax function will produce a large difference in their output values, which reflects the confidence of the prediction [27].

7.4.4. Rectified Linear Unit: ReLU Function

The Rectified Linear Unit (ReLU) is a commonly used activation function in the field of machine learning. It is defined as a piecewise linear function, where for an input value greater than zero, the output is equal to the input value. In constrast, for an input value less than or equal to zero, the output is zero. The ReLU function can be mathematically represented as shown in Equation 7.12:

$$\sigma(z_i) = \text{ReLU}(z_i) = \max(0, z_i) \tag{7.12}$$

ReLU is favored for its simplicity and computational efficiency compared to other activation functions, particularly in between hidden layers. It can also alleviate the vanishing gradient problem in deep neural networks, where the gradient becomes extremely small and makes it difficult for the network to learn. However, the ReLU function has a drawback known as the ”dying ReLU” problem, where if a ReLU neuron receives a negative input, it remains inactive and produces a constant output of zero. This can cause a large portion of the network to become ineffective and negatively impact the learning process. To address this issue, variants of the ReLU function have been proposed, such as the leaky ReLU [27].

$$\sigma(z_i) = \text{ReLU}(z_i) = \max(0, x) \tag{7.13}$$

7.5. Loss functions

In a neural network, the loss function plays a crucial role in determining how well the model is performing. It provides a measure of how far the predictions of the model are from the true values in the training data. As mention in prevous sections, the objective of the training process is to minimize the average value of the loss function, so that the prediction error of the model is as small as possible. This is done by updating the weights of the model and biases in the direction of minimizing the loss. The gradient of the loss function is used to determine the direction of update, and optimization algorithms such as gradient descent are used to perform the update [20].

The loss function is a scalar value that summarizes the difference between the predictions of the model and the true values in the training data. It is calculated for each training example and then averaged over the entire training set. The choice of loss function depends on the specific problem being solved and the type of data being used. This function are used as a feedback signal for the optimization algorithm to adjust the parameters of the neural network. During training process, the optimization algorithm iteratively updates the parameters of the

model to minimize the value of the loss function. Once the optimization algorithm has found the optimal set of parameters, the model can be used to make predictions on new, unseen data [20]. In the rest of this section, some popular loss functions are explained.

7.5.1. Mean Square Error

The Mean Squared Error (MSE) loss function is a commonly used measure of the difference between the actual and predicted outputs of a machine learning model, particularly in regression problems. In a neural network, the MSE loss is calculated as the average of the squared differences between the predicted outputs and the actual outputs. This function can be expressed as in Equation 7.14:

$$\mathcal{L}\left\{y_i, \sigma(z_i)\right\} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y_i})^2 \tag{7.14}$$

The MSE loss function is widely used because it is differentiable and convex, which makes it easy to optimize. It also provides a clear measure of the accuracy of the predictions of the model. However, the MSE loss function can be sensitive to outliers, meaning that a single large error in the prediction can significantly impact the overall loss value. This can be mitigated by using a robust loss function, such as the Huber loss, which is less sensitive to outliers [39].

7.5.2. Cross-Entropy

The Cross-Entropy loss function is a widely used especially in the field of deep learning. It is primarily used for training multi-class classification models, as it provides a measure of the distance between the predicted class probabilities and the true class label. The basic idea behind Cross-Entropy is to penalize the model for making confident, but incorrect predictions. This is achieved by calculating the negative logarithm of the predicted class probabilities and averaging it over all the classes. The resulting loss is then backpropagated through the neural network to update the model's parameters and reduce the prediction error. The Cross-Entropy loss function is defined as follows in Equation 7.15:

$$\mathcal{L}\left\{y_i, \sigma(z_i)\right\} = -\sum_{i} y_i' \cdot \log(y_i) \tag{7.15}$$

Where y_i is the set of probabilities predicted by the model and y_i' is the true label, which is what we wanted the model to predict. Thus, equation 7.15 measures the average information content or uncertainty of the predicted class probabilities, relative to the true class label. Minimizing this loss function results in the model making more accurate predictions, as it converges towards the true class label [39].

