University of Manouba
National School of Computer Sciences

Report of

the Design and Development Project

Subject: Automatic ECG Diagnosis For Arrhythmia Identification

Authors :

Mr. Ahmed Amara Mr. Khalil Boubaker
Mr. Khalil Regaieg

Supervisor :

Dr. Leila Nasraoui

Academic Year : 2020 / 2021

Assessments and signature of supervisor

Acknowledgements

Through this report, we would like to express our sincere gratitude and thanks to everyone who has contributed to the development of this work, in particular, our supervisor Dr.Leila Nasraoui for her continuous support and fruitful advice and guidance throughout the project.

We are also very grateful to all the teachers of the National School for Computer Sciences for their great efforts in transferring their valuable knowledge to us.

Finally, we would like to express our deep respect to the members of the jury for their precious time to evaluate our work and for their constructive comments.

Contents

List of Figures

List of Tables

List of Acronyms

AI	*Artificial Intelligence*
ANN	*Artificial Neural Network*
API	*Application Programming Interface*
CNN	*Convolutional Neural Network*
CSV	*Comma-Separated Values*
DL	*Deep Learning*
ECG	*Electrocardiogram*
IDE	*Integrated Development Environment*
ML	*Machine Learning*
RNN	*Recurrent Neural Network*

Introduction

Over the past few years, automation has invaded the world and helped us to speed some of the tasks that required a lot of time and energy consumption. Medical artificial intelligence is used to assist doctors by detecting anomalies and diseases such as cancer, Alzheimer and heart conditions.

This is the context in which this design and development project takes place. This project is intended to classify electrocardiogram (ECG) heartbeat signals into 5 classes using Convolutional Neural Networks.

This report is divided into four main chapters.

- In the first chapter, we will define some of the concepts needed for understanding our project. Then, we will present the existing solutions to similar problems, the characteristics, and limitations of each one and we will present our solution.
- Moving to the second chapter, we will identify the actors, The functional and non-functional requirements of the system and we will elaborate some diagrams to further understand the system.
- Next, in the third chapter, we will detail the steps of designing our solution starting with data pre-processing, then the model architecture, and the way it's trained and finish with explaining some testing metrics.
- In the fourth chapter, we present the software development environment, explain how the model is implemented and trained, and display some performance metrics.

Finally, by giving future perspectives and potential improvements for our work a conclusion will close this report.

Chapter 1

Preliminary Study

In this chapter, we present the technical terms of the field of artificial intelligence and the needed medical terms to understand the electrocardiogram signal. We then discuss the existing solutions for automatic heart disease diagnosis and we finish with introducing the proposed solution.

1.1 Artificial Intelligence

Artificial Intelligence (AI) is a field combining the advances in computer science with the rapidly increasing volumes of data to enable automated problem-solving [URL1].
A concept that is always highlighted is the similarity AI algorithms have with the qualities of the human mind like their capacity to recognize certain patterns, solve problems and learn.
This field also encompasses other sub-fields such as machine learning and deep learning.

1.1.1 Machine Learning

Machine Learning (ML) is a subfield of AI which solves a problem by assembling a data set and building a mathematical model based on the nature and type of that data set [URL2].
Learning can be divided into four types, supervised, unsupervised, semi-supervised, and reinforcement learning.

Supervised Learning

This type of learning is used when we are working with labeled data. The algorithm can learn from this data by comparing the predictions it makes with the actual values contained in the labels. This method needs a considerable amount of data to yield satisfying results

[URL2].
Supervised learning problems can be separated into two kinds, classification and regression:

- **Classification :** in this type of problem, we categorize the data into classes in the target column and the algorithm predicts the class of given data point. The classes are also called labels, targets, or categories. The classification is referred to as **binary** when we have only two classes, **multi-class** when we have multiple classes and **multi-label** when some of the data points are given more than one label.

- **Regression :** this type of algorithm tries to predict a continuous quantity, this technique is implemented on data to calculate the best fit relation between independent or dependent variables. Its essential goal is to predict an output from numerical data like stock market data and housing prices [URL3].
  Some of the most used regression algorithms are linear regression, polynomial regression, gradient descent.

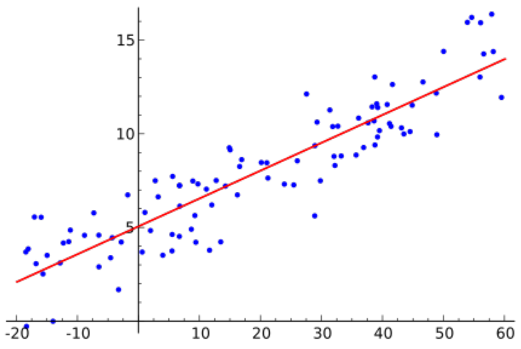


Figure 1.1: Linear regression example
[URL4]

Unsupervised Learning

These algorithms are used when working with data that is not labeled. The way they work is by discovering similarities and differences making it ideal for exploratory data analysis.
One of the most used approaches of unsupervised learning is clustering which is used to process raw unlabeled data into groups composed of the data points sharing a specific pattern of feature [URL2].

Semi-supervised Learning

This method can benefit from a data-set containing a small amount of labeled data and a big amount of labeled data [URL2].

Reinforcement Learning

In reinforcement learning, datasets are not used in training the model. Instead, the machine takes certain steps on its own, analyzes the feedback, and then tries to improve its next step to get the best possible outcome [URL2].

1.1.2 Deep learning

Deep learning (DL) is a subclass of ML that uses artificial neural networks (ANN) to extract prevalent features from the data based on which a conclusion or a decision will be formulated. Figure 1.2 shown below is an example of the structure of an ANN.
The main differences between DL and ML are the lower need for human intervention, the structure of ANN, and the higher amount of required data [URL5].

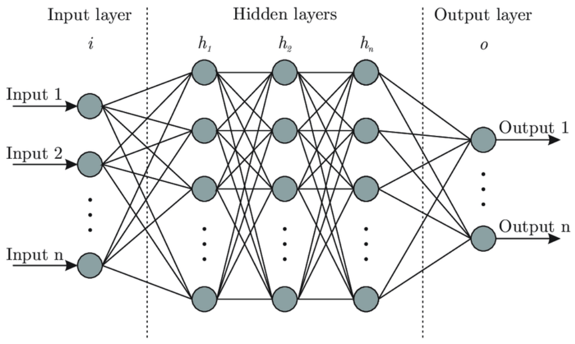


Figure 1.2: ANN basic structure
[URL6]

ANNs are composed of:

- **Neurons :** each artificial neuron has multiple inputs and one output generated from an activation function (Sigmoid, Relu..). Every one of these units is connected to other neurons through a connection link [URL7]. Figure 1.3 contains a visual representation of an artificial neuron.

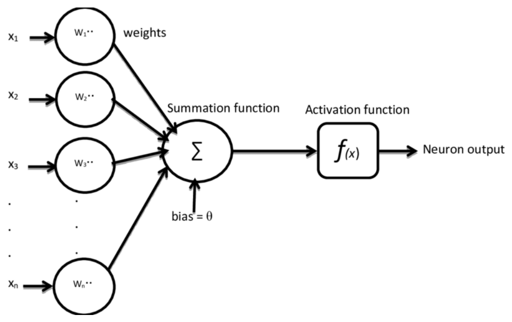


Figure 1.3: Artificial neuron
[URL8]

- **Connections and weights :** every connection linking two neurons has a specific weight which indicates the importance of the connection in the neural network.

- **Propagation functions :** it is used to transfer values through the neurons in each layer of the ANN. We can add a bias term to the summation function,the Figure 1.4 shown bellow represents different activation functions [URL9].

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

$$Relu(x) = \max(0, x)$$

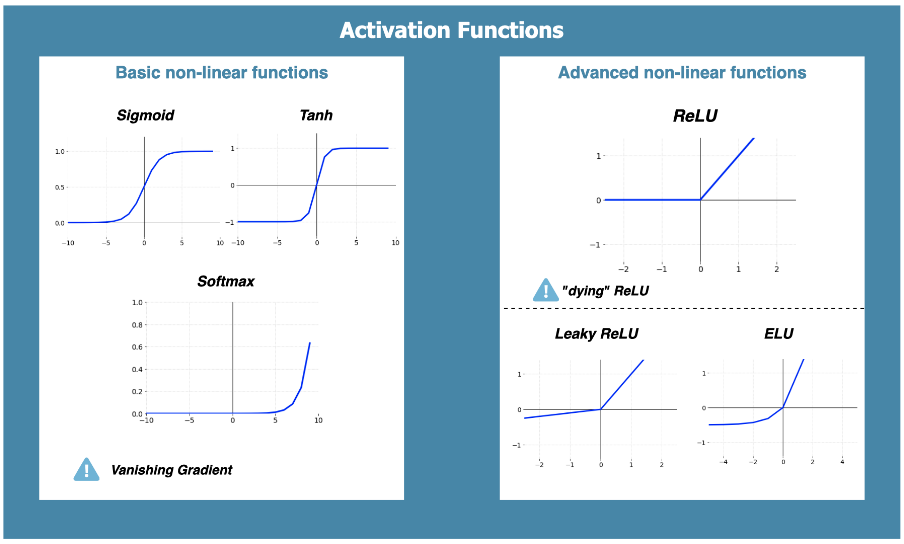


Figure 1.4: Different activation functions
[URL9]

1.1.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of ANN mostly used in image and audio to automatically detect different features and patterns in order to distinguish one from another.
The CNN is basically formed by stacking a combination of specific layers of neurons. As shown in Figure 1.5, we can see that this type of ANN relies mainly on three kinds of layers, the convolutional layers, the pooling layers, and the fully connected layers[URL10].

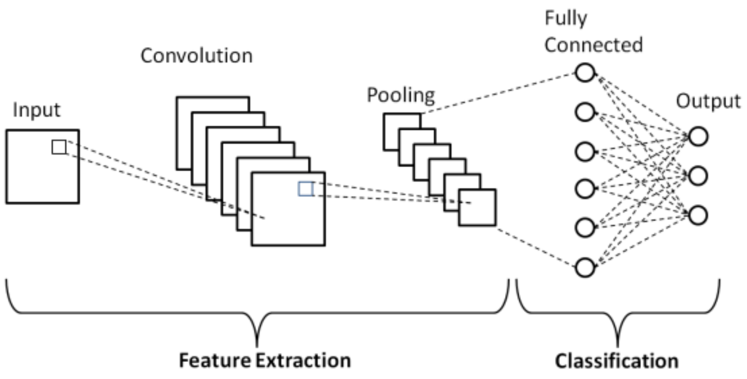


Figure 1.5: CNN structure
[Phung and Rhee, 2019]

- **Convolutional layer :** Convolutional layers are the major building blocks of a CNN. They are composed of several kernels which generally are small matrices initialized with random parameters. These kernels are slid across the height and width of the input to generate an output that consists of the dot product of the kernel and the corresponding portion of the input [URL11].

  In order to increase the non-linearity in the output an activation function like the ReLU or Tanh is used at the end of every convolutional layer

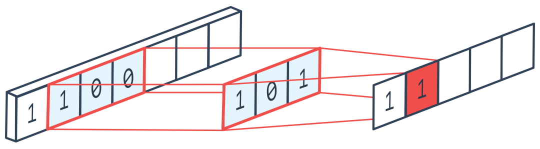


Figure 1.6: One dimensional convolution
[URL12]

- **Pooling layer :** This operation involves sliding a filter of a specified size over each channel of the input coming from the previous layer thus highlighting the features contained in the area currently covered by the filter [URL11].
  Pooling layers are used to reduce the size of the feature maps reducing the number of the parameters the model is required to learn and the amount of computation needed.
  In most cases, pooling layers are stacked right after convolutional layers because they are effective at summarising the feature maps generated by the previous layer.
  Some of the most used types of pooling are Max Pooling and Average Pooling.

  - **Max pooling :** In max pooling, as shown in Figure 1.7, in every region of the feature map the size of the filter is replaced by its maximum element before sliding to the next area. This operation is repeated on the whole input feature map.

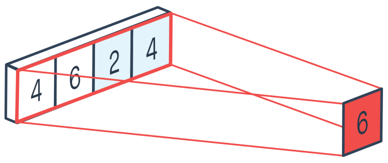


Figure 1.7: Max pooling
[URL13]

– **Average pooling :** In the case of average pooling and as shown in Figure 1.8, the specified region is instead replaced with the mean value of this region.

