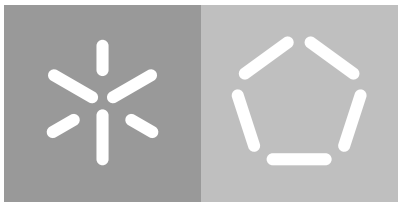


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Cláudia Vanessa Martins de Brito

Cloud-based Analytics for Monitoring and Classification of Arrhythmias

2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Cláudia Vanessa Martins de Brito

Cloud-based Analytics for Monitoring and Classification of Arrhythmias

Master dissertation
Master Degree in Medical Informatics

Dissertation supervised by
Professor Doutor António Luís Pinto Ferreira de Sousa

2018

Anexo 3

DECLARAÇÃO

Nome Cláudia Vanessa Martins de Brito

Endereço eletrónico: a71575@alunos.uminho.pt Telefone: 925422089 /

Número do Bilhete de Identidade: 14810064 3 ZY9

Título dissertação [X] /tese [ ]

Cloud-based Analytics for Monitoring and Classification of Arrhythmias

Orientador(es): António Luís Sousa

Ano de conclusão: 2018

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

Mestrado Integrado em Engenharia Biomédica – Ramo de Informática Médica

É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

Universidade do Minho, ___/___/_____

Assinatura: _____

ACKNOWLEDGEMENTS

First, I would like to thank Dr. Jorge Marques and Technician Ana Cabreiro for the warm welcoming and for taking their time to explain me the process of Electrocardiograms and Holters.

It is known people come and go, and accepting this approach to life has shown to increase happiness, however, I, for once, have been lucky enough to have found friends who have accepted to not let go even if life keeps us all apart. To those friends, I would like to say "Thank you" for coming along on this magnificent ride with me and for all the support given particularly during this academic journey.

I would also like to thank the HASLab family for their constant support and warm welcoming for the past few months, filled with morning coffees and good mood.

Without further ado, I would like to express my gratitude to Professor António Sousa, for accepting to make this project a reality, for all the insightful meetings over the course of the last year and for the most important part, helping me to keep my mental sanity.

Last but not least, I would like to thank my parents and brother for making the possibles and the impossibles to give me everything I ever needed and allowing me to pursue my dreams without losing the sense of home. Moreover, I would like to thank my godmother for all the support provided during this master thesis. Being the family the fundamental basis of existence, I'm thankful for all the support I received from my family even if it came by questions like "Is it done yet?". Everyone likes to receive this kind of boost.

In memoriam: Francisco Brito

"Happiness can be found, even in the darkest of times, if one only remembers to turn on the light".
J.K.Rowling

ABSTRACT

Real-time monitoring has become one of the most important and clinically relevant tasks in medical settings, yet one of the most repetitive and tiresome tasks is the analysis of 24-hour ECG records. One way to automate this long task is to convert this process into a real-time process with the automatic classification of the heart rate and with this, the classification of arrhythmias.

Thus, this master thesis focuses on the study of Deep Learning models for the classification of arrhythmias and data processing tools for the streaming and processing of data in real time. Consequently, this master's thesis comprises several phases. In the first place, a more theoretical part is presented which is the ground truth of the use of the tools later used for the development of the system. The development of the system includes a more practical part of data streaming composed by an IoT middleware, Apache Kafka as an intermediate agent between this middleware and Apache Spark, and ElasticSearch for real-time data storage for visualization. On the other hand, in the main scope of this thesis, two models of Deep Learning were created, one for the classification of arrhythmias and another one for their forecast.

The results obtained are promising with the arrhythmia classification yielding 98% accuracy in the classification of each beat in one of the four classes used. When the model was tested in data obtained directly from the Hospital of Braga, it was not possible to obtain such good results, however, the model after new training was able to obtain accuracy values of 81% for the testing dataset. This deep learning model was also tested with the integration of Apache Spark, in order to create data parallelism and increase the speed of the deep learning process, which tends to be very time consuming, without neglecting its performance.