7.5.3. Sparse Categorical Cross Entropy

This is a variant of the Cross-Entropy loss function, with the main difference being that the target values are given as integers instead of one-hot encoded vectors, which makes the model computationally more efficient. This function calculates the difference between the predicted class probabilities and the true class labels. This is done by first converting the true class labels into a one-hot encoded vector and then computing the Cross-Entropy loss between this vector and the predicted class probabilities. Subsequently, the loss value is computed for each example in the batch and then averaged over the batch size to obtain the final loss value for a single training step [39].

$$\mathcal{L}\{y_i, \sigma(z_i)\} = -\sum_i y_i' \cdot \log(y_i) \tag{7.16}$$

Being C the amount of classes in the classification and y_{int} the integer label for a data point (i.e., an integer between 0 and C−1), then the one-hot encoded label y can be derived as:

$$\begin{cases} 1 & \text{if } c = y_{\text{int}} \\ 0 & \text{otherwise} \end{cases}$$

7.6. Backpropagation Algorithm

The backpropagation algorithm is a method used to update the weights in a neural network. It is a supervised learning algorithm that is used in the training process of a neural network. The goal of backpropagation is to minimize the error between the predicted output of the neural network and the actual target output. The algorithm accomplishes this by updating the weights of the network in such a way that the prediction error is reduced. As a first step, the algorithm starts by making a forward pass through the network, which involves passing the input through the network and making predictions using the current weights. After making predictions, the algorithm calculates the error with the loss function between the predicted output and the actual target output. This error is then propagated backwards through the network, where the weights are updated in such a way that the error is reduced [24].

Subsequently, the algorithm uses the gradient descent optimization algorithm to update the weights of the network. The gradient descent algorithm calculates the gradient of the error with respect to the weights, and updates the weights in the direction of the negative gradient. This reduces the error between the predicted output and the actual target output, and eventually leads to convergence to a minimum error value. Finally, the algorithm iteratively updates the weights until the error between the predicted output and the actual target output is reduced to an acceptable level. This procedure is considered to be computationally efficient and is widely used in training neural networks. It is one of the most important algorithms in deep learning, and is the basis for many popular deep learning models such as convolutional neural networks and recurrent neural networks [39].

7.6.1. Optimizers

During the error backpropagation algorithm, that is, when the training is being accomplished, the optimizer allows to train the neural network efficiently. Optimization algorithms in backpropagation algorithm are used to update the model's parameters, such as weights and biases, based on the gradients computed during the backpropagation process. There are various optimization algorithms such as Stochastic Gradient Descent, Adagrad, Adadelta, Adam, among others. The choice of optimization algorithm often depends on the specific problem and data being modeled [24].

Gradient decent Method

The Gradient Descent Method that involves iteratively updating the weights of the network in the direction of the negative gradient of the loss function with respect to the weights. The magnitude of the update is determined by a learning rate, which determines the step size of the update. The method is based on the idea of continuously decreasing the error by moving towards the minimum of the loss function. This iterative process continues until the gradient

of the loss function with respect to the weights is close to zero or a stopping criteria is met. The Gradient Descent Method is widely used due to its simplicity and computational efficiency. The update rule for the parameters in gradient descent is given by the following equation:

$$w_{t+1} = w_t - \alpha \cdot \nabla \mathcal{L}(w_t) \tag{7.17}$$

where $w(t)$ represents the model parameters at iteration t, $\nabla \mathcal{L}(w(t))$ is the gradient of the cost function with respect to the parameters and α is the learning rate, which determines the size of the update step [40].