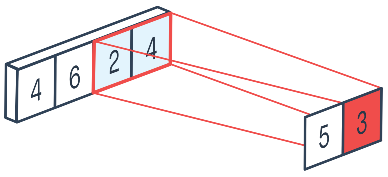


Figure 1.8: Average pooling
[URL14]

The type of pooling used can have a different effect on the resulting output. Max pooling highlights the most important features while average pooling gives a smoother feature extraction.

- **Fully connected layer :** This is the last layer of CNN. It is a feed-forward neural network. The output of the previous pooling or convolutional layer is flattened and then fed to this network. After going through the fully connected layers, a **softmax** function is used to determine the probability of the input being in a certain class which is classification.

$$softmax(x)_i = \frac{\exp(x)}{\sum_{j=1}^{K} \exp(x_j)} \tag{1.1}$$

One of the most common issues we face when building a neural network is **over-fitting**. This concept occurs when the model fits the training data too closely so it doesn't perform well at all when facing new unseen data.
An array of regularization techniques have been proven to improve the performance of CNNs and reduce over-fitting issues. In the next part, we will be focusing on the batch normalization and dropout techniques.

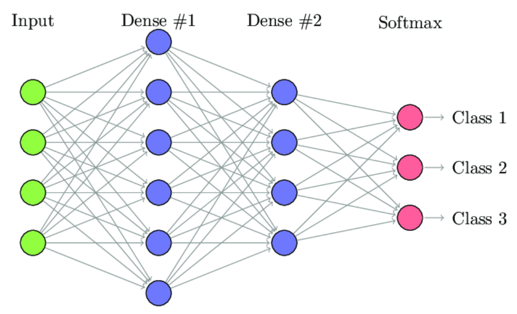


Figure 1.9: Fully connected layer
[URL15]

- **Batch Normalization :** is a process that will make neural networks faster and more stable. This technique is achieved through a normalization step that consists of fixing the means and variances of each layer's input [URL16] .

- **Dropout :** In the fully connected layer, there are a lot of parameters that makes neurons very co-dependent during training causing over-fitting to the training data. To reduce this co-dependency between neurons, the dropout method was developed. The way to do this is to basically drop some randomly chosen neurons during training acting like they don't exist for the duration of a certain forward or backward pass[URL16].

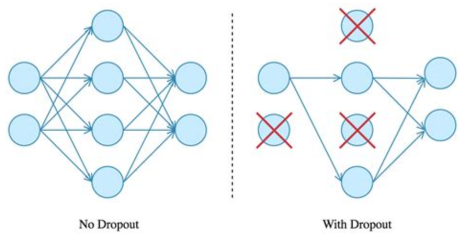


Figure 1.10: Dropout technique
[URL16]

1.2 Electrocardiogram

In this section, we introduce the ECG signal and some of its characteristics. We then present heart arrhythmia, how the ECG helps to detect it, and some of its most common types.

1.2.1 Introducing the Electrocardiogram

An electrocardiogram (ECG) is a record of the electrical activities of the heart. As shown in Figure 1.11, it is usually presented in the aspect of a graph of voltage value versus time, and it's generated by measuring the amplitude differential between two electrodes during the depolarization and repolarization phase of the heart [URL17].

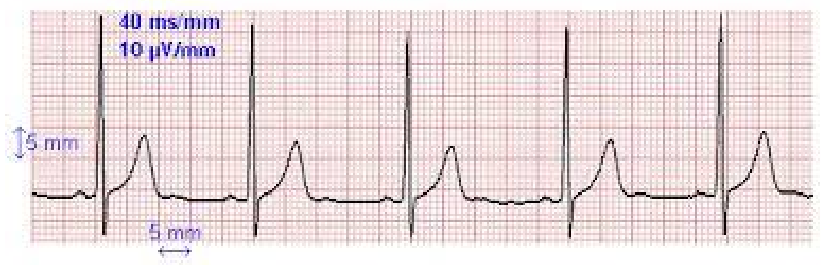


Figure 1.11: ECG signal
[URL18]

This signal is commonly used in medicine due to its simple acquisition procedure and the valuable diagnostic insight that it gives which can help in identifying the pathophysiological condition of the cardiac muscle.

There exist many variations of the ECG signal but the standard 12 leads ECG which it's generated by 10 electrodes placed on the chest and limbs of the patients as shown in the figure below.

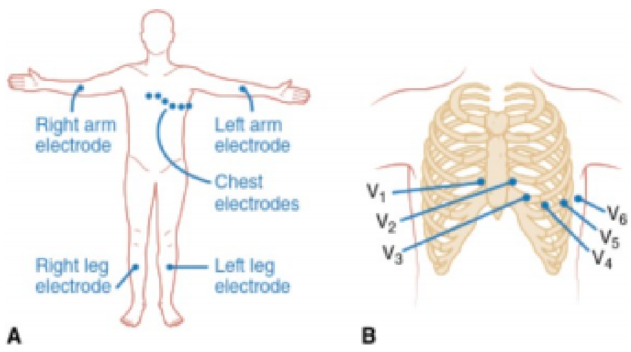


Figure 1.12: (A) Placement of electrocardiogram (ECG) electrodes, (B) Close-up view of chest electrodes placement[Lilly, 2015]

1.2.2 Characteristics of the ECG Signal

For a healthy patient, the ECG signal is composed of three major compartments shown in Figure 1.13 [URL19].

- **P wave:** The P wave represents the depolarization of the atria.
- **QRS complex:** Following the P wave, the tracing returns to its baseline to then introduce the QRS complex which corresponds to depolarization of the ventricles.
- **T wave:** After the QRS complex, the tracing returns to baseline once again and after a brief delay, repolarization of the ventricular cells is followed by the T wave.

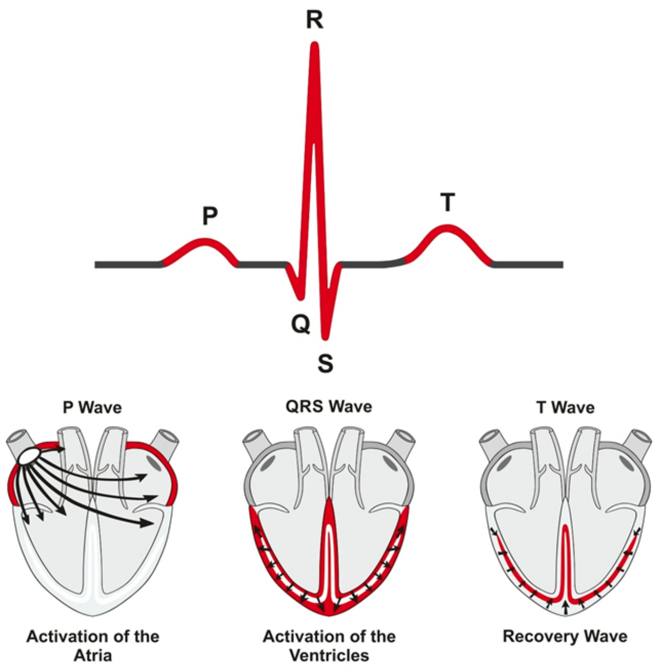


Figure 1.13: ECG of a healthy heart
[URL20]

For patients with heart illness, a variety of cardiac abnormalities may occur producing some mutation in the aspect of the ECG. Some of these diseases require immediate diagnosis that can only be provided by a specialist. So, to decrease the rate of mortality caused by cardiac diseases, the development of an efficient and reliable automatic diagnosis tool has become an absolute necessity.

1.2.3 Arrhythmia

Arrhythmia is a heart condition in which the heart beats rapidly, slowly, or with an irregular pattern. Several types of arrhythmia can be identified from the shape of the ECG signal, such as supraventricular arrhythmias, ventricular arrhythmias, and premature heartbeats [URL21].

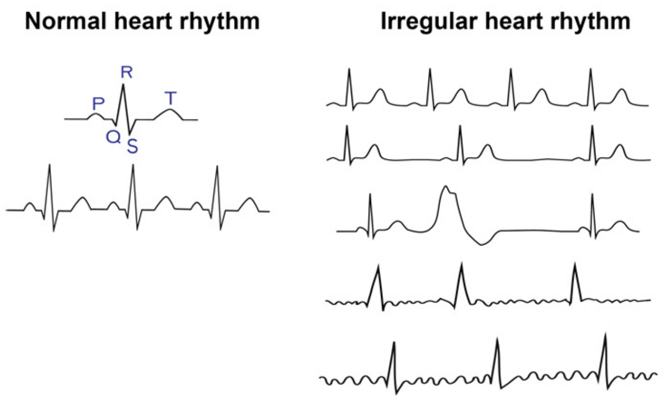


Figure 1.14: Arrhythmia
[URL22]

1.3 Existing Solutions for Arrhythmia Detection

We dedicate this section to present solutions that have a similar purpose to ours. We will compare them, highlight what they lack and justify the need for the functionalities of our project.

1.3.1 Classification of ECG Arrhythmia Using RNN

This project was carried out by Shraddha Singh, Saroj Kumar Pandey, Urja Pawar, and Rekh Ram Janghel from the National Institute of Technology, Department of Information Technology, Raipur, India [URL23].
The MIT-BIH Arrhythmia Database is used in this project and it was divided into a training set and test set in a ratio of 70% and 30% respectively. Recurrent Neural Networks (RNN) have been applied to classify the ECG recordings. As shown in figure 1.14 the model is composed of only three layers of RNN, the first layer contains 128 neurons, the second contains 256 neurons and 100 neurons for the third layer. To each layer was added a dropout rate of 0.2 with a linear activation function and Mean Squared Error was used as the loss function.
The advantage of this model is the simplicity and low computational cost, however, the accuracy of this model was 85.4% which is relatively low for this problem.

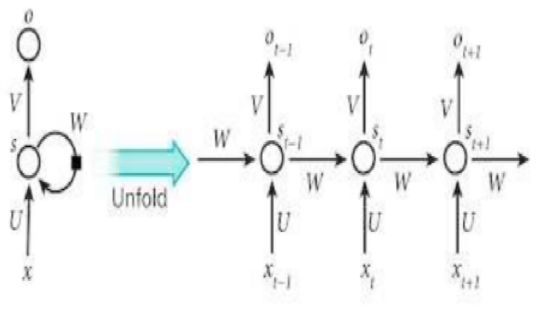


Figure 1.15: RNN model
[URL23]

1.3.2 Cardiologist-Level Arrhythmia Detection CNN

This project was carried out by Andrew Y. Ng, Codie Bourn, Masoumeh Haghpanahi, Awni Y. Hannun and Pranav Rajpurkar [URL24].
The data set was collected from 30000 different patients which is a very large data set compared to other data sets of its kind and it was divided into a training set and a validation set in a ratio of 90% and 10% respectively. As shown in figure 1.15 The model used in this project was a large CNN containing 33 layers of convolution followed by a fully connected layer and a softmax.
The accuracy of this model was 98.27% which is great, however, the data set is very large and the CNN architecture used is big compared to the RNN solution presented previously which will cause a big computational cost.

