UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN INGENIERÍA BIOMÉDICA

TRABAJO FIN DE MÁSTER

DESIGN AND DEVELOPMENT OF A DECISION SUPPORT SYSTEM TO PREDICT CARDIAC DISEASES ON 12-LEAD ECGs USING DEEP LEARNING

ANTONIO OCHOTORENA LAYNEZ

2021

TRABAJO FIN DE MÁSTER

Título:	Diseño y desarrollo de un Sistema de Ayuda a la Decisión para predecir enfermedades cardiacas en ECGs de 12 derivaciones usando técnicas de Aprendizaje Profundo
Título (inglés):	Design and development of a Decision Support System to predict cardiac diseases on 12-Lead ECGs using Deep Learning
Autor:	Antonio Ochotorena Laynez
Tutor:	Gema García Sáez
Departamento:	Tecnología Fotónica y Bioingeniería

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN
Departamento de Tecnología Fotónica y Bioingeniería

TRABAJO FIN DE MÁSTER

DESIGN AND DEVELOPMENT OF A DECISION SUPPORT SYSTEM TO PREDICT CARDIAC DISEASES ON 12-LEAD ECG USING DEEP LEARNING

Antonio Ochotorena Laynez

June - 2021

Agradecimientos

En primer lugar, me gustaría darle las gracias a Gema por dejarme participar en este gran proyecto.

Gracias a mi familia, por permitirme estudiar este máster y apoyarme en todo momento. Gracias a mis amigos, por animarme cuando nadie más puede.

Además, me gustaría darle las gracias a todos los profesores que me han enseñado a lo largo de todos estos años. Finalmente, gracias a Helena por estar ahí siempre.

Acknowledgement

I would like to offer my special thanks to Gema for her guidance and her valuable advice during the whole project.

Thanks to my family, for giving me the chance to study this master and supporting me in every way possible and to my friends, for pushing me forward every day. Besides, thanks to all the professors that educated me throughout the years.

Ultimately, thanks to Helena for being there when I need it the most

Resumen

Este proyecto tiene como objetivo abordar el problema de la clasificación automática de Electrocardiogramas (ECGs) de 12 derivaciones. Normalmente, tras un ataque cardíaco, esta es la primera prueba que se realiza, ya que con ella podemos estudiar el comportamiento del corazón desde una gran variedad de ángulos. Sin embargo, estos procedimientos requieren una enorme cantidad de tiempo que, de ser resuelto, facilitaría en gran medida la labor de los profesionales sanitarios.

Los datos utilizados para esta tarea provienen del desafío Physionet/CinC 2020. En esta competición se ofrecieron cinco conjuntos de datos de varios lugares del mundo, con el objetivo de desarrollar sistemas de clasificación. Además de los ECGs sin procesar, los datos contenían información demográfica de los pacientes, como la edad y el sexo. Una vez que se extrajeron los datos, comenzamos a desarrollar el modelo de Aprendizaje Automático.

El primer paso fue preprocesar los datos. Para ello, desarrollamos dos métodos diferentes que dieron como resultado dos conjuntos, siendo su principal diferencia que las derivaciones fueran agrupadas o se tomaran en cuenta de manera independiente. Posteriormente, construimos una Red Neuronal, la cual está formada por dos bloques en paralelo, uno con una SE-ResNet y otro con una serie de LSTMs bidireccionales.

Tras el entreno, obtuvimos resultados satisfactorios similares a los de otros equipos de la competición, demostrando que nuestra arquitectura va en la dirección correcta. Para el problema de clasificación planteado, nuestro conjunto de datos contenía 27 etiquetas diferentes que podrían aparecer simultáneamente en las muestras y además existían grandes desequilibrios entre clases.

Una vez que terminado el modelo final, desarrollamos una herramienta de Ayuda a la Decisión utilizando Flask para construir el servidor y React para representar la información. Gracias a ello, el médico puede acceder a través del navegador web para analizar diferentes señales de ECG sin preocuparse en absoluto por los métodos que se están ejecutando en segundo plano.

Palabras clave: Aprendizaje Profundo, SE-ResNets, ECGs de 12 derivaciones, Sistemas de Ayuda a la Decisión, Physionet/CinC Challenge, LSTMs

Abstract

This project aims at tackling the automatic classification problem over 12-lead ECGs. After heart attacks or strokes, doctors usually start looking for anomalies to detect malfunctioning segments. However, these procedures take a tremendous amount of time. This is why they are about to be replaced thanks to the new advances in technology where Artificial Intelligence (AI) may be able to outperform the interpretation accuracy of healthcare practitioners.

The data used for this task came from the Physionet/CinC 2020 challenge. They offered a set of 5 different datasets from various places worldwide, aiming to develop robust classification systems. Apart from the raw ECGs, the data contained demographic information of the patients like the age and gender. Once all data was extracted, we started developing the Machine Learning (ML) pipeline.

The first step was to preprocess the data. For that purpose, we developed two different methods that resulted in two datasets. The main differences fall into the fact that the leads were taken into account as a group or independently. Afterwards, we built a Neural Network pipeline, which consisted of a modified SE-ResNet block parallel with a set of bidirectional LSTMs. We also included the demographic data by using a fully connected layer in between.

After training, we obtained satisfactory results similar to those of other teams in the competition, showing that our architecture is heading in the right direction. For the classification problem posed, the dataset contained 27 different labels which could appear simultaneously in the samples. Not to mention the presence of significant data imbalances between classes.

Once the final model was ready, we developed a Decision Support tool using Flask at the back-end and React at the front-end. There, the physician could access the application via the web browser to analyze different ECG signals and get prediciton results.

Keywords: Deep Learning, SE-ResNets, 12-lead ECGs, Decision Support Systems, Physionet/CinC Challenge, LSTMs

Contents

Resumen **I**

Abstract **III**

Contents **V**

List of Figures **IX**

List of Tables **XIII**

Glossary **XV**

1 Introduction **1**

1.1 Context 1

1.2 The heart 3

1.2.1 Conductive system 4

1.2.2 The electrocardiogram 4

1.2.3 Analysing the signal 7

1.3 Recent approaches to automatically classify ECGs 9

2 Objectives and document structure **11**

2.1 Objectives of the project 11

2.2 Structure of the document 12

3 Enabling Technologies **13**

3.1 What is Machine Learning? 13
3.1.1 Types of Machine Learning tasks 14
3.1.2 Other considerations 16
3.2 Introducing Neural Networks 17
3.2.1 Introduction 17
3.2.2 The Multilayer Perceptron (MLP) 18
3.3 Convolutional Neural Networks 21
3.4 Recurrent Neural Networks 25

4 PhysioNet/CinC datasets **29**
4.1 Dataset overview 30
4.2 Exploring dataset's diseases 33
4.3 Multi-labeled data 38

5 Development of classification models **41**
5.1 Pre-processing phase 41
5.1.1 Demographic data 42
5.1.2 Signal preprocessing 42
5.2 Development of the Neural Networks 46
5.2.1 The final architecture 47
5.3 Results summary 52

6 Defining the Decision Support System **55**
6.1 Components 55
6.2 Architecture 56
6.2.1 Back-end 56
6.2.2 Front-end 57

7 Conclusions and future work **61**

7.1 Conclusions 61

7.2 Future Work 62

A Impact of this Project **i**

A.1 Social impact i

A.2 Economic impact ii

A.3 Environmental impact iii

A.4 Ethical impact iii

B Cost of the system **v**

B.1 Physical resources v

B.2 Human resources vi

B.3 Licenses vi

B.4 Taxes vii

C Libraries and others **ix**

C.1 Tools and libraries ix

C.2 Cross Validation (CV) xi

C.3 Troubleshooting with Google Colaboratory xiii

C.4 Inverted T wave patterns xiv

C.5 Ensembles xiv

C.5.1 Averaging ensembles xv

C.5.2 Sequential ensembles xv

C.6 Complete summary of results xvi

C.6.1 Computation of the Area Under the Curve (AUC) and the Receiver Operating Characteristics (ROC) curves xvi

C.6.2 Execution times xvii

C.6.3 Summary table and hyperparameter testing xx
C.7 ECG printable & Features Extraction xx
C.8 Installation guide xxi

Bibliography **xxv**

List of Figures

1.1 Proportional mortality in Spain 2

1.2 Anatomy of the heart 4

1.3 Conductive system of the heart 4

1.4 12 Lead ECG angles intuition 5

1.5 Einthoven's triangle intuition 7

1.6 The Electrocardiogram (ECG) signal 8

1.7 The Pan Tompkins algorithm 9

3.1 Types of ML models based on type of classification 15

3.2 Example of Perceptron's architecture 18

3.3 Activation functions 19

3.4 Example of a typical CNN architecture 22

3.5 Convolution operation of a CNN 23

3.6 ResNet-34 RU architecture 24

3.7 SE block architecture 24

3.8 Unrolling through time of a simple RNN 25

3.9 Types of RNNs based on inputs and outputs 26

3.10 LSTM memory cell architecture 28

4.1 Final dataset's distribution 31

4.2 Bar graph of the age distribution 32

4.3 Top 10 distribution of diseases over the full dataset 32

4.4 Six second representation of AF 34
4.5 Six second representation of IAVB 34
4.6 Axis deviation features 35
4.7 LBBB wave patterns 35
4.8 Six second representation of PR 36
4.9 Confusion matrix showing labels relationships 40

5.1 Pre-processing steps pipeline 42
5.2 Filtering process 43
5.3 First harmonization method 44
5.4 Random crop example where signal is greater than window size 45
5.5 SE-ResNet architecture 48
5.6 Explaining the SE block 49
5.7 Bidirectional LSTM architecture 50
5.8 Final NN architecture & prediction process 51
5.9 Accuracy test on both models 52

6.1 DSS Sequence Diagram 56
6.2 Back-end main functionalities 57
6.3 Snapshot of the upload page with two files submitted 58
6.4 Snapshot of the resulting diagnosis page 58
6.5 Downloaded ECG printable 59
6.6 Snapshot of the prediction 60
6.7 Snapshot of the extracted ECG features 60

B.1 Cost of building the Deep Learning Model & Application vii

C.1 Snapshot of the moment when Colabs force quits an account xiii
C.2 AUC comparison of the different models xvii

C.3 ROC curves from the m1 top graphs xviii

C.4 ROC curves from the m1 bottom graphs xix

C.5 Averaged execution time per epoch of the different models in minutes . . . xx

List of Tables

1.1 Parts of the heart studied with each derivation 6

3.1 How to avoid learning errors 17

4.1 Sources for the Physionet/CinC 2020 challenge 30

4.2 Description of PhysionetCinC features 31

5.1 Analysing the NNs parameters 52

5.2 Model results with a fixed threshold of 0.5 53

5.3 Model results with m1 54

5.4 Model results with m1 54

C.1 AUC predictive analysis xvii

C.2 Summary of the models tried thoughout the project xx

Glossary

AF: Atrial Fibrillation
AFL: Atrial Flutter
AI: Artificial Intelligence
AUC: Area Under the Curve
ANNs: Artificial Neural Networks
AV: Atrioventricular Node
BN: Batch Normalization
BNNs: Brain Neural Networks
Brady: Bradycardia
CV: Cross Validation
CVDs: Cardiovascular Diseases
CNN: Convolutional Neural Network
CRBBB: Complete Right Bundle Branch Block
DL: Deep Learning
DSS: Decision Support System
DOM: Domain Object Model
ECG: Electrocardiogram
EHR: Electronic Health Records
EU: European Union
GRU: Gated Recurrent Unit
GDPR: General Data Protection Regulation
HTML: HyperText Markup Language
HTTP: Hypertext Transfer Protocol
IAVB: 1st Degree AV Block
IRBBB: Incomplete Right Bundle Branch Block
JSON: JavaScript Object Notation
kNN: k-Nearest Neighbors
LAD: Left Axis Deviation
LAnFB: Left Anterior Fascicular Block
LBBB: Left Bundle Branch Block

LOO: Leave One Out
LPO: Leave P Out
LPR: Prolonged PR Interval
LQRSV: Low QRS Voltages
LQT: Prolonged QT Interval
LR: Learning Rate
LSTM: Long-Short Term Memory
ML: Machine Learning
MLP: Multilayer Perceptron
NLP: Natural Language Processing
NN: Neural Network
NSIVCB: Nonspecific Intraventricular Conduction Disorder
NSR: Normal Sinus Rhythm
PAC: Premature Atrial Contraction
PR: Pacing Rhythm
PVC: Premature Ventricular Contractions
QAb: Q Wave Abnormal
RAD: Right Axis Deviation
RBBB: Right Bundle Branch Block
ReLu: Rectified Linear Unit
REST: Representational State Transfer
RNN: Recurrent Neural Network
ROC: Receiver Operating Characteristics
RL: Reinforcement Learning
ResNet: Residual Network
RU: Residual Unit
SA: Sinoatrial node
SA: Sinus Arrhythmia
SB: Sinus Bradycardia
SD: Standard Deviation
SSD: System Sequence Diagram
STach: Sinus Tachycardia
SVM: Support Vector Machines
SVPB: Supraventricular Premature Beats
TAb: T Wave Abnormal
TInv: T wave Inversion
UIs: User Interfaces

VPB: Ventricular Premature Beats

WFDB: WaveForm DataBase

WHO: World Health Organization

CHAPTER 1

Introduction

This chapter will serve as a broad introduction to the reader. Firstly, in Section 1.1, we will describe the current state of Cardiovascular Diseases (CVDs) and how practitioners deal with them in the biomedical world. Secondly, Section 1.2 explains the heart's structure and the standard procedure to study it. Then, Section 1.3 introduces AI, and some of the ultimate approaches focused on interpreting the heart signals. Finally, in Section 2.2, you can find a quick summary of the whole project.

1.1 Context

Cardiac diseases represent the leading cause of death in the world, out-numbering other serious pathologies like cancer. The main problem that these diseases have is that there are no underlying symptoms that make them appear in the first place. After heart attacks or strokes, doctors usually start looking for anomalies to detect the malfunctioning segments. This outdated methodology is about to be replaced thanks to the new advances in technology where AI and Personalized Medicine are conquering the clinical scene.

Firstly, let us talk about the main habits that lead to CVDs. All of them are well known by the population. However, even nowadays, people tend to neglect these proven facts and

choose to maintain harmful habits (Figure 1.1). Those are an unhealthy diet, excessive consumption of tobacco/alcohol, and the lack of physical exercise. These tendencies foster the so-called intermediate risk factors: high glucose, high blood pressure and overweight. However, more determinants escape our control and are also causes of the apparition of CVDs, for instance, stress, hereditary factors and poverty.

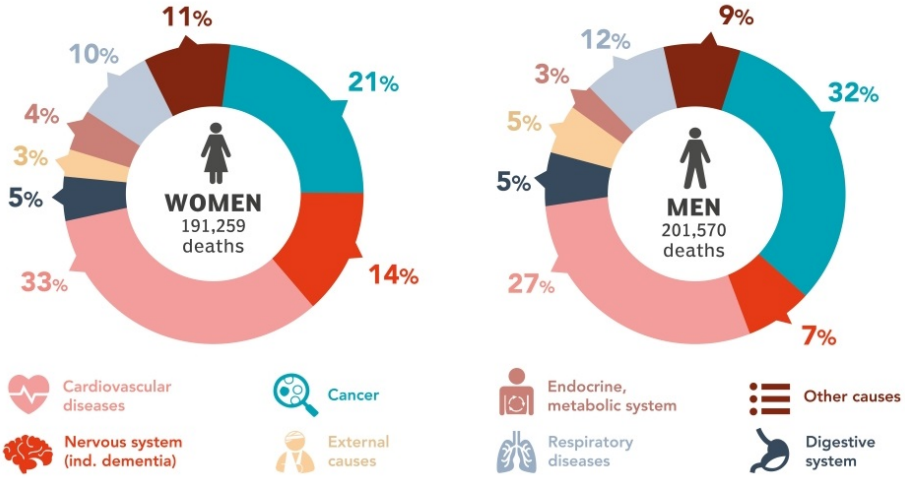


Figure 1.1: Proportional mortality in Spain [1]

The World Health Organization (WHO) has defined several policies and treatments as 'best buys' to prevent CVDs. The three main actions they promote are taxation to reduce harmful practices, medicines for prevention (such as aspirins and beta-blockers), and surgical operations (bypass, valve repair, heart transplant and balloon angioplasty). Sadly, as you might have already noticed, they did not mention AI in their article [2].

One of the reasons behind this is that State-of-the-Art Systems for CVDs diagnosis are far from the human performance level. However, it is a matter of time that we get there. Data extraction has always represented an obstacle in biomedical research; every hospital has databases with tons of information about health conditions, images and treatments. Unfortunately, this information is hard to extract due to the strict General Data Protection Regulation (GDPR) laws.[3]

Furthermore, imagine going through all this painful process to get the data; bureaucracy, anonymization and so forth, to discover that only a small percentage of it is useful. Data represent the base from which AI is built [4]. Hence, it is of extreme importance to teach physicians how to store it. Additionally, to make a substantial change, we need generalization, which is achieved by collecting data from several places to build a representative distribution that will lead researchers to create a robust and efficient ML model.[5]

But before we get there, it is vital to understand the cardiovascular system's principal performer, the heart. Lying beneath the second rib, we can locate the heart's base. The distal end extends downwards and is called the apex, sitting near the fifth intercostal space. Its primary function is distributing nutrients throughout the body by pumping blood. With a specialized cardiac muscle called the myocardium, the heart can beat each day tirelessly. This differentiated muscle that does not fatigue explains it all; if the heart fails, it is over.

The main procedure to study this organ's activity is by recording an ECG. With it, the practitioner can detect a vast range of diseases in a non-invasive way. Taking this recording does not take time and is an inexpensive procedure. However, reading through it can take some time and several hours of training, as we can see in this article from Jason Waechter et al. [6]

> 'Students would need to spend an average of **112 minutes** and complete about thirty-four practice cases to obtain **75%** of an ECG rhythm strip exam'

As we can imagine, this time-consuming practice has room for improvement with AI's introduction in physicians working pipelines.

1.2 The heart

As a starting point, it is vital to explain the hearts' anatomy (Figure 1.2). We can divide this organ into four hollow chambers; the two upper sections are called the atria and the other two, the ventricles. The left and right side are separated by what we call the septum. Each side is connected with valves designed in a way that backflow is not allowed.

In contrast with other species, our circulatory system is dual, which means that the blood passes twice through the heart. In the right atrium comes the deoxygenated blood from the superior and inferior vena cava. Then, it passes through the tricuspid valve into the right ventricle. From this point starts the pulmonary circuit where the blood is oxygenated and carbon dioxide released. Afterwards, the blood comes in again through the pulmonary veins into the left atria and passes the mitral valve towards the left ventricle. At this point, the blood starts what we call our systemic circuit, reaching out all our organs and tissues.

It is also important to mention the pulmonary and the aortic valve whose function is to prevent blood backpropagation during the ventricular relaxation[7].

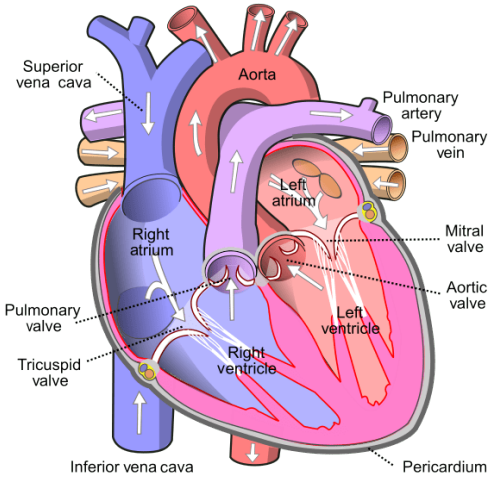


Figure 1.2: Anatomy of the heart[8]

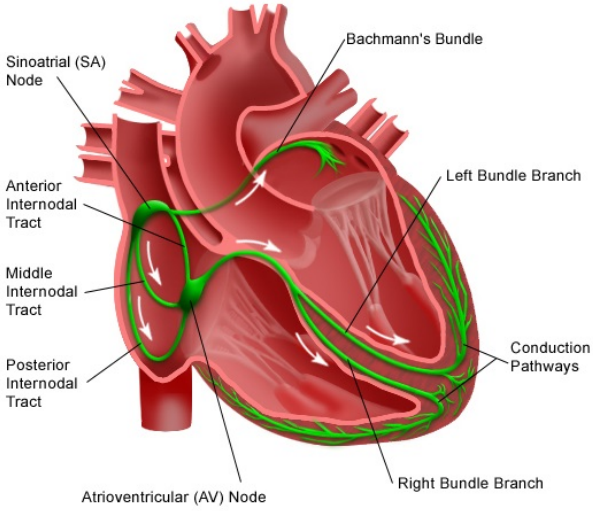


Figure 1.3: Conductive system of the heart[9]

1.2.1 Conductive system

A synchronous beat is crucial to maintain our main body functions intact; if it has irregularities, several diseases may arise as a consequence. How does our body achieve this?

It all starts in the SA node (a mass of specialized tissue beneath the epicardium close to the vena cava opening). Its cells, left alone, can reach the required electrical threshold to produce a beat by stimulating the atrial syncitium. The fact that this node controls the heart's rhythm is why we call it the pacemaker.

Secondly, the pulse reaches the next node called Atrioventricular node (AV), located in the inferior interatrial septum. It manages the atrial syncitium connection with the ventricular one, introducing a small delay due to the fibres' diameter. This delay is convenient so that the atria can clear out the blood into the ventricles before its contraction occurs.

From here, the impulse travels to the bundle of His, located in the upper part of the interventricular septum. This bundle is divided into two branches of Purkinje fibres that go to the apex, curve around the ventricles and passes the lateral walls. Besides, each pathway has numerous terminations to become continuous with the muscle fibres(Figure 1.3).[7]

1.2.2 The electrocardiogram

Doctors use several tools to look for cardiovascular anomalies, varying from imaging techniques to more invasive ones. As we have explained before, the ECG recording is the

standard procedure. As its name suggests, this technique consists of measuring the electrical signal emitted from the heart. There are many different ways of recording it, here we will explain the different approaches and types of ECGs.

We can say that the ECG signal is originated by measuring the heart's depolarization and polarization. In a cellular level is the result of the action of Na/K bombs which, as a final result, charge the cells generating an electric potential. It is essential to notice that the myocardial tissue contractions are the ones that produce the electrical signal to be measured. The conductive system described above generates these contractions, but the electrical potential they generate is too small to be captured.

The fact that the depolarisation occurs in an ordered sequence allow us to define the heart's electrical axis. Furthermore, it is good to note that the amount of energy measured is proportional to the number of active cells, which means that we will recognize the ventricles' signals as more significant than the atria ones (particularly the left ventricle).

The body contains ionic fluids which are indispensable to measure electrical signals throughout our skin, allowing us to 'watch' the heart in a non-invasive way by placing electrodes on it. The conventional procedure is to make a 12 lead ECG, which is the one that we will treat in-depth in this project. We can classify the leads in three groups, where each of them represents the heart's contractions from different angles (Figure 1.4).

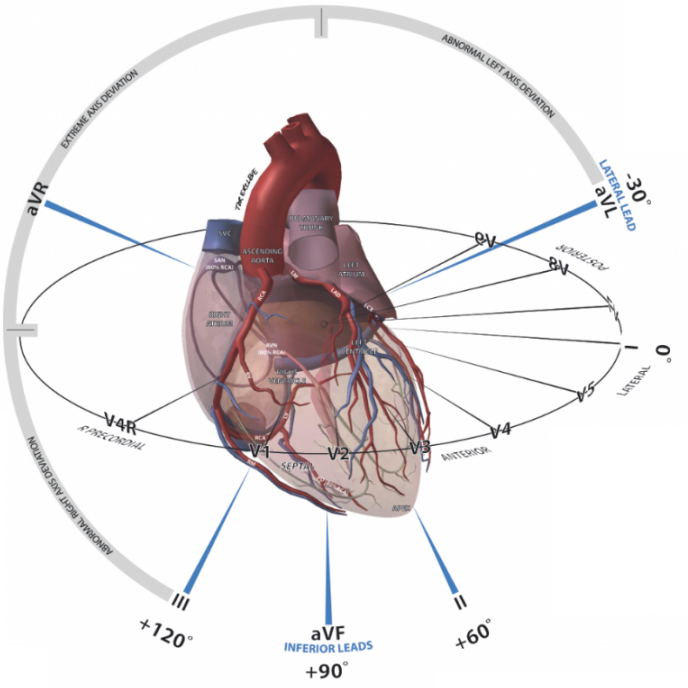


Figure 1.4: 12 Lead ECG angles intuition [10]

Table 1.1: Parts of the heart studied with each derivation

I	Lateral	**aVR**	-	**V1**	Septal	**V4**	Anterior
II	Inferior	**aVL**	Lateral	**V2**	Septal	**V5**	Lateral
III	Inferior	**aVF**	Inferior	**V3**	Anterior	**V6**	Lateral

We define a derivation as the position of two electrodes within the body. It is useful to see them as vectors where their relative position establishes the heart's angle from where we measure. The module of that vector will be proportional to the signal's strength, and its direction goes from the negative to the positive electrode.

Three derivations define the Einthoven's triangle (Figure 1.5), which we will call I, II and III derivation, also named as bipolar limb leads. They are commonly placed in the right arm, left arm and left leg. Thanks to our bodies conductive feature described in the previous paragraph, the signal captured from the arms is comparable to the one measured in the shoulders, and the legs is to the one taken from the groin. We need to define a reference or ground in every circuit, which, in our case, we will place on the right ankle. As we can see in the image, the Left arm electrode acts as positive for the lead I and as negative for the third derivation III, and if we think of this as vectors the Einthoven's law states the following:

$$Lead\ I + Lead\ III = Lead\ II \tag{1.1}$$

The final objective of the ECG is to plot the mean electrical axis of the heart. We achieve this with the bipolar leads, but only in a coronal plane, which means we cannot diagnose specific pathologies whose effects are hidden from this view, in the end, we have to think of the heart in a 3D way. This requisite explains why the 12 lead ECG is the most used technique, as it manages to plot the mean electrical axis in a three-dimensional space, capturing all the electrical currents that occur during each heartbeat.