The development of the model for the prediction of arrhythmias was done based on Long-Short Term Memory layers, in order to create a neural network with memory, the results obtained in the records with 30 minutes were not high. Despite the less good results in the first dataset, when the model was tested in the 24-hour records, the results obtained were quite high which demonstrated that the model can predict if it is based on a longer record. Nevertheless, these results were obtained individually because the Electrocardiograms can be an object of human identification.

Based on the results obtained it was possible to conclude that more tests should be done increasing the spectrum of arrhythmias to be classified so that this process becomes fully

automatic, without neglecting the precision of the results since human lives may depend on it.

RESUMO

A monitorização em tempo-real tem vindo a tornar-se numa das mais importantes e clinicamente relevantes tarefas em ambientes médicos, todavia, uma das tarefas mais repetitivas e cansativas é a análise dos registos de Eletrocardiogramas de 24 horas. Uma das formas de automatizar este longo processo é convertendo este processo num processo em tempo-real com a classificação automática do ritmo cardíaco e com isto a classsificação de arritmias.

Desta forma, o objetivo primordial desta tese de mestrado passa pelo estudo de modelos de Deep Learning para a classificação de arritmias assim como de ferramentas de processamento de dados para a transferência e processamento dos dados em tempo real. Consequentemente, esta tese de mestrado é composta por várias fases. Em primeiro lugar é apresentada uma parte mais teórica que fundamenta a utilização das ferramentas posteriormente usadas para o desenvolvimento do sistema. O desenvolvimento do sistema engloba uma parte mais prática de transmissão de dados composta por um middleware IoT, Apache Kafka como agente intermediário entre este middleware e o Apache Spark e ainda o ElasticSearch para armazenamento em tempo-real dos dados para visualização dos mesmos. Por outro lado, no âmbito principal desta tese, dois modelos de Deep Learning foram criados, um para a classificação de arritmias e outro para a previsão destas.

Os resultados conseguidos mostraram-se promissores, com a classificação de arritmias a obter resultados com precisão de 98% na classificação de cada batimento numa das quatro classes usadas. Quando o modelo foi testado em dados obtidos diretamente do Hospital de Braga não foi possível obter diretamente bons resultados contudo após novo treino conseguiu-se obter valores de precisão de 81% para o dataset de teste. Este modelo de deep learning foi também testado com a integração do Apache Spark por forma a criar o paralelismo de dados e aumentar a rapidez do processo de deep learning, que tendencialmente é muito moroso, sem descurar a performance do mesmo, ponto que foi conseguido.

Por outro lado, o desenvolvimento do modelo para a previsão de arritmias foi feito tendo por base camadas Long-Short Term Memory, de forma a criar uma rede neuronal com memória, os resultados obtidos nos registos com 30 minutos não foram muito altos ainda assim foi possível concluir que os Eletrocardiogramas podem ser a base de estudos mais pormenorizados no sentido de serem considerados biomarcadores. Apesar dos resultados menos bons no primeiro dataset, quando o modelo foi testado nos registos de 24 horas, os resultados obtidos foram bastante altos o que demonstrou que o modelo consegue prever caso tenha por base um registo mais longo. Não obstante, estes resultados foram

obtidos individualmente assentando no facto de que os Eletrocardiogramas são um objeto de identificação humano.

Com base nos resultados obtidos foi possível concluir que mais testes devem ser feitos tendo em conta o aumento do espetro de arritmias a serem classificadas de forma a que este processo se torne totalmente automático, sem desprezar a precisão dos resultados, uma vez que vidas humanas poderão depender disso.