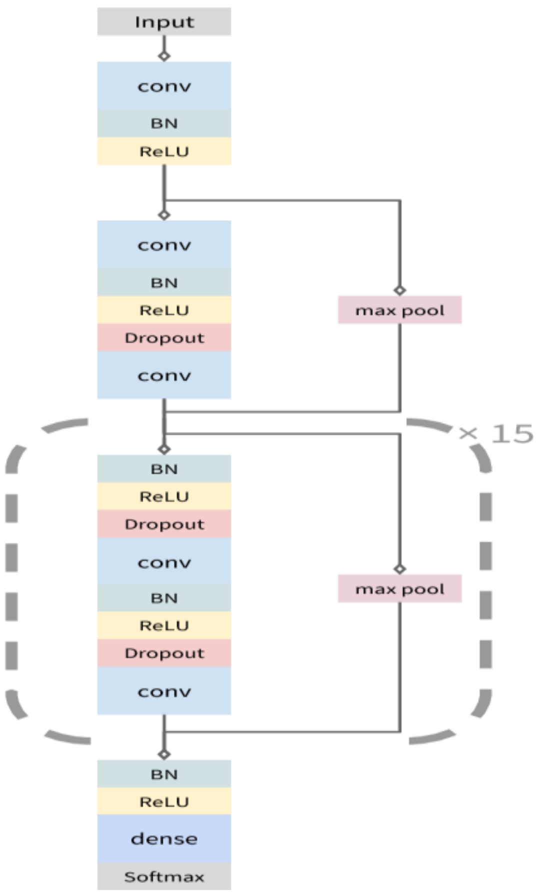


Figure 1.16: CNN architecture for the Cardiologist-Level Arrhythmia Detection model [URL24]

1.4 Proposed Solution

Our goal is to build an effective solution that doesn't require immense computational power and capable of automatically classifying an inputted ECG signal in order to assist with cardiovascular diagnosis.
After thoroughly studying some of the existing algorithms and used models, we opt for the use of a basic one dimensional CNN architecture seeing how this type of neural networks has been proven to be efficient for automatic time series classification problems.
The architecture of our model consists of a DNN that takes an ECG fragment as an input and gives its classification into one of five predefined heartbeat classes. These classes consist of a normal healthy heartbeat class and four other classes that contain certain abnormalities which will further be detailed in the next chapter.
Regarding our work methodology, we use an iterative approach where we start with the implementation, then move to the testing phase, and then the maintenance. This process is repeated until the requirements are met.

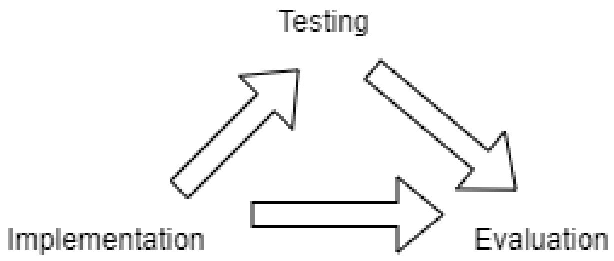


Figure 1.17: Iterative process methodology

1.5 Conclusion

In this chapter, we defined some concepts in the fields of AI and medicine we need for our project, listed some already existing solutions for ECG classification and we finally presented our proposed solution.
The following chapter will be presenting our system's requirements and specifications.

Chapter 2

Requirements' Analysis and Specification

Having presented the needed theoretical concepts and definitions, we will now move on to the technical specification step. We will begin with presenting the actors and listing the system's requirements. And then we will introduce some diagrams to further explain the behavior of the system.

2.1 Requirements' Analysis

In this section, we identify the actors and then define the requirements the system needs to meet in order to provide for their needs.

2.1.1 Actors Identification

User

This application is designed to be used by the medical staff needing assistance in diagnosing heart diseases.

Admin

The admin is the party responsible for managing the data set in addition to the configuration of the CNN model and its training.

2.1.2 Functional Requirements

Functional requirements are a description of the main functions the software must fulfill. Our work has to satisfy these specifications in order to fulfill the user's needs.

In our context, the system needs to satisfy a set of requirements for both its user and the admin.

Functional Requirements for User

The system must satisfy the following requirements for its user:

- **Allow the user to input an ECG beat for the algorithm to classify :** the loaded ECG heartbeat must be in the form of a one-line CSV file where the values represent the amplitude of the ECG signal after every sampling period. The file must contain the signal of a single heartbeat. In case these conditions aren't fulfilled the model's ability to be executed correctly is uncertain and the results provided are probably incorrect.

- **Give a diagnosis to the ECG :** the given ECG heartbeat signal is fed into the CNN model to be classified into one of the following classes which are predefined in the labeled data :

  - Normal Heartbeats
  - Supraventracular Heartbeats
  - Ventricular Ectopic Beats
  - Fusion Beats
  - Unknown Beats

- **Display the results of the classification :** the output given by the CNN is displayed on the screen as it represents the diagnosis of the ECG heartbeat that was given by the user.

Functional Requirements for Admin

The system must also provide these services to the admin :

- **Manage data-set :** the admin must be able to easily manage the training and testing data set. He is given the ability to add new data, delete data that is no longer needed and prepare the data to be used in training the model.

- **Manage CNN model :** the admin has to be able to reconfigure the model at any given point, to retrain it and evaluate its performance.

2.1.3 Non-Functional Requirements

Non-functional requirements are the criteria that describe the application's capabilities and potentials that improve its functionality to give its users a better experience.
For our system, these requirements are the following :

- **Performance :** the ECG diagnosis procedure should take less than 5 seconds.
- **Reliability :** the ECG should be classified with a high precision rate of at least 90 percent.
- **Re-usability :** the model on which this application is built should be able to train with different data sets.

2.2 Requirement's Modeling

In this section, we will be modeling the interactions between the user and the system as well as the system's behavior in order to get a better understanding of our project's requirements.

2.2.1 Modeling language

For modeling purposes, we will be utilizing the Unified Modeling Language (UML) to obtain a standard visualization of the system's requirements. UML is a standard modeling language used in the field of software engineering that provides a standard visualization of the system [URL25].

2.2.2 Use Case Diagram

The use case diagram (see Figure 2.1) defines the expected behavior of the system and its interactions with both actors.
The user can import a CSV file containing the needed ECG data into the system and get a classification result on the input signal while the admin has, in addition to the normal user's capabilities, the ability to change the data-set and retrain the model.

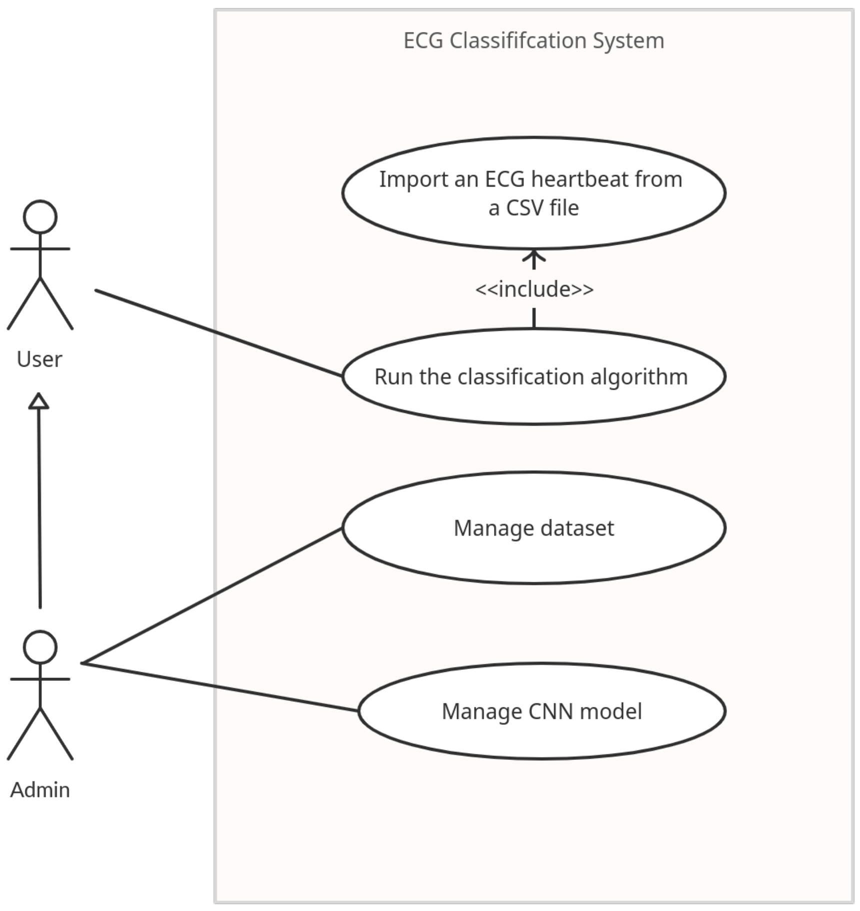

ECG Classififcation System
Import an ECG heartbeat from a CSV file
<<include>>
Run the classification algorithm
Manage dataset
Manage CNN model
User
Admin

Figure 2.1: Use case diagram

2.2.3 Scenarios Description Using Sequence Diagrams

Sequence diagrams help us to visually depict the dynamic interactions taking place between users and the system. This section contains the sequence diagram describing each use case scenario.

Use Case "Import ECG heartbeat from a CSV file"

Figure 2.2 describes the use case scenario "import ECG heartbeat from a CSV file" in which the user imports a heartbeat signal in the form of a CSV file, this file is then loaded into the system to be reshaped in a way that fits the model's input. Following this operation, the classification algorithm is called to give the final prediction about the state of the heartbeat signal.

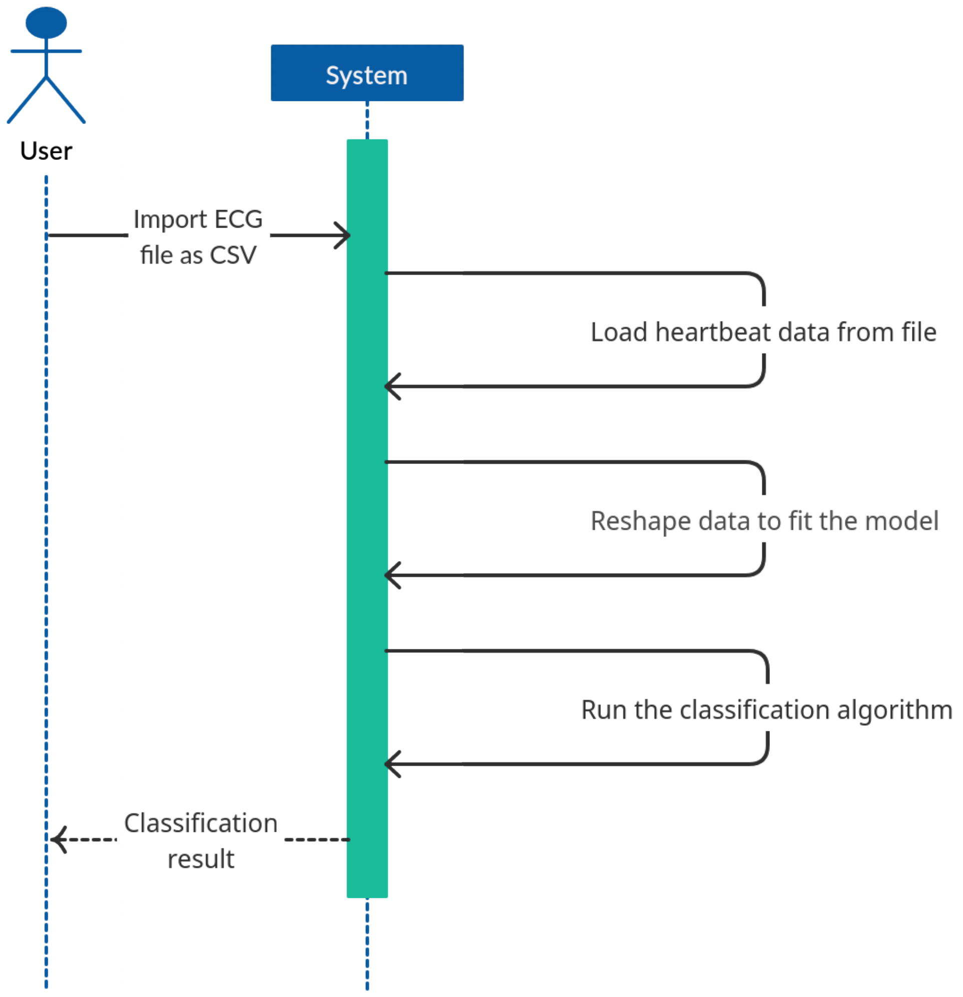

User
System
Import ECG file as CSV
Load heartbeat data from file
Reshape data to fit the model
Run the classification algorithm
Classification result

Figure 2.2: Sequence diagram describing use case "Import ECG heartbeat from a CSV file"

Use Case "Manage data set"

Figure 2.3 is an illustration of the use case diagram for the scenario "Manage data set". The admin is capable of adding new data, deleting parts that are no longer needed as well as preparing it in order to be fed into the CNN model for training.