The next group of leads that we will talk about are the augmented limb leads, those are unipolar and measure the heart in the same plane as the limb leads. The main difference is that instead of using one electrode as a negative reference, this augmented leads averages the remaining negatives leads, creating a vector that goes from the middle of each side of the Einthoven's triangle towards the positive electrode. Since we have three sides, we will have aVR, aVL and aVF.

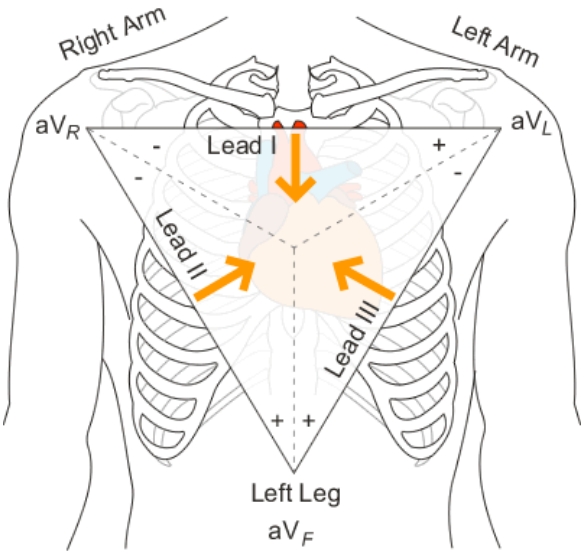


Figure 1.5: Einthoven's triangle intuition [11]

We have explained six of the twelve leads by now. We call the remaining six, precordial, those are unipolar and measure the heart from a transversal plane. They are placed in the chest's surface around the heart in a serial way that allows a clear interpretation from one extreme to the other. If we analyze V1 and V6, we can see that the remaining leads represent a transition between them. V1 and V2 focus on the anteroseptal region, V3-V4 the apical and V5-V6 the anterolateral.

1.2.3 Analysing the signal

We will then interpret the signal and associate it with the physiological events within each heartbeat, which results in this familiar pattern that we can see on several leads.

On it, we can appreciate segments, intervals and complexes. We call a segment to the region between two waves; an interval to the duration that includes a segment and one or more waves. Lastly, we define a complex by a combination of signals. It is also important to remember that we will have a positive deflexion when the resulting vector goes towards the positive electrode and a negative one when the opposite situation occurs. The x-axis represents time; therefore, we will follow this reference to describe each ECG components.

- **P wave:** it is a positive deflexion characteristic of the atria's contraction, with 200 ms long, this signal presents an ill-defined morphology very susceptible to noise.
- **PQ-segment**: A characteristic isoelectric line shows the delay present when the signal travels from the SA node to the VA one.

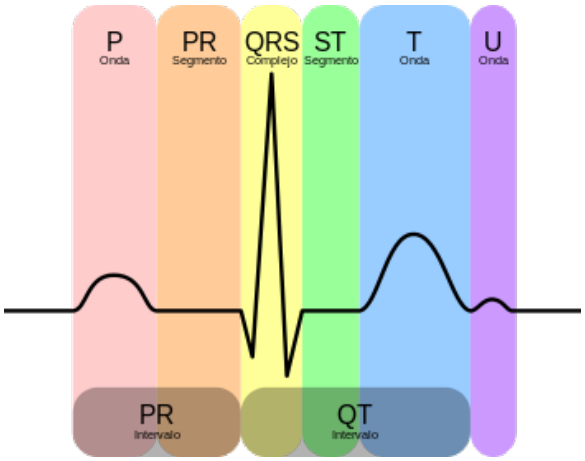


Figure 1.6: The ECG signal [12]

- **QRS complex**: a series of negative, positive and negative deflexions corresponding to the ventricles' contractions. It is the sharpest signal within the ECG, and its shape can vary from one lead to another.
- **ST-segment:** It represents the time until the starting point of the repolarization of the ventricles. It is usually studied within the T wave because its starting point might be blurred.
- **T wave:** This one corresponds to the ventricular repolarization. It represents a positive deflection with a significant ascending segment. If it is too fast, this wave can be confused with the next P wave's starting point.
- **U wave:** It is a wave that appears right after the T one, but it is not necessarily noticed or indicative of underlying pathologies. We can watch this wave easier from the precordial leads.
- **QT interval:** It represents the time that passes from the ventricular contraction's starting point to its final repolarization. We need to express its value fixed due to frequency dependency.
- **RR interval:** It is one of the most studied ones as we use it to analyze the cardiac frequency and its variability.

As we are going to develop a Decision Support System, it is vital to understand and study how cardiologists read ECGs. They use a special graph paper where the ECG signal is plotted. We represent voltage on the vertical axis, where 1 mm equals 0.1 mV. We define time on the horizontal axis, where each square represents 0.2 seconds; therefore, each 5 mm

equals a second. These measurements are not standard; physicians can modify them to study specific pathologies where other smaller parts of the signal are of interest.

1.3 Recent approaches to automatically classify ECGs

As a quick summary, we have discussed that the heart produces a periodic signal called ECG, indicating either the presence or absence of CVDs. The current efforts focus on automatically detecting these abnormalities on the ECG and transforming them into a possible diagnosis.

Overall we would say that there are two main currents for studying the ECG signal. The first focuses on feature extraction, processing the ECG and extracting important measurements that practitioners may find useful for their analysis. The second one, we could say that it is more straightforward and has a more meaningful goal: using AI to diagnose CVDs based on the complete ECG signal. Both methods have their pros and cons; thus, we see more and more projects that combine both methodologies.

The first and foremost known method to study the ECG signal is the Pan Tompkins algorithm, which focuses on the signal's periodicity. We can see the model's pipeline in the next picture.

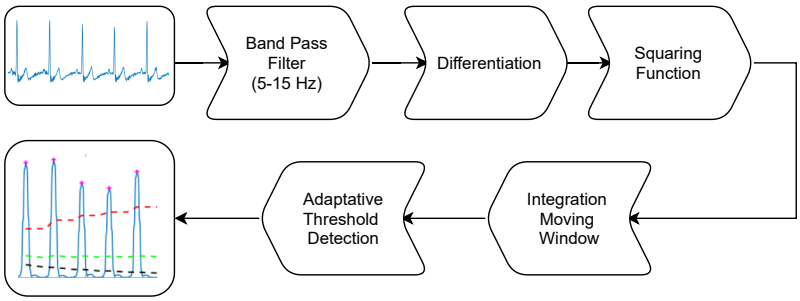


Figure 1.7: The Pan Tompkins algorithm

The Pan Tompkins algorithm's final objective is to detect the position of the QRS complexes. One of its applications is estimating the heart rate by counting the number of R peaks. As we only want to examine the QRS complex, the first step is to filter the other waves' frequencies by treating them as noise. Afterwards, the efforts will focus on amplifying the peaks and detecting them by an integration window. The results were astonishing as they obtained 99.3% accuracy in detecting these peaks on an arrhythmias dataset [13]. We can also adjust this algorithm to study variability on the sampling frequency, noise, diseases and detecting the remanent waves using the wavelet transforms.

Talking about the second method, nowadays, researchers are obtaining meaningful results using Neural Networks to predict CVDs. These models are faster than the former as they do not suppose additional workload for practitioners. The sequential nature of the ECG makes them susceptible to study with either Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN) as the signal possesses either a defined shape and a time-dependent behaviour.

Physionet and Computing in cardiology have carried out a series of challenges to tackle clinically challenging questions that remain unsolved or whose performance is mediocre. In 2020 they carried out 'Classification of 12 leads ECGs', more than a hundred teams participated from several universities and groups worldwide. The results were published in September of 2020. Thanks to them, we could research numerous innovative approaches not only referring to the Neural Networks or the ML algorithms but also to the data preprocessing steps which are crucial to develop a successful model.

We decided to focus on the top ten papers as they were the ones that achieved the highest scores so far on the test sets. Between these ten, only one chose to combine feature engineering with Neural Networks; those were the 'prna team' [14]. They extracted over 300 features from lead II and applied a random forest classifier to analyse feature importance. Finally, they combined them (22) with a deep Transformer Neural Network (NN) architecture.

Other teams performed a milder preprocessing step by simply using the raw ECG signal. In what refers to the NN architectures, we saw common patterns as many groups were using Residual Network (ResNet) as their central neural unit. However, when evaluating their models, we could see many approaches like the definition of an adaptative threshold, customized loss functions, rule-based models, and reward matrices. We averaged the results from the top ten teams, and they obtained a 63.5% accuracy on the validation set. However, when the organization evaluated their models with the hidden test sets, the scores dropped to 45%. Interestingly, the higher validation scores teams performed worst in the secret test set, proving a growing lack of generalization. Here you can find access to all the papers published during the competition.[15]

It is also important to mention that reproducible results were not plausible due to the amount of computing capacity we had. In the following chapters, you will see in great detail the arrangements we took to perform the classification task. In the end, we decided to use a different scoring metric than the ones proposed by the competition as we did not want to give distinct priorities to specific classes. In this paper, you can find a detailed analysis of the whole challenge if the reader is interested.[16]

CHAPTER 2

Objectives and document structure

2.1 Objectives of the project

The main objective is to design and develop a Decision Support System (DSS) that helps physicians in the diagnosis of CVDs based on ECG recordings. Before reaching that point, we defined the following sub-objectives:

1. Firstly, we need to design and develop a robust Deep Learning (DL) model. In this phase, the development is divided into three parts:
   (a) We begin with a pre-processing step to extract the main features of the different ECGs.
   (b) Afterward, we build different NNs architectures using the Keras API.
   (c) Finally, we iterate over the different NNs models based on various scoring metrics to assess model performance.
2. Then, we create the final Decision Support System (DSS) with two main modules:
   (a) The back-end, where we integrate the Deep Learning model along with the logic to predict diseases.

   (b) The front-end, which will be in charge of displaying the data in a straightforward and intuitive interface.

3. The final goal is to deploy an operative version of the DSS at the University servers. This final product is intended to be tested and improved by the physicians' feedback in posterior phases of the development.

2.2 Structure of the document

In this section, we will outline the incoming chapters so that the reader can get an overall view of the projects' structure. Chapter 3 will introduce NNs and all the techniques that allowed the model's realization. Afterwards, in Chapter 4, we will explore the datasets' features and labels (Cardiac Diseases). In Chapter 5, we will explain the model architecture from the preprocessing steps towards the models' results. Finally, in Chapter 6, we will describe the design and development of the Decision Support System (front-end and back-end).

CHAPTER 3

Enabling Technologies

In this chapter, in Section 3.1 we will introduce Machine Learning and the basis from which it is built. Firstly, we will explain simple Artificial Neural Networks (ANNs) (Section 3.2) to gain some intuition on how they work. Finally, we will describe the architecture of advanced neural structures, which will be of interest to our particular use case. In Section 3.3, we will explain in great detail Convolutional Neural Network (CNN) and their most novel architectures. Then in Section 3.4, we will define Recurrent Neural Network (RNN), which are of great importance in sequence analysis, which in the end, is what ECGs are.

3.1 What is Machine Learning?

Arthur Samuel, in 1959 defined Machine Learning (ML) as the field of study that gives computers the ability to learn without being explicitly programmed. This definition was too shallow; therefore, in 1997, Tom Mitchell defined machine learning technically:

"A computer program learns from experience (E) concerning some tasks (T) and some performance measure (P), if its performance on (T), as measured by (P), improves with experience (E)."

Before, the typical procedures for 'Machine Learning models' involved creating rules based on human perception. However, that so-called 'perception' is limited. Human experts are good at recognizing patterns but limited to the dimensionality of these problems (<=3). The data quantity and complexity of the medical field generate tasks with sometimes more than a thousand dimensions. Sophisticated algorithms and decent computing capacity are needed to overcome these issues.

Computer algorithms needed to evolve to solve these dimensionality problems, and they did it exceptionally well. Nowadays, let us inspect the technologies we use. Almost all of them apply ML algorithms, for instance, spam detection on e-mail platforms, recommender systems on YouTube, Reinforcement Learning on videogames, Natural Language Processing (NLP) for subtitling films, and we can keep going. Even if at first glance they seem missing, like on a physical product, its main features might be coming from a ML study.

With this said, ML encountered another problem, which is data. Making a computer learn from it, is not an easy task; this data must fulfill a series of characteristics to be valid. Two of them are quality and quantity, and having both is not an easy task.

3.1.1 Types of Machine Learning tasks

There are various ways to classify machine learning systems (Figure 3.1). This section will explain them briefly as it is useful when deciding the starting point in a new project.

The first type of classification involves whether or not a ML system is trained with human supervision. Here, we could split the cases into four categories:

- **Supervised learning:** in this situation, the data fed to the computer includes the desired solutions, also called labels. The two most typical tasks are classification and regressions. The difference between these two is that on classification, we have a fixed set of possible labels. Simultaneously, on regressions, the computer is programmed to give a value called predictor given a specific input. The most common supervised algorithms are k-Nearest Neighbors (kNN), Support Vector Machines (SVM), Decision Trees, and some NNs.

- **Unsupervised learning:** as its name suggests, we do not need to have our dataset labeled, which means that just by gathering data, our computer will try to gain knowledge over the different instances. This section deals with clustering, anomaly and novelty detection, visualization, dimensionality reduction, and association rule learning.

- **Semi-supervised learning:** Sometimes, labeling a massive amount of data can be a time-consuming task. Therefore, you might find your dataset half labeled. These systems work by combining supervised and unsupervised algorithms. Some of them are deep belief networks and restricted Boltzmann machines.

- **Reinforcement Learning (RL)**: Treat an entirely different problem. Here we have an agent that performs actions. These actions can be rewarded or penalized, and by recreating sets of agents, the final one would be able to perform the desired task correctly without human intervention.

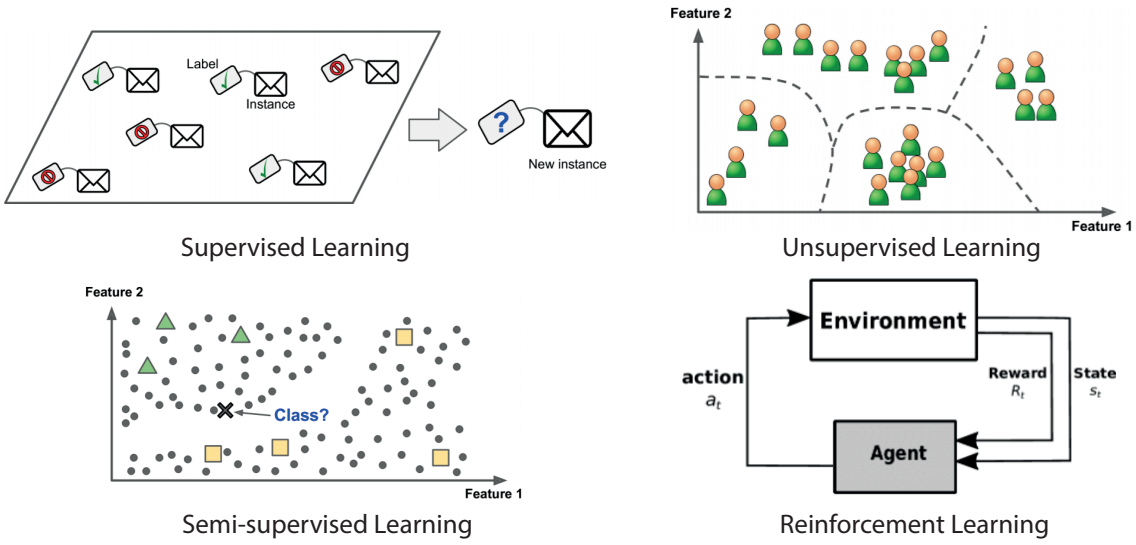


Figure 3.1: Types of ML models based on type of classification [17]

Another possible variety is batch vs. online learning. In this one, the main difference is how we deal with large amounts of data. If our machine cannot cope with such quantity, the logical approach would be to learn by batches from an external data source. We will call this procedure online learning and, in the opposite situation, batch learning.

Finally, we have instance vs. model-based algorithms, whose differences remain in how they learn from the data they receive. If it generalizes based on some similarity measure, we will call them instance-based. On the contrary, on model-based algorithms, a 'model' is constructed, and if the data fulfills its criteria, the predictions would be confident towards that solution.

3.1.2 Other considerations

We can sum up ML into two fundamental tasks, selecting a learning algorithm and training it. Therefore, there are only two things that can go wrong: inadequate datasets or algorithms.

Firstly, let us talk about an insufficient quantity of training data. A computer needs thousands or even millions of samples to extract relationships over samples. Therefore, having a small dataset is not an option when learning complex tasks like image or speech recognition. However, there is a partial solution to this problem, which is using transfer learning. This technique uses an existing model to reduce the task to a simpler one. For example, suppose we want to analyze the car registration numbers on an image. In that case, we could combine current models that recognize cars and focus only on the letters and numbers of the car's image.

Secondly, we would like to talk about non-representative data. In every ML task, we want to achieve generalization. Thus, it is crucial to gather samples that are representative of the cases we want to generalize. Cleaning up the data is worth the effort; removing outliers, missing values, and wrong measurements can help the algorithm finding correct data patterns.

Thirdly, a good ML project comes with a good selection of features and reduces the least important ones. Feature engineering has two main phases, which are feature selection and feature extraction. Feature selection is the process of choosing the most promising values from data. Feature extraction combines the existing features to reduce the problem's dimensionality and helps the algorithm discover meaningful patterns.

Finally, we will talk about overfitting and underfitting, mainly correlated to the learning algorithm and data. When we overfit a model, it means that it loses the ability to generalize to new cases, which is our final goal. It appears when the model is too complex relative to the amount of noisiness of the training data. Underfitting the training data refers to the opposite situation. It happens when the model is too simple to learn the underlying structure of data (Table 3.1).

Table 3.1: How to avoid learning errors

Reduce overfitting	Reduce underfitting
Simplifying the learning model	Increase model complexity
Gathering more data	Extract better features
Cleaning the data	Reduce constraints on the model
Use regularization	

3.2 Introducing Neural Networks

3.2.1 Introduction

We could crudely say that neural networks are a specific type of ML algorithm. However, due to its growth in the last years, people are starting to differentiate between ML and DL.

In the past, people thought DL was limited and could not improve due to its basic blocks. However, as new technologies emerged and computers improved, this theory has already been proven wrong and, ironically, is DL the one taking the lead.

Our brain is our most precious and unknown organ; with it, we control our body functions autonomously, solve complex problems, and communicate with each other. People might be wondering what this organ has to do with computers. Well, NNs were first inspired by the brain. Even though we know little about how it works, the first models were motivated by their biological units, also called neurons.

With the help of specific biomolecules, these specialized cells transmit electrical impulses, leading to specific effects in our body. One of the main features that make our brain so powerful is that, even though a neuron alone cannot perform complex operations, one unit can be connected to thousands of other neurons, which grants such an incredibly computational capacity. Such is its complexity that Brain Neural Networks (BNNs) are still the object of active research; even though some areas of our brains have been mapped successfully, the organ itself remains a mystery.

Current NNs differ significantly from the brain, and because of this, some researchers argue that this analogy should be dropped. One characteristic of NNs that makes them ideal for ML tasks is scalability. By using NNs, we can solve complex problems where the

amount of data needed is immeasurable, and theoretically, the more data we get, the better our model will do.

3.2.2 The Multilayer Perceptron (MLP)

In 1943 McCulloch and Pitts proposed simplified NNs where each of its inputs was binary and proved that they could generate every logical operation available with combinations of such units.

This trend leads to the development of one of the most straightforward NN architectures called the Perceptron (Figure 3.2). Here the concept of binary inputs was enhanced and in exchange, what happened is that each input, representing each instance of the data, was multiplied by a certain weight. At the end of this process, a Threshold Logic Unit computes a weighted sum and applies what we will call an activation function.

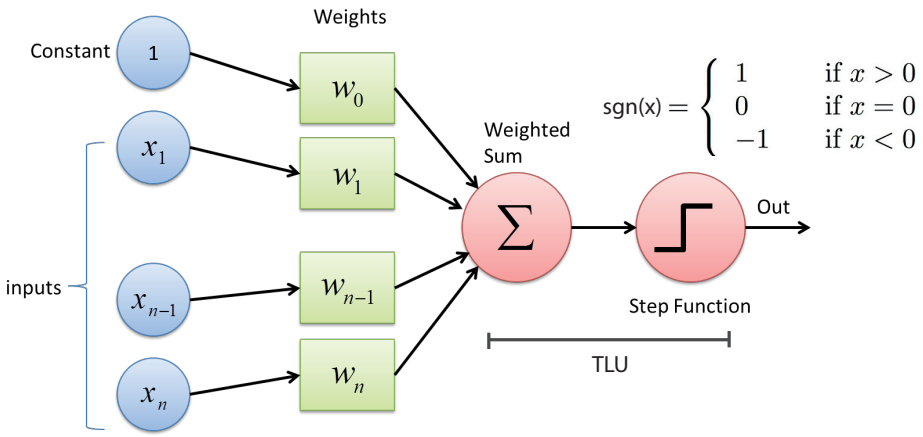


Figure 3.2: Example of Perceptron's architecture [17]

First, we will talk about the inference process of this ANNs, and for this purpose, we suppose that it has already updated the weights with data and is ready to make predictions. If we imagine a set of inputs X and a matrix of weights W, the final result will be given by:

$$\begin{aligned} h_{(W,b)} &= \sigma(XW + b) \\ w_{i,j} &= w_{i,j} + \eta(y_j - \hat{y}_j)x_i \end{aligned} \tag{3.1}$$

However, how does this network 'learns'? The Perceptron learning rule was based on the biological assumption of Siegrid Lowel [18], where she stated that 'cells that fire together, wire together' leading to the following learning rule where minor errors reward the current weights, and big ones penalize them.

The Perceptron was limited, and only by constructing a multilayer one, further improvements could be made. Combinations of these basic Perceptron units end up forming a network called the Multilayer Perceptron. Its first layer is the input, the middle ones are called hidden, and the final one is the output layer. The name hidden refers to the fact that we do not necessarily need to know its inner values (like a black box).

An obstacle that prevented this NN from improving was the learning algorithm, solved years later with the backpropagation implementation. The MLP steps are commonly used today and are the basis from which more complex NNs were developed. Firstly, let us go through the learning steps of the Gradient Descent algorithm used to minimize the error between the real values and the predicted ones:

$$min(J_{w,b}) = min(\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^i, y^i)) = \nabla J_{w,b} = (\frac{\partial J}{\partial w^{[1]}}, ... \frac{\partial J}{\partial w^{[L]}}, \frac{\partial J}{\partial b^{[1]}}, ..., \frac{\partial J}{\partial w^{[L]}}) \tag{3.2}$$

$$w^{[l]} = w^{[l]} - \alpha\frac{\partial J}{\partial w^{[1]}} \tag{3.3}$$

$$b^{[l]} = b^{[l]} - \alpha\frac{\partial J}{\partial b^{[1]}} \tag{3.4}$$

$$\frac{\partial \mathcal{L}^i}{\partial a^{[L](i)}} = \frac{-y^i}{a^i} + \frac{1-y^i}{1-a^i} \tag{3.5}$$

On Equation 3.1, σ corresponds to the activation function, an element of the NN that implements non-linearity to the architecture, the application of these functions is fundamental on a NN. Without it, neurons would perform linear transformations of the inputs, and having several layers of them stacked would be like having one. Thanks to these activation functions, we can theoretically approximate our models to any continuous function.

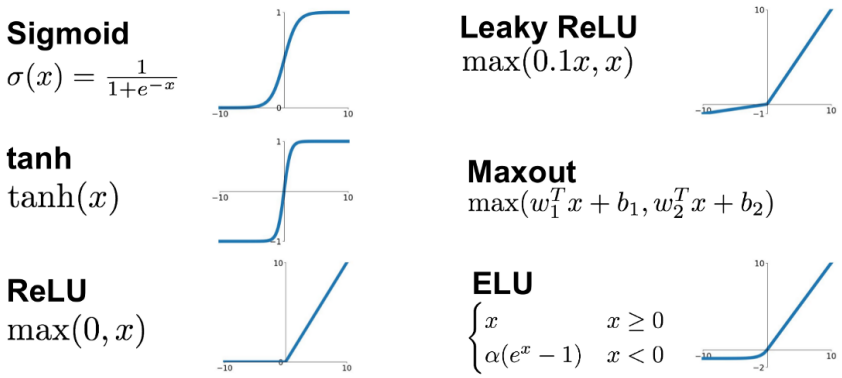


Figure 3.3: Activation functions [19]

Training a NN is not an easy task; when the net grows larger and larger, several problems start to appear. In our project, we will have to deal with some of them.

Firstly we have the vanishing/exploding gradient problem, where the algorithm cannot adjust its weights during the forward or the backward pass. The cause of it was unclear until two researchers found two possible culprits, one of them was the activation function and the other the weight initialization.

The activation function caused the values to saturate when the inputs were large. Therefore, increasing the number of layers was not improving the performance as was theoretically expected. The standard weight initialization scheme was based on the normal distribution (μ=0 and σ^2=1). Computing the backward propagation, in the end, is like performing the derivative of the activation function, which on the extremes, as we can see on the sigmoid function on Figure 3.3, has a zero value. Therefore, when updating the weights, the result was either unchanged (vanishing) or oscillative (exploding).