Standard Stochastic Gradient Decent

Standard Stochastic Gradient Descent (SGD) is a variation of the Gradient Descent optimization algorithm. This algorithm randomly selects a single samples from the training data and compute the gradient with respect to the loss function for that single samples. This gradient is then used to update the model parameters in a single step. Subsequently, the process is repeated iteratively, using a different random samples from the training data each time, until the model reaches convergence, or the training accuracy reaches a satisfactory level. Mathematically, it can be represented as shown in Equation 7.18:

$$w_{t+1} = w_t - \alpha \cdot \nabla \mathcal{L}(w_t, x_i) \tag{7.18}$$

Where w_t is the weight of the model at iteration t, α is the learning rate, $\nabla \mathcal{L}(w_t, x_i)$ is the gradient of the cost function $\mathcal{L}$ with respect to the weights w_t for a single training example x_i. It can be seen that Equation 7.17 and Equation 7.18 are similar. However, the difference between the two methods lies in the way the gradients are computed. The standard Gradient Descent algorithm computes the average of the gradients over all the training examples, while the Stochastic Gradient Descent computes the gradient with respect to a single training sample. The use of a single sample in SGD results in a more efficient optimization process and also helps the algorithm to converge faster by providing a more stochastic optimization process, which can escape from local minima. However, the increased randomness in the optimization process also increases the risk of overfitting compared to gradient descent [40].

7.6.2. Adam Optimizer

The Adam optimizer is a popular optimization algorithm used in the training of neural networks in machine learning. It is a combination of two optimization methods, Stochastic Gradient Descent and Root Mean Square Propagation, which are combined to form a computationally efficient and effective optimization method. At its core, Adam optimizer uses a running average of the gradient, as well as a running average of the squared gradient, to dynamically adjust the learning rate for each weight in the network. This helps to avoid the oscillation of the optimization process and converge to the optimal solution more effectively. The learning rate is updated adaptively, allowing the algorithm to make larger updates early in the optimization process, when the gradient is still large, and smaller updates as the optimization process approaches convergence. In comparison to other optimization algorithms, such as SGD, the Adam optimizer has been shown to converge faster and provide more stable convergence in a wide range of applications. Additionally, the algorithm requires relatively little computational overhead, making it an attractive choice for deep neural networks and other large-scale machine learning problems [41].

7.6.3. Receiver Operating Characteristic (ROC) Curve

The ROC graph illustrates the performance of the model by showcasing the balance between the True Positive Rate or sensitivity (Equation 7.19) and the False Positive Rate which can be calculated with Equation 7.21.

$$\text{TPR} = \text{Sensitivity} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}} \tag{7.19}$$

$$\text{Specificity} = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)} + \text{False Positives (FP)}} \tag{7.20}$$

$$\text{FPR} = 1 - \text{Specificity} \tag{7.21}$$

The optimal ROC curve of a model approaches the top-left corner, denoting high sensitivity without loss of specificity. The Area Under the Curve (AUC) serves as a comprehensive metric to quantify this performance. An AUC with values close to 1, as seen in Figure 7.11, indicates superior discriminatory power, while values near 0.5 suggest no better performance than random chance. A steeper ROC curve signifies enhanced sensitivity with minimal specificity loss. Analyzing the ROC and AUC metrics for the studied algorithms provides insights into their precision in differentiating bacterial categories to determine the optimal approach choice, particularly in scenarios with class imbalances [42].