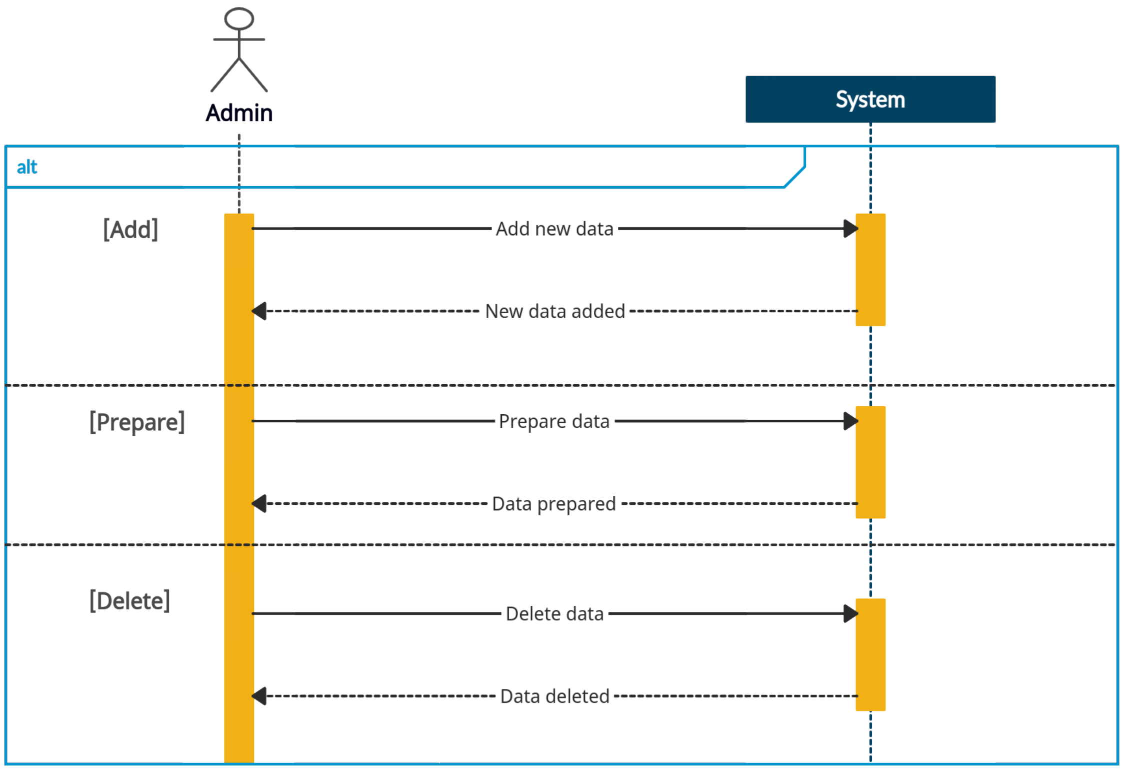


Figure 2.3: Sequence diagram describing use case "Manage data set"

Use Case "Manage CNN model"

Figure 2.4 shows the ability that the admin has in managing the model whether through configuring its parameters, training it on the provided dataset, or evaluating its performance on unseen test data.

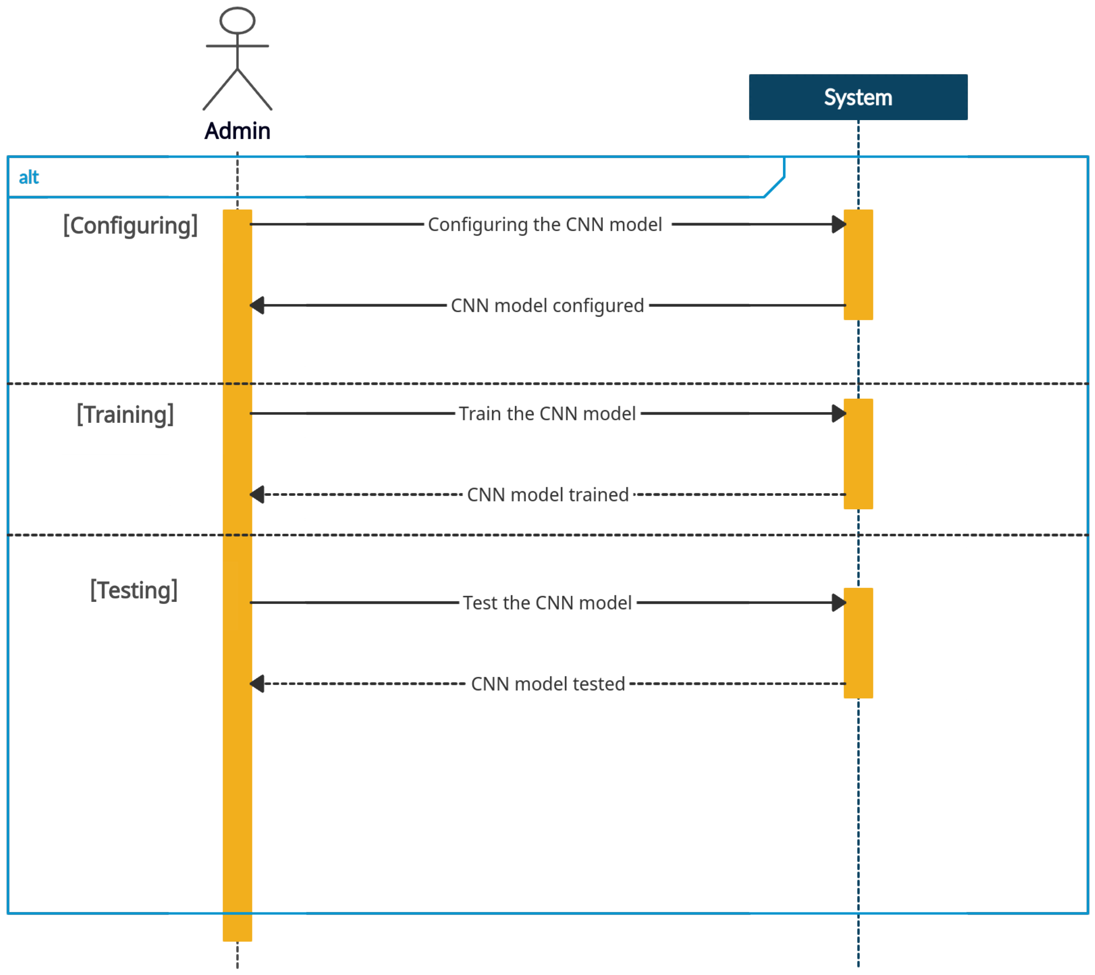


Figure 2.4: Sequence diagram describing use case "Manage CNN model"

2.2.4 Activity Diagram

Activity diagrams are helpful in visualizing the dynamic aspects of the system by modeling the flow from one activity to the next.

Figure 2.5 describes the behavior of the system. First, the data is loaded and then reshaped to fit the input layer of the model. Then, the input data is fed to the CNN model and passed through its different layers turning it into feature maps. Finally, the system classifies the initially loaded ECG into one of the previously mentioned heartbeat classes.

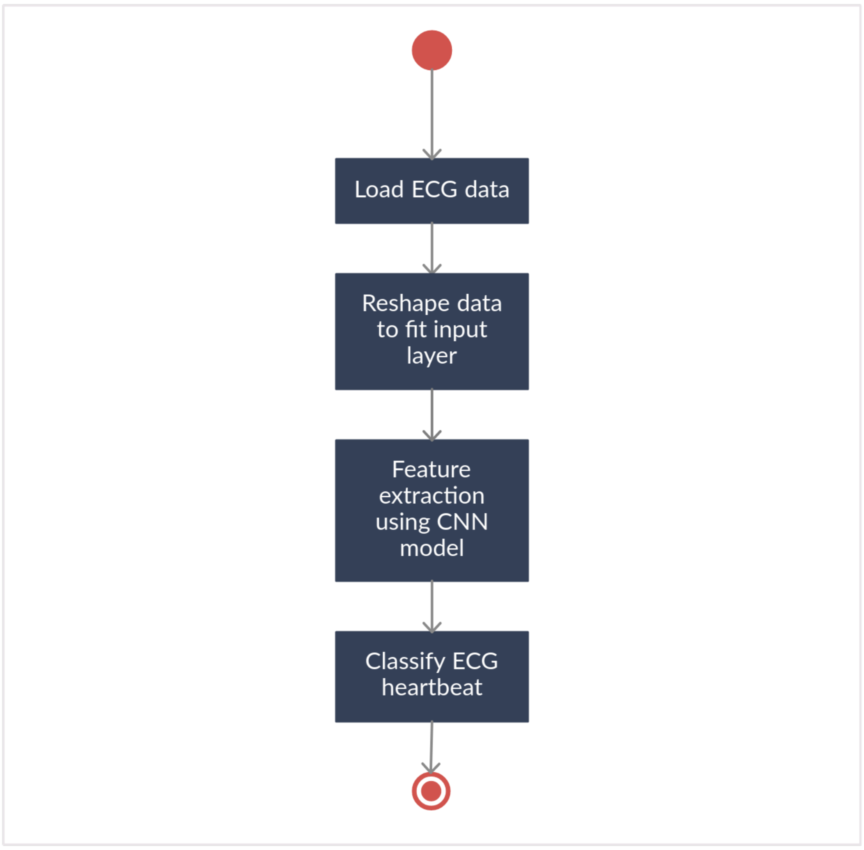

Load ECG data
Reshape data to fit input layer
Feature extraction using CNN model
Classify ECG heartbeat

Figure 2.5: Activity diagram

2.3 Conclusion

This chapter exposed the functional and non-functional requirements of this project and presented multiple diagrams to visualize the system's main aspects.
In the next chapter, we will be delving into the details of these operations.

Chapter 3

Design of the CNN Solution

CNN models were proven to be powerful in the task of classifying time series data. In this chapter, we will go into the details of the steps required to construct a reliable model. To accomplish this task, first, we will start by pre-processing the data which is the phase of preparing the data. Following this part, we will move into the model configuration phase which is considered the most crucial task in the totality of the project mainly because it determines the model's performance in the training phase which is the third stage of this process. Finally, we will sum by giving an estimation of our model's performance.

3.1 Data Pre-processing

Data pre-processing is a very important part of every artificial intelligence project, it could be considered the most complicated part depending on the type of data and the problem at hand. The hardships of this step mainly consist in transforming the data into types of data structure that can be fitted perfectly to the required input of the model.
This section focuses on the data pre-processing step. The first part is a presentation of the used dataset and the second part details the pre-processing operations applied to the training data.

3.1.1 Dataset

The Physio-Net MIT-BIH Arrhythmia Database [URL26] contains 47 ECG recordings from different patients with a sampling rate of 360 Hz. Every recording contains 48 half-hour segment of two-channel ambulatory ECG recordings
Each one of the recordings is examined by at least two cardiologists to be labeled into one of the five categories detailed in Table 3.1.

In our case, the data used [URL27] as a source for ECG records is composed of CSV files where single heartbeats were extracted from the raw ECG recordings of the Physio-Net MIT-BIH Arrhythmia Database. Figure 3.1 contains a sample beat obtained after

Category	Annotations
N	Normal Left/Right bundle branch block Atrial escape Nodal escape
S	Atrial premature Aberrant atrial premature Nodal premature Supra-ventricular premature
V	Premature ventricular contraction Ventricular escape
F	Fusion of ventricular and normal
Q	Paced Fusion of paced and normal Unclassifiable

Table 3.1: Summary of mappings between beat annotations and AAMI EC57 categories.

extraction.

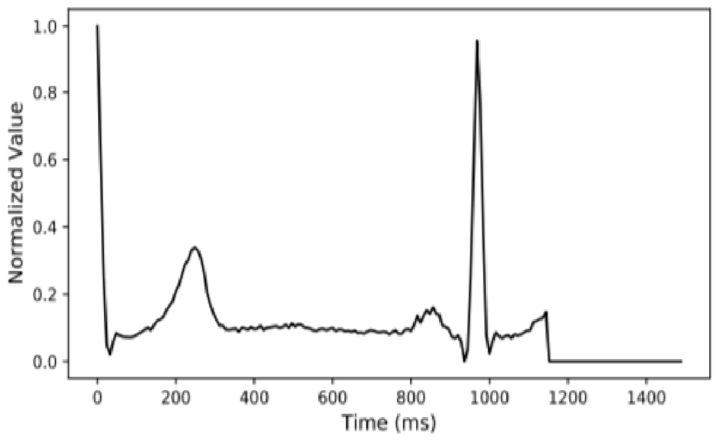


Figure 3.1: A single heartbeat extracted from the raw data [Kachuee and Fazeli, 2018]

The data used in our project is comprised of two CSV files, the first contains the training data and the second, the testing data.
The first file comprises 87554 labeled heartbeat samples and the second file contains 21892 samples reserved for testing. Each heartbeat sample is a set of 187 floating values in the $[0, 1]$ interval.
The first file will later be split into two parts. The first, with a 0.8 ratio is allocated for

training. The second, with a 0.2 ratio is used for validation.