There are several ways to solve this; one of them uses different initialization techniques that focus on adjusting the inputs and outputs on each layer so that the μ and σ (Standard Deviation (SD)) remain constant. Another one could be to use other activation functions whose derivative is monotonous throughout its whole domain. Finally, we can opt to use Batch Normalization, which consists of calculating the mean and the standard deviation on each mini-batch. This final technique has been proven to be decisive for achieving regularization, increasing the learning rate, and in the end, speeding up the training process.

Training a deep NN can take an immeasurable amount of time. The techniques exposed above represent specific ways to reduce such intervals. Moreover, there is one more parameter that we have not mentioned yet, which is the optimization algorithm. Everyone knows about Gradient Descent as a reliable one. However, over time, newer and more efficient algorithms have been developed, for instance, Nesterov Accelerated Gradient, AdaGrad, RMSProp, Adam, and Nadam.

Now let us focus on overfitting, since reducing it, is not an easy task. It happens when the network has too many parameters, and especially, if our dataset is noisy or small. Some viable solutions have been covered so far, but not all. Regularisation and dropout are the remanents. Dropout is one of the newer methods to reduce overfitting; its functioning relies on turning off specific neurons by defining a certain probability 'p'. It is working exceptionally well for deep networks, where it has been proven that even on state-of-the-art models, adding this layer optimizes the network by 1% or more. This might seem like a minor improvement, but if we have a model of a 95% accuracy, this change might be equivalent to dropping the error rate by 20% onwards (from 5% to roughly 3%).

3.3 Convolutional Neural Networks

A book called '*Hands on Machine Learning with Scikit-Learn, Keras & Tensorflow*' [17] served as a guideline for the development of these sections. Therefore, if you are interested in a complete explanation of a specific topic, we recommend reading through it.

These networks emerged from the study of the brain's visual cortex. For us, perception might look like a simple job. However, obtaining reproducible results on computers is a challenging task to achieve if we think about it. CNNs are present in many DL models; they are used on image search services, self-driving cars, and automatic video classification systems. Furthermore, tasks that might be disconnected for image analysis, such as voice recognition and Natural Language Processing, are still object of study on CNNs. Conventional networks were not enough to handle images, although they can be used. If we are working, for example, on a dataset of 100*100 pixels images, this would mean that to feed the whole dataset into the first layer of the neural network, we would need the first layer of 10000 units which, computationally talking, is not viable.

The distinct components that make a CNN work are two specific layers, which are called convolutional and padding (Figure 3.4). We will introduce them briefly.

In convolutional layers, the concept is that we perform a convolution operation within each step towards deeper layers. Which in terms of an image means that in shallow parts, the network will study low-level features (horizontal and vertical lines), and by going deeper, the network will start learning bigger 'chunks' of the image, for instance, eyes, noses, and ears (in a face recognition task).

The structure of the convolutional layers will be determined by three mandatory parameters: the filter, the padding, and the stride. The filter or convolution kernel represents the sliding box that will be performing the convolution operation. The stride is the number of steps that the filter takes vertically and horizontally. Finally, with the padding, we add zeros on the edges of the image, increasing the slider range for cases where the data is not squared or its shape and the stride create an information loss on the edges.

The combination of filters and images allows the computer to create feature maps, where the neurons within each map share the same parameters (weights). This way, we reduce the network's complexity and recognize a particular pattern on every part of the image, which was not possible with conventional networks.

Another big problem with CNNs is memory usage during training because we need to keep the values of each layer to compute backpropagation in the following steps. To give

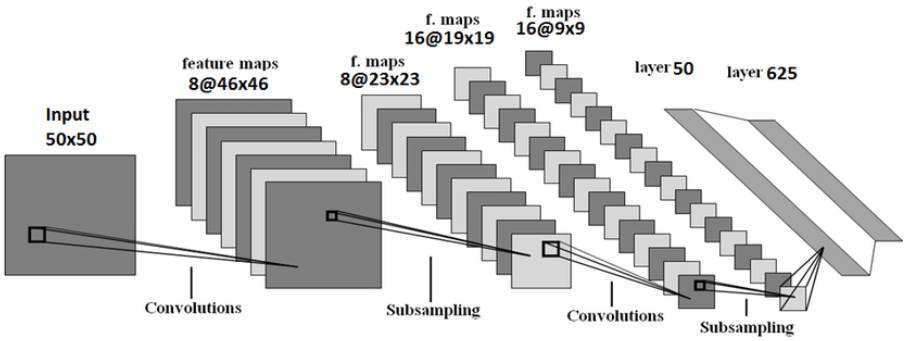


Figure 3.4: Example of a typical CNN architecture [20]

a practical example, imagine that we have a $(12 * 256)$ 'image' (which will be the case we will deal with in our project). Each row corresponds to an ECG signal which in the end, grouped, can be plotted as a unique signal. Supposing we want 200 feature maps per filter and that we are storing its values as type float32 will be $(256 * 12 * 200 * 32) = 19.66$ MB, and if we use a mini-batch size of 32, which will be the case, we will be using up to 0.6GB of RAM per layer, which is quite a lot. This constraint does not represent a burden as Google Colabs' C.1 capacity is quite decent.

The goal of pooling layers is to subsample the input image to reduce overfitting, and also, in parallel, to achieve a reduction in memory usage and computational load. Each of its layers works similarly to convolutional ones except that they do not possess weights. We have to define the padding, the stride, and the size of the sliding window. At this point, instead of performing a convolution operation, the layer can compute the maximum or the average of its receptive field. Therefore if we have a 2x2 pooling layer with a stride of 2 and no padding, this will result in a halved image both in height and width. Pooling layers also introduce invariance, allowing our model to generalize given different rotations and scalings, which translates into a robust classification (Figure 3.5).

Now let us talk about how to combine layers to get robust CNNs. The architecture is quite repetitive; for brevity, we will illustrate the typical components: Convolutional layers followed by Rectified Linear Unit (ReLu) activation functions, typically followed by a pooling layer. After combining these aggregates, the objective will be to flatten the last layer's result and concatenate dense layers to estimate class probabilities. Once this structure is built, the hard part would be to define correct filter sizes to maintain a balance between losing spatial information and improve computational costs.

The architecture of CNNs has drastically evolved with time. In particular, the most famous neural architectures are LeNet5, AlexNet, GoogleLeNet, VGGNet, Xception, ResNet,

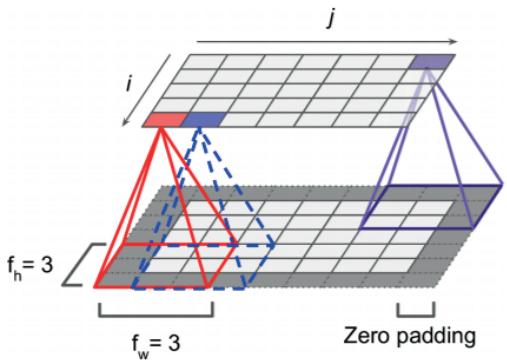


Figure 3.5: Convolution operation of a CNN [17]

and SENet. This last one is the state-of-the-art network in image classification tasks showing promising results in ECG classification, combining ResNets units with Squeeze and Excitation blocks [21]. As it can be seen, the field of CNNs is quite complex; we would not explain all the architectures described in this chapter apart from the ones that will be used in our final model.

Next, we would like to talk about ResNets. The power of this network belongs to the fact that it can be built very deep without comprising the learning schedule. The depth barrier was one of the reasons that prevented deep learning from progressing. The key to reaching that depth without losing model performance is using skip connections where the signal fed to one layer jumps to an output layer located a bit further in the stack.

When training a NN, the goal is to make it model a target function. When we add skip connections, the network is forced to model the following $f(x) = h(x) - x$ rather than $h(x)$. On each iteration of a CNN, the units are very rarely uploaded, and having those skip connections speeds up learning considerably. Finally, another key feature of these networks is that compared to those seen before, if training is stopped at any part of the network, the skip connections can make it through towards the subsequent layers.

Its architecture is based on stacks of RU units composed of two convolutional layers, batch normalization, and a ReLu activation one. The convolutional layers use 3x3 filters, 'same' padding, and a stride of 1.

As we get deeper into the network, the feature maps are doubled. Hence, to match shapes, some modifications on the network need to be made to allow learning (Each time we change the number of feature maps, we will have to make this operation). This architecture is specific for the ResNet-34 (Figure 3.6), although the structure needs to be changed accordingly to reach deeper layers, for instance, on ResNet152 (three convolutional filters).

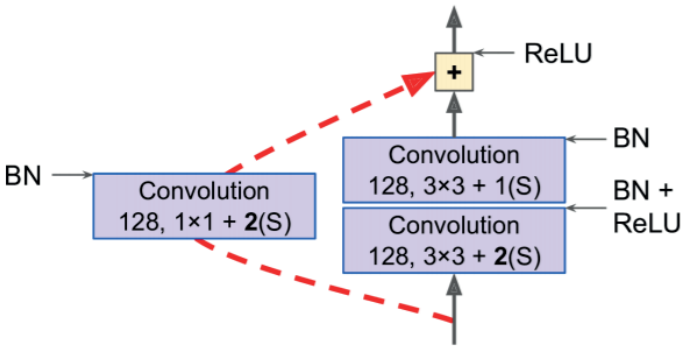


Figure 3.6: ResNet-34 RU architecture [17]

To end with CNNs, we will introduce the trendiest CNN architecture, the SENet. It combines the previous architecture (ResNets) and others not discussed (Inception), in the end surpassing their successors by far. Our project will be using SE-ResNets as the literature showed that it is a good approach towards ECG classification. The reason for this improvement is thanks to the Squeeze and Excitation block, which is added to every residual unit of the architecture.

Its work is to gather all the feature maps resulting from the Residual Unit (RU) and analyze them in terms of depth (not looking into spatial patterns), learning which features are more active together. To explain it with a practical example, imagine building a classifier to distinguish between different animals. Usually, to differentiate them, we will need to focus on their most characteristic parts (mouth, tail, legs, and fins). A set of feature maps will have plenty of units with no information about these. Thus the work of the SE block would be to boost (mouths, tails, legs, and fins feature maps) and reduce the irrelevant ones that do not possess such frames. These units comprise a global average pooling layer, a dense layer with a ReLu activation function, and another dense layer with a sigmoid one.

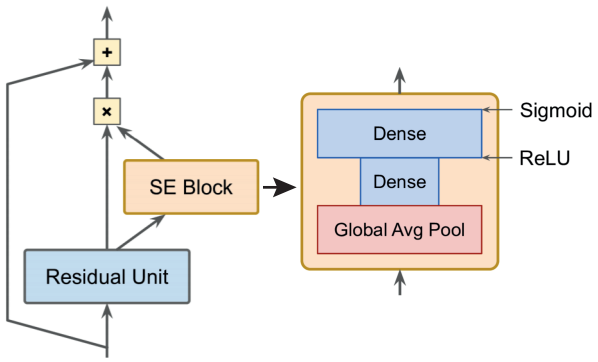


Figure 3.7: SE block architecture [17]

3.4 Recurrent Neural Networks

The last type of architecture that we are going to study in this chapter are RNNs. Their purpose is to make predictions given a previous sequence, for instance, weather forecasting, trajectories, stock prices, parse texts (as words tend to appear in sequential order), audio analysis. In general, we could say that these networks are good at analyzing sequences extracting patterns that can serve as a reference to build incredibly robust and disruptive models. In the end, a sequence is made of 1D data, which is the perfect feed for conventional NN. In short sequences, they can perform exceptionally well. On the contrary, in large series such as audio recordings, CNNs can do it reasonably well. Therefore, what is the point of RNNs?

The only difference between RNNs and conventional ones is that the first has connections pointing backward. It is originating what we call a recurrent connection where the neuron's output is added to its input at the same time. It is vital to understand how they work to apply this potential to sequential data. The best way of seeing it is by unrolling the neuron in the time axis. Consequently, each neuron will have two sets of weights, one for the inputs and another for the outputs (Figure 3.6).

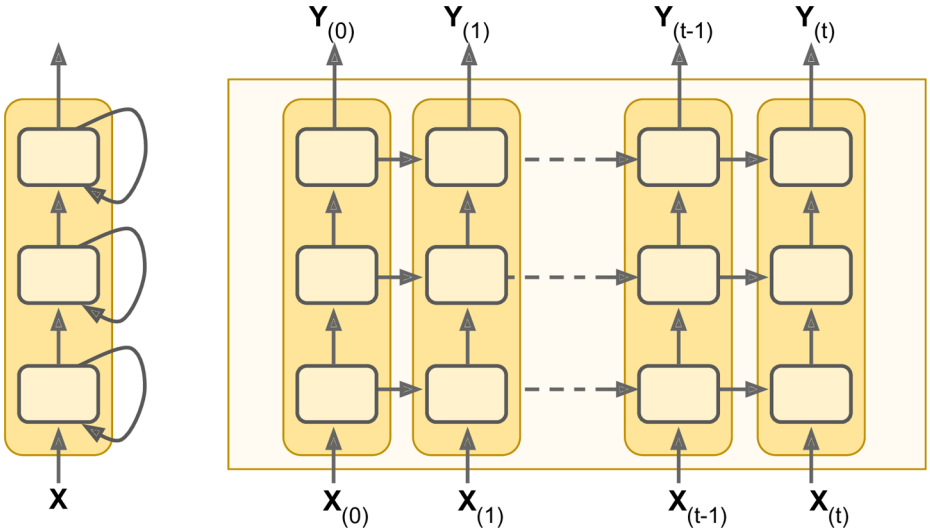


Figure 3.8: Unrolling through time of a simple RNN [17]

By having a recurrent connection, we could say that the architecture has memory as every output Y is going to be represented by the function:

$$Y(t) = \prod_{i=0}^{t} X(t)Y(t-i) = X(t)Y(t-1) = X(t)X(t-1)Y(t-2) * ... \tag{3.6}$$

Allowing the model to 'remember' approximately ten steps to the past depending on the

task at hand. We will call these layers memory cells being the ones explained on equation 3.6 the simpler ones.

In RNNs, we will call this memory at a time t, the state, which in our previous example was equal to the output y, although we will see that this is not the case to all the architectures. This fact allows the construction of four different types of RNNs (Figure 3.9), which are:

1. A sequence to sequence network is good for tasks like time series prediction (stock prices).

2. A sequence to vector network is good to study speech/texts where the output could be the opinion of the speaker/writer (sentiment analysis).

3. A vector to sequence network is in charge of outputting a particular sequence given a specific vector, for instance, a text given an image.

4. The combination of sequence to vector networks with vectors to sequence networks represents what we call an encoder-decoder architecture, being one of its multiple uses language translation.

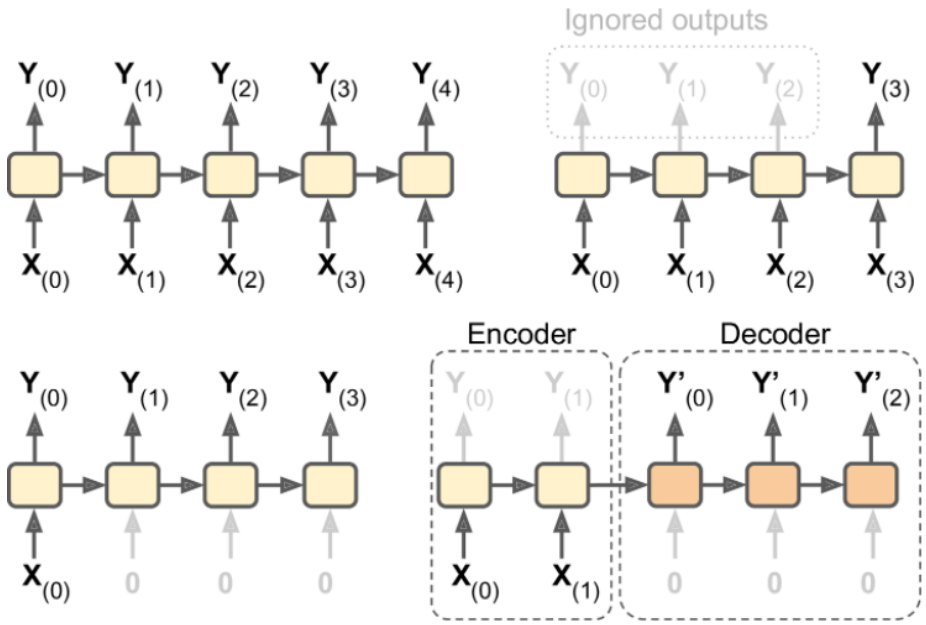


Figure 3.9: Types of RNNs based on inputs and outputs

The training of an RNN is relatively straightforward as the only thing we have to do is unroll it through time and perform the forward pass along with the backpropagation step as

we did in previous networks. The problem that appears at backpropagation is that at each neuron, we would have to calculate the gradient not only for the output of the last neuron but for all of them at the same time (python libraries automatically solve this problem).

As it happens with other NN, the deeper we go, the more problems start to appear. However, as we saw in CNNs, RNNs have several approaches to overcome all these issues.

The first problem is the unstable gradient which can be solved with parameter initialization, dropout, regularization, learning rate, and optimization. Implementing Batch Normalization between layers could seem like a good idea; however, it has been proven that this does not improve performance. As an alternative, the literature suggests using layer normalization instead, which in contrast with Batch Normalization (BN), the operations are performed across the feature space and not the batches. However, another problem is the non-saturating activation functions that tend to increase the weights' values until they explode, which is why in RNNs, the standard activation function is the hyperbolic tangent(saturating function).

Implementing Batch Normalization between layers could seem like a good idea; however, it has been proven that this does not improve performance. As an alternative, the literature suggests using layer normalization instead, which in contrast with BN, the operations are performed across the feature space and not the batches.

The second problem to deal with is the short-term memory problem that these cells have. As a solution, researchers developed the so-called Long-Short Term Memory (LSTM) cells, which outperformed their ancestors (explained above), leaving them unused. To put it simply, the only difference from the previous cells is that the state is divided into two vectors h(t) for the short term state and c(t) for the long term.

However, as we open up the box, we see how these cells discern what elements to remember and forget, and it is not that simple. We explain below how this process works.

As it can be seen in Figure 3.10, we have three different inputs, which are the previous vector c(t-1), the previous state h(t-1), and the input signal x(t). The last two are fed into four different fully connected layers whose outputs will be called f(t), g(t), i(t), and o(t). The g(t) function contains the main features from the input signals x(t) and h(t-1), whereas the other three are called gate controllers whose purpose is to determine whether to close or open the forget gate (f(t)), the input gate (i(t)) and the output gate (o(t)). Another renowned cell very similar to the LSTM one is the Gated Recurrent Unit (GRU), a simplified version of the LSTM that performs identically well. Hence, it is more used

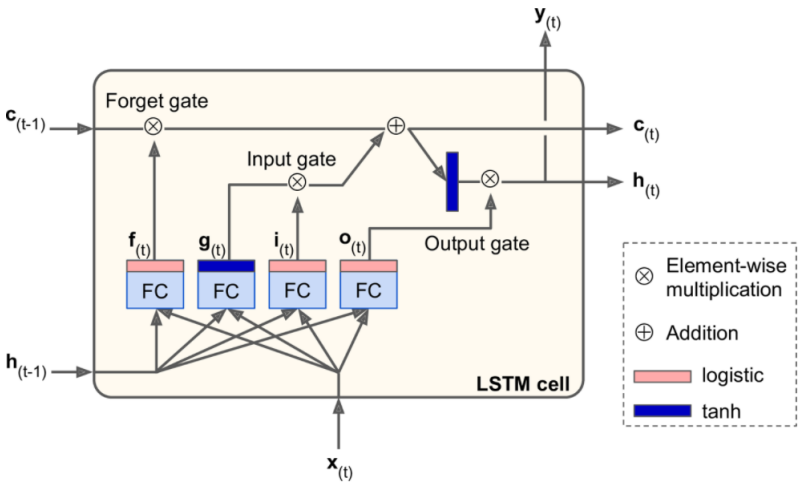


Figure 3.10: LSTM memory cell architecture [17]

nowadays.

$$\tau_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \qquad \hat{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{3.7}$$

$$tau_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \qquad c^{<t>} = \tau_u * \hat{c}^{<t>} + \tau_f * c^{<t-1>} \tag{3.8}$$

$$\tau_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o) \qquad a^{<t>} = \tau_o * \tanh c^{<t>} \tag{3.9}$$

Although these networks have been improved in terms of memory, they start to lose control on signals over a hundred samples, which are standard nowadays. In this project, in particular, our raw ECG signals can have sampled over 150K steps. One approach to elude this problem is to preprocess the data shortening the input size by using 1D convolutional layers, reducing the feature space considerably, and introducing LSTMs or GRUs. In this project, the first approach that we will take will be to preprocess the signal to shorten its length and prepare it to feed our architecture, as we will see in Chapter 5.

Another concept to note is that even though some predictive tasks require only a causal analysis (looking in the past and the present), in some others like speech and NLP, it is also crucial to look at the future. For example, while analyzing the syntax of a sentence, some of its elements do not necessarily appear ordered the same way. What is worse, sometimes we need parts of the end of the sentence to label others from the beginning. A brilliant idea that is pretty straightforward nowadays to tackle this problem is to use bidirectional RNNs. Explaining them is simple; the input sequence needs to be reversed and then fed to the standard RNN layer and combine its results within another network with the data showed in its logical order. The only difference between using only one LSTM is that the number of outputs is doubled as we use x2 more networks.

CHAPTER 4

PhysioNet/CinC datasets

The internet is a powerful tool for extracting data in many different fields. The problem is that this data usually needs a large number of working hours to be useful for ML. Finding a dataset that fits the requirements of a plan is a difficult task due to the heterogeneity of data. Usually, a final dataset for a specific purpose results from merging more than two or more completely different sets. Thankfully, the project task at hand was already gathered up for us. In this chapter, we will study the dataset that we will be using in the project. Firstly, in Section 4.1, we will reference the source of the data, and along with it, we will do a basic analysis of its features. Secondly, in Section 4.2, we will go through the different classes of the dataframe, explaining each of the diseases we will be dealing with; this will allow us to better understand the classifier's results when performing error analysis. Finally, in Section 4.3, we will explain a key characteristic of the final dataset, which is critical to consider during the whole project, multi-labeling.

4.1 Dataset overview

As we explained in section 1.1, the best tool for clinical diagnosis of CVDs is the ECG. We require skilled practitioners to interpret such recordings; therefore, having a machine helping them in this job would represent an incredible progression in the field. In our project, the final dataset contains several recordings of 12 lead ECGs. The big problem that its detection has faced through time is that even though some algorithms show promising results, their performance drops drastically in heterogeneous datasets. The purpose of the Physionet/CinC 2020 challenge was to address this heterogeneity by gathering up five datasets from different centers worldwide. Even though there were some standards when uploading them, we could find important differences in the type of recording. In this section, we will extract every source of data and analyze it independently. Finally, we will merge it (the preprocessing steps will be shown in Chapter 5, here we will describe the data as it was given in the challenge).

Table 4.1: Sources for the Physionet/CinC 2020 challenge

Database	**Records**	**Sample frequency (Hz.)**	**Duration (per record)**
China Physiological Signal Challenge (CPSC)	6 877	500	6-60 s.
St. Petersburg INCART	75	257	30 min.
Physikalisch Technische Bundesanstalt (PTB)	549	1000	10 s.
PTB_XL	21 837	500	10 s.
Georgia database	10 344	500	10 s.

As we can see in Table 4.1, the fundamental frequency of the instances would be 500Hz. However, as the objective is to build a robust classifier, we would like to use every recording as long as it keeps on with the data standards. While working further on the project, we decided to put aside the Holter ECGs (INCART database) as their instances were too different from the rest (256 Hz. and 30 minutes long and only 57 recordings). In the end, we have 43 027 recordings, where each of them follows the format shown in Table 4.2.

Each recording is stored in a WaveForm DataBase (WFDB) format which contains the following extensions: the header, the signals, the annotations, and the calibration files. In our project in particular, as we can see in the following table, we have only two of them: the header and the signal files. In the first one, we can find descriptive parameters such as age(Age), sex (Sex), diagnosis (Dx), prescription (Rx), history (Hx), and symptoms or surgery (Sx). On the other hand, we also have values referring to the signal calibration

Table 4.2: Description of PhysionetCinC features

			Type	*Description*
WFDB	**.hea**	***Age***	Integer	Age of the patient
		Sex	String	Sex of the patient
		Lead	String	Derivation of the recording
		Disease	Integer	Snomed CT code
	.mat	***Recording***	Sequence	Serialized values from each lead

method, which will be repeated once for each lead, for instance: the number of bits, offset value, amplitude, signal resolution, baseline value, the first value of each signal, a checksum value and a recording name (as the WFDB format suggests, these values should have been recorded in the calibration file instead).

Now we will explore some dataset features graphically to gain more profound insights into the data (Figure 4.1). Firstly, we could see that most of the final dataset samples will be from PTB_XL (57.2%), which contradicts a little the aim of the challenge, which is having a well-distributed dataset. Afterward, we can see that the age histogram is tail-heavy, reflecting the predisposition of suffering from heart diseases as you get older (Figure 4.2). However, this tail-heavied shape usually makes the prediction task harder; to solve it, we would have to transform the data into a more bell-shaped distribution. Finally, the sex values seem balanced, which is usually the ideal case when performing a classification task.