CONTENTS

1 INTRODUCTION 2
1.1 Context and Motivation 2
1.2 Objectives 4
1.3 Contributions 4
1.4 Structure of the Document 5
2 STATE OF THE ART 6
2.1 Introduction 6
2.2 Machine Learning 7
2.2.1 Artificial Neural Networks 8
2.2.2 Deep Learning 10
2.2.3 Deep Networks Architectures 15
2.2.4 Deep Learning Applications 18
2.2.5 Deep Learning Frameworks 19
2.3 Distributed Data Processing and Streaming 24
2.3.1 Overview 24
2.3.2 Frameworks 25
2.3.3 Summary 34
2.4 Data Visualizer 35
2.4.1 ElasticSearch (ES) 35
2.4.2 Kibana 36
2.5 The Human Heart 37
2.5.1 Hearts' Anatomy 37
2.5.2 Electrocardiogram (ECG) 38
2.5.3 Arrhythmia 39
2.6 Related Work 41
2.6.1 Arrhythmias: feature extraction, detection and classification 42
2.6.2 Preprocessing of an Electrocardiogram 47
2.6.3 Cloud-based and IoT systems applied to Healthcare and Electrocardiograms 48
2.7 Personal Data Protection in Healthcare 50
2.8 Summary 51
3 RESEARCH METHODOLOGY 52
3.1 Introduction 52
3.2 Design Science Research Methodology - DSRM 52

3.3 Practical Application 54
4 SYSTEM ARCHITECTURE 55
4.1 Introduction 55
4.2 Proposed Approach 56
4.3 System Architecture 57
4.3.1 Retrieval and Storage Architecture 58
4.3.2 Deep Learning Architecture 58
4.3.3 Analysis and Visualization Architecture 58
5 DEVELOPMENT AND IMPLEMENTATION 59
5.1 Decisions 59
5.1.1 Deep Learning 59
5.1.2 Data Streaming and Visualization 63
5.2 Implementation 63
5.2.1 IoT Platform 63
5.2.2 Deep Learning 63
5.2.3 Data Streaming and Visualization 65
5.3 Summary 67
6 RESULTS 68
6.1 Experiment setup 68
6.2 Deep Learning 69
6.2.1 Dataset 69
6.2.2 Evaluation Metrics 71
6.2.3 Training Methodology - Classification 72
6.2.4 Testing Methodology - Classification 73
6.2.5 Training and Testing Methodology - Prediction 73
6.3 Results and Discussion 74
6.3.1 Training Results - Classification 74
6.3.2 Testing Results - Classification 77
6.3.3 Training and Testing Results - Prediction 79
7 CONCLUSION 81
7.1 Conclusions 81
7.2 Prospect for future work 82

A SUPPORT MATERIAL 100
A.1 Detection of the QRS complex 100
A.2 Evaluation Metrics Results 101
A.2.1 Keras without Apache Spark 102
A.2.2 Keras with Apache Spark 104

A.2.3 Test Dataset Results 107
A.3 Apache Spark Jobs 109

LIST OF FIGURES

Figure 1 Handwritten digits sample. 8
Figure 2 Comparison between a biological neuron and an artificial neural network. 10
Figure 3 Relationship between Deep Learning and Artificial Intelligence. 11
Figure 4 Comparison between a Neural Network and a Deep Neural Network. 11
Figure 6 Example of a neuron. X1, X2 and X3 represent the inputs; W1, W2, W3 represent the weights, b represent the biases; a is the output. 14
Figure 7 The convolution operation. Obtained from [1] 15
Figure 8 Schematic of the Long Short-Term Memory block. Obtained from [1]. 17
Figure 9 Anatomy of a Kafka Topic with Producers and Consumers. 25
Figure 10 Example of how Kafka Connect is applied. 27
Figure 11 Schematic of a MapReduce job. 28
Figure 12 Storm topology example 30
Figure 13 Samza architecture 30
Figure 14 The Apache Spark architecture platform 31
Figure 15 The Apache Flink architecture 33
Figure 16 Structure of the heart. Obtained from [2] 37
Figure 17 Ideal *Electrocardiogram (ECG)* with key features; (a) Schematic representation of normal sinus rhythm; (b) Electrocardiogram features, how each feature is generated [3]. 39
Figure 18 Premature Contractions present in an ECG; (a) Atrial Premature Contraction (lead I); (b) Supraventricular Premature Contraction (lead III); (c) Premature ventricular Contraction (lead I and II). Obtained from Guyton and Hall [2]. 42
Figure 19 Proposed prototype architecture represented by three blocks. The first block represents the data sources, the acquisition of the biosignals. Secondly, the middle block represents the *Internet of Things (IoT)* gateway and platform as well as the cloud storage. The third and last block represents the analytic stage and the dashboard visualization of the obtained results. 56