3.1.2 Data Re-Sampling

Data re-sampling is used when we have an imbalanced class repartition. Thus, we need to regulate these imbalances by removing samples from the preponderant class which is also called down-sampling and/or increase the data of the minority classes, also referred to as up-sampling. Figure 3.2 contains a visual presentation of these two operations. In our Case, when comparing the number of heartbeats contained in each class, we can see that our data-set is severely imbalanced. In order to adjust the imbalances in our classes, we re-sample each class to 20000 examples to obtain equal data partitioning.

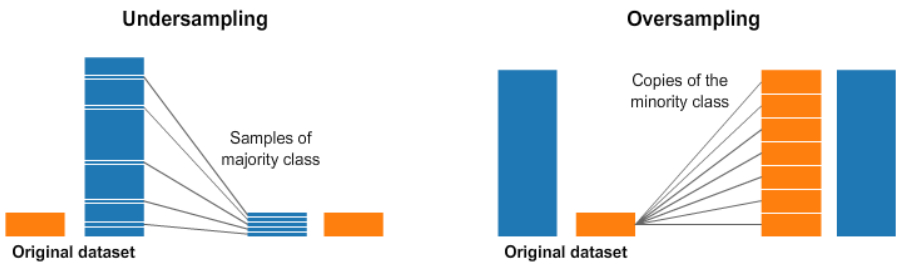


Figure 3.2: Visual representation of re-sampling operations [URL28]

3.1.3 Data Augmentation

Data augmentation is a very important part of the data prepossessing step. It is used to slightly change the existing training data or to add more changed copies. Due to its great effect on the learning ability and classification performance, data augmentation techniques are highly used for signal processing like biological signals which our case falls into.
Using data augmentation to slightly alter the composition of the ECG hear-beat signals in the training data makes the data more diverse and helps increase its size without too much repetition making the model more robust overall.
So, we apply various of these techniques to our training data by defining the following methods to slightly alter the ECG composition:

- **Adding Gaussian noise to the signal :** this function adds up to the original ECG signal a randomly generated noise signal to add relatively small fluctuations as follows :

$$output(t) = input(t) + noise(t) \tag{3.1}$$

Where noise(t) is an array of randomly generated float numbers in the range of $[-0.05, 0.05]$ having the same length as the input heartbeat signal. The figure below (Figure 3.3) shows a heartbeat signal with Gaussian noise.

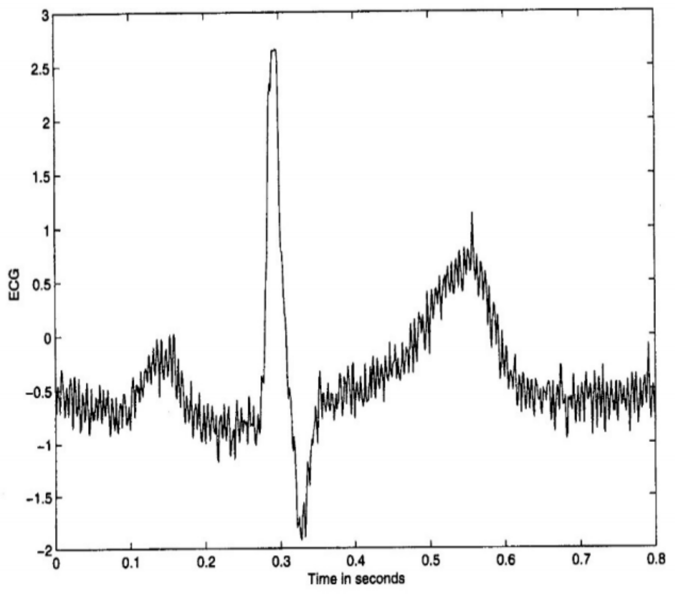


Figure 3.3: An ECG heartbeat with high frequency noise
[URL29]

- **Slightly stretching or compressing the length of the ECG beats :** in this case, we randomly increase or decrease the length of the heartbeat by a small margin. In the case of compression, we compensate the lost length by adding zeros padding, and in the case of stretching, we truncate the values that surpass the normal length. The new length is randomly generated as follows :

$$n' = \left\lfloor n \times (1 + \frac{\alpha - 0.5}{2}) \right\rfloor \tag{3.2}$$

Where n is the original number of samples in the heartbeat, n' is its new length and α is a randomly generated number in the interval $[0, 1]$.
After generating the new number of samples, we use the scipy function ***scipy.signal.resample(original-signal, new-sample-number)*** [URL30] to generate the new modified signal.
Figure 3.4 is a visual representation of the stretching operation applied on the ECG signal.

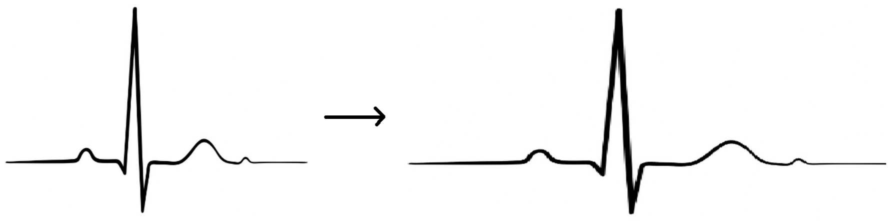


Figure 3.4: Heartbeat signal before and after stretching

- **Amplifying the heartbeat signal :** this function slightly increases the amplitude of the original ECG signal by multiplying it with a randomly generated factor as explained in equation (3.3) below.

$$output(t) = input(t) \times (-\alpha \times input(t) + (1 + \alpha)) \tag{3.3}$$

Here, *alpha* is a randomly generated float in the interval $[0, 1]$ and *input* and *output* are respectively the original ECG signal and the amplified signal after the operation. Figure 3.5 below presents the amplification of an ECG heartbeat.

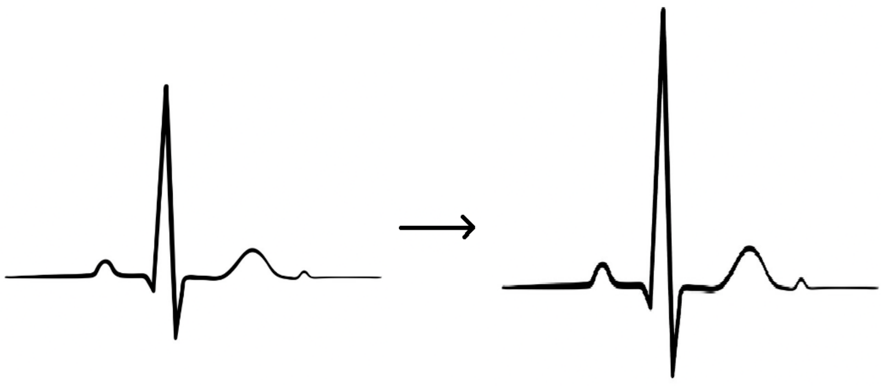


Figure 3.5: Heartbeat signal before and after amplification

The way we approach data augmentation is by applying randomly varying combinations of the three previously explained functions on the training data.
For each sample in the training set, we generate a random number in the $[0, 1]$ interval, and depending on its value, we either keep the signal unaltered, apply one of the three defined augmentation operations or apply a combination of two or three of them.

3.2 Model Architecture

Given the nature of our problem which consists of time series classification, we came to the realization that the best-suited type of architecture is CNNs. So, the first thing we did was trying some classic CNNs but they didn't yield satisfying results. The next thing step was to look at some research papers that addressed ECG diagnosis with DL algorithms. After some trial and error, we found a CNN model [Özal Yıldırım et al., 2018] that we tweaked to fit our needs. We ended up with the model shown in the figure below that gave the most promising results.

For this project, a 16-layer deep CNN was designed to classify the ECG signals. This deep network model assures an automatic classification of input chunks through an end-to-end structure with no need for manual intervention in specifying the features. As presented in the block diagram in Figure 3.6, the architecture of the model consists of a standard CNN layers with the first layer being a 1D convolution characterized with 128 weight vectors. The outputs of this layer are then normalized using a batch normalization layer. Following this process, the feature maps obtained from the previous layer are then introduced to a 1D max pooling layer which will generate a reduced feature map that will help us lower the computational cost. In the next layers, the operations of convolution and pooling are repeated until we reach the 14th layer, the flatten layer, which is responsible for transforming the multidimensional input feature vectors into one-dimensional output data. The features obtained from the flattened layer are passed to a fully connected layer with 512 units. In the last layer, a $softmax$ function will predict to which class the input data belongs. In addition to all of this, we added a dropout parameter to the dense layer. This aids in avoiding over-fitting issues during the training phase.

3.3 Model Training

After preparing the data for training and configuring the model architecture, the next step is training the CNN model as shown in the activity diagram in Figure 3.7.
First, the dataset is imported. Then, the data preprocessing previously detailed is applied to the training data making it ready to be passed through the model's multiple convolutional and pooling layers.
The model training is done in batches each containing a number of data samples specified by the admin. An epoch is a term used to refer to one cycle through the full training dataset being completed. Each epoch is comprised of a number of batches.
The first step of the training process is initializing the various weights of each layer, then the activation functions are calculated. After crossing the fully connected layer, the error is calculated and the weights are updated accordingly. This is repeated for each batch until the epoch is finished. At the end of every epoch, the model is evaluated with validation data and the current values of its weights are stored in a file if the validation accuracy is improved.
This process is repeated for a number of epochs specified by the admin. After completing all epochs, the model with the best accuracy can be loaded from the last saved file and then evaluated on the test data.

3.4 Evaluating the model

After completing the training process, the model is evaluated using the unseen data that was previously reserved for testing.
Two metrics were used to evaluate the CNN model, the accuracy and the loss.

3.4.1 Accuracy

A model's accuracy can be defined as the percentage of the correct predictions. In other words, it is the result of the correct predictions divided by all predictions.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{3.4}$$

3.4.2 Loss Function

The loss is the sum of the errors for every example on the training set and validation set. It measures how far a value estimated by the model is from the true value. Intuitively, we want to reduce the loss for better performance.
The loss function used in our model is **Categorical Cross-Entropy** given that it is

adapted for classification tasks. Also called Softmax Loss, it is a Softmax activation plus a Cross-Entropy loss.

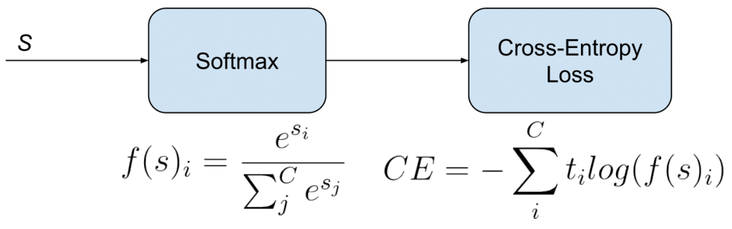

s
Softmax
Cross-Entropy Loss
$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$
$CE = -\sum_i^C t_i log(f(s)_i)$

Figure 3.8: Categorical cross-entropy

3.5 Conclusion

This chapter detailed the four main steps of the project starting with data pre-processing, then presenting the CNN model's architecture and its training process, and ending with its evaluation.