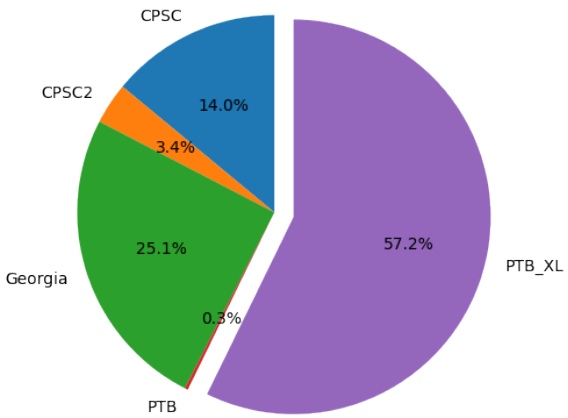


Figure 4.1: Final dataset's distribution

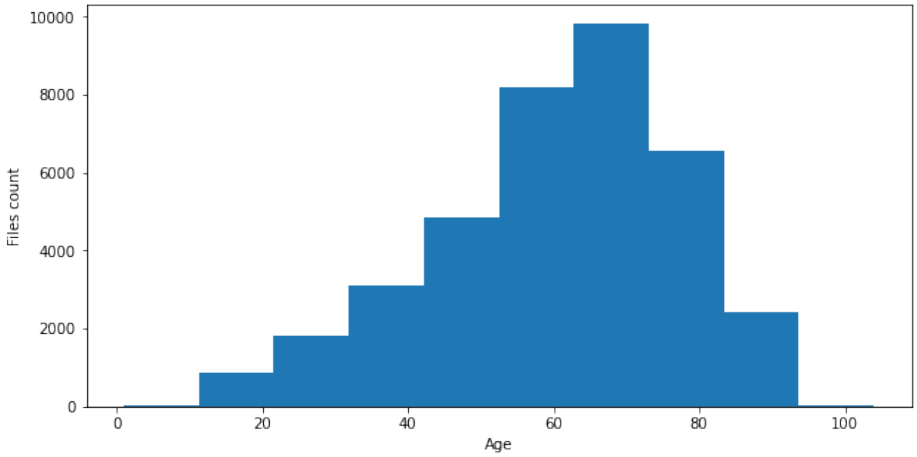


Figure 4.2: Bar graph of the age distribution

The remaining data consists of two more columns, the ECG signals coming from each derivation, and the data labels of each record. The signal analysis will be treated in chapter 4. Lastly, referring to the data tags, 111 possible diagnoses are present in the whole dataset. However, due to variability and noise, the challenge's organization decided to reduce the labels to focus on the most harmful/frequent ones, which leaves us with 27 different pathologies to classify (Figure 4.3).

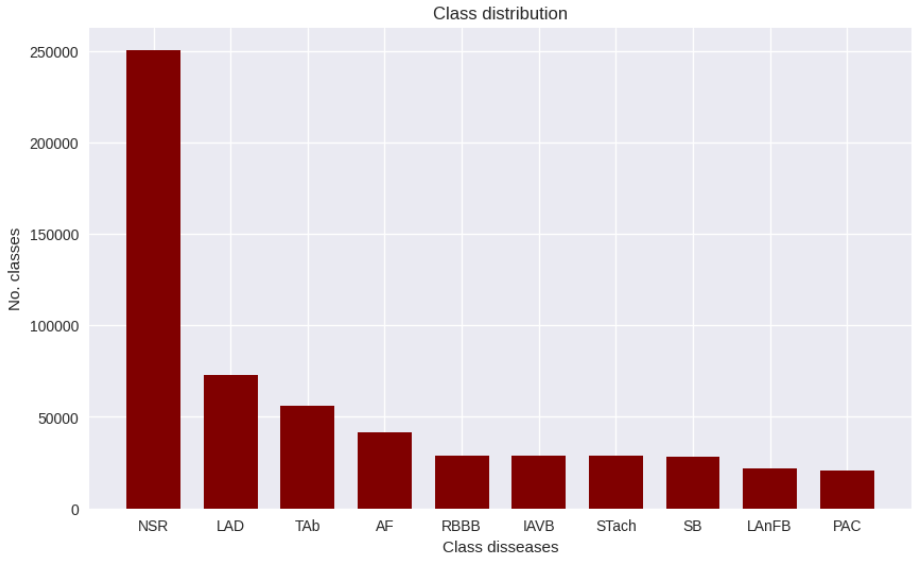


Figure 4.3: Top 10 distribution of diseases over the full dataset

4.2 Exploring dataset's diseases

We will briefly develop the medical categories to explain some of the most common patterns within the data and their specific features. When a physician is studying an ECG, he must pay attention to a varied set of anomalies within each wave, referred to as the form, duration, and polarization. With all these targets in mind, he must discern between a wide range of possible wave abnormalities. Several classes have been extracted from this source.[22]

Normal Sinus Rhythm (NSR): Before explaining them, it is logical to justify the reference from which the rest of the signals will be called pathological. The signal emitted from a healthy and well-functioning heart is called Normal Sinus Rhythm. It is characterized by a cardiac frequency that goes from 60-100 beats per minute. The QRS complex is preceded by a P wave which is positive on I and II derivations and inverted in the precordial ones. The PR intervals remain constant, and the QRS step takes about 100 ms. We will try to show you how the leads represent each pathology. This way, we will be able to review the classifier's results with more references.

Bradycardia (Brady): Its direct implication in the ECG signal is the reduction of the cardiac frequency. The apparition of delayed heartbeats can be mainly attributed to a malfunctioning SA node. However, the causes that provoke this are varied: damaged tissue, infections, hypothyroidism, and medicines.[23]

Atrial Flutter (AFL): It starts in the heart's atrial chambers. Its direct implication in the ECG signal is the significant rise of the cardiac frequency, which is why it is classified in the same group as the **Supraventricular tachycardias (SVTs)**.

Atrial fibrillation (AF): The main features of this wave are its irregularities. It does not follow a specific pattern, but its characteristics are the lack of P waves, the absence of an isoelectric line, an irregular ventricular rhythm, and fibrillation waves that can be misinterpreted as P-ones (Figure 4.4).

First degree AV block (IAVB): It is characterized by a PR interval higher than 200 ms, which sometimes causes overlap between the T and P waves from consecutive beats, leaving them unrecognizable. The anatomic cause of this delay is the blockage of the SA and AV node (Figure 4.5).

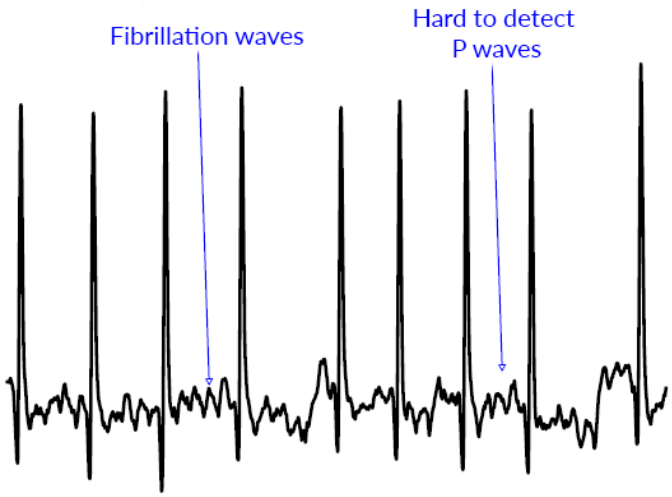


Figure 4.4: Six second representation of AF

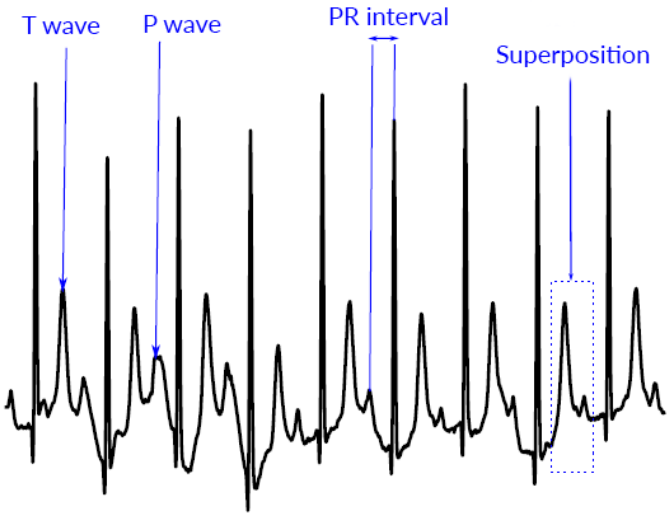


Figure 4.5: Six second representation of IAVB

Left Axis Deviation (LAD): It is a condition where the mean electrical axis of the ventricular contraction of the heart lies in a frontal plane direction between -30º and -90º. This is reflected by a QRS complex positive in derivation I and negative in aVF and II (Figure 4.6).

Left Anterior Fascicular Block (LAnFB): It is an abnormal condition from the left ventricle of the heart, similar to the LBBB. Its main features on the ECG have LAD usually between -45º and -60º, qR patterns in leads I and aVL, rS pattern in II, III and aVF.

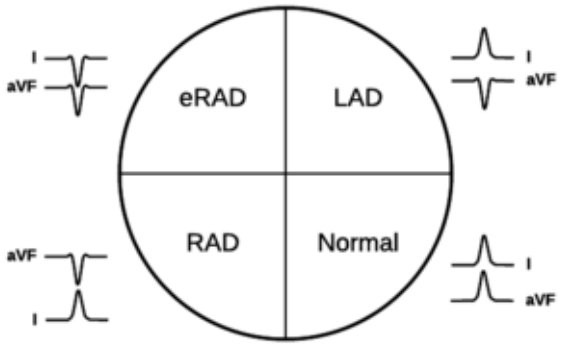


Figure 4.6: Axis deviation features

Left Bundle Branch Block (LBBB): It appears when the activation of the left ventricle is delayed. In the ECG, we can see that the QRS complex is longer than 120 ms, a QS or rS complex in V1, and an M-shaped R wave in lead V6 (Figure 4.7).

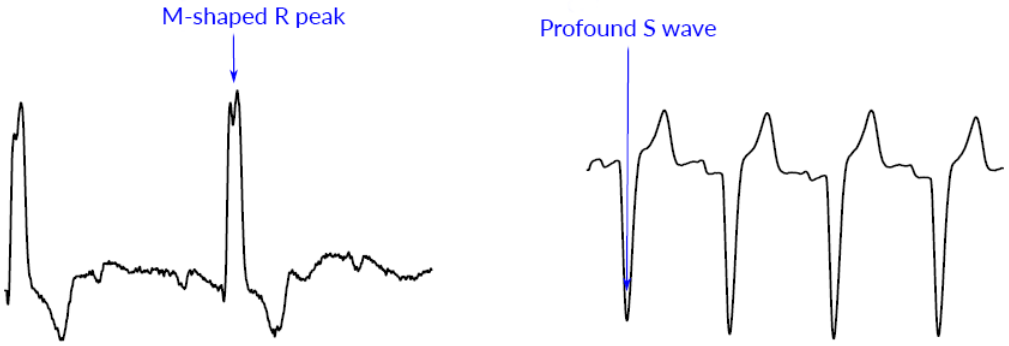


Figure 4.7: LBBB wave patterns

Low QRS Voltages (LQRSV): It happens when the amplitude of the QRS complexes is inferior to 5mm, or in the precordial leads smaller than 10mm.

Nonspecific Intraventricular Conduction Disorder (NSIVCB): It is diagnosed when LBBB and RBBB are discarded. Therefore, its definition might be a little blurry. In adults, it is defined by a QRS duration greater than 110ms and when the two conditions described before are rejected. As it might be imagined, diagnosing it is challenging as several anomalies are related to widened QRS complexes.

Pacing Rhythm (PR): It occurs when the signal emitted from the patient's ECG presents a newer spike-shaped wave due to a pacemaker's presence. The position of the spike varies

depending on the location where the pacemaker is doing the job. We can find atrial, AV sequential, ventricular, and demand pacemakers (Figure 4.8).

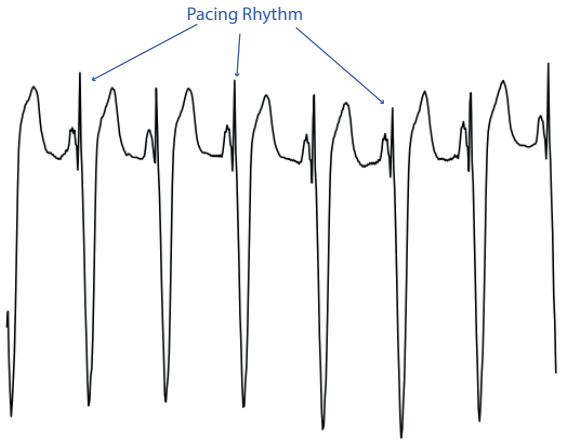


Figure 4.8: Six second representation of PR

Premature Atrial Contraction (PAC) - Supraventricular Premature Beats (SVPB): It appears when cardiomyocytes emit irregular pulses inside the atrium. Its main feature is the apparition of an abnormal P wave whose shape might vary depending on the location of the malfunctioning cardiomyocytes. On some occasions, it can affect the SA node, provoking a skipped beat known as a palpitation.

Premature Ventricular Contractions (PVC) - Ventricular Premature Beats (VPB): Analogous to the one described above, but its asynchronous activity comes from the ventricles. Its features are a prolonged QRS complex (>130ms), ST discordant segments with a compensatory pause, and changes in the T wave.

Prolonged PR Interval (LPR): The PR interval measures the distance between the starting point of the P wave (atrial depolarization) to the beginning of the QRS complex. The average times of this interval range between 120-200ms. A long PR interval is usually a symptom of having an IAVB.

Prolonged QT Interval (LQT): It happens when there is a problem during the repolarization phase. Its diagnosis is not an easy task because this interval is highly variable in both healthy and pathologic patients. As a solution to this problem, a scoring system called Schwartz is usually used where it checks ECG features along with genetic ones.

The first approach to diagnose it is by measuring the QTc. The QTC is normal in 95% of males when its signal lasts less than 450 ms, and some suggested values between 470 and 490, corresponding with the 99th percentiles.

Q-wave abnormal (QAb): Q waves are considered pathologic if they are either too long (>200ms) or too deep (>5mm). Q waves that are pathologically deep but not wide are representative of ventricular hypertrophy. If both pathological signs are present, it may imply that there is a myocardial infarction.

Right axis deviation (RAD): It occurs when the QRS electrical axis is shifted to 90^{o}-180^{o}. The causes of this shift can be RBBB, lung disease, right-sided heart strain, and right ventricular hypertrophy.

Right Bundle Branch Block (RBBB): Caused by a delay in the right ventricle, this kind of wave can be diagnosed by finding ST depressions, T wave inversions in V1, V2, and V3, and a secondary R peak with an M-shaped form.

In this project, the Complete Right Bundle Branch Block (CRBBB) and RBBB were identically treated in scoring and diagnosis. Even though their SNOMED CT code is different, their samples refer practically to the same kind of wave. We can also observe the **Incomplete Right Bundle Branch Block (IRBBB)**, which is thought to be associated with abnormalities of the peripheral Purkinje system. Its ECG features are very similar to the ones described in the RBBB.

Sinus Arrhythmia (SA): It is considered to be a normal variation found between young adults. The main characteristics of this wave are an R-R interval greater than 120ms, P waves showing uniform morphology, and a P-P intervals greater than 120ms. This last feature has brought controversy as sinus arrhythmia is thought to be regulated by the stimulus of the vagal nerve. Thus, increasing or reducing P-P with exhalation and inhalation.

Sinus Bradycardia (SB): It is represented by a slow heartbeat that occurs when the SA node starts firing at a slower rhythm. The average activity would be to have between 60 to 100 bpm. However, in these cases, the beats are significantly slowed (lower than 60). This state is usually not pathological, and it frequently happens in athletes, children, and young adults.

Sinus Tachycardia (STach): In sinus tachycardia, we can find a normal P wave on lead II preceding every QRS complex. Its diagnosis is usually made by measuring an atrial and ventricular rate greater than 100 bpm.

T-wave Abnormal (TAb): A standard T wave has the following characteristics: It is upright in all leads and inverted in V1 and aVR, smaller than 5mm in limb leads and <10mm in precordial leads (varies a little from male to female), and the typical measurement to study the T wave is the QT interval.

We can classify T wave abnormalities within six different categories:

- Peaked T waves: A typical pattern in hyperkalemia with tall narrow, and symmetrically peaked T-waves.
- Hyperacute T waves: to measure them, the physician must pay attention to their relative size compared with the QRS complex. It is also detected when there is a loss in precordial T-wave balance.
- Inverted T waves: a pathological inverted T wave is usually symmetrical and deep (>3mm). It is normal to see inverted T waves in children where, given their physiology, the ventricular forces are dominant.
- Biphasic T waves: caused by myocardial ischemia and hypokalemia. In the first one, we have a positive wave preceding a negative one, whereas the opposite occurs in hypokalemia.
- Camel Hump T waves: occur when the T wave is mixed either by a prominent U wave or its preceding P.
- Flattened T waves: represent the absence of a T wave as it happened in biphasic T waves where the causes can be ischemia and hypokalemia.

T-wave Inversion (TInv): It is a subgroup within abnormal T waves. It can occur in all these situations: myocardial ischemia, BBB, ventricular hypertrophy, pulmonary embolism, hypertrophic cardiomyopathy, raised intracranial pressure. In Annex C.4 we will develop all these conditions.[24]

4.3 Multi-labeled data

Another remarkable characteristic of the dataset is the presence of multi-labeled documents.

When we are talking about real world-data, uniqueness is not a standard feature[25]. For example, a document in sentiment analysis can express positive and negative emotions without a clear distinctive of which one is dominant. However, due to the task's reduced dimensionality (positive, neutral, or negative), several methods have been developed to differentiate between these varied situations.

If the dimension of the labels increases, the appearance of more complex relationships between classes becomes a reality. In our project, with 27 different labels, several inputs are placed in more than one class. When studying the ECG signal, we can find one or more cardiac anomalies as each beat can be unique, which is why the data developers assigned multiple tags to the dataset documents.

The next step will be to plot on a confusion matrix (Figure 4.9) the different relationships within our dataset. With this data, we can infer some correlations between labels. It can also serve as a tool to study both the classifier's results and the fundamental anatomic principles that make these cardiac anomalies appear together, which is quite important. It can help us check if the classifier is acting correctly or making completely incorrect assumptions. The process goes as follows:

1. Once we have successfully extracted every record, we start associating them by building a matrix that shows the relative appearance of every disease given each of the classes.
2. The advantage of this approach is that we can quickly check if this operation was performed successfully by checking if the matrix is symmetrical within its diagonal.
3. Next, if we plot these numbers in a color map, the diagonal values will be brighter than the rest as you are querying a particular disease within itself. Therefore, the approach chosen was to convert every number from the main diagonal to zero.
4. Finally, some diseases, due to their relative appearance, impede the rest of relationships. Therefore, after writing them down, the approach was to transform them to zero to reveal the remaining connections within the data (e.g., Left Anterior Fascicular Block (LAnFB) and Left Axis Deviation (LAD)).

By looking at the resulting confusion matrix (Figure 4.9), we could easily see some relationships between the diseases. Our aim in this section is to explore disease relationships, not their frequency of appearance:

- **T Wave Abnormal (TAb):** It is one of the most common ECG patterns related

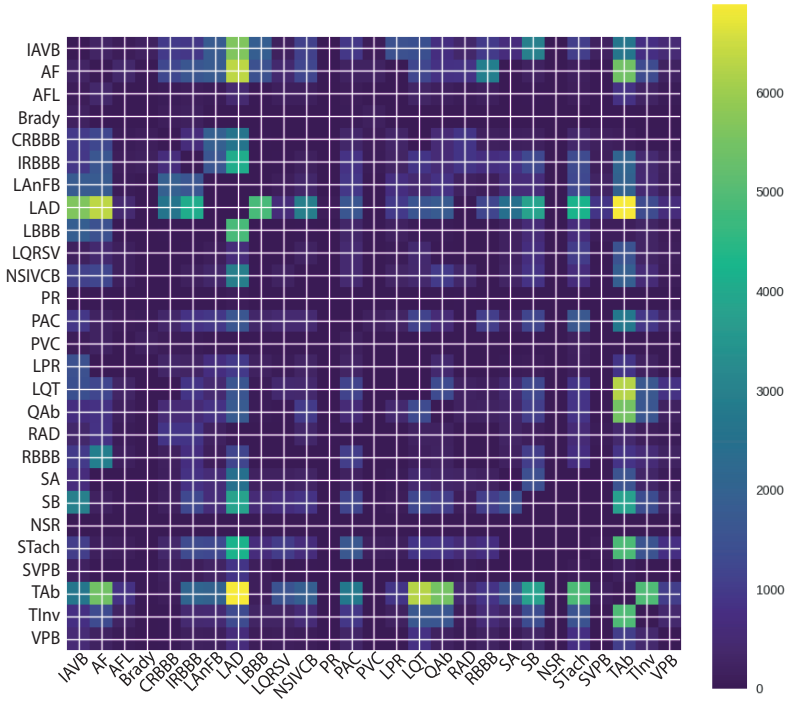


Figure 4.9: Confusion matrix showing labels relationships

to many diseases. One of the main reasons for that is that the causes of abnormal T wave patterns can come from various sources: the Q wave, the beats per minute (if it's a tachycardia or bradycardia), premature atrial contractions, atrial fibrillations, and many others.

- **Left Axis Deviation (LAD):** Common causes of this condition are the LAnFB and the LBBB. This pathology is related to variations of the mean electrical axis of the ventricular contraction. In addition to it, we can see other strong relationships like IAVB and AF. However, this last one might be boosted by its frequency of appearance.
- **1st Degree AV Block (IAVB):** We can see it is strongly related to Sinus Bradycardia which makes sense as the blockage of the AV node leads to a slower heartbeat.
- **Atrial Fibrillation (AF):** we can see that it is strongly related to RBBB and LBBB. However, looking at the theory, it makes no sense that RBBB was more correlated than LBBB as AF is mainly associated with systolic heart failures and LBBB. The main motive found that justifies these relations are the dataset's imbalances.
- **T wave inversion (TInv):** It is good to explore these categories in particular as they serve as sanity checks. T wave inversion is a subgroup of T Wave Abnormal (TAb). Thus, it is logical that the confusion matrix showed them strongly correlated.

CHAPTER 5

Development of classification models

This chapter will explain the process that drove this project to its final Neural Network architecture. As a starting point, in Section 5.1, we will talk about the signal pre-processing step, where we tested several ways to group and organize the data. Then, Section 5.2 talks about the final NNs architectures explained in Chapter 3. Finally, in Section 5.3, we will talk about the results from the most promising one, which will be the one to be implemented in the Decision Support System (DSS).

5.1 Pre-processing phase

As discussed in Chapter 4, the datasets gathered come from five different sources representing different diseases and recording techniques. The majority of the data has been filed with unified criteria. However, there are some discrepancies that we should take into account in the pre-processing steps. In addition, it is essential to note that we discarded the ECG recordings from the INCART dataset (Holter ECGs) as they are signals whose recording procedures are far from the majority of the dataset, introducing more noise and having an extremely large duration.

Initially, we performed format checks so that we could create generalized methods to

extract data features from the '.hea', '.mat' files, and check the feasibility of merging them successfully. Here is an outline of the whole process:

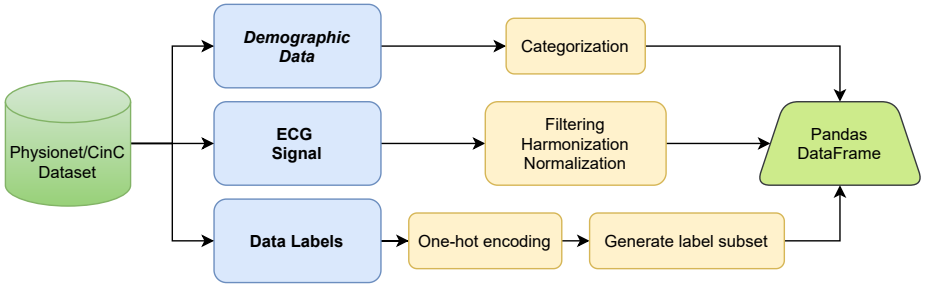


Figure 5.1: Pre-processing steps pipeline

5.1.1 Demographic data

The next step was to decide the format of the different features. From the header file, the age will be extracted as an integer. The first problem that we encountered in this field was the apparition of missing values. We could either remove the samples with missing ones or replace them with an averaged age to preserve the dataset's integrity. We decided to follow the second option. The sex field was of type string. Thus, we transformed it to a binary column where 0 corresponded to Male and 1 to Female. The diagnosis field (Dx), which has the tags from the dataset, will be treated in the following paragraph. Other features were prescription (Rx), history (Hx), and symptoms/surgery (Sx), which turned out to be empty fields; hence we did not take them into account.

Finally, we will talk about the diagnosis preprocessing step. As we stated in Chapter 4, there were a total of 111 diseases in the data. However, to diminish class imbalances, the organization decided to reduce them to 27. Consequently, we had to extract from the dataset the data that contained those classes (passing from 43027 to 37716 recordings). In the beginning, this might seem like an easy task to perform; however, the multi-labeled characteristics of the dataset tangled its removal. The final result turned into a set of sparse binary arrays of shapes (1, 27), with ones representing the presence of a specific wave pattern of the ECG recording.

5.1.2 Signal preprocessing

In this section, we will describe the preprocessing steps applied to the ECG signal. As we explained in Chapter 3, the signals from the .mat file contain 12 different ECG recordings,

each of them associated with one lead. We decided to divide the signal preprocessing into three distinct steps.