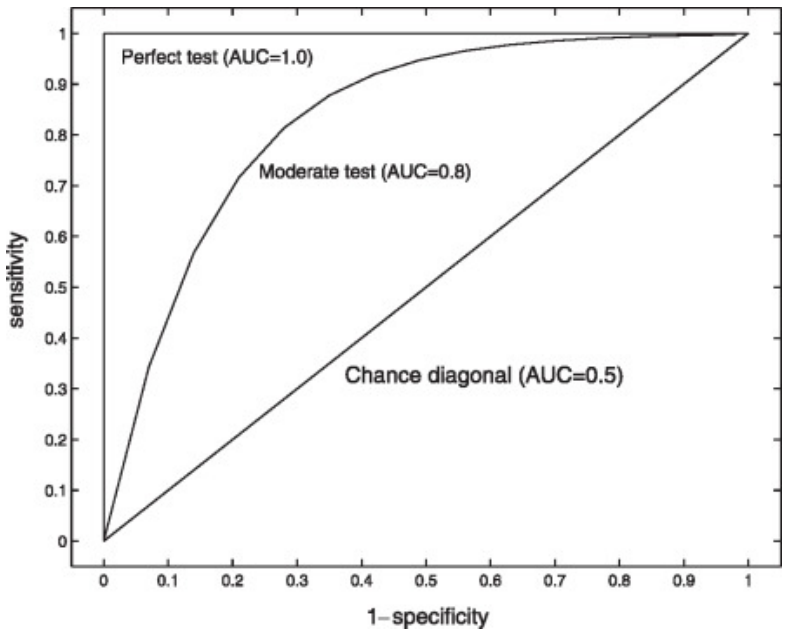

Figure 7.11: ROC curve displaying various scenarios. The AUC, or Area Under the Curve, measures diagnostic precision. An AUC of 0.5 signifies chance performance (represented by a diagonal line), while a maximum value of 1.0 reflects a perfect classifier, where sensitivity is maintained across all specificity values. [42]

References

1. Hrabák, J., Chudác □ková, E. & Walková, R. Matrix-Assisted Laser Desorption Ionization–Time of Flight (MALDI-TOF) Mass Spectrometry for Detection of Antibiotic Resistance Mechanisms: from Research to Routine Diagnosis. *Clinical Microbiology Reviews* **26,** 103. ISSN: 08938512. /pmc/articles/PMC3553667/%20/pmc/articles/PMC3553667/?report=abstract%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3553667/ (Jan. 2013).
2. Allicock, O. M. *et al.* Baccapseq: A platform for diagnosis and characterization of bacterial infections. *mBio* **9,** 1–10. ISSN: 21507511 (Sept. 2018).
3. March-Rosselló, G. A. Rapid methods for detection of bacterial resistance to antibiotics. *Enfermedades Infecciosas y Microbiología Clínica (English Edition)* **35,** 182–188. ISSN: 2529-993X. https://www.elsevier.es/en-revista-enfermedades-infecciosas-microbiologia-clinica-english-428-articulo-rapid-methods-for-detection-bacterial-S2529993X17300606 (Mar. 2017).
4. Fournier, P. E., Couderc, C., Buffet, S., Flaudrops, C. & Raoult, D. Rapid and cost-effective identification of Bartonella species using mass spectrometry. *Journal of medical microbiology* **58,** 1154–1159. ISSN: 0022-2615. https://pubmed.ncbi.nlm.nih.gov/19528172/ (Sept. 2009).
5. Hrabák, J., Walková, R., Študentová, V., Chudáčková, E. & Bergerová, T. Carbapenemase Activity Detection by Matrix-Assisted Laser Desorption Ionization-Time of Flight Mass Spectrometry. *Journal of Clinical Microbiology* **49,** 3222–3227. ISSN: 0095-1137 (Sept. 2011).
6. Biswas, S. & Rolain, J. M. Use of MALDI-TOF mass spectrometry for identification of bacteria that are difficult to culture. *Journal of Microbiological Methods* **92,** 14–24. ISSN: 0167-7012 (Jan. 2013).
7. Cockerill, F. R. *et al.* Optimal testing parameters for blood cultures. *Clinical Infectious Diseases* **38,** 1724–1730. ISSN: 10584838. https://academic.oup.com/cid/article/38/12/1724/304691 (June 2004).
8. Lee, A., Mirrett, S., Reller, L. B. & Weinstein, M. P. Detection of bloodstream infections in adults: how many blood cultures are needed? *Journal of clinical microbiology* **45,** 3546–3548. ISSN: 0095-1137. https://pubmed.ncbi.nlm.nih.gov/17881544/ (Nov. 2007).
9. Pallen, M. J., Loman, N. J. & Penn, C. W. High-throughput sequencing and clinical microbiology: progress, opportunities and challenges. *Current Opinion in Microbiology* **13,** 625–631. ISSN: 1369-5274 (Oct. 2010).
10. Xie, Y. *et al.* Influences of graphene on microbial community and antibiotic resistance genes in mouse gut as determined by high-throughput sequencing. *Chemosphere* **144,** 1306–1312. ISSN: 0045-6535 (Feb. 2016).
11. Ventola, C. L. The Antibiotic Resistance Crisis: Part 1: Causes and Threats. *Pharmacy and Therapeutics* **40,** 277. ISSN: 1052-1372. /pmc/articles/PMC4378521/%20https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4378521/ (2015).
12. Alonso-Sarduy, L. *et al.* Real-Time Monitoring of Protein Conformational Changes Using a Nano-Mechanical Sensor. *PLOS ONE* **9,** e103674. ISSN: 1932-6203. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0103674 (July 2014).