Figure 20 System architecture for data collection, processing, analytics and monitoring. It comprises the Kaa IoT, HBase, Kafka, Spark, a Deep Learning Stack and ElasticSearch and Kibana for the last block. 58
Figure 21 Network Structure of the model for the classification task. Overall the network contains 12 layers of convolution followed by a fully-connected layer and a softmax. 64
Figure 22 Network Structure of the model for the prediction task based on LSTMs. 66
Figure 23 Visualization of one Electrocardiogram after passing through the IoT platform. As seen, this electrocardiogram has not suffer the filtration it needs and therefore it is a noisy record. 67
Figure 24 Environmental setup. This setup comprises six machines, five of them run Ubuntu 16.04. The Kaa Cluster has three machines with 2 CPU cores, 4 GB RAM, 20 GB Memory. The visualization machine has 4 CPU cores, 120 GB of disk space and 8GB of RAM memory. The deep learning machine has 250GB of disk space, 20 CPU cores and 32GB of RAM memory. The other processes were performed on a machine running Fedora 28, with 4 CPU cores, 100GB of disk space and 8GB of RAM memory. 69
Figure 25 Splitting the dataset. The splitting is done in two steps: first, 20% for test subset and, from the 80% left, 65% is reserved for training subset and 35% for validation. 70
Figure 26 Comparison between a before and after filtering of one beat from file 101 of MIT-BIH Database Arrhythmia. 71
Figure 27 Training and testing results regarding the accuracy of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Train Results on model without Apache Spark integration; b) Test Results on model without Apache Spark integration. 74
Figure 28 Testing results regarding the F1-score of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Test Results on model without Apache Spark integration using Stochastic Gradient Descent for each class; b) Test Results on model without Apache Spark integration using Adaptive Moment Estimation for each class. 75
Figure 29 Training results regarding the accuracy of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Train Results on model with Apache Spark integration; b) Test Results on model with Apache Spark integration. 75

Figure 30 Testing results regarding the F1-score of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Test Results on model with Apache Spark integration using Stochastic Gradient Descent for each class; b) Test Results on model without Apache Spark integration using Adaptive Moment Estimation for each class. 75
Figure 31 Training results regarding the elapsed time of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Test Results on model without Apache Spark integration; b) Test Results on model with Apache Spark integration. 76
Figure 32 Testing results regarding the classification of arrhythmias. This graphic representation presents the accuracy of the classification for each record. 77
Figure 33 Training and testing results regarding the forecasting of arrhythmias. This graphic representation presents the accuracy of the forecasting for each record on the MIT-BIH Arrhythmia Database. 79
Figure 34 Training and testing results regarding the forecasting of arrhythmias. This graphic representation presents the accuracy of the forecasting for each record on the collected records from the Hospital of Braga. 80
Figure 35 Apache Spark Job with four workers running locally. 109

LIST OF TABLES

Table 1 Comparison of Deep Learning Frameworks based on Licenses, CPU and GPU Support, Multi-Language Support, Supported Neural Networks and Visualization Integration. 24

Table 2 Comparison of Data Streaming Frameworks based on Streaming Model, Guarantees, Fault Tolerance Mechanism, Latency, Throughput and Maturity. 34

Table 3 Summary of number of beats of each class. 70

Table 4 Summary of number of beats of each class for the new train dataset. 78

Table 5 Results obtained for the new dataset created based on 11 records from the hospital for training and the five worst records from the hospital for testing. 78