5.1.2.1 Noise filtering

The ECG recordings from the dataset are in a raw format, which means that we can find low-frequency noises from the patients' movements and high ones given the electronic elements. We filtered the signals using a Butterworth filter of order 1 with cut-off frequencies of 0,001 Hz and 15 Hz (These values were taken from the Challenge's sample code [26]). Then we converted those values to their normalized frequency, applying the Nyquist theorem where $fn = fs/2$. Lastly, the delay introduced by the filter is corrected (Figure 5.2).

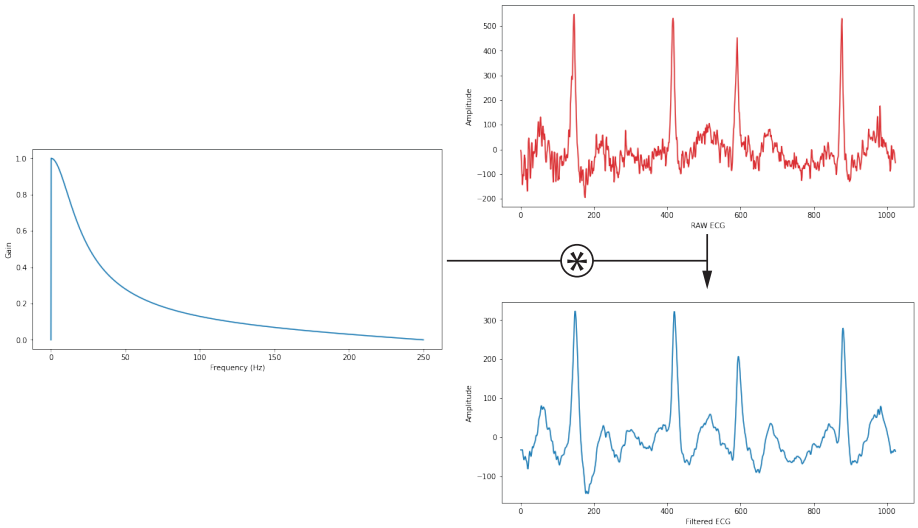


Figure 5.2: Filtering process

5.1.2.2 Input harmonization

One important feature that should be present in every inputted sample is shape uniformity. We developed a varied set of methods to obtain this feature. In the end, the objective was that each instance must have the same duration. Several examples from the dataset have different sample frequencies; depending on the signal at hand, we have to perform various transformations so that the ECGs are equally long.

In this project, we proposed two different ways to proceed with this task where the chosen approach will ultimately determine the data structure that we will be feeding into the NNs.

- **Crop and sample**

  In this method, we will crop the signal into equally sized segments of 3000 and 6000 samples (Figure 5.3), which was decided by studying the averaged length of the inputs. In terms of duration, this corresponds to 6 seconds fragments from the ECGs. However, having such lengths generate a dataset computationally expensive to train. As a solution, we decided to resample it to 256 samples in exchange for losing some quality (we used powers of two to speed-up computations). In 1000 Hz inputs, the resample affects the signal twice as much, having a subsample ratio of 23,43 and 11,72 in 500Hz ones.

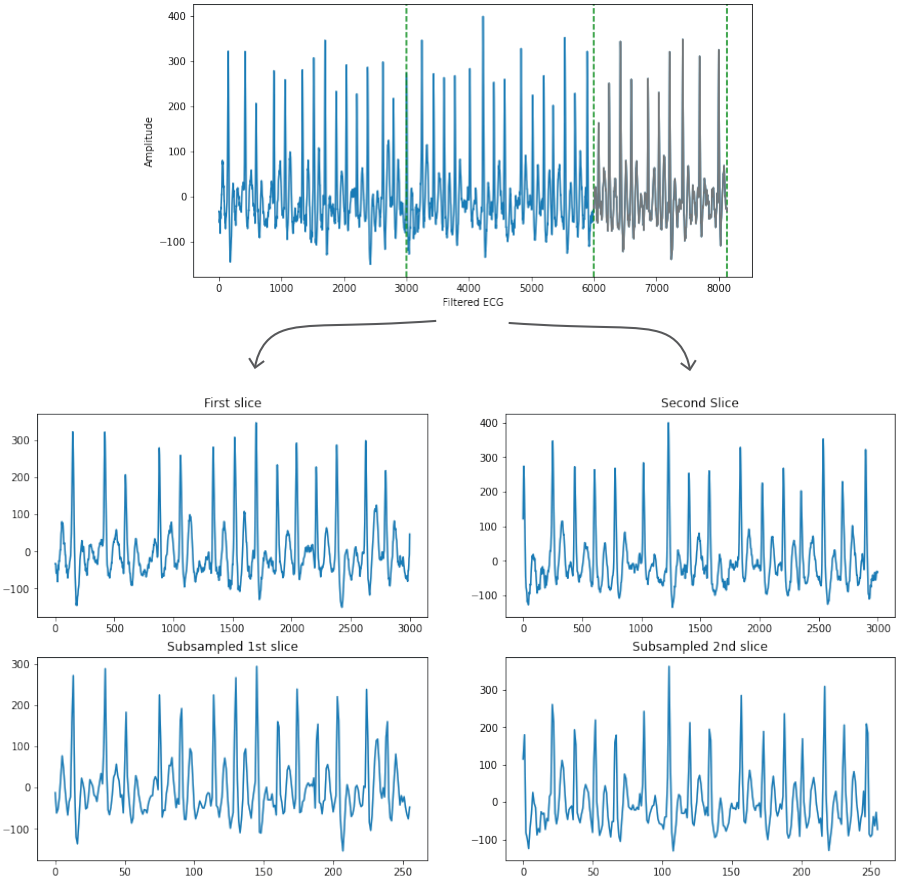


Figure 5.3: First harmonization method

- **Random sampling**

  The objective of random sampling is to obtain a single representation of each input. With this, we are not cropping the signal in slices of a predefined shape but defining a window that will be in charge of randomly extracting the ECGs information (Figure 5.4). Identically to the previous paragraph, the sliding windows will be of 6 seconds (which in terms of samples refer to 3000 and 6000 from signals with 500 Hz

and 1000 Hz respectively). In this method, we have to consider two different case scenarios:

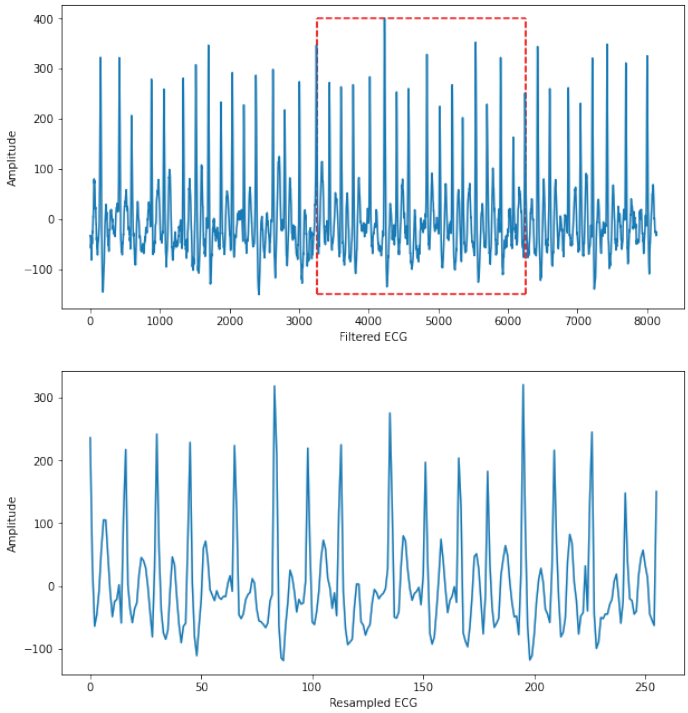


Figure 5.4: Random crop example where signal is greater than window size

- **Signal size greater than the window:** In this case, the objective will be to define a window within the signal's range (without exceeding their sample boundary).
- **Signal size smaller than the window:** Here, as we do not have enough samples to fill our entire window, the process carried out will be to randomly place the signal into an array of zeros where its size was frequency reliant.

We have two possible options, taking each lead as independent ones by defining a random window for each of them or defining a specific one for each group of 12 lead inputs. We decided to carry out the second option because, in the former one, the sequential data of each lead will not be consistent with each recording.

5.1.2.3 Signal normalization

Another essential step will be to normalize the signal as scale is a factor that strongly affects NNs and their computation speed. The algorithm to perform this operation is the following:

$$x_N = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5.1}$$

Finally, we will merge the demographic data, the ECGs, and the class labels using Pandas. This library allows us to transform and extract the data to be well-organized and replicable in online servers like Google Drive.

5.2 Development of the Neural Networks

From the previous section, we generated three different datasets. The first one will be called m1 as we extracted it with the first harmonization method. The other two will be called m2_256 and m2_512 as we gathered its signals with two subsampling resolutions (256, 512). Thus, we developed different NNs architectures as the input and output features are slightly different depending on the datasets. However, before moving on to the specific characteristics of the network, we will explain the general approaches followed to build them.

We will use the Google Colab (C.1) environment to train and develop our models. The servers from Google grant an enormous computational power compared with our laptops. In particular, with a free account, we can work with 13GB of RAM and a GPU NVIDIA Tesla T4 with 16GB of dedicated memory. In contrast, working with this Google cloud environment have constraints like a time limit of 12 hours and a limited number of GPUs (if the cloud is overcrowded, they will not let you a GPU resulting in a low computation speed). To know more about the troubleshooting of these problem go to Annex C.3.

We are using Keras (Annex C.1) to construct Neural Networks, a library that works on top of Tensorflow, allowing us to replicate various architectures intuitively and practically. However, before feeding the data to the network, we need to carry out different indispensable steps.

Firstly, the test data from the Physionet/CinC competition is private. Hence, we will need to split our dataframe into three different sets that we will call train, validation, and test using the renowned Scikit Learn library (C.1). We will use the train and validation sets to train the different NNs and the test set to evaluate the performance over unseen data. Once we had the final selected model, we could prepare the network with the whole data before uploading it to the DSS. The splits carried out in this project were: 80% train, 10% validation, and 10% test. Before splitting, we will shuffle the data as some merged sources tend to have different class distributions. Additionally, as the ECGs and the demographic

data are different, we split the three groups into qualitative and sequential samples.

Secondly, we have to transform the data regarding the different NNs. Here, the reshape method from the NumPy library will be of great utility. For this task, we had to focus on the inputs and the output shapes, whether we are using RNNs or CNNs (2D/1D), and also that the class labels from the output layer were ideally suited.

Regarding the architecture, in general, our model will have three main branches that will converge towards a series of Dense layers. Each of the branches will correspond to either the ECG signals or the demographic data. Using the Concatenate layer from Keras, we can build the architecture that we desire. We can even combine all the different approaches in parallel or in a sequential order. An additional feature to bear in mind is the configuration of the output layer. Our output samples are multi-labeled. Therefore, our NN will not converge using the Softmax activation function as they will always try to output joint probabilities. As a solution, we needed to implement a layer that treats each of the 27 different outputs independently, and for that purpose, we selected the Sigmoid activation function.

Finally, before starting training, we need to define our loss function, the optimizer that will update the weights, and the different metrics that will show how our NNs are performing. Generally, we will use 'binary cross-entropy' as our loss function, 'adam' as the optimizer, and metrics such as the AUC and the accuracy to track improvements (Annex C.6.1). Furthermore, to avoid overfitting and deal with the 12-hour constraint from Google Colab, we will also implement specific callbacks in the model, such as early stopping, model checkpoints, and a Learning Rate (LR) scheduler. These features will be explained better in the following section.

As we have discussed so far in the previous chapters, the two main features of ECGs are its periodicity and its shape. In DL, in general, there are two types of networks that excel at recognizing patterns and analyzing different forms (explained in chapter 2) those are CNNs and RNNs. Therefore, with the two methods described at the beginning of the section and the whole methodology proposed, we planned a series of tests that combines these architectures to see which combinations are the most promising ones.

5.2.1 The final architecture

After testing many different NNs, reviewing the data preprocessing steps, and reading various articles, we ended up creating our final architecture, which consists of a parallel combination of a Squeeze and Excitation Residual Network with bidirectional LSTMs. In

the end, all the data is merged into an ensemble of fully connected layers that leads to the final prediction layer.

First of all, we are going to talk about the SE-ResNet architecture (Figure 5.5). It consists of a convolutional layer followed by a series of RUs. Each of them contains two convolutional layers followed by a SE block. After every two residual blocks, we increase the number of filters by a factor of two. Additionally, when performing this change, as dimensions from the bypass connection from the ResNet do not match the output, we needed to carry out a computational trick explained in (Section 3.3)(it consists of performing a 1x1 convolution operation over the bypassed input so that it matches the shape of the output).

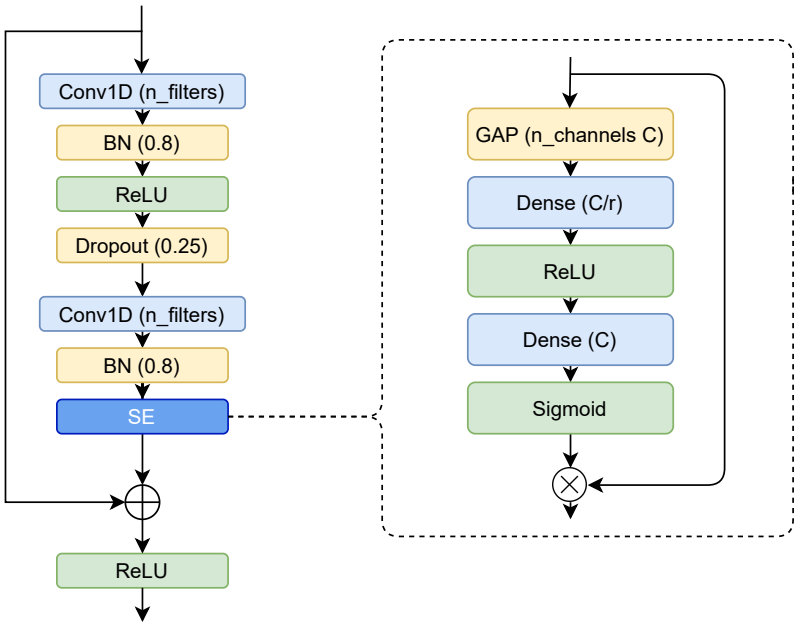


Figure 5.5: SE-ResNet architecture

If we unravel each convolutional layer, we would see that we made them up of:

- Batch normalization layers to normalize weights and speed up the learning process. With a 'momentum' of 0.8.
- ReLu activation functions, because it is a default element used in CNNs given its uniformity for negative values.
- Max pooling layers generate a set of pooled outputs that extract the most critical features of the previous CNN to reduce overfitting and speed up computations. We used the default value of 2 for the pooling layer. Therefore the inputs are halved on each iteration.

We could say that each of the RU is quite similar to the scheme described above, but on the inside, apart from the previous layers, we can find:

- Dropout layers turn off a given percentage of neurons solely to reduce overfitting. We set this parameter to a 25% dropout rate.
- Squeeze and excitation block (described below).

The SE block (Figure 5.6) aims at extracting channel importance and relationships, which translates into better predictions. Given its structure, it does not suppose an additional computational burden. Firstly, the data coming from the RU is 'squeezed' by using a Global Average Pooling layer so that the dimensions are reduced channel-wise. Afterwards, this data is fed into a multilayer perceptron whose number of units will be halved by a reduction factor of r=16. However, this value act as a hyperparameter that can be tuned.

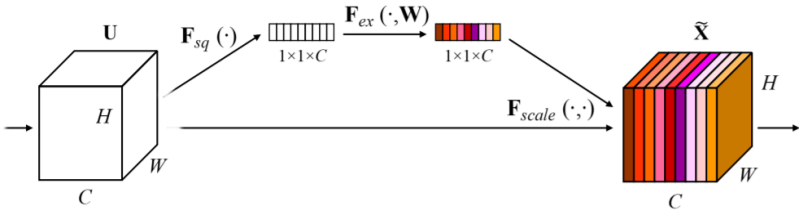


Figure 5.6: Explaining the SE block [21]

In parallel to this SE-ResNet block, we have two layers of Bidirectional LSTMs of 16 units each (Figure 5.7). We wanted to use the weights for further predictions, changing the 'return_sequences' parameter to True. In reality, when we define a bidirectional block, we introduce the input twice, one forward and the other reversed, which is why the resulting shape is doubled. Given previous research, these layers have proven to be excellent for predicting wave patterns on ECGs. Furthermore, we can use them thanks to the resample method applied in the preprocessing step. However, with signals larger than 1K samples, LSTMs' memory capacity drops significantly, which is why they do not use them in other teams from the competition.

Then, we will introduce the demographic parameters, sex, age, and leads (the leads parameter is only used in the m1 dataset) using a fully connected layer of 16 units.

Finally, all the outputs from the three different branches (SE-ResNet, BiLSTMs, and Demographic_FC) are merged using the Concatenate and Flatten method from the Keras library. The shapes that needed to be transformed came from the ECG data (SE-ResNet and BiLSTMs). All these outputs are then fed into a combination of three FC blocks of

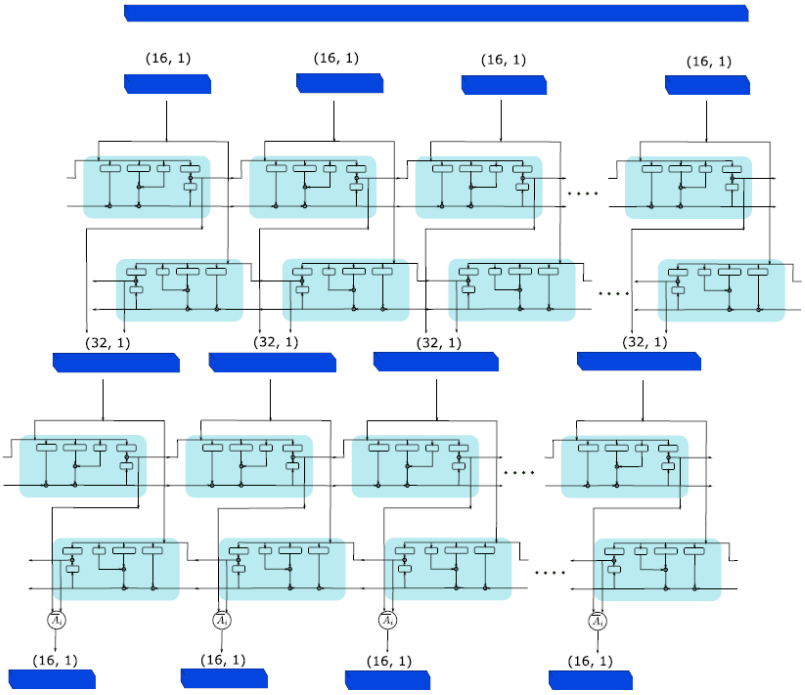


Figure 5.7: Bidirectional LSTM architecture [22]

1024, 512, and 64 units respectively that converge into a 27 sigmoid activation layer in charge of making predictions.

We have to define various callbacks to control the training process. This will allow us to save time and effort as the model will be automatically controlled by these conditions. The callbacks used in the project were:

- Early stopping: here, we monitored the validation loss with the patience of 5 epochs. If its value has not improved, the training will automatically stop.
- Save model on checkpoints: saves the file which achieves the best validation loss as it usually is the model which has generalized better over the dataset.
- LR scheduler: it is in charge of reducing the learning rate by a factor of ten if the validation loss had not improved for ten epochs. The initial value of our learning rate will be 0.1.

Before moving on to the results, we needed to design a method to transform the probability predictions from the sigmoid function into actual labels. We defined a fixed threshold of 0.5 that returns true if some output surpasses this value and gives false otherwise. In the next page you will see a final view of the m1 architecture (Figure 5.8)

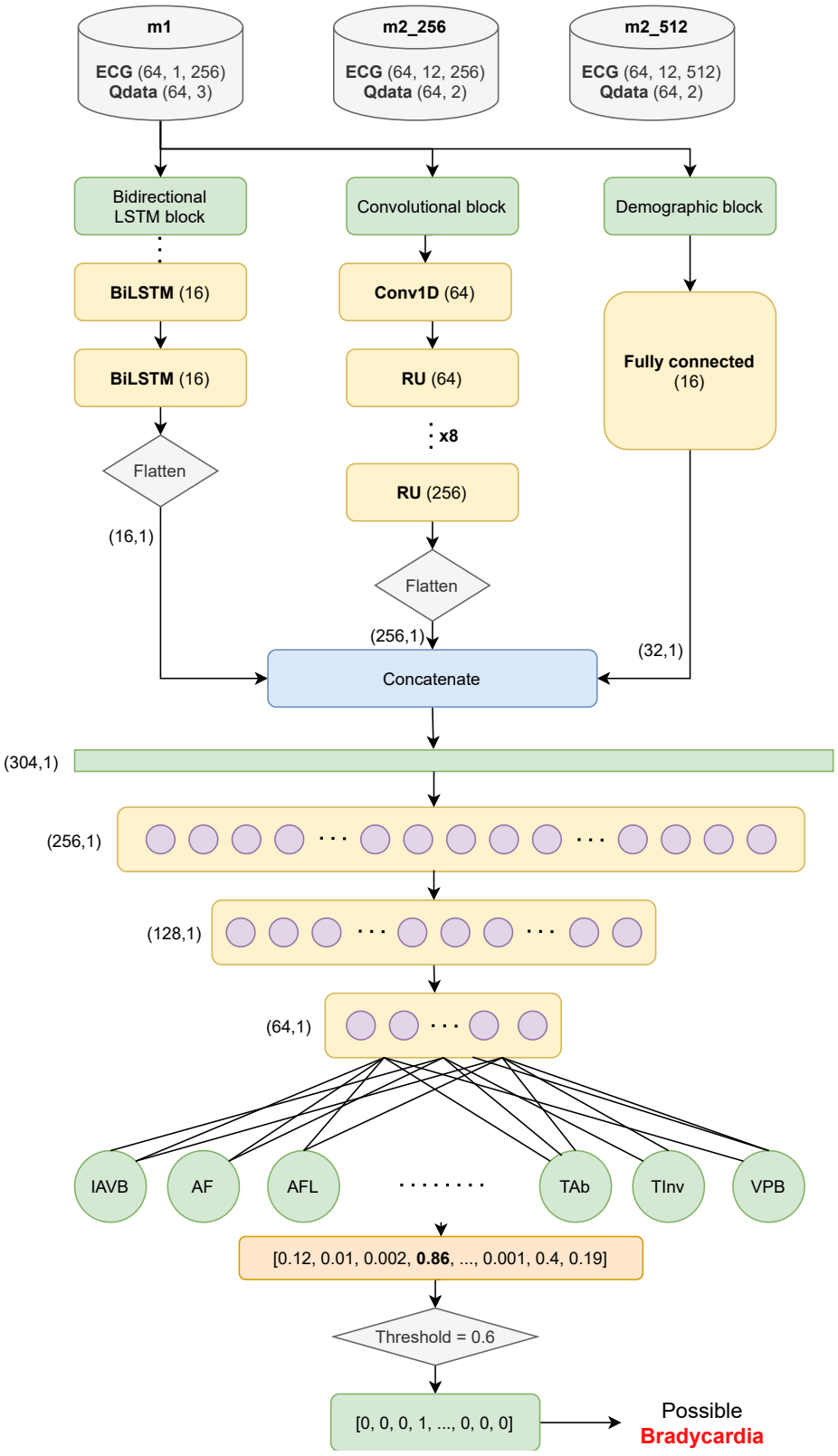


Figure 5.8: Final NN architecture & prediction process

5.3 Results summary

As it could be seen in Figure 5.8, we prepared three different datasets for testing purposes. In addition, we had to perform slight changes to the NN architecture to allow training. Therefore, the final splits on the data play a fundamental role in this part as they will be a feature to predict good performance.

Table 5.1: Analysing the NNs parameters

	Training shape	Validation shape	Test shape	Trainable parameters	Non trainable parameters
m1	439.840	54.980	54.981	3.243.191	7.104
m2_256	30.172	3.772	3.772	2.857.173	4.800
m2_512	30.172	3.772	3.772	2.857.173	4.800

In the m2 databases from table 5.1, the number of parameters is highly superior to the number of samples, which is a clear indicator of overfitting. However, in the other set, m1, thanks to the preprocessing step, we can see a much more balanced relationship between the samples and the NN parameters as we have approximately increased the number of inputs by a factor of twelve.

While evaluating the models, we checked the validation accuracy and loss against the training one, which is the fastest way to verify if there is overfitting in our data (Figure 5.9). After carrying out this process, we could see clear signs of overfitting after approximately 20 epochs on m2_256, which triggers the early stopping callbacks. The database m2_512 accuracy is not shown in this section as their results are very similar to the ones obtained with m2_256 (please, find the complete results in Annex C.6).

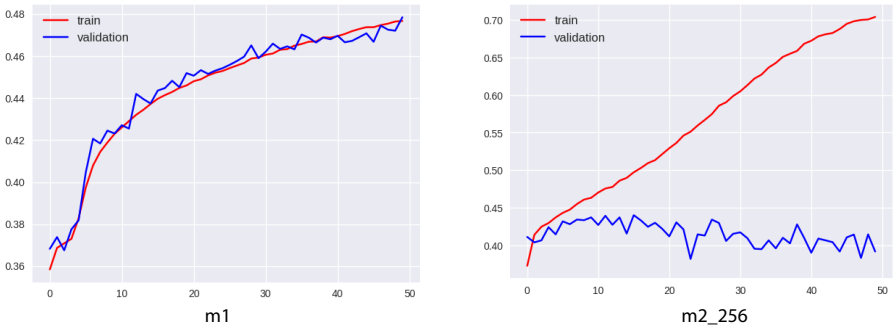


Figure 5.9: Accuracy test on both models

Now we will focus our attention on the m1 dataset. The first thing we would need

to do is define the metrics used for evaluating performance. Given previous works on the matter and previous projects in ML, we decided that the optimal metrics for this task will be precision, recall, and the F1 score. Afterward, we will examine the performance of m1 over each of the 27 diseases to spot weaknesses and possible improvements as it is the model that we will implement into the final DSS.