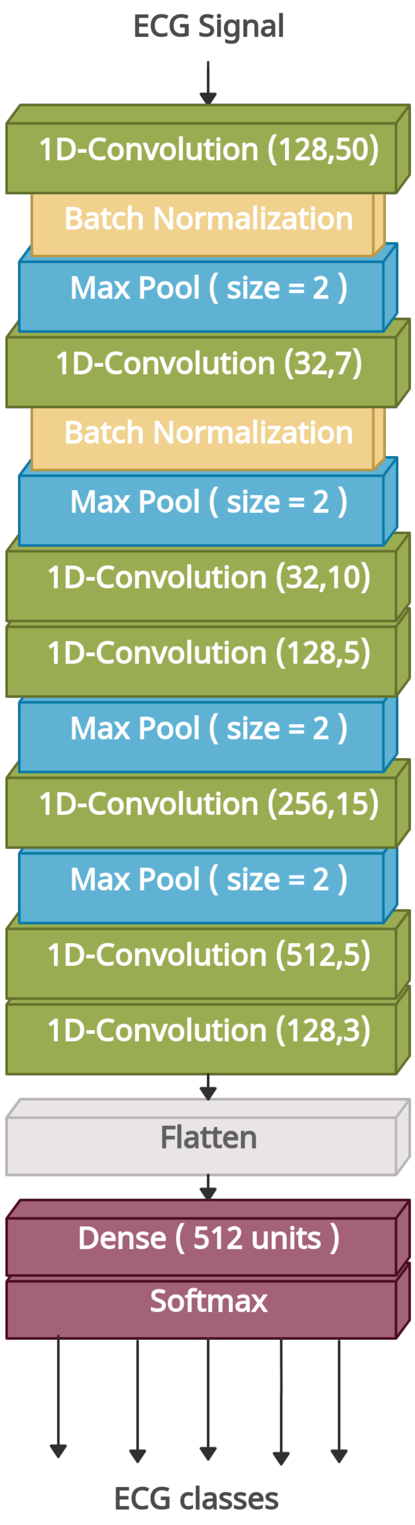


Figure 3.6: Block diagram of the proposed 16 layer CNN model

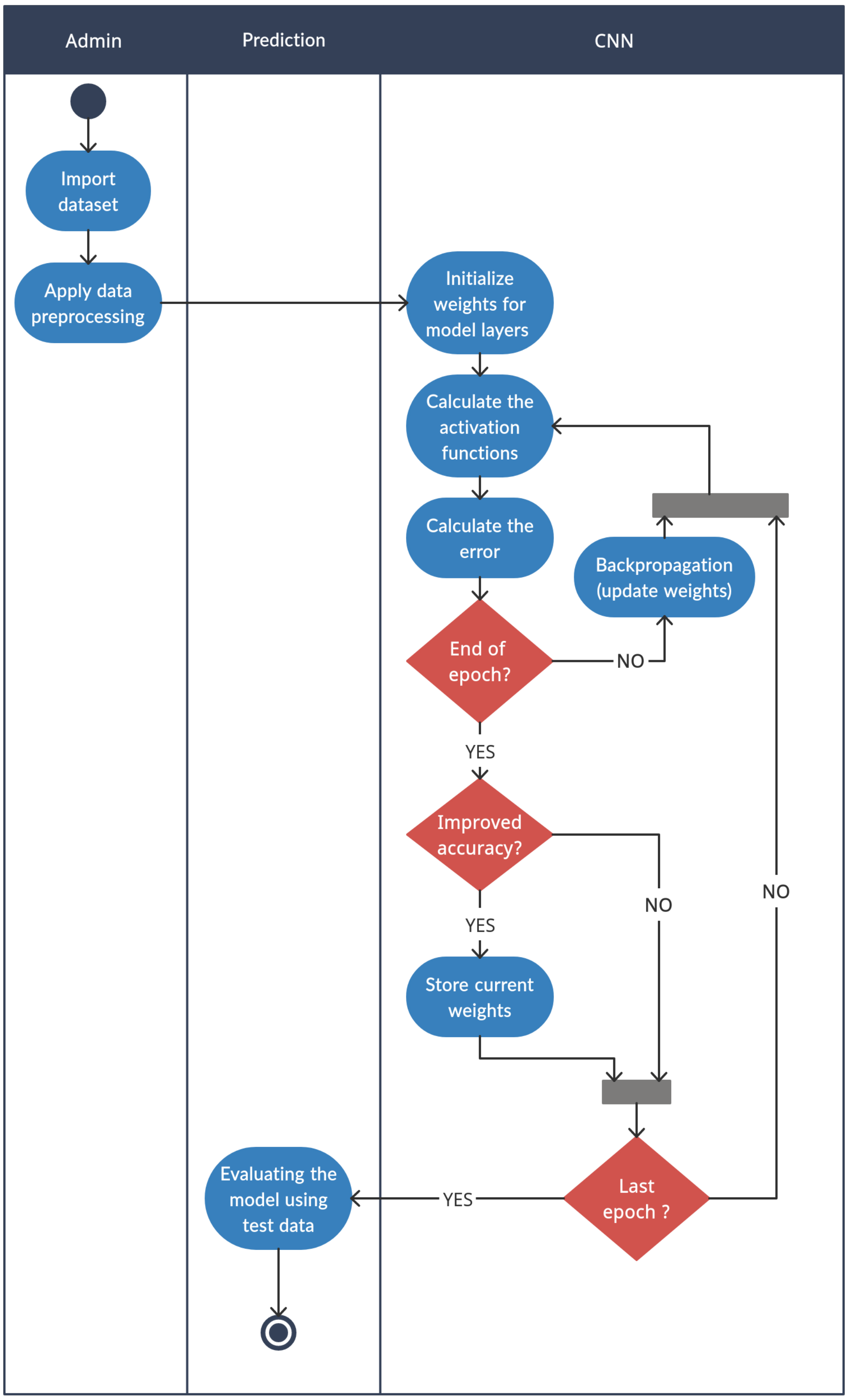


Figure 3.7: Activity diagram of the training phase

Chapter 4

Implementation & Test

In this chapter, we will showcase our working environment. Then, we will display the impact that prepossessing had on the morphology of the data and the re-partitioning of classes. In addition, we will present the evaluation metrics throughout the training, validation and testing phase. Lastly, we will mention some of the difficulties we encountered throughout the whole process.

4.1 Development Environment

In this part, we will be presenting the software environment that helped us complete this project.

Programming Language:

Python: it is a high-level interpreted programming language used mainly for ML, DL, and Data Science projects.
We opted for Python in our project given its array of features

Libraries:

- **Numpy :** a library used for the manipulation of matrices and arrays.
- **Pandas :** provides powerful tools for data manipulation and analysis.
- **Keras :** it's used mainly for neural networks and machine learning algorithms.
- **Scikit-learn (sklearn) :** a python library providing various machine learning tools.
- **SciPy :** a scientific library that uses numpy and brings more functionality.
- **Matplotlib :** mainly used for plotting and visualizing the data.

- **Seaborn :** a library based on matplotlib, it provides a high-quality interface for statistical data visualization.

Integrated Development Environment(IDE)

We used **Google Colaboratory** which is a free Google Research product that provides a cloud-based Jupyter notebook environment. We choose this environment seeing how it provides high GPU based training speeds for DL models and stores the notebook

Modeling tool

For drawing the various UML models, we used **Creately** which is an online modeling tool that provides an easy-to-use graphical interface for drawing various organizational charts and UML diagrams.

4.2 Environment Set-Up

Before starting anything, we need to set up the run-time environment. First of all, we open a new Google Colab session and we download the CSV files forming our dataset using the Kaggle API. Then, we upload a copy of the dataset files which are "mitbih-train.csv" and "mitbih-test.csv" to Google Drive for easier access in later sessions. Then we load these files into two Pandas DataFrame objects so that we can apply all the data-prepossessing procedures detailed in a previous section.

4.3 Data Pre-Processing

After loading the data from the CSV file, it is split into two parts in order to keep a 20% chunk for validation. And then, re-sampling and data augmentation are applied to the training data.

4.3.1 Data Re-sampling

The pie charts in Figure 4.4 visualizes the before and after effect of applying the re-sampling operation on the class re-partition. It shows how, at first, a big percentage of the data samples belong to the preponderant class, and after applying the re-sampling step, the data samples are equally partitioned between the various classes.

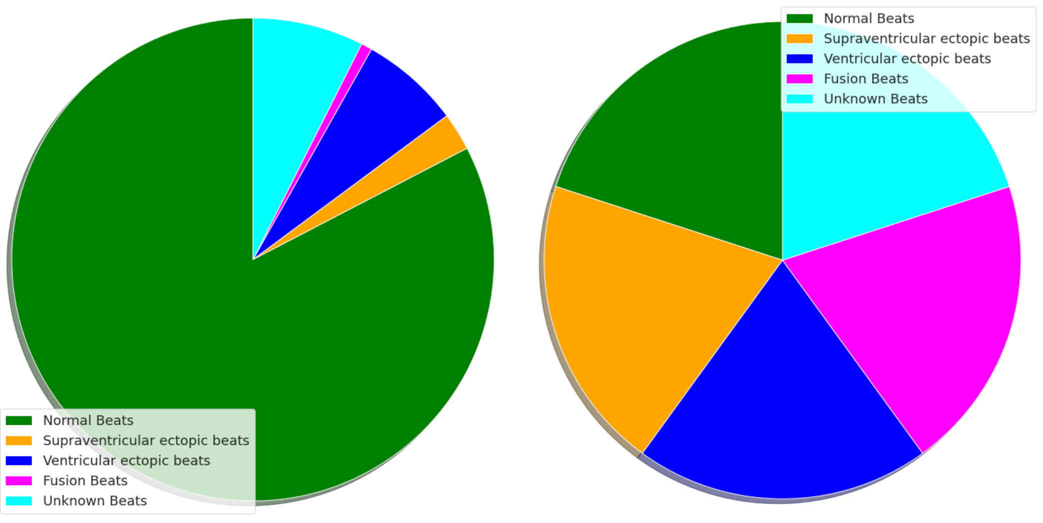


Figure 4.1: Classes partitioning before and after data re-sampling

4.3.2 Data Augmentation

Now having an equal subdivision of the training data samples between the five classes, we randomize the order of the samples and then apply a random combination of the three data augmentation operations we detailed in the previous chapter on each data sample. The three figures below showcase how the application of each data augmentation operation affects a heartbeat sample taken from the dataset.

Adding Gaussian Noise to the ECG Hear-beats

Figure 4.1 illustrates the application of Gaussian noise on one of the heartbeat samples from the data set.

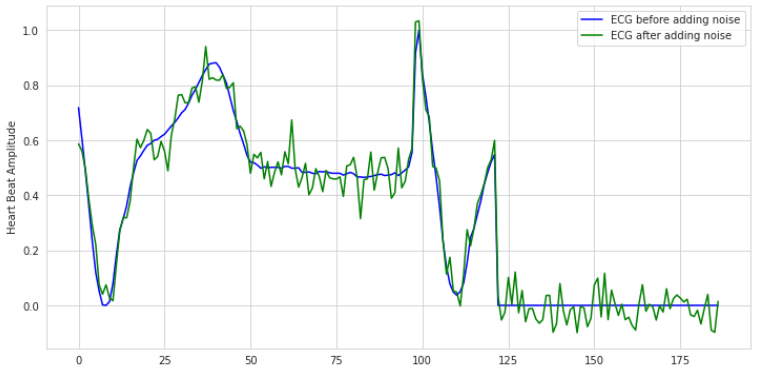


Figure 4.2: Heartbeat before and after adding Gaussian noise

Stretching or Compressing the ECG Heart-beats

Figure 4.2 shows the operation of stretching performed on a heartbeat signal extracted from the dataset.