Table 6 Representation of the effectiveness of each approach to detect the R-peak in the electrocardiogram signals. 100

Table 7 Confusion Matrix for each number of epochs tested on the model without Apache Spark integration and with SGD and Adam as optimizers. 102

Table 8 Precision results obtained from the tests of the deep learning model without Apache Spark integration regarding each class and with SGD and Adam as optimizers. 103

Table 9 Recall results obtained from the tests of the deep learning model without Apache Spark integration regarding each class and with SGD and Adam as optimizers. 103

Table 10 F1-Score results obtained from the tests of the deep learning model without Apache Spark integration regarding each class and with SGD and Adam as optimizers. 103

Table 11 Confusion Matrix for each number of epochs tested on the model with Apache Spark integration and with SGD as optimizer. 104

Table 12 Confusion Matrix for each number of epochs tested on the model with Apache Spark integration and with Adam as optimizer. 105

Table 13 Precision results obtained from the tests of the deep learning model with Apache Spark integration regarding each class and with SGD and Adam as optimizers. 105

Table 14 Recall results obtained from the tests of the deep learning model with Apache Spark integration regarding each class and with SGD and Adam as optimizers. 106
Table 15 F1-Score results obtained from the tests of the deep learning model with Apache Spark integration regarding each class and with SGD and Adam as optimizers. 106
Table 16 Confusion Matrix from the 8 records obtained from the Hospital of Braga and classified with the best model obtained previously. 107
Table 17 Precision from the 8 records above-mentioned obtained from the Hospital of Braga and classified with the best model obtained previously for each class. 108
Table 18 Recall from the 8 records above-mentioned obtained from the Hospital of Braga and classified with the best model obtained previously for each class. 108
Table 19 F1-Score from the 8 records above-mentioned obtained from the Hospital of Braga and classified with the best model obtained previously for each class. 108

LIST OF ACRONYMS

A

ADAM Adaptive Moment Estimation

AI Artificial Intelligence

ANNS Artificial Neural Networks

C

CNNS Convolutional Neural Networks

CS Computer Science

D

DL Deep Learning

DNNS Deep Neural Networks

DSRM Design Science Research Methodology

E

ECG Electrocardiogram

ES ElasticSearch

I

IOT Internet of Things

IT Information Technologies

L

LSTM Long Short-Term Memory

M

ML Machine Learning

R

RMSPROP Root Mean Square Propagation

RNNS Recurrent Neural Networks

S

SGD Stochastic Gradient Descent

1

INTRODUCTION

As the world grows in technology and innovative approaches, the human population becomes increasingly more pressured by its surroundings. The pressure felt by humans is observed by the also increased health problems. The stress promotes bad habits, bad habits promote health problems, this cycle is recreated on a large scale around the world. Known as the main organ, the Heart suffers from all these circumstances leading to even more problems. The Cardiovascular Diseases are known as one of the deadliest conditions and should be carefully addressed and treated. Focusing in this thematic tapering it to arrhythmic heart beats, this master thesis further reveals some of the depths of learning with past cases to analyze bio-signals that detect arrhythmias in real-time.

The above Section, Section 1.1, presents a brief contextualization of the overall thematic and a motivation for the development of this project. Followed by an explanation of the main goals. Section 1.3 exposes the contributions this project had this past year. The last Section, 1.4, describes the document's structure.

1.1 CONTEXT AND MOTIVATION

The amount of information created and ingested in a medical environment goes beyond the ability of doctors to evaluate all the situations of their patients. In such scenario, it is necessary to create a way to automate some processes and some diagnoses in all areas. Nevertheless, Cardiology has brought the attention of the public eye since it is known that sudden cardiac arrest and other major cardiac conditions may lead to death. In 2017, Cardiovascular Diseases caused one in three deaths in the United States of America and Coronary Heart Diseases account for most deaths [4]. In Portugal, sudden cardiac arrest represents 20% of deaths in a year, showing a higher death rate than HIV, Breast Cancer and Pulmonary Cancer all summed up [5].