There is always a trade-off between precision and recall (increasing one translate into diminishing the other). This is why it is so important to define the requisites of the DSS so that it is prepared for that particular use case. As we did not have clinical feedback in our project, we went for the result that maximizes the F1 score. This value measures a model's performance incredibly well because it calculates the harmonic mean between precision and recall. If one of these values is low, it will heavily punish the F1 score. As the final point was to maximize that number, we iteratively tried several threshold values using a grid search method from Scikit Learn until the best F-score was found. The highest result obtained so far was with the m1 model, and it achieved a 55% score after tweaking the threshold parameter (which turned out to be optimal in 0.4).

Table 5.2: Model results with a fixed threshold of 0.5

	Micro average			**Macro average**		
	Precision	Recall	F-score	Precision	Recall	F-score
m1	0.73	0.38	0.55	0.59	0.15	0.19
m2_256	0.53	0.45	0.49	0.26	0.22	0.23
m2_512	0.55	0.44	0.49	0.24	0.19	0.21

Now we will analyze the different classes and how the model performed on each of them separately. As we have 27 labels, the images and values will not fit in this section, so we will only show and comment on the top performances and failures. While analyzing the results in-depth, we also have to consider the factor of data imbalances.

Our top 6 performances come from some of the most populated labels of the samples. As it was expected, the classifier learned to discern NSR with very high precision. However, in the top 6, we could find excellent results from low populated labels that the classifier learned to identify with a high chance. We were happy to see infrequent classes surpassing the top frequent ones because it means that the classifier, even with the imbalances, is trying to maximize every class probability (LBBB and PR). A reason for it might be related to unique wave patterns that differ significantly from the rest, like in the PR.

Table 5.3: Model results with m1

	Precision	**Recall**	**F1-score**	**Support**	**AUC**
NSR	0.76	0.84	0.80	27799	0.87
STach	0.78	0.73	0.76	3215	0.97
AF	0.60	0.63	0.62	5894	0.93
LBBB	0.67	0.44	0.53	1614	0.93
PR	0.63	0.33	0.43	374	0.93

In Table 5.4, we can see the most challenging wave patterns to classify. Some of them are related to the low appearance of the dataset. However, as we could see in the top 5 scores, some others proved this argument wrong. We can see in this group several categories that need an experienced cardiologist to diagnose them correctly, for instance, the IRBBB and the prolonged PR interval.

Table 5.4: Model results with m1

	Precision	**Recall**	**F1-score**	**Support**	**AUC**
LPR	0.24	0.01	0.02	394	0.87
LQRSV	0.41	0.02	0.04	643	0.84
QAb	0.36	0.02	0.05	1196	0.80
IRBBB	0.46	0.03	0.05	2070	0.74
TInv	0.48	0.04	0.07	1350	0.80

Finally, the last effort to improve the model will be to use ensemble techniques. There are three main variations of them which are bagging, stacking, and boosting (Annex C.5). However, given the time taken to train the m1 models, we finally discarded the completion of this task. It was expected to improve results by approximately 1%. In the last pages of the document the reader can find detailed confussion matrices of each of the methods.

CHAPTER 6

Defining the Decision Support System

This chapter will take a drastic shift in the content as it is focused on the DSS developed. Firstly, in Section 6.1, we will describe the methodology followed to build the system effectively. Secondly, in Section 6.2.1 we will explain how our back-end server works with the different methods and processes that have been implemented. Finally, Section 6.2.2 will describe the modules used to build the front-end application and some snapshots of the main parts from the resulting system. If the reader is interested in installing the DSS module go to Annex C.8.

6.1 Components

The DSS system will be based on an API Representational State Transfer (REST) architecture. With it, a specific user can make requests to the back-end server via the HTTP protocol to perform CRUD operations (create, read, update and delete) over data. Which in terms of Hypertext Transfer Protocol (HTTP) queries translates to the GET, POST, PUT and DELETE methods. This is thought to be one of the most straightforward ways for the physician, as accessing this app can be done through the web browser.

With all this, the application will be stateless, meaning that the only thing it will do

is accept ECG data and translate it into a possible diagnosis. Thanks to the modular feature of the API REST, we could build additional modules leaving the project open for improvement. In the following section, we will deepen into the different sides of the API.

6.2 Architecture

The first step will be to define the API structure by developing a System Sequence Diagram (SSD), where we can see the events that the external actors generate and their order (Figure 6.1). The only action required by the user would be to send the ECG file. Then, the rest of the process would be straightforwardly handled by the back-end part. Below, it can seen a sequence diagram of the SSD system implemented:

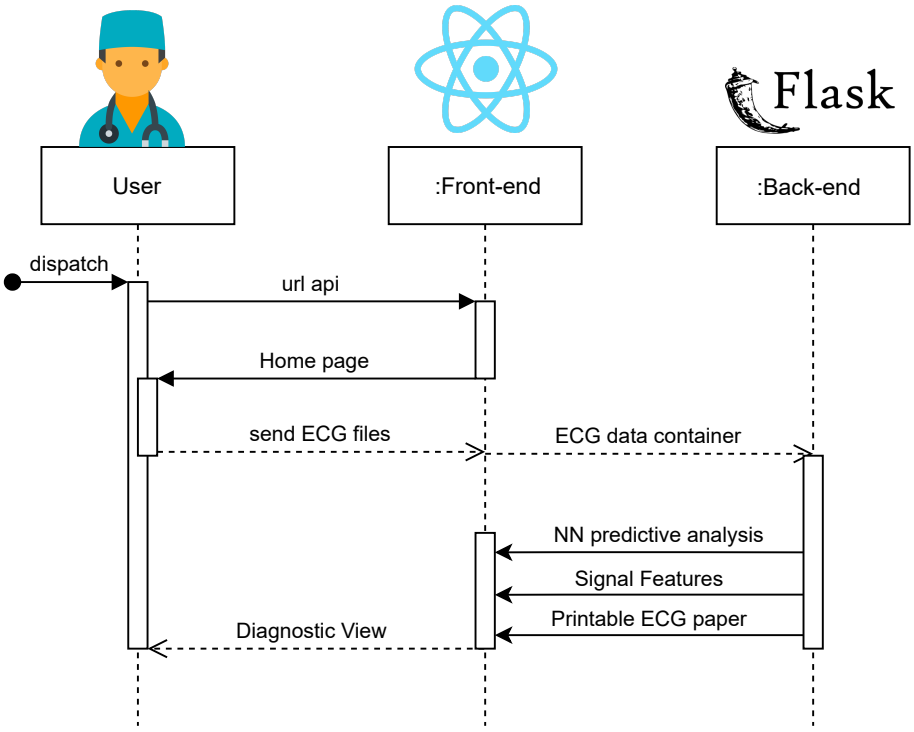


Figure 6.1: DSS Sequence Diagram

6.2.1 Back-end

We programmed the analytical methods (The model based on NNs, feature extractors, and the ECG plotter) in Python code. Therefore, we needed an environment able to act as a server that supported such language. We were between choosing Django or Flask, but we finally decided to go with Flask as it is well-documented.

Flask is a web framework that provides tools to manage HTTP requests from the user side (its other functionalities are irrelevant to this Master thesis). It is considered to be a micro framework that is deemed to be more secure. Why is that? Flask only works with the help of two dependencies which are Jinja2 and Werkzeug WSGI (Web Server Gateway Interface). As the chances of becoming obsolete are immensely reduced, its resilience to bugs and its lightweight structure makes it ideal for deploying in hospitals, famous for its strict security protocols.

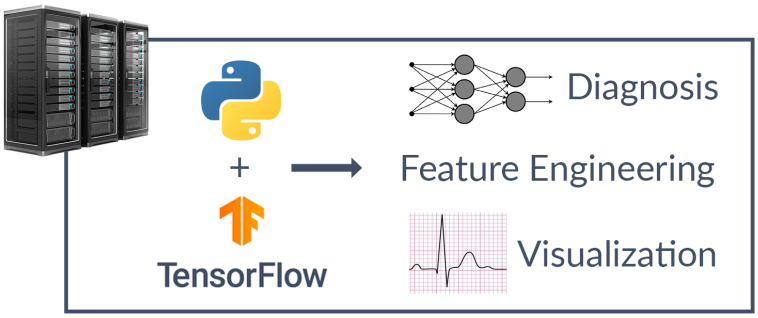


Figure 6.2: Back-end main functionalities

6.2.2 Front-end

Thanks to previous work, the construction of the front-end was very straightforward [22].The physician side was built using React, which is a Javascript library used to build User Interfaces (UIs). Two of its main advantages are that it uses a declarative language (easy to debug) and that its structure is based on components, keeping the state out of the Domain Object Model (DOM). Therefore, we can create encapsulated elements that handle their state and turn them into complex user interfaces, which we can upgrade with a bit of effort in posterior iterations.

We defined two main views in the UI: upload data (Figure 6.3) and diagnosis environment (Figure 6.4). The first one consists of an upload page where the physician must drag both the .hea and the .mat file. Then, when the back-end pipeline finishes, its state is updated, and we can see the diagnostic interface with the result from the NN, some features from the uploaded sample (like the beats per minute), and varied views of the ECGs.

- Upload Data

  In the top right corner of the page, we can see two icons. The first one displays information about the different disease categories that our app can classify (including their abbreviations). The other one is a button that manages the page's state, allowing the

user to switch between light and dark mode. The main body is made of a drag & drop container where the user can upload the required files for the analysis. Additionally, the rest of the page comprises HyperText Markup Language (HTML) code whose purpose is to guide the physician through the app to avoid possible errors. Finally, there is a 'Diagnose' button in the bottom right corner, which will be disabled until the two required files have been uploaded.

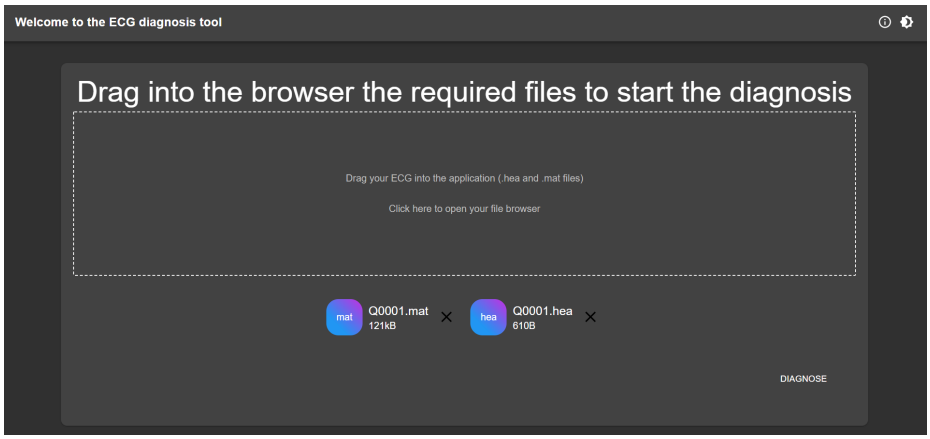


Figure 6.3: Snapshot of the upload page with two files submitted

- Diagnosis page

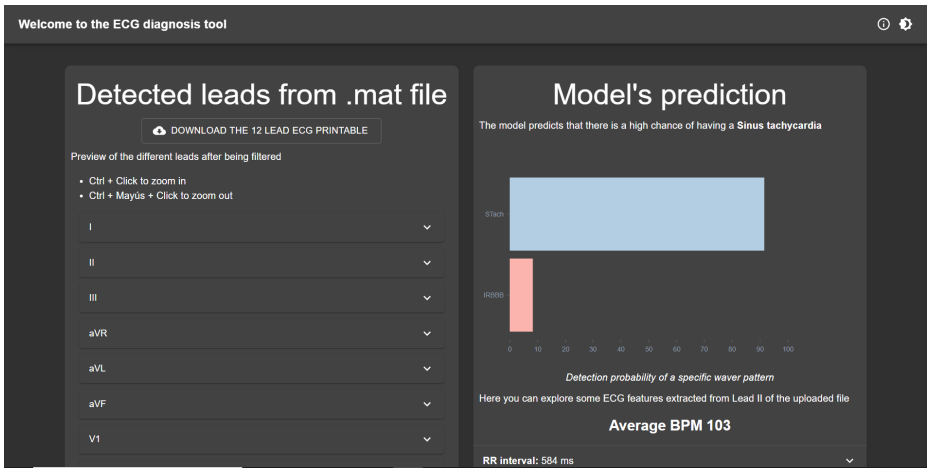


Figure 6.4: Snapshot of the resulting diagnosis page

Once the server has processed the data, it is programmed to send the response (in a JSON format) to the React app. All the information is then displayed according to the nature of the data in different modules. As we wanted to plot insightful graphs, we decided to use an interactive library called Nivo, which works based on D3.js and allows the developer to display complex visualizations at ease.

For example, on the left side, we will see 12 drop-down cells corresponding to each of the ECG samples fed into the NN so that the physicians themselves could inspect the different leads looking for anomalies. Additionally, within each derivation, the user could zoom in and out in order to look at the peaks more closely. We added a 'Download the 12 lead ECG printable' button to get the ECG signal printed in a millimetric grid where the physician could inspect the signal on the time axis (Figure 6.5).

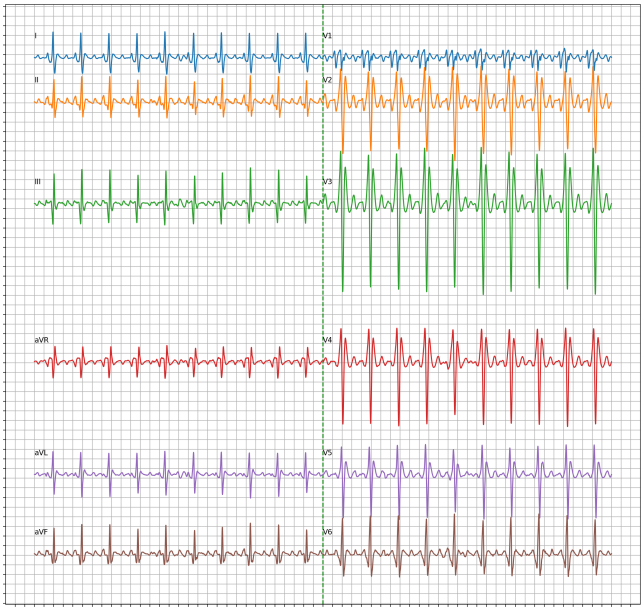


Figure 6.5: Downloaded ECG printable

Then, on the right side, we can see the prediction coming from the trained Neural Network, shown as a percentage. Thanks to Nivo, the user can hover over the different graphs to get additional information from the results (Figure 6.6).

Finally, a set of Accordion containers from MaterialUI will display the different features extracted from the ECG in the bottom right corner (RR distances, PR intervals, bpm...), all taken from lead 2 (II). The calculations have been done using the complete signal so that we could use the information in posterior phases of the project to train newer NNs.

As it can be seen in the picture the containers can be expanded to see detailed information of the results, in case there are some bugs in the calculations (Figure 6.7). It

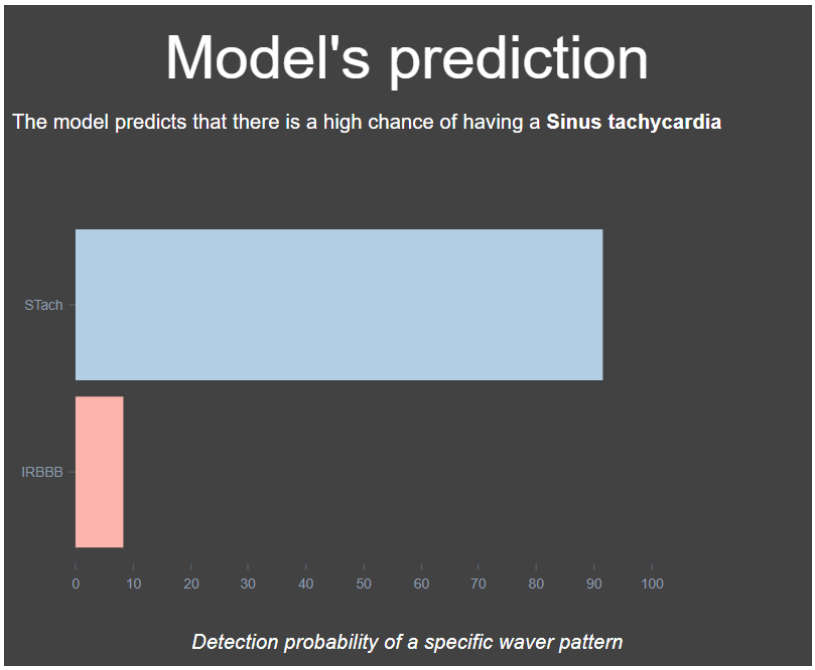


Figure 6.6: Snapshot of the prediction

can be noticed that some sets are disabled, which means that the implemented code has not been able to extract those features successfully.

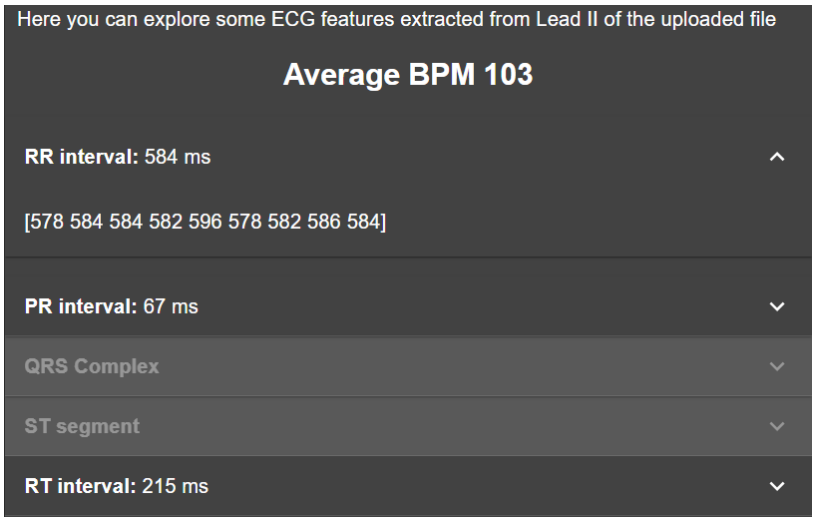


Figure 6.7: Snapshot of the extracted ECG features

In the final Chapter 7, we will discuss possible improvements of the current DSS which will be possible thanks to the modular features that React and Flask have to offer. There is more information about the ECG printable and features in Annex C.7.

CHAPTER 7

Conclusions and future work

In this project, we have tested the computers' ability to classify raw ECG samples. Thanks to Deep Learning techniques and a decent computer capacity, we have obtained satisfying results compared to the other team competitors (given the available computational power). In Section 7.1, we will draw the different conclusions at which we have arrived. Finally, in Section 7.2, we briefly propose possible future lines of development that can arise from this work.

7.1 Conclusions

The first characteristic noted when working with ECGs is the complexity between concepts present in a single signal. In this particular project, we are dealing with diseases that have associated a wide distribution of wave patterns. Even for a formed professional, the task of detecting them could be challenging. Therefore, solving this with a classification model that can depict all these relations in a matter of seconds can represent an impactful change in medicine.

Another important thing learned from this project is that before going straight to work with the datasets, we have to explore them in depth, which means not only reading the

description available in the site but also exploring the different classes making sure there are no mistakes on them.

It is impossible to go through all of it one by one. However, the procedure taken in this project allowed us to confirm that the data was truthful and had few mistakes on it. In addition, we designed a series of tests for checking the functionality of the methods developed. This helped us spot several unnoticed errors until the very end as we performed multiple iterations through the whole ML process.

The final score achieved was 55% (F-score), using the m1 dataset with the architecture described in Chapter 5. If we suppose that the average number of positive labels per sample is two, this would mean that if we use a random classifier over our data, the probability that it correctly classifies the given input would be $\frac{1}{27}\frac{1}{27}$. Therefore, if we compare it with our NN results over the test set, we would be surpassing the random classifier by a factor of 400. Lastly, checking other articles scores from the winning teams of the competition confirmed that our approach is in the right direction.

The Decision Support tool built was also an excellent element to facilitate access to the prediction model and visualization of results after analyzing a concrete ECG signal for the practitioner. The physician only has to connect to the server where the app is installed via a web browser like Firefox. The server handles all the complex procedures so that the user can only focus on the diagnosis task itself.

All the areas of the project were initially unknown. Therefore, apart from coding, we researched many different subjects like Deep Learning, the Flask environment, and React. In the end, this was a time-consuming process that paid off. Additionally, we spent a significant amount of time automatizing the notebooks and documenting all the code to lessen the learning curve and facilitate progress in further works.

7.2 Future Work

The prospect of the biomedical field is up-and-coming. In particular, the final objective of our project would be to implement this software into health centers. However, to achieve such a thing, we need to design a method to transform the ECGs registered at a specific Hospital into usable inputs. This is not a trivial task given several constraints of the recording equipments.

Another option would be to develop specific classifiers that focus on complex wave patterns like T wave anomalies, which are some of the most difficult ones to locate.

Finally, we could focus the efforts on the DSS by adding additional visualization tools and functionalities that will serve the physicians to save time and improve the diagnosis. In this part, we thought about integrating the system with the Electronic Health Records (EHR), as well as the development of a database where the cardiologist can keep track of the different ECGs performed on the patient. With time the ECG drawings start to decay; therefore, having them digitalized could significantly improve the existing system.

Additionally, the samples from the patients could come from different distributions as there are multiple types of recordings (Holter, stress, resting, ambulatory). The tool could have sets of classifiers installed depending on the situation, leading to better decisions. However, to build them, we will require a significant amount of recordings.

To conclude, there is no doubt that DL will dramatically change the way cardiologists work. However, we are still far from the human performance level in many different wave anomalies, which is why physicians are sometimes doubtful about the results given by these systems.

APPENDIX A

Impact of this Project

The different techniques applied in this project play an essential role in the construction of DSS. There are visible interests placed on the creation of algorithms that can classify ECGs in a fast and organised way. Nowadays, workers have to analyse this data manually; with the development of these systems, the time spent on those matters could be drastically releasing the practitioners to spend their time doing other meaningful tasks. Distinctly, in the biomedical world, this data managing tools can help in many different fields, for instance, imaging, genetics, research and diagnosis. Robust decision systems can improve immensely the value provided to physicians and as final beneficiaries, to patients. However, errors in those applications could lead to misinformation and cause grave consequences. In this appendix, we study those different impacts in terms of society A.1, economy A.2 and environment A.3. Lastly, we discuss the different ethical implications in section A.4.

A.1 Social impact

The introduction of DSS into a company has severe implications concerning their workers. The first thing to note is that every change in the daily routine of a physician supposes an

extra amount of hours that the worker will have to spend learning how to use the program. Not all the operators accept those changes positively because it will suppose a variation in their routine that has not been altered for years. Therefore, the implementation can be a slow process that produces extra stress in every practitioner but more intensely in those reluctant to such changes.

The inability to learn new technologies or the amount of time that requires apprehending them also plays an essential role in such matters. For example, in hospitals, the EHR which represented a drastic and extremely positive change in how hospitals worked, was first developed in 1965. Still, it was not until the 2000s when they started implementing them, and even nowadays, some doctors keep using paper as the tool for storing patient's information.

Data is an essential requisite in these projects, and taking into account that we are talking about medical information, the vast majority of it comes from patients. A physician needs to recover tons of personal information from subjects to help them overcome their diseases. The GDPR regulates the way this information is stored, helping to keep this knowledge unavailable to external sources. If we want to work with this information, it implies having it pseudo-anonymised so that no one knows from which person is coming. A leak on this data can have several implications to the patient affected and also several punishments to the institution that facilitates it.

A.2 Economic impact

Developing these systems can be an expensive task. Firstly, we have to make sure that the data quality and quantity are sufficient for the duty, which may require hiring an expert in healthcare data. Secondly, we need software engineers and data scientists to develop the operating program. Thirdly, we expect someone to teach professionals about how to use our system, to reduce costs, it could be a developers' extra task. Finally, those practitioners need time to learn the new software, taking out hours of their daily schedule could lead to a deterioration of their previous assignments. A plausible solution to this drawback could be augmenting their salary.

If we are talking about the benefits, the first clear one is that an increase in the efficiency means a consequent improvement in the capacity of the company which can be reflected in attending more patients, developing clinical notes faster and facilitating work to researchers. Increasing productivity directly affects earnings. However, for a project to be viable, those earnings should be significantly higher than the costs described in the preceding paragraph.

A.3 Environmental impact

The first environmental consequence that our systems may produce is an increase in electricity consumption for having a computer or a server operative. This consumption induces an increase in the demand for energy sources, both non-renewable and renewable, affecting the environment negatively.

Either if we use computers or severs, those machines have built-in obsolescence which means that at the time they stop being useful they need to be dumped. Several components of PC's such as batteries, the power source, plastic covers, cables and internal circuits have contaminant products (selenium, cadmium, chrome, cobalt and mercury). Recycling them should be a widespread practice to reduce the impact that these devices may cause to nature.

Finally, having an excellent DSS, can reduce the number of tests that a patient has to take before getting a diagnosis which can be harmful to the patient (ECGs, chemotherapy, X-rays, surgeries). Also, if we look at this problem through the hospital perspective, reducing the number of tests implies a reduction in the expenditure of sanitary materials, reducing costs and contamination.