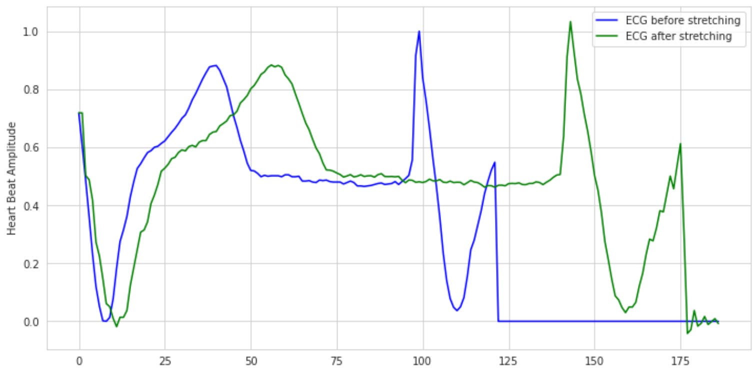


Figure 4.3: ECG signal before and after stretching

Amplifying the ECG Heart-beats

Figure 4.3 demonstrates the effect of the amplification operation on the ECG signal

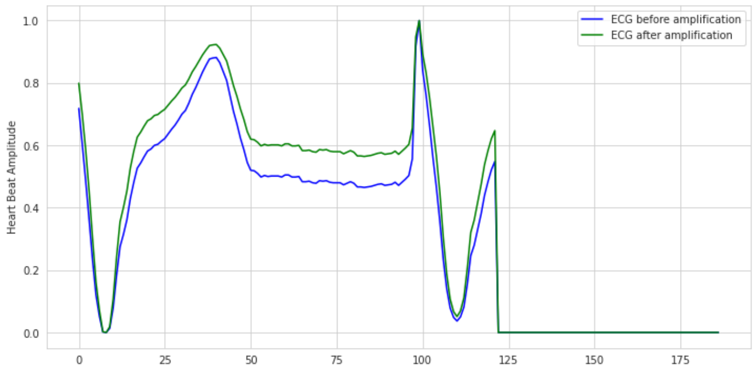


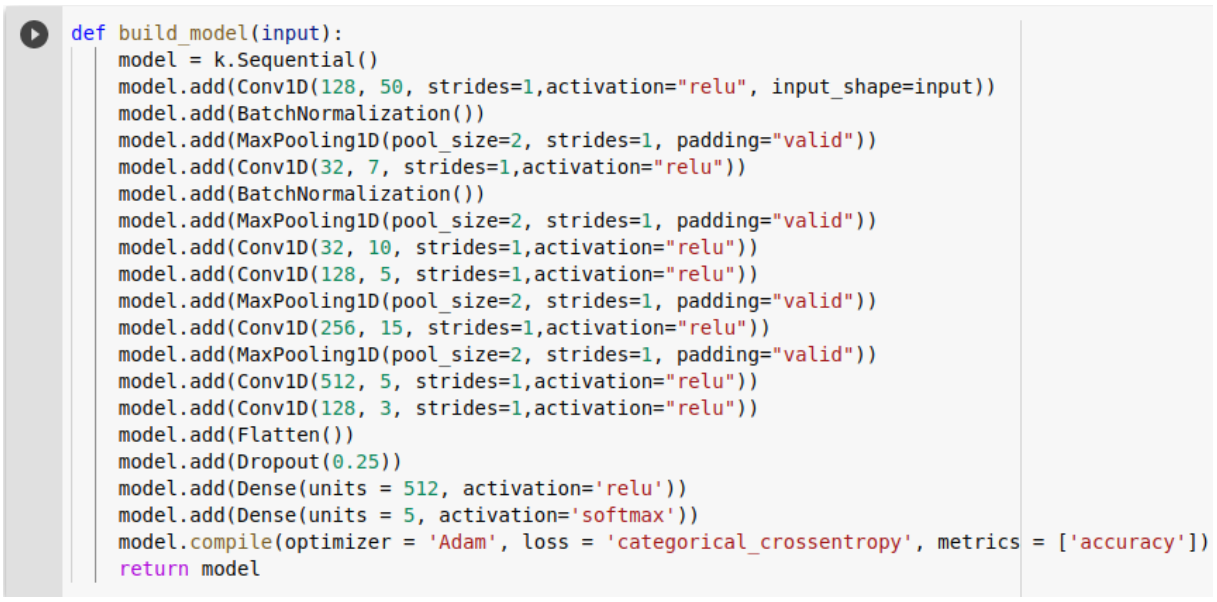
Figure 4.4: ECG signal before and after amplification

4.4 Model Configuration

This section will detail the steps that precede the model's training. In this project, we realized ECG classification using the Keras DL API for model implementation and the Google Colab for training, and Google Drive to store the dataset.

4.4.1 Model Definition

Having set up the environment and prepared data for training, we then implement the CNN architecture we detailed in the previous chapter. In the figure below is the function defined to implement the CNN model using the Keras API :

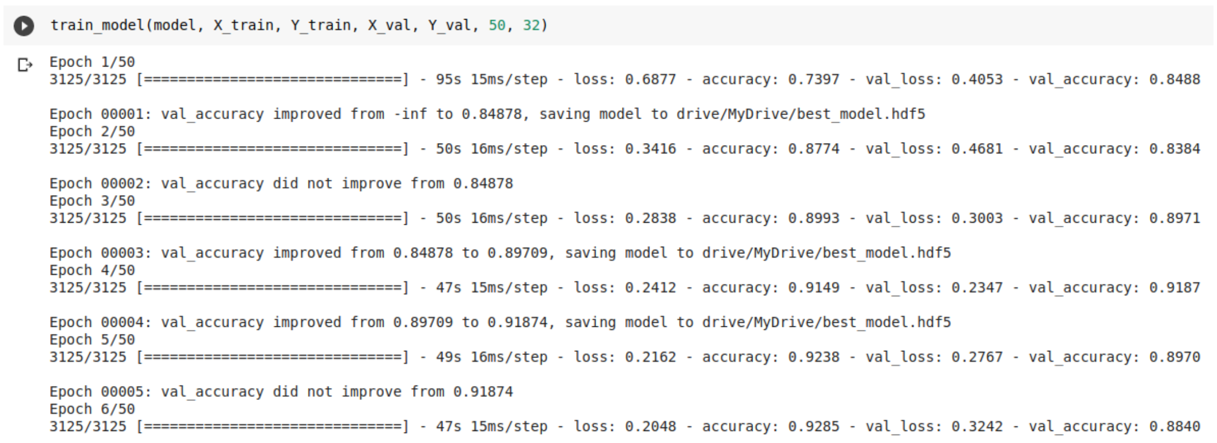


```python
def build_model(input):
    model = k.Sequential()
    model.add(Conv1D(128, 50, strides=1,activation="relu", input_shape=input))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2, strides=1, padding="valid"))
    model.add(Conv1D(32, 7, strides=1,activation="relu"))
    model.add(BatchNormalization())
    model.add(MaxPooling1D(pool_size=2, strides=1, padding="valid"))
    model.add(Conv1D(32, 10, strides=1,activation="relu"))
    model.add(Conv1D(128, 5, strides=1,activation="relu"))
    model.add(MaxPooling1D(pool_size=2, strides=1, padding="valid"))
    model.add(Conv1D(256, 15, strides=1,activation="relu"))
    model.add(MaxPooling1D(pool_size=2, strides=1, padding="valid"))
    model.add(Conv1D(512, 5, strides=1,activation="relu"))
    model.add(Conv1D(128, 3, strides=1,activation="relu"))
    model.add(Flatten())
    model.add(Dropout(0.25))
    model.add(Dense(units = 512, activation='relu'))
    model.add(Dense(units = 5, activation='softmax'))
    model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
    return model
```

Figure 4.5: Model implementation

4.4.2 Model Training

After defining the model we start the training phase bypassing the reprocessed data set into the CNN for a total of 50 epochs with 32 samples per batch as shown in Figure 4.6. After every epoch, the validation accuracy and loss are calculated and the current model is saved directly to the "best_model.hdf5" file in Google Drive if there is an improvement in the validation accuracy.



```
train_model(model, X_train, Y_train, X_val, Y_val, 50, 32)

Epoch 1/50
3125/3125 [==============================] - 95s 15ms/step - loss: 0.6877 - accuracy: 0.7397 - val_loss: 0.4053 - val_accuracy: 0.8488

Epoch 00001: val_accuracy improved from -inf to 0.84878, saving model to drive/MyDrive/best_model.hdf5
Epoch 2/50
3125/3125 [==============================] - 50s 16ms/step - loss: 0.3416 - accuracy: 0.8774 - val_loss: 0.4681 - val_accuracy: 0.8384

Epoch 00002: val_accuracy did not improve from 0.84878
Epoch 3/50
3125/3125 [==============================] - 50s 16ms/step - loss: 0.2838 - accuracy: 0.8993 - val_loss: 0.3003 - val_accuracy: 0.8971

Epoch 00003: val_accuracy improved from 0.84878 to 0.89709, saving model to drive/MyDrive/best_model.hdf5
Epoch 4/50
3125/3125 [==============================] - 47s 15ms/step - loss: 0.2412 - accuracy: 0.9149 - val_loss: 0.2347 - val_accuracy: 0.9187

Epoch 00004: val_accuracy improved from 0.89709 to 0.91874, saving model to drive/MyDrive/best_model.hdf5
Epoch 5/50
3125/3125 [==============================] - 49s 16ms/step - loss: 0.2162 - accuracy: 0.9238 - val_loss: 0.2767 - val_accuracy: 0.8970

Epoch 00005: val_accuracy did not improve from 0.91874
Epoch 6/50
3125/3125 [==============================] - 47s 15ms/step - loss: 0.2048 - accuracy: 0.9285 - val_loss: 0.3242 - val_accuracy: 0.8840
```

Figure 4.6: Model during training

When the training is done, we plot (see Figure 4.7 and Figure 4.8 below) the variation of both the training and validation accuracy and loss metrics in order to get a view of the way the model is improved during its training.
We can see that the training accuracy and loss values are steadily improving throughout the 50 epochs while the validation accuracy first rises quickly during the first epochs and then starts oscillation around the 0.93 marks reaching a global maximum of 0.95089. As for the loss function, it rapidly drops in values at first and then starts stays in the $[0.2, 0.3]$ range.
These two plots give us some insight on the model's ability to generalize what it has learned during the training phase using the validation data. They also show us the moment the model starts over-fitting and is no longer improving.

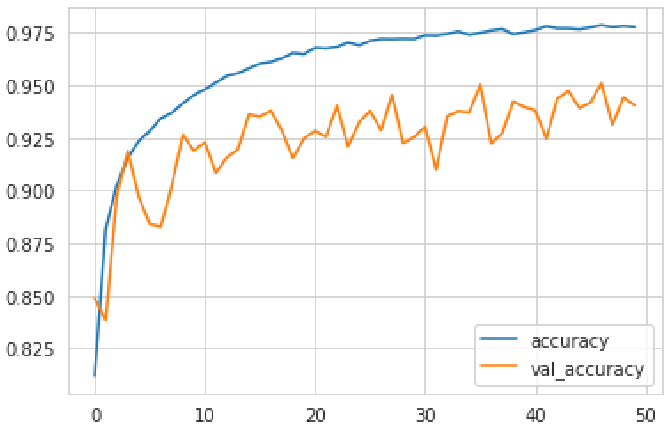


Figure 4.7: The variation of the training and validation accuracy during the 50 epochs of training

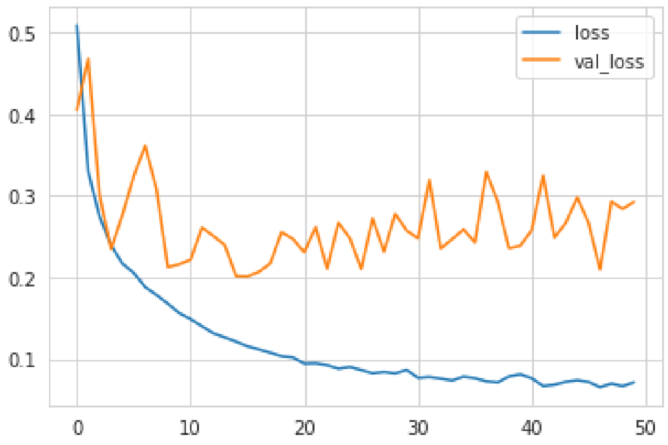
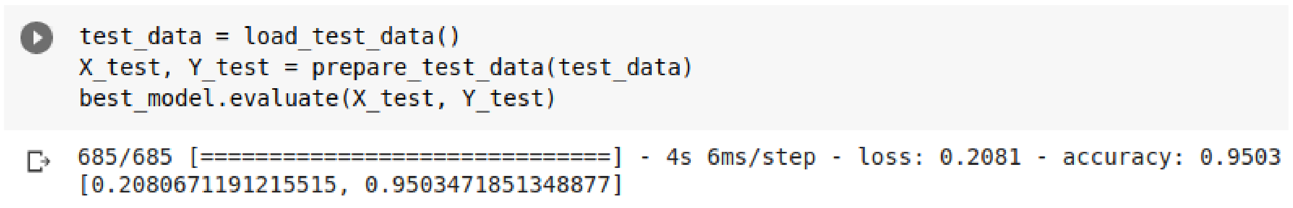


Figure 4.8: The variation of the training and validation loss during the 50 epochs of training

4.5 Model Performance

After the training is complete, we load the test data from the second CSV file, prepare it to fit the model's input layer, and then use it to evaluate the performance.
As shown in Figure 4.9, we obtained an accuracy of **0.95** which satisfies our initial goal of getting an accuracy higher than 0.9 and a **0.2** loss.



```
test_data = load_test_data()
X_test, Y_test = prepare_test_data(test_data)
best_model.evaluate(X_test, Y_test)
```

```
685/685 [==============================] - 4s 6ms/step - loss: 0.2081 - accuracy: 0.9503
[0.2080671191215515, 0.9503471851348877]
```

Figure 4.9: Evaluating the trained model

4.5.1 Confusion Matrix

The confusion matrix is a great performance measurement for ML algorithms in the case of classification problems. So, in order to get a better visualization of our model's performance, we plot the confusion matrix shown in Figure 4.10.
It is a NxN matrix where N is the number of target classes (in our case $N = 5$) where the rows represent the true labels and the columns represent the labels predicted by the model.
As we can see, most predictions are contained in the main diagonal of the confusion matrix which means that most of the samples in the test data were classified correctly.