For this matter, it is crucial to be on top of every bio-signal that may induce an early diagnosis of major heart conditions that may lead to death. Due to the measuring of the electrical activity of the heart, Electrocardiograms (ECG or EKG) help in the diagnosis of the

abnormal heart rhythms, such rhythms may lead to Ventricular Fibrillation and thereafter cardiac arrests [2, 6]. However, early diagnosis of this condition may prevent this outcome.

Arrhythmias are the most common diagnoses in this medical area and are composed of electrical changes that cause the normal heart rhythm to change. These changes can cause the heart to beat faster (tachycardia), slower (bradycardia), or at an irregular beat. Arrhythmias are widely classified, and their classification may depend on the factors described above and on the place where they occur, ventricles or atria. These changes can even lead to sudden death from a stroke or cause other types of damage because of the inability of the heart to pump enough blood into the body and consequently cause damage to the brain, heart, or other organs [2, 6].

The causes of these changes are diverse and should be studied individually for each patient. From excessive consumption of alcohol, heart disease or diabetes, obesity and sleep apnea, all these factors can influence the emergence of arrhythmias and therefore, the heart rhythm should be properly monitored.

To this extent, the importance of monitoring systems increases with the extra goal of improving patient care and the speed with which such care is provided. One objective of these systems should also contemplate the prediction of arrhythmic episodes.

One way to monitor these patients is using sensors. These sensors usually have a small memory for signal processing and the microprocessors used have a limited processing capacity, the power of these sensors is also questioned because their durability is quite limited. The concepts of IoT and Cloud Computing seem to be able to add a significant contribution to solving some problems identified in this area of medical diagnosis. The sensors are connected to an IoT agent that can collect all the sensor data and send them to the Cloud for processing and storage.

After storing the data and creating the forecasting system, it is necessary to apply two concepts, *Artificial Intelligence (AI)* and *Machine Learning (ML)*. These two concepts should be applied to the data in order to analyze and pass the raw data to useful information to doctors and nurses. This information is important since it allows these caregivers to be alerted to abnormal values detected or to the timely probability (prediction) of the occurrence of an arrhythmic episode. This system may allow a better functioning of the organization and the storage of the data.

This project aims to solve the main issue discussed above. Going from the process of analyzing an ECG to the detection of arrhythmic episodes, it is intended to create a platform which can store and, using *Deep Learning (DL)*, analyze these records. Not only the computational analysis matter but also the visualization of these results may offer a bigger overview of the problem detected in the patient.

1.2 OBJECTIVES

This project has the goal of dealing with a major condition in the world population. Without focusing on a specific population, it is intended to deal with all backgrounds reaching the same purpose. These assumptions are part of the main aim of this master thesis, dealing with all types of arrhythmias and being able to process it in real-time, giving the doctors a real-time feedback of which type of arrhythmia is the patient suffering in that moment and further, the longtime objective is to be able to predict these arrhythmias and prevent sudden cardiac arrest originated by arrhythmias.

With this in mind, the objectives of this project comprise:

- Assess the adoption of IoT systems and deep learning models for healthcare environments with emphasis in ECG records:
  - Collect and analyze bibliographic documents about IoT systems dedicated to ECG monitoring;
  - Assessment of open-source frameworks for real-time data processing and streaming;
  - Assessment of the adoption of Machine Learning techniques for classification and/or prediction of arrhythmias;
  - Assessment of security measures in a healthcare environment;
  - Assessment of visualization tools for real-time indexing and visualization.
- Design and implement a system architecture for the overall system:
  - Design of a proposed architecture based on frameworks suitable for medical sensors and general medical environment;
  - Implementation of the designed architecture.
- Evaluate the performance of the developed system:
  - Performance tests execution;
  - Evaluation of the results in comparison with related works.

1.3 CONTRIBUTIONS