A.4 Ethical impact

In the end, the objective of ML is outperforming workers thanks to computational skills. This first reason makes us think that one of the first ethical implications while developing machine learning systems is that we will be removing someone's job. However, DSS, as its name says, is designed to support the professionals as a helping tool, not as a replacement. Also, new technologies create new jobs giving workers the chance to adapt to changes by acquiring new skills.

Another ethical implication can be related to the medical GDPR. Collecting data from patients requires a sophisticated system that ensures that data is protected and encrypted so that no one else has access to it. Extracting this information for ML is a difficult task with many requirements. In hospitals, for example, data is stored in servers inside the building so that any outsiders have access to them. We have to be physically in the hospital to work with the information and also extract it anonymised in the best possible way.

Finally, as we are developing a classifier that may support decision systems in medicine, there are ethical implications related to the production of mistaken predictions. These errors

can have enormous consequences on patients health; therefore, to reduce this impact, we need to make sure that those systems are reliable. However, it is essential to remember that those algorithms are prepared to help professionals as tools. Depending on the situation, it is always the final choice of the practitioner to follow or not those decisions.

APPENDIX B

Cost of the system

If we want to sell the software developed in this project, first we need to improve the results. Doing so could be achievable by acquiring powerful GPUs as well as a more significant amount of data. In this chapter, we will discuss the different costs associated with the development of a DSS software that will use the models designed in this work. Section B.1 talks about the price of the physical infrastructure necessary to compile our software. Then, section B.2 explains the different professionals that will be needed for the creation of the software. Finally, sections B.3 and B.4 talks about the licenses and taxes required for the commercialization of our product.

B.1 Physical resources

The resources needed to ensure the correct functioning of the proposed software are strongly related to the amount of data used. Our dataset supposes a storage capacity of 1-2GB. Hence, a computer or server with a powerful GPU will be necessary for the fulfilment of this task. However, as we said before, if we want to commercialise a product of such kind we will need much more data in order to make a robust and truthful model. With the dataset used, the estimated requirements are:

- **Hard disk:** 1TB
- **RAM:** 16GB (2x8GB)
- **CPU:** Intel i7-11700K 3.6 GHz
- **GPU:** GeForce RTX 3070 8GB GDDR6

The cost of a machine with these characteristics could be around 1200€, **without considering the GPU** and other components. These features are only symbolic since, as we have said before, if we want to commercialise it, we will need more data and larger computing capacity to develop such software.

B.2 Human resources

To develop an application of this kind, we estimate that hiring a data scientist specialised in healthcare data (biomedical engineer) would be enough for its completion.

We decided to divide the project into two parts within a year. In the first half, the data scientist will be in charge of extracting meaningful features from the data as well as training the NNs. Finally, in the half left, the engineer will develop an application that uses the learned model and test it in real case scenarios. As a final result, we expect the commercialisation of the software to biomedical companies and hospitals. If we suppose that the biomedical engineer earns 20€/hour, the total costs related to the workforce of the project will be around 8000€.

B.3 Licenses

We are going to divide the licenses of the project into three sections:

1. The first one is related to the obtention of medical datasets. There are many different sources of free data like PubMed, data.gov, medicare, etc. However, many others require a subscription or a license to be accessible.
2. The second part, which is the one related to this project, does not require license since all the software used is of free access (Python, Google Colab and VS Code). PyCharm can also be an interesting tool to work with Python code since it has an excellent debugger. The drawback is that the professional version that includes compatibility with Jupyter Notebook requires a license.
3. Finally, we did not include any particular program in the application's construction since React and Flask are free access software.

B.4 Taxes

If we consider selling the product as downloadable software, we will have to apply 21% taxes supposing that our company is established in Spain. Additionally, the acquisition of an application represents a service which means that selling it to foreign countries is not considered exportation. Besides the seller, we have to consider the residence of the vendor, if it is from the European Union (EU) the tax remains the same. However, if it is from a foreign country selling this software will have additional fees.

Workforce	Hours	€/hour	Total
Biomedical Engineer	405	20,00 €	8.100,00 €

Material Costs	Price	Monthly Usage	Amortization	Total
Personal computer	2.519,28 €	12 months	4 years	629,82 €
Google Colab Pro	59,94 €	NaN	NaN	59,94 €

Direct and Indirect Costs	Percentage	Total
General expenses	15%	1.318,46 €
Industrial benefit	6%	527,39 €

Subtotal budget	10.635,61 €
VAT (21%)	2.233,48 €
Total Budget	**12.869,09 €**

Figure B.1: Cost of building the Deep Learning Model & Application

APPENDIX C

Libraries and others

This chapter will explain the different software applications used in the development of this work. Additionally, we will describe several concepts and results that helped reach the final solution and were not shown on the main corpus.

C.1 Tools and libraries

Google Colaboratory Previously, scientific articles lacked reproducibility in terms of programming. Publications were full of prose, without specific tools to show the code that supported those conclusions. From this need raises the Google project and with it the development of Google Colaboratory. This tool covers both the prose and the code section, being able to be specific and facilitate reproducibility.

The code is organised in cells that can be individually modified and compiled. The result of each cell appears below as a part of the document, improving its readiness. Additionally, Google Colab not only covers markdown code but also ways for representing mathematical equations and interactive graphs. Finally, markdown cells and the code ones can store comments which makes a notebook rich in specifications about either the code or the purpose of the writing.

These notebooks support multiple programming languages, which communicate using back-end programs that have a standard protocol. The first programming code used with this notebook was from a Python kernel. The access to the software can be done through a web browser; however, nowadays, some applications like Visual Studio Code and PyCharm support it. The documents, stored with a JavaScript Object Notation (JSON) format and an extension 'ipynb' can be published via Github recovering the programming environment and facilitating the execution to other researchers [27].

NumPy In Python, NumPy arrays are the way of representing numerical data. Using them facilitates the computation of extensive datasets by optimising the times of 'for' loops significantly. The three fundamental techniques that achieve this are the vectorising calculations, the avoidance of copying data in memory and the minimisation of operation counts. NumPy matrices also called ndarrays, uses the following attributes [28]:

- Data pointer: the memory address of the first byte in the array.
- Data type description: defines the nature of the element stored in the array.
- Shape: stores the dimension of the array in the form of coordinates.
- Stride: the number of bytes skipped to proceed to the next element.
- Flags: define if we are allowed to modify the array or not.

This efficiency, adopted by the Pandas library allows computing effortlessly operations in large datasets. Usually 'for' loops represent the bootle neck in the computation time, managing correctly with NumPy arrays avoids this problem.

Pandas It is a Python library used in many different fields, preferably on structured sets. It has all the means to perform an exhaustive analysis on data by performing manipulations and queries. It also has the tools to allocate the dataframes by converting .csv files into the notebook and vice-versa. To import the dataset, we have to create a Dataframe object on which we will initialise our required variables. Once it is uploaded, the data can be reshaped and transformed for the required task (ML in our case) [29].

In our particular situation, we developed the methods to extract the signal from the Challenge's files. In this case, the library was practical, not only to store the information efficiently but also for visualization and testing purposes. Thanks to ndarrays, we were able to transform the datasets without complications allocating the new dataframes in a '.pkl' format. Afterward, thanks to NumPy and Scikit-learn, we were able to adjust the data so that it could feed into our NNs models.

The operations are not very efficient if we are using Dataframes or Series. Therefore, the best way to avoid this inconvenient is to work directly with the ndarrays by extracting the data with the attribute 'values'. Statistical functions like 'mean' and 'std' have been overridden to avoid missing values which is a common feature of many datasets.

Scikit-learn It is a Python module that implements a varied range of ML algorithms (supervised and unsupervised). It focuses on bringing these methods to a high-level language so that it is easily understandable by non-specialists; putting a strong emphasis on clear and complete documentation.

Its code provides a reliable implementation with a consistent set of parameters and variables which reduces the learning curve of the library. It also shows strict adherence to the Python coding guidelines and the NumPy style documentation. The main components of the library are estimators, transformers, the cross-validation iterator and a varied range of scoring algorithms. All of it made this an ideal tool not only for research but also as building blocks for a diverse range of approaches like medical imaging and signal analysis [30].

Keras It is an API designed to lessen the cognitive load that carries using TensorFlow. Thanks to it, data scientists can build complex DL models at a great rhythm, empowering them to try more ideas than their competition, and faster. For example, the top five teams in Kaggle, a website with hundreds of ML challenges, use Keras as their main API. It is not only meant for research but also to deploy its models into a wide range of applications:

> "Keras has the low-level flexibility to implement arbitrary research ideas while offering optional high-level convenience features to speed up experimentation cycles."[31]

C.2 Cross Validation (CV)

In this project we could not use CV due to the GPU/TPU limitations from Colab, as a solution, we divided the data into train/validation/test, which is the traditional approach. The objective of the CV is to avoid predicting something that was previously learned by the computer. Training a model with the gross data is a common mistake on someone who starts studying ML. There are many different ways of overfitting, some of them, more noticeable than others. The one described above is the first typical example, computing the

score on this overfitted model, will show an accuracy close to 100%.

The initial idea to solve this problem is to divide the data into what we call a test and a training split. Doing so allows the computer to prove its model with unseen data. Unfortunately, this method lacks generalisation performance because the only representation of the model is based in a specific split. CV allow us to generalise our models and make them more representative of our data.

Another overfitting problem occurs if we use GridSearch on the training data. This overfitting is harder to notice because it does not enhance the results drastically. Optimising the training split for a given test set will improve the results of the classifier, giving a false sensation of improvement. A solution to this kind of overfitting is to create a validation set to evaluate the GridSearch. However, this procedure implies an extra partitioning of the data which worsens the variety in the training and testing. Thanks to CV, this additional set is no longer needed.

There are many different types of CV iterators, some of the most common ones are:

- **CV for independent and ideantically distributed(i.i.d.) data**
  This approach assumes that all samples have been extracted from the same generative model. In real life, this approach is not very robust because data can be time dependant or modelled by a generative model with a grouped structure. Some of the algorithms used for this group are K-fold, Leave One Out (LOO) and Leave P Out (LPO).
- **CV with stratification**
  If the distribution of tags is unbalanced, this iterator creates each fold based on percentages assuring that the relative frequency between labels remains constant. Stratified K-fold is the most renowned method.
- **CV for grouped data**
  If the distribution of tags is unbalanced, this iterator creates each fold based on percentages assuring that the relative frequency between labels remains constant. Given the multilabeled characteristics of the dataset this technique could not be performed.
- **Time series split**
  This particular iterator can be used in a data model that has a constant flow of data. Therefore, what it does is putting the new inferences in the test set and trains the model with the previous samples.

C.3 Troubleshooting with Google Colaboratory

In this section we will try to explain the possible problems that can arise using Google Colaboratory and how to solve them. First of all we will enumerate the three main circumstances that can happen, and then we will propose some workarounds:

1. **RAM usage:** it is not much we can do about it. The RAM limit, available using the default Colab environment, is 12.7 GB. If the code exceeds this limit, try deleting variables iteratively or divide the process into several batches. Another option could be upgrading to Colab Pro, which grants the user with 25 GB. However, by now, in Spain, this is not a viable option because Google does not offer this package. This service is available in Germany, Brasil, Canada, United States, France, India, Japan, United Kingdom, and Thailand (Using a VPN has not been tested).

2. **Long runtime disconnections:** occur when the user have been running Colab for several hours. Google has designed a system to log the user out for inactivity (this does not count executing a cell but interacting with the notebook). Previously, the solution was to insert a JS code in the console to maintain the session opened automatically. However, after discovering this, Google launched a random Captcha to stop the execution in case of not being completed. The best solution to avoid problems is to define the model training with a Keras callback called ModelCheckpoint [32] and save the model in the Google Drive folders. It is necessary to make sure to save it with the parameter 'save_weights_only' turned false. This will keep the optimizer, allowing to rerun the model from where it was left.

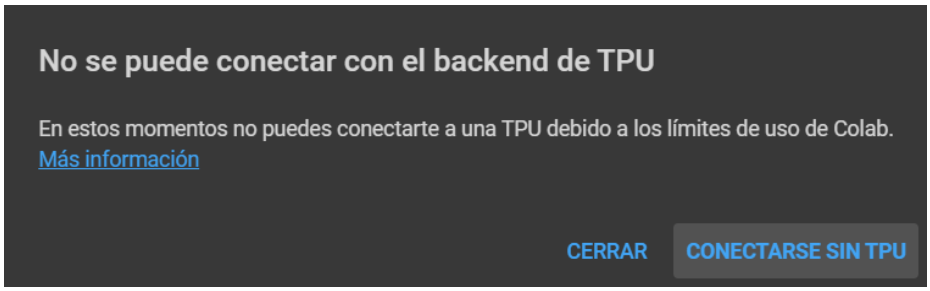


Figure C.1: Snapshot of the moment when Colabs force quits an account

3. **Google force quit:** The last problem encountered is that Google keeps track of computational consumption. Therefore, if the user has been using his servers for an intensive session, it is most likely that Google forces him out, displaying the message that there are no GPUs/TPUs available at the moment. The only solution known for

the moment is to create a new Google account and configure it so that the code could be reproducible, which means making the desired folders in Drive. If this is not the case, the only thing the user can do is wait until it is available again.

C.4 Inverted T wave patterns

The multiple situations that could lead to the apparition of inverted T waves are very extense. Some could represent a present threat, and others might be harmless, so we decided to dedicate a whole section to this wave pattern.

- **Paediatric T waves:** It is a common finding in the right precordial leads (V1 to V3) of children where the ventricular forces are dominant. These waves may persist into adulthood. In what refers to its morphology, they tend to be asymmetric and shallow <3mm.
- **Myocardial ischaemia:** They tend to appear in contiguous leads over the area of infarction (whether it is inferior lateral or anterior). Their relationship leads-location can be seen in the table 1.1 from Chapter 1. Dynamic T wave inversions are seen in acute myocardial ischemia and fixed ones following infarctions.
- **Bundle Branch Blocks:** RBBB produces inverted waves in V1, V2, and V3, and in the LBBB in I, aVL, V5 and V6.
- **Ventricular Hypertrophy:** The left one has similar characteristics to the inverted wave in the LBBB. Hence, they can be easily misinterpreted.
- **Pulmonary embolism:** It can be reflected in the right precordial and inferior leads, corresponding to V1-V3 and II, III, and avF.
- **Hypertrophic cardiomyopathy:** Deep inverted T waves have a presence in all the precordial leads from V1 to V6.
- **Raised Intracranial pressure:** A rise in the intracranial pressure like a subarachnoid hemorrhage can produce waves with abnormal morphology (long waves).

C.5 Ensembles

Ensemble models are in charge of combining the different outputs from a varied set of classifiers so that the overall performance could be improved. There are two main categories

of ensembling methods, averaging and boosting. The first one averages the predictions of a set of classifiers separately and combine the results in a parallel way; the second builds a sequence of classifiers that learn by correcting the bias from the prior in an iterative process.

C.5.1 Averaging ensembles

Bagging: Uses a set of N classifiers that train by extracting subsamples from the training set with replacement, doing so, ensures diversity and robustness in the decisions. The typical algorithms used in bagging are linear, for instance, SVM and logistic regression. Depending on the sampling, there are four possible variations: the pasting, the random subspaces and the random patches. In the case of large datasets, bagging is not the best because it is only optimal for small datasets, and as an alternative, pasting works better. In the end, a majority voting decides the best prediction from all the trained classifiers.

Stacking: This renowned ensemble technique improves single classifiers and combines them by reducing bias and over-fitting. These first classifiers, also called layer 1, learn by using a part of the training set and the rest for predicting. Then, it combines these resulting predictors via a meta-classifier that uses the results as new inputs for a new learning process.

C.5.2 Sequential ensembles

Boosting: Iteratively trains a set of 'N' weak classifiers, lowering the training error on each step. In each cycle, the mistaken samples from the previous phases move on to the next classifier to learn better from the most conflictive features. Adaboost is one of the most used algorithms; its working is similar to the original boosting method but differs from it in the use of weights for training and decision-making. On each training phase, the correct classified samples do not represent a big problem to the model, as a consequence, the algorithm focuses on learning from the mistaken inputs resulting in a reduction on the training error. It also repeats an iteration until such error rate gets lower than a 50% threshold. Finally, to make predictions, boosting methods perform a weighted average of the N estimates.

Ensembling is the last resort in machine learning that comes up when there is nothing left to try. All the previous steps involved, such as cleaning the data, feature extraction and multiple classifiers trials, are vital to ensure a successful ensemble. The predicting ability of a model is entirely dependant of the features; therefore, having extracted a varied set of them is fundamental to exploit the data successfully.

C.6 Complete summary of results

C.6.1 Computation of the AUC and the ROC curves

ROC curves are an excellent tool to represent the performance of classification models. They have a 2-D graphic representation where the X-axis corresponds to the False Positive Rate (FPR) and the Y-axis to the True Positive Rate (TPR).

$$Sensitivity = TPR = \frac{TP}{FP + FN} \tag{C.1}$$

$$1 - Specificity = FPR = \frac{FP}{FP + TN} \tag{C.2}$$

The plotted curves show the effect on varying a threshold that goes from 100% to 0% FPR. If the model is perfect, which means that it reaches an FPR of 0% with a TPR of a 100%, we will see a squared function that starts at the origin. However, if we cannot discern between the True Positives and False Positives at all, the plot will be a linear function of $y = x$.

To compare different ROC curves, the standard thing to do is computing the AUC, which means calculating the area under the curve from 0 to 1. The area of a square function will represent a perfect AUC with value one and in the worst-case scenario, the result will be the half of it (0.5). Therefore, calculating the AUC is interesting for choosing which categorisation method is better for a given task by measuring which one has a higher value. In a multilabel/multiclass classification, ROC curves are computed based on the number of labels being tested in the dataset, in our case 27.

Firstly, we will show a graph that summarises the AUC scores from the ROC curves of the top-performing models. Here we can see that as the results showed in Section 5.3, the m1 model outperforms the m2 ones. However, by looking at the values independently, we can see that the m2 model performs better for some classes, which is an important point that proves that ensemble techniques would likely improve the final model.

With this graph, we were also able to prove that the m2_512, an improved version of m2_256, produces better results. Given GPU/RAM problems, we were not able to try out m2_1024 and m2_2048. However, given the pattern, if we expected the averaged AUC to increase proportionately like from m2_256 to m2_512, the estimated improvement would be:

To conclude this section, we collected the ROC curves of the m1 top and bottom classes

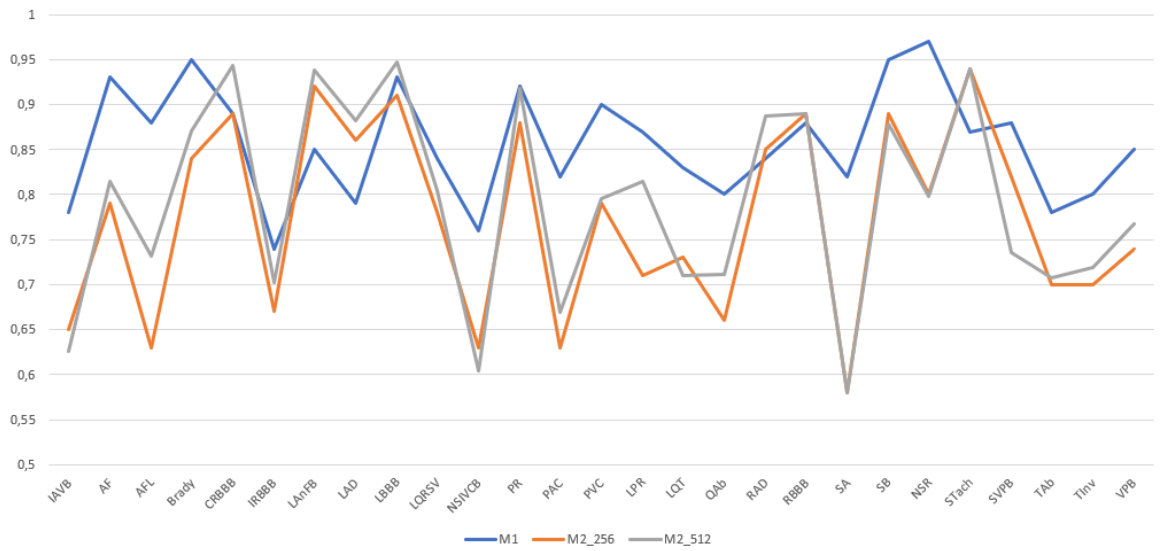


Figure C.2: AUC comparison of the different models

	m1	**m2_256**	**m2_512**	**m2_1024**	**m2_2046**
AUC (%)	**0.93**	0.87	0.89	0.91	**0.93**
Drive Storage (GB)	1.22	0.89	1.74	3.48	6.96

Table C.1: AUC predictive analysis

(Figure C.3 & Figure C.4) so that the reader could gain some intuition into how ROC and AUC are related. The more flattened the worst performance it has.

C.6.2 Execution times

It is also essential to check which model was faster in terms of execution times. The quicker it converges to a solution, the faster it would be for the user to try different architectures and hyperparameters. Thanks to the compression factor of 12 offered by the m2 models, we achieve an incredible reduction of learning time per epoch. Thanks to this, we were able to test the same architecture of the m1 model faster, to debug parts, and check different aspects of the network. It is required to know that the times vary a lot as Google is continually limiting and monitoring the GPU usage. Therefore the values might not be as consistent as we wanted them to be (m2_512 has two times more information, therefore, it was logical that it should have run two times slower, however this was not the case).