13. Scheuring, S. *et al.* High resolution AFM topographs of the Escherichia coli water channel aquaporin Z. *The EMBO Journal* **18,** 4981–4987. ISSN: 1460-2075. https://onlinelibrary.wiley.com/doi/full/10.1093/emboj/18.18.4981%20https://onlinelibrary.wiley.com/doi/abs/10.1093/emboj/18.18.4981%20https://www.embopress.org/doi/10.1093/emboj/18.18.4981 (Sept. 1999).
14. Kohler, A. C., Venturelli, L., Longo, G., Dietler, G. & Kasas, S. Nanomotion detection based on atomic force microscopy cantilevers. *The Cell Surface* **5,** 100021. ISSN: 2468-2330 (Dec. 2019).
15. Longo, G. *et al.* Rapid detection of bacterial resistance to antibiotics using AFM cantilevers as nanomechanical sensors. *Nature Nanotechnology 2013 8:7* **8,** 522–526. ISSN: 1748-3395. https://www.nature.com/articles/nnano.2013.120 (June 2013).
16. Pelling, A. E., Sehati, S., Gralla, E. B., Valentine, J. S. & Gimzewski, J. K. Local Nanomechanical Motion of the Cell Wall of Saccharomyces cerevisiae. *Science* **305,** 1147–1150. ISSN: 0036-8075 (Aug. 2004).
17. Rosłoń, I. E., Japaridze, A., Steeneken, P. G., Dekker, C. & Alijani, F. Probing nanomotion of single bacteria with graphene drums. *Nature Nanotechnology 2022 17:6* **17,** 637–642. ISSN: 1748-3395. https://www.nature.com/articles/s41565-022-01111-6 (Apr. 2022).
18. González, C., Jensen, E. W., Gambús, P. L. & Vallverdú, M. Poincaré plot analysis of cerebral blood flow signals: Feature extraction and classification methods for apnea detection. *PLoS ONE* **13.** ISSN: 19326203. /pmc/articles/PMC6286008/%20/pmc/articles/PMC6286008 / ?report = abstract % 20https : / / www . ncbi . nlm . nih . gov / pmc / articles / PMC6286008/ (Dec. 2018).
19. Hayakawa, K., Heima, A., Ozaki, M. & Yoshida, S. A Development of AI Predictive Maintenance System using IoT Sensing, 2021 (2021).
20. Duke University. *Introduction to Machine Learning - Coursera Course* https://www.coursera.org/learn/machine-learning-duke.
21. Tawhid, M. N. A. *et al.* A spectrogram image based intelligent technique for automatic detection of autism spectrum disorder from EEG. *PLOS ONE* **16,** e0253094. ISSN: 1932-6203. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0253094 (June 2021).
22. Gröchenig, K. *Foundations of Time-Frequency Analysis* ISBN: 978-1-4612-6568-9. http://link.springer.com/10.1007/978-1-4612-0003-1 (Birkhäuser Boston, Boston, MA, 2001).
23. Charniak, E. An Introduction to Deep Learning. *Perception* **48,** 759–761. ISSN: 0301-0066 (Aug. 2019).
24. Cilimkovic, M. Neural Networks and Back Propagation Algorithm.
25. Suthaharan, S. Support Vector Machine, 207–235. https://link.springer.com/chapter/10.1007/978-1-4899-7641-3_9 (2016).
26. Developers Breach. *Convolutional Neural Network - Deep Learning - Developers Breach* https://developersbreach.com/convolution-neural-network-deep-learning/.
27. Sharma, S., Sharma, S. & Athaiya, A. ACTIVATION FUNCTIONS IN NEURAL NETWORKS. *International Journal of Engineering Applied Sciences and Technology* **4,** 310–316. http://www.ijeast.com (2020).
28. Jurtz, V. I. *et al.* An introduction to deep learning on biological sequence data: examples and solutions. *Bioinformatics* **33,** 3685–3690. ISSN: 1367-4803. https://academic.oup.com/bioinformatics/article/33/22/3685/4092933 (Nov. 2017).