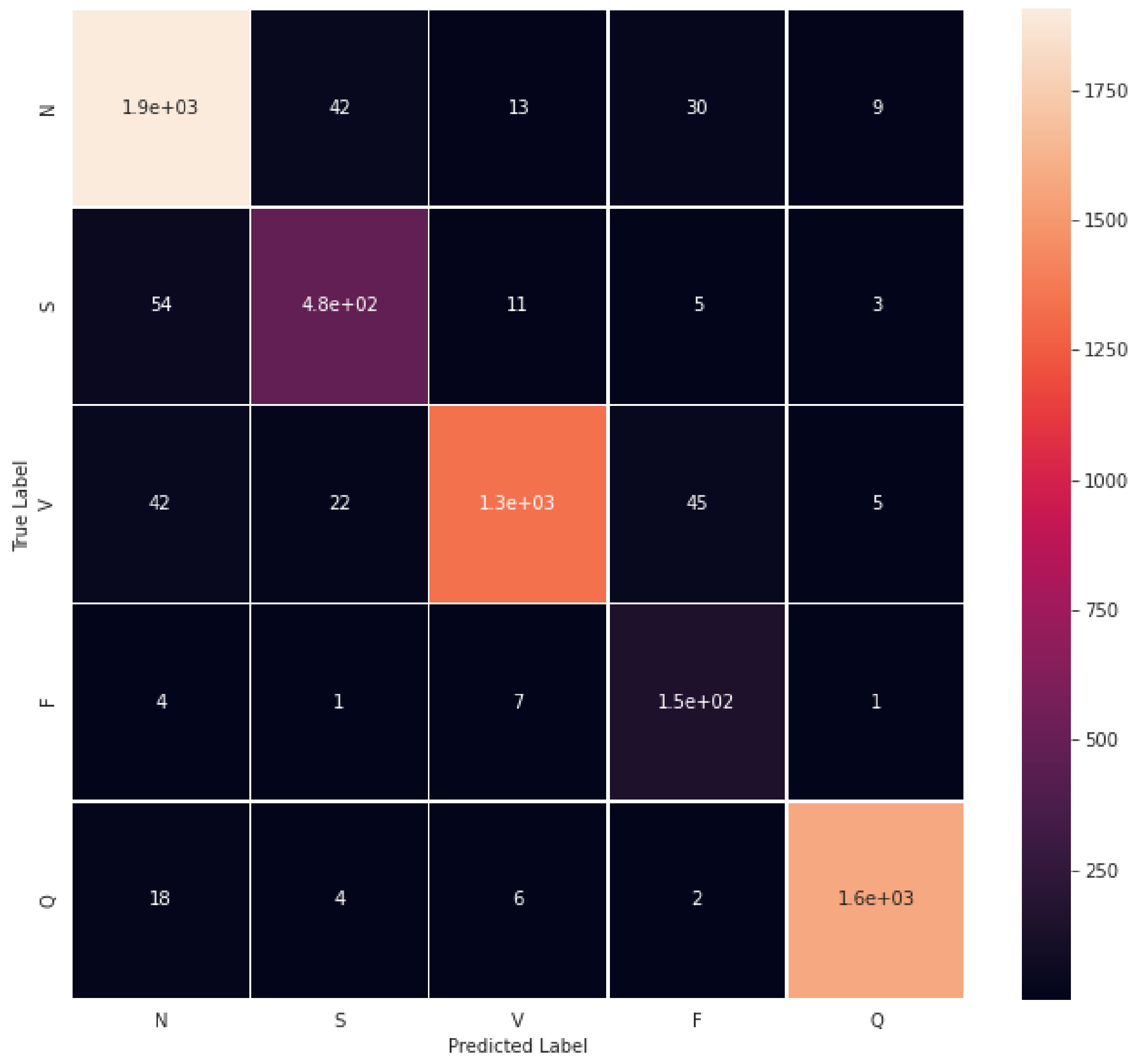


Figure 4.10: Confusion matrix

4.6 Encountered Challenges

As is the case with every ML project, we encountered numerous challenges that prevented us from achieving the metrics that we strove for in the analysis phase:

- The choice of the data set was one of the most challenging aspects of this project as we found many datasets most of them extracted from Physio-Net databases. These datasets required some heavy signal processing before being ready to use for classification which, with the time frame provided, was far from achievable. So, after some trial and error, we decided to use two CSV files belonging to the same database that requires less processing than the other variations.

- The CNN model configuration was also a little bit tricky as the model architecture. we took inspiration from what wasn't fully compatible with the dimensional shape of our data. Therefore, some tweaking in the layer's parameters was crucial in order to get the model running correctly.

4.7 Conclusion

During this chapter, we devoted the first part to presenting the development environment by indicating the programming language, the libraries, and the IDE used for this project. Then we presented the steps of the environment set-up. Next, we explained the implementation of the data pre-processing and the model configuration and we finished by showcasing the model's Performance.

Conclusion & Perspectives

The leading cause of deaths in today's world are cardiovascular diseases which made the medical community very determined in developing new techniques and trying different approaches to attenuate the number of deaths due to heart conditions.

In this context, the purpose of our work is to develop a system that assists the doctors in the ECG signal interpretation and diagnosis.

In this report, we started by defining some keywords and concepts and presenting the problem and the proposed solution. Next, we identified the actors, mentioned the functional and non-functional requirements, explained the communication between the system and the user via the use case diagram and the various sequence diagrams, showcased the internal behavior of the system with an activity diagram. Afterward, we explained in detail the steps we took in designing the project. Following the design phase, we presented the development environment, the model configuration and showed some of the performance metrics. Finally, we shed some light on the most challenging difficulties we faced when working on this project.

However, we can always develop and improve this model and enhance its performance and upgrade its functionalities for future projects by working on a larger data set and a deeper CNN which will enable us to increase the number of output classes with higher accuracy and a lower loss.

In conclusion, we would like to emphasize the importance of this experience on our technical and personal skills and we hope it reaches the satisfaction of the jury members.

Bibliography

[Kachuee and Fazeli, 2018] Kachuee, M. and Fazeli, S. (2018). Ecg heartbeat classification: A deep transferable representation. *arXiv.org*.

[Lilly, 2015] Lilly, L. (2015). *"Pathophysiology of heart disease: A collaborative project of medical students and faculty"*.

[Phung and Rhee, 2019] Phung, V. H. and Rhee, E. J. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Science*.

[Özal Yıldırım et al., 2018] Özal Yıldırım, Pławiak, P., Tan, R.-S., and Acharya, U. R. (2018). Arrhythmia detection using deep convolutional neural network with long duration ecg signals. *Computers in Biology and Medicine*.

Netography

[URL1] https://www.ibm.com/cloud/learn/what-is-artificial-intelligence, last revised April 5, 2021.

[URL2] https://www.educba.com/types-of-machine-learning/, last revised April 26, 2021.

[URL3] https://www.analyticsvidhya.com/blog/2021/01/a-quick-overview-of-regression-algorithms-in-machine-learning, last revised April 17, 2021.

[URL4] https://www.upgrad.com/blog/types-of-regression-models-in-machine-learning, last revised April 10, 2021.

[URL5] https://www.ibm.com/cloud/learn/deep-learning, last revised Mai 12, 2021.

[URL6] https://jintensivecare.biomedcentral.com/articles/10.1186/s40560-019-0393-1/, last revised Mai 9, 2021.

[URL7] https://towardsdatascience.com/what-is-an-artificial-neuron-and-why-does-it-need-an-activation-function-5b4c1e971d80, last revised Mai 11, 2021.

[URL8] https://www.researchgate.net/figure/The-structure-of-the-artificial-neuron_fig2_328733599, last revised Mai 11, 2021.

[URL9] https://datascience.aero/aviation-function-deep-learning/, last revised Mai 12, 2021.

[URL10] https://theappsolutions.com/blog/development/convolutional-neural-networks/, last revised Mai 13, 2021.

[URL11] https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac, last revised Mai 18, 2021.

[URL12] https://towardsdatascience.com/pytorch-basics-how-to-train-your-neural-net-intro-to-cnn-26a14c2ea29, last revised Mai 21, 2021.

[URL13] https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-global-max-pooling, last revised Mai 19, 2021.

[URL14] https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/1d-average-pooling, last revised Mai 23, 2021.

[URL15] https://www.researchgate.net/figure/Example-of-fully-connected-neural-network_fig2_331525817, last revised Mai 14, 2021.

[URL16] https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[URL17] https://www.mayoclinic.org/tests-procedures/ekg/about/pac-20384983, last revised Mai 16, 2021.

[URL18] http://thierry.col2.free.fr/restreint/exovideo_lycee/ex_seconde_physique/ch11_electrocardiogramme.htm, last revised Mai 16, 2021.

[URL19] https://ecgwaves.com/topic/ecg-normal-p-wave-qrs-complex-st-segment-t-wave-j-point/, last revised Mai 16, 2021.

[URL20] https://www.carolinaheartandleg.com/arrhythmia/arrhythmia-2/, last revised Mai 16, 2021.

[URL21] https://www.heart.org/en/health-topics/arrhythmia, last revised Mai 16, 2021.

[URL22] https://www.mountelizabeth.com.sg/healthplus/article/arrhythmia-guide, last revised Mai 16, 2021.

[URL23] https://www.sciencedirect.com/science/article/pii/S1877050918307774, last revised Mai 16, 2021.

[URL24] https://www.semanticscholar.org/paper/Cardiologist-Level-Arrhythmia-Detection-with-Neural-Rajpurkar-Hannun/b96de5f21c0449b23e7e256affc794f4f0404a5d, last revised Mai 16, 2021.

[URL25] https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml, last revised Mai 16, 2021.

[URL26] https://physionet.org/content/mitdb/1.0.0/, last revised Mai 16, 2021.

[URL27] https://www.kaggle.com/shayanfazeli/heartbeat, last revised Mai 16, 2021.

[URL28] `https://hersanyagci.medium.com/random-resampling-methods-for-imbalanced-data-with-imblearn-1fbba4a0e6d3`, last revised Mai 16, 2021.

[URL29] `http://www.jscholaronline.org/articles/JBER/Signal-Processing.pdf`, last revised Mai 16, 2021.

[URL30] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.resample.html`, last revised Mai 16, 2021.

Résumé — L'objectif de ce projet est de concevoir et de développer une solution intelligente pour aider les professionnels de la santé dans le diagnostic des arythmies cardiaques. Pour cela, nous avons utilisé un réseau neuronal convolutif pour effectuer la classification des électrocardiogrammes.
Mots clés : Apprentissage Profond, Classification des Electrocardiogrammes, Classification des l'Arythmie cardiaque, Réseau Neuronal Convolutif.

Abstract— The purpose of this project is to design and develop an intelligent solution for assisting the medical staff with diagnosing heart Arrhythmia.
For this, we used a convolutional neural network to classify Electrocardiogram signals.
Key words : Deep Learning, Electrocardiogram Classification, Arrhythmia Classification, Convolutional Neural Network.