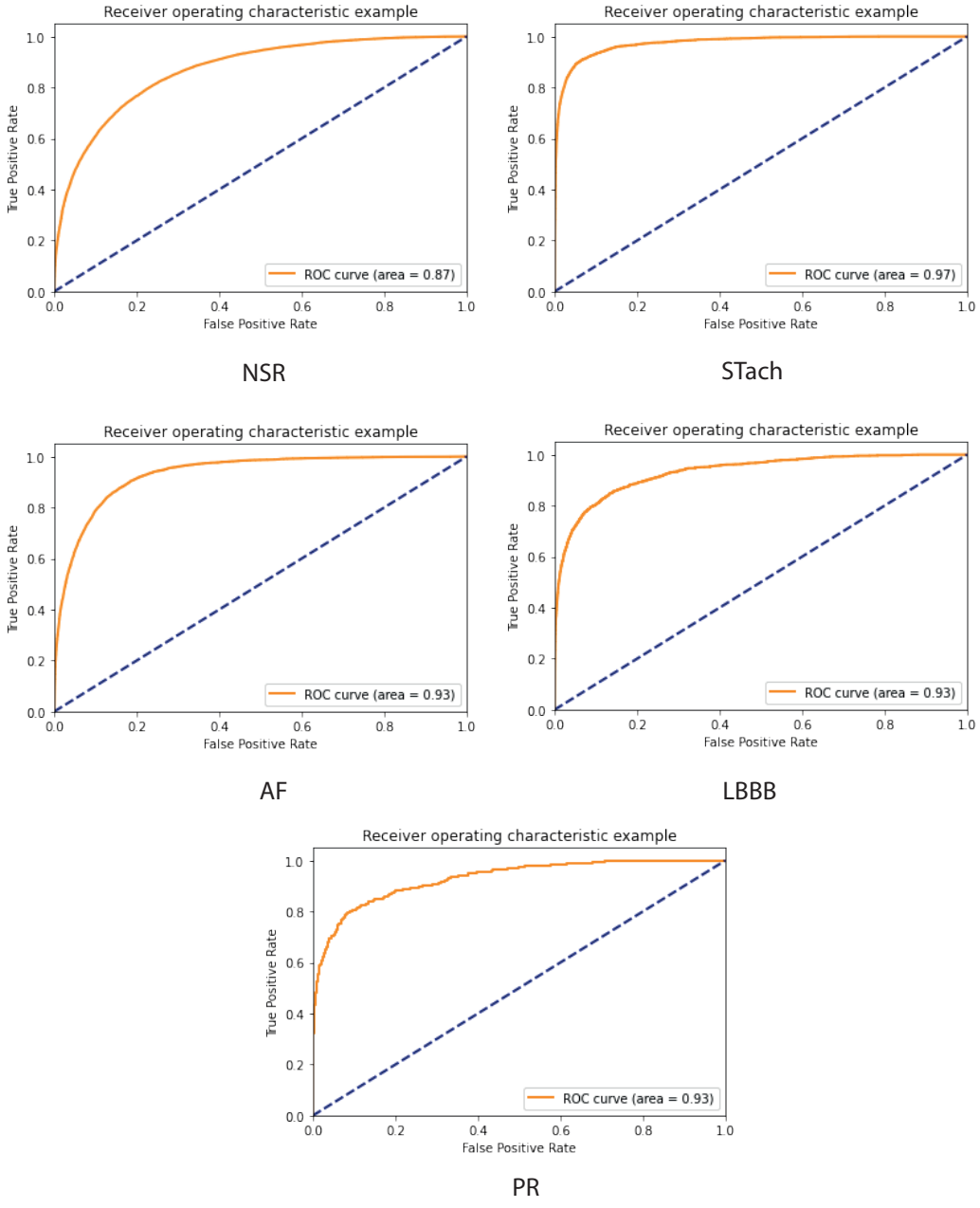


Figure C.3: ROC curves from the m1 top graphs

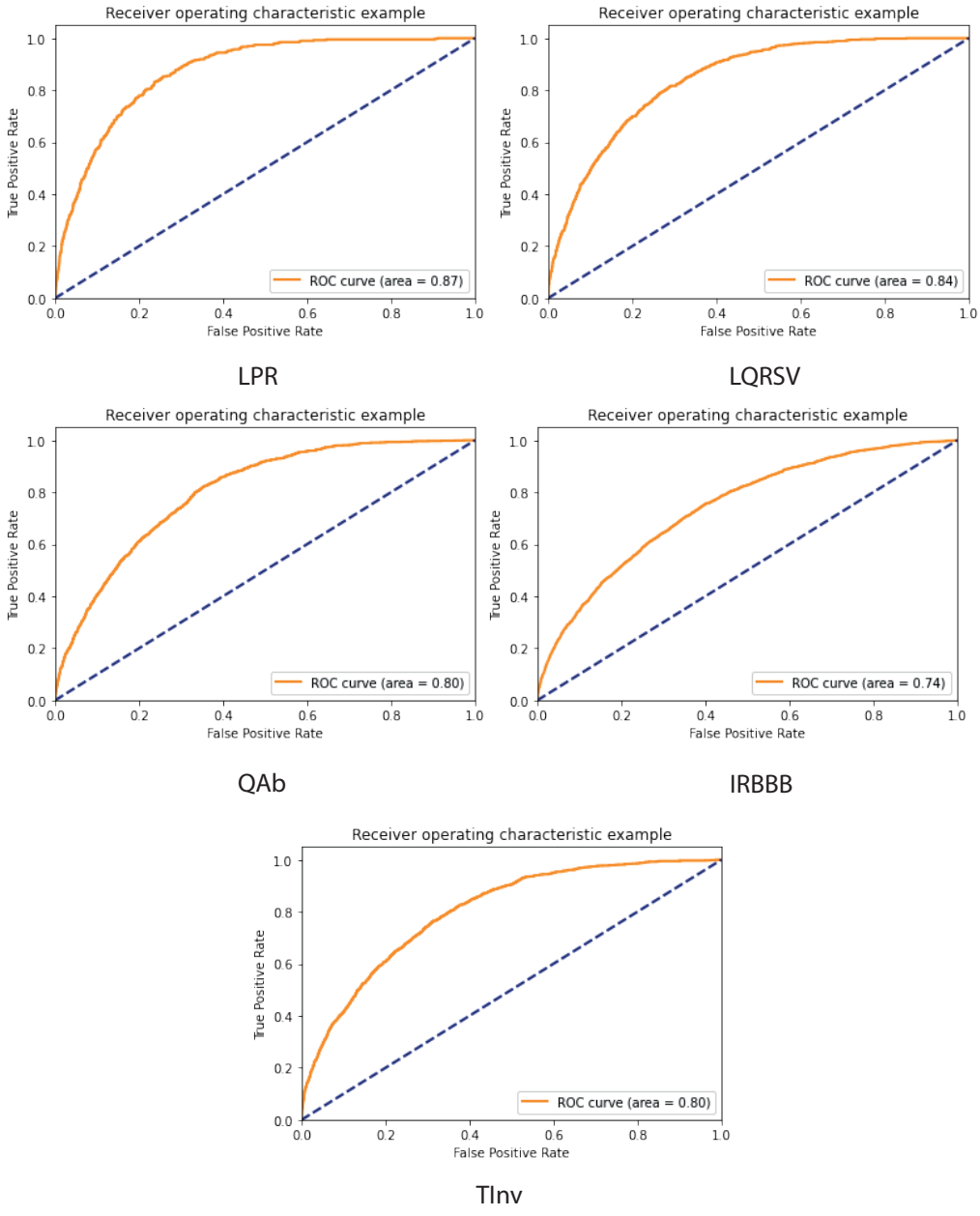


Figure C.4: ROC curves from the m1 bottom graphs

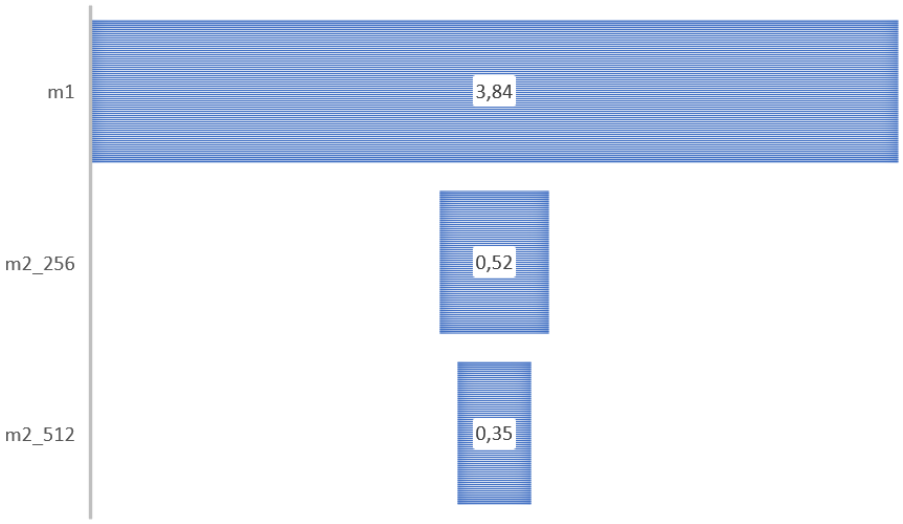


Figure C.5: Averaged execution time per epoch of the different models in minutes

C.6.3 Summary table and hyperparameter testing

This last part of the document will wrap up different models and some hyperparameters that we tuned throughout the project. We wanted to perform more intensive Grid Searches to fine-tune our best models; however, this task could not be fully completed given time constraints.

The main hyperparameters that have been tuned are: the preprocessing parameters (type of filter, number of samples...), the Learning Rate, the batch size, the dropout rate, the batch normalization and the final threshold.

Table C.2: Summary of the models tried thoughout the project

Models				
Conv1D	Conv2D	LSTM	Conv1D+LSTM	Conv2D +LSTM
BiLSTM	Conv2D + BiLSTM	Conv2D + BiLSTM	SE_ResNet	SE_ResNet + BiLSTM

C.7 ECG printable & Features Extraction

The ECG printable is generated from a cropped normalized signal. We carried out the normalization because the values of the precordial leads have high variance. We used the

biosppy library [33] to remove noise and correct the basal line so that the signal representations have a strong correlation to those printed in real case scenarios. The code is prepared to build the grids based on the signal frequency; based on those lines, the squared grids are created.

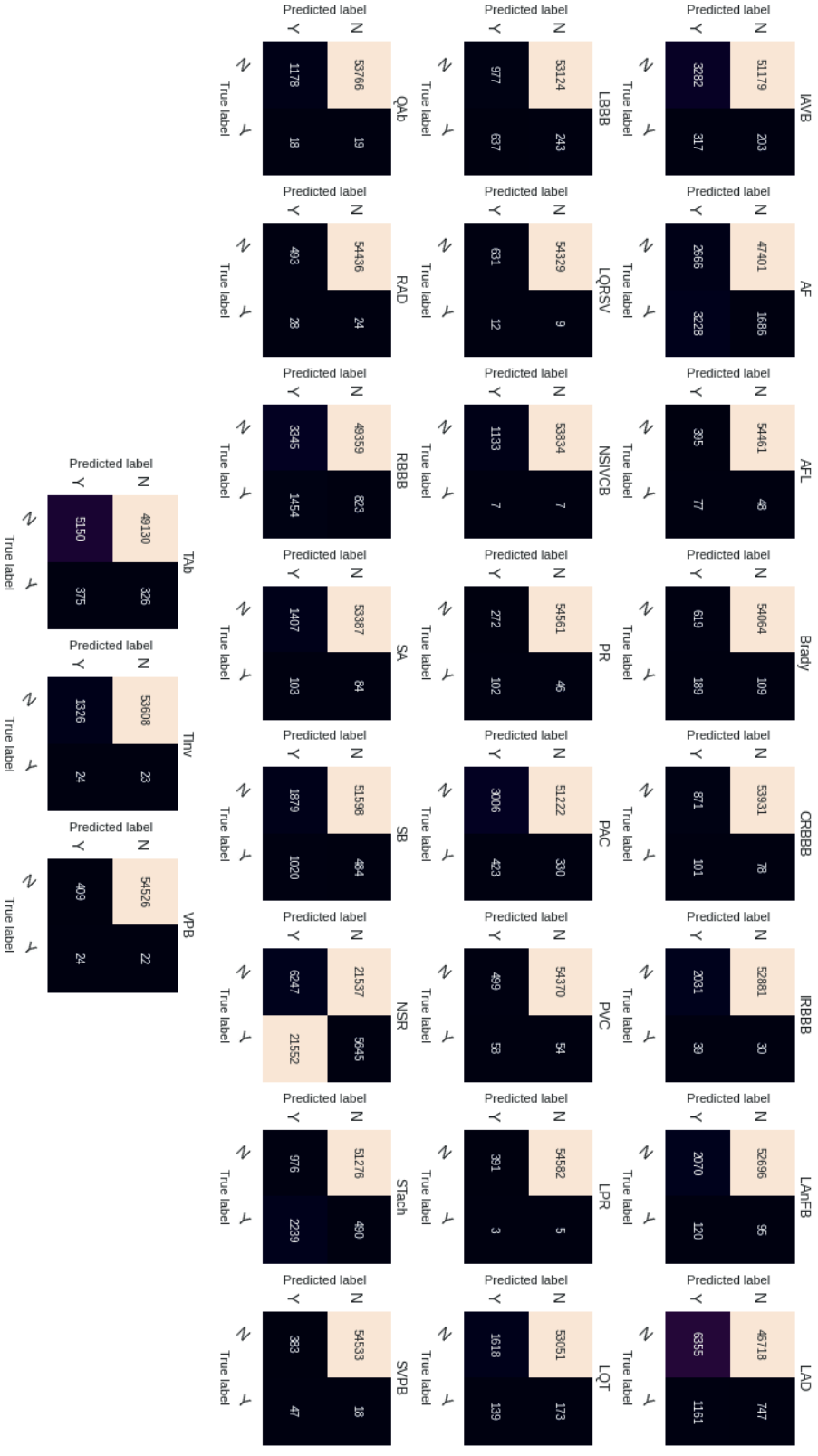
The ECG features are extracted using a library called *neurokit2* [34]. We can locate the position (in sample units) of the different peaks of the ECG signal with it. The algorithm works by detecting the R peaks and based on those, the remaining waves are estimated. Once we have all the points, we designed several methods to transform the signal to the time domain and prepare the information to extract it as a .json file allowing the front-end to fetch it when needed.

C.8 Installation guide

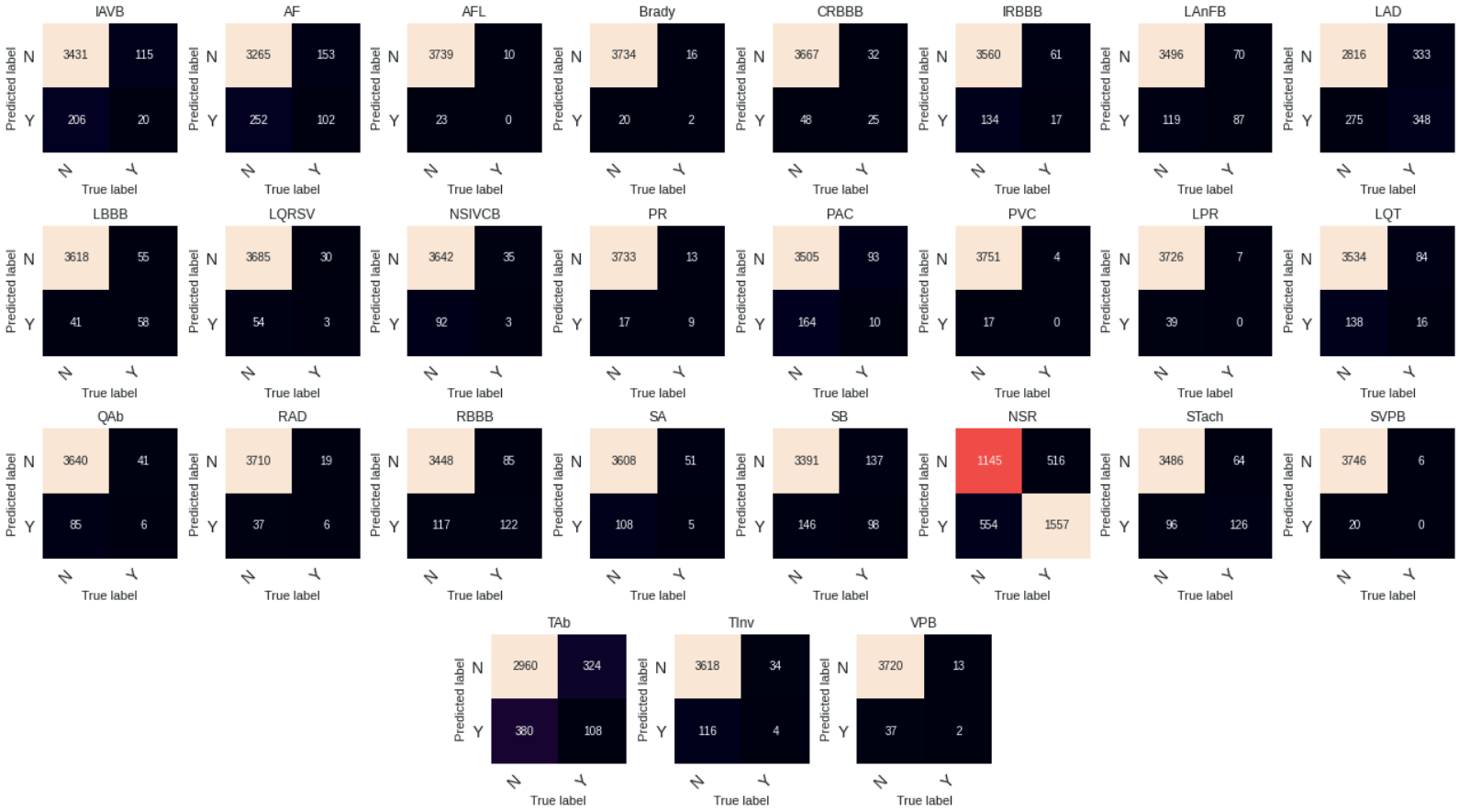
The first step to install the DSS in the computer will be to download the corresponding files in GitHub [35]. Before continuing with the installation, the user needs to make sure that Python 3.8.9 and Node JS are installed. If that is not the case, follow these links to install them [36] [37]. Once installed, we will divide the steps into two parts: the front-end and the back-end.

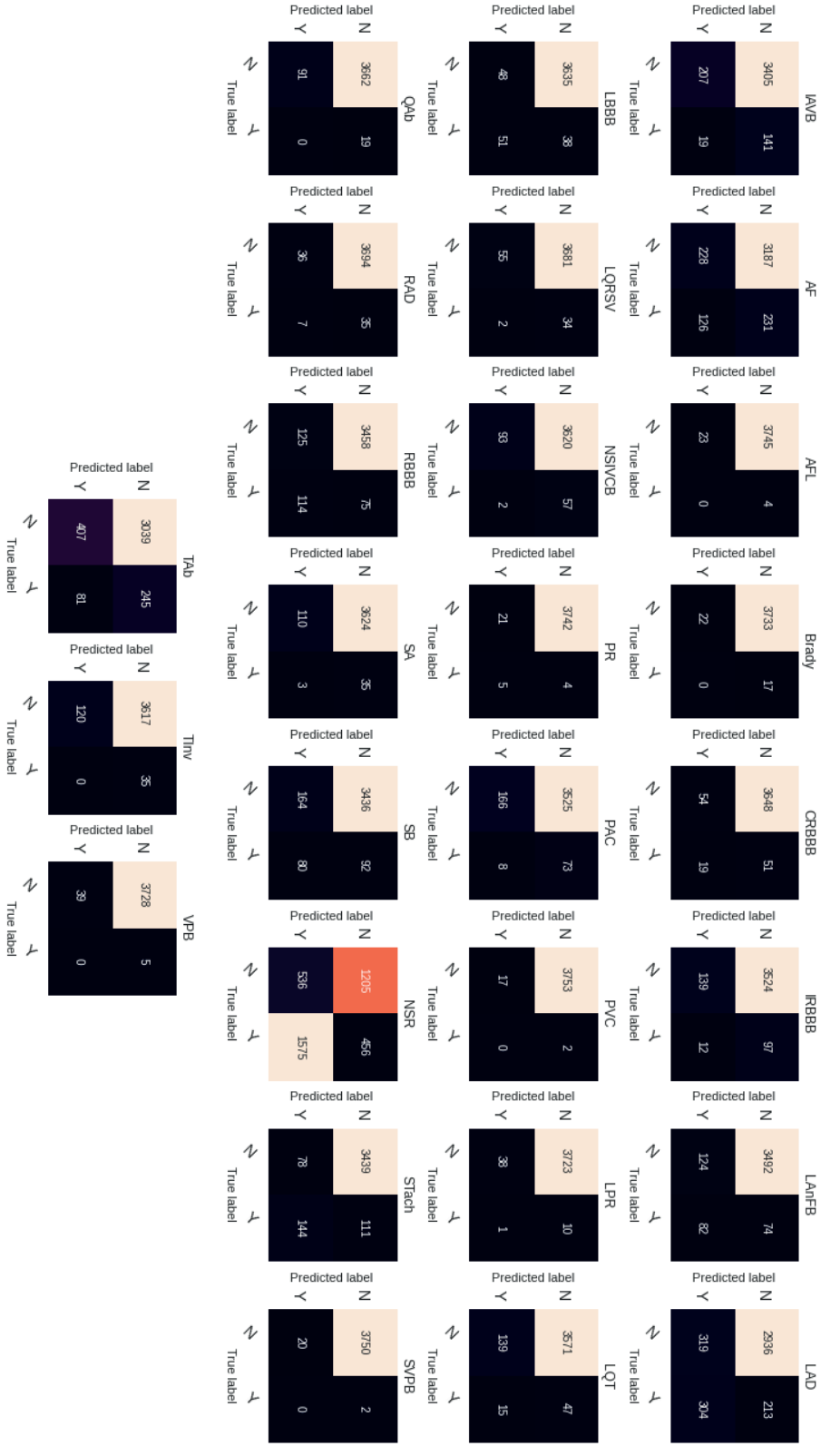
In the front-end, the user will need to get into the *./front/'* route using the *cd* command. Then, once there, the user should run **npm install**, which will create the node modules folder with all the required dependencies to run the front-end. The process will take time because it is a heavy download. Once done, the user can just run the **npm start** command in the same route to start the front-end in the browser in the address *http://localhost:3000/*.

The back-end installation will be done using a *requirement.txt* file. That will install all the python dependencies directly by running **pip install -r requirements.txt**, which is a folder that has all the required libraries along with their correspondent versions. If the installation of all packages is not completed for some reason (like the Operative System version), the user will have to debug the errors and solve them one by one. To check if the server is working, execute inside the *./server/* route the command **python app.py**. In case everything is working correctly, the server will be launched at *127.0.0.1:5000*.


M1 classifier - Individual confussion matrices
IAVB
AF
AFL
Brady
CRBBB
IRBBB
LAnFB
LAD
LBBB
LQRSV
NSIVCB
PR
PAC
PVC
LPR
LQT
QAb
RAD
RBBB
SA
SB
NSR
STach
SVPB
TAb
TInv
VPB
Predicted label
True label
Y
N

M2_256 - Individial confusion matrices




M2_512 - Individual confussion matrices
IAVB
AF
AFL
Brady
CRBBB
IRBBB
LAnFB
LAD
LBBB
LQRSV
NSIVCB
PR
PAC
PVC
LPR
LQT
QAb
RAD
RBBB
SA
SB
NSR
STach
SVPB
TAb
TInv
VPB
Predicted label
True label
N
Y

Bibliography

[1] Pharmaboardroom — top causes of death in spain. https://pharmaboardroom.com/articles/top-causes-of-death-in-spain/. (Accessed on 02/05/2021).

[2] Cardiovascular diseases (cvds). https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds). (Accessed on 01/28/2021).

[3] Arthur Jochems, Timo M Deist, Johan Van Soest, Michael Eble, Paul Bulens, Philippe Coucke, Wim Dries, Philippe Lambin, and Andre Dekker. Distributed learning: developing a predictive model based on data from multiple hospitals without data leaving the hospital–a real life proof of concept. *Radiotherapy and Oncology*, 121(3):459–467, 2016.

[4] Tamim Alsuliman, Dania Humaidan, and Layth Sliman. Machine learning and artificial intelligence in the service of medicine: Necessity or potentiality? *Current research in translational medicine*, 68(4):245–251, 2020.

[5] Stanfordmedicinehealthtrendswhitepaper2017.pdf. https://med.stanford.edu/content/dam/sm/sm-news/documents/StanfordMedicineHealthTrendsWhitePaper2017.pdf. (Accessed on 01/29/2021).

[6] Jason Waechter, David Reading, Chel Hee Lee, and Mathieu Walker. Quantifying the medical student learning curve for ecg rhythm strip interpretation using deliberate practice. *GMS journal for medical education*, 36(4), 2019.

[7] Jahangir Moini. *Phlebotomy: Principles and Practice*. Jones & Bartlett Publishers, 2013.

[8] Heart -anatomy, function, facts, diseases, diagnosis – leogenic healthcare pvt ltd. https://www.leogenic.com/heart-anatomy-function-facts-diseases-diagnosis/. (Accessed on 02/06/2021).

[9] Heart conduction system — medical education for undergraduate md students. https://mdmedicine.wordpress.com/2011/04/24/heart-conduction-system/. (Accessed on 02/06/2021).

[10] The 12 leads of christmas: V2 — ems 12 lead. http://ems12lead.com/2015/01/06/12-leads-of-chistmas-v2/#gref. (Accessed on 02/09/2021).

[11] Bipolar leads - ecg lead placement - normal function of the heart - cardiology teaching package - practice learning - division of nursing - the university of nottingham. https://www.nottingham.ac.uk/nursing/practice/resources/cardiology/function/bipolar_leads.php. (Accessed on 02/06/2021).

[12] Electrocardiograma - wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Electrocardiograma. (Accessed on 02/06/2021).

[13] Jiapu Pan and Willis J. Tompkins. A real-time qrs detection algorithm. *IEEE Transactions on Biomedical Engineering*, BME-32(3):230–236, 1985.

[14] Annamalai Natarajan, Yale Chang, Sara Mariani, Asif Rahman, Gregory Boverman, Shruti Vij, and Jonathan Rubin. A wide & deep transformer neural network for 12-lead ecg classification. 2020.

[15] Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020 — physionet/cinc challenges. https://physionetchallenges.org/2020/papers/. (Accessed on 06/21/2021).

[16] Erick A Perez Alday, Annie Gu, Amit J Shah, Chad Robichaux, An-Kwok Ian Wong, Chengyu Liu, Feifei Liu, Ali Bahrami Rad, Andoni Elola, Salman Seyedi, et al. Classification of 12-lead ecgs: the physionet/computing in cardiology challenge 2020. *Physiological measurement*, 41(12):124003, 2020.

[17] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

[18] Siegrid Lowel and Wolf Singer. Selection of intrinsic horizontal connections in the visual cortex by correlated neuronal activity. *Science*, 255(5041):209–212, 1992.

[19] Why sigmoid?. ever since you have started to learn... — by swain subrat kumar — núcleoml — medium. https://medium.com/n%C3%BAcleoml/why-sigmoid-ee95299e11fd. (Accessed on 05/05/2021).

[20] IE Poletaev, KS Pervunin, and MP Tokarev. Artificial neural network for bubbles pattern recognition on the images. In *Journal of Physics: Conference Series*, volume 754, page 072002. IOP Publishing, 2016.

[21] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[22] Pablo Martín Redondo. DiseÑo y desarrollo de un sistema de ayuda al diagnÓstico de enfermedades cardiovasculares a partir de electrocardiogramas. June 2020.

[23] Sinus bradycardia — cedars-sinai. https://www.cedars-sinai.org/health-library/diseases-and-conditions/s/sinus-bradycardia.html. (Accessed on 06/13/2021).

[24] T wave litfl ecg library basics. https://litfl.com/t-wave-ecg-library/. (Accessed on 06/11/2021).

[25] Antonio Ochotorena Laynez. Application of natural language processing techniques for biomedical document classification. May 2020.

[26] Github - physionetchallenges/python-classifier-2021: Python classifier for the physionet/cinc challenge 2021. https://github.com/physionetchallenges/python-classifier-2021. (Accessed on 06/13/2021).

[27] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian E Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica B Hamrick, Jason Grout, Sylvain Corlay, et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pages 87–90, 2016.

[28] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[29] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14(9), 2011.

[30] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[31] Keras: the python deep learning api. https://keras.io/. (Accessed on 06/10/2021).

[32] tf.keras.callbacks.modelcheckpoint — tensorflow core v2.5.0. https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint. (Accessed on 06/10/2021).

[33] Welcome to biosppy — biosppy 0.6.1 documentation. https://biosppy.readthedocs.io/en/stable/. (Accessed on 06/22/2021).

[34] neurokit2 · pypi. https://pypi.org/project/neurokit2/. (Accessed on 06/22/2021).

[35] 8toz/dss_12lead_ecg_detection: Implementation of a dss based on a trained neural network. the repository contains colab notebooks for the deep learning task and two additional folders containing a react frontend and a flask backend in charge of implementing the dl model. https://github.com/8toz/DSS_12Lead_ECG_detection. (Accessed on 06/23/2021).

[36] Download python — python.org. https://www.python.org/downloads/. (Accessed on 06/22/2021).

[37] Descarga — node.js. https://nodejs.org/es/download/. (Accessed on 06/22/2021).

[38] Cuiwei Li, Chongxun Zheng, and Changfeng Tai. Detection of ecg characteristic points using wavelet transforms. *IEEE Transactions on biomedical Engineering*, 42(1):21–28, 1995.