29. Mikołajczyk, A. & Grochowski, M. Data augmentation for improving deep learning in image classification problem. *2018 International Interdisciplinary PhD Workshop, IIPhDW 2018,* 117–122 (June 2018).

30. Nisar, S., Khan, O. U. & Tariq, M. An Efficient Adaptive Window Size Selection Method for Improving Spectrogram Visualization. *Computational Intelligence and Neuroscience* **2016.** ISSN: 16875273 (2016).

31. Sharda, R., Voß, S. & Suthaharan, S. *Machine Learning Models and Algorithms for Big Data Classification* tech. rep. (2016). http://www.springer.com/series/6157.

32. Shlens, J. A Tutorial on Principal Component Analysis. https://arxiv.org/abs/1404.1100v1 (Apr. 2014).

33. Groß, S. *et al.* Improved broad-spectrum antibiotics against Gram-negative pathogens via darobactin biosynthetic pathway engineering. *Chemical Science* **12,** 11882–11893. ISSN: 2041-6539. https://pubs.rsc.org/en/content/articlehtml/2021/sc/d1sc02725e%20https://pubs.rsc.org/en/content/articlelanding/2021/sc/d1sc02725e (Sept. 2021).

34. Navon-Venezia, S., Kondratyeva, K. & Carattoli, A. Klebsiella pneumoniae: a major worldwide source and shuttle for antibiotic resistance. *FEMS Microbiology Reviews* **41,** 252–275. ISSN: 0168-6445. https://dx.doi.org/10.1093/femsre/fux013 (May 2017).

35. Chhaya, K., Khanzode, A. & Sarode, R. D. Advantages and disadvantages of Artificial Intelligence and Machine Learning: A literature Review, 9–10. ISSN: 2277-3584. http://www.iaeme.com/IJLIS/index.asp30http://www.iaeme.com/IJLIS/issues.asp?JType=IJLIS&VType=9&IType=1JournalImpactFactor%20www.jifactor.comhttp://www.iaeme.com/IJLIS/issues.asp?JType=IJLIS&VType=9&IType=1.

36. Das, S., Dey, A. & Roy, N. Applications of Artificial Intelligence in Machine Learning: Review and Prospect. *International Journal of Computer Applications* **115,** 975–8887 (2015).

37. *1.4. Support Vector Machines — scikit-learn 1.2.1 documentation* https://scikit-learn.org/stable/modules/svm.html.

38. *Logistic Regression Scikit Learn 1.2.1 Documentation* https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

39. Wang, Q., Ma, Y., Zhao, K. & Tian, Y. A Comprehensive Survey of Loss Functions in Machine Learning. *Annals of Data Science* **9,** 187–212. ISSN: 21985812. https://link.springer.com/article/10.1007/s40745-020-00253-5 (Apr. 2022).

40. Gupta, N., Khosravy, M., Patel, N., Gupta, S. & Varshney, G. Evolutionary Artificial Neural Networks: Comparative Study on State-of-the-Art Optimizers, 302–318. https://link.springer.com/chapter/10.1007/978-981-15-2133-1_14 (2020).

41. Kingma, D. P. & Ba, J. L. Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings.* https://arxiv.org/abs/1412.6980v9 (Dec. 2014).

42. Devos, A. *et al.* Classification of Brain Tumours by Pattern Recognition of Magnetic Resonance Imaging and Spectroscopic Data. *Outcome Prediction in Cancer,* 285–318 (Jan. 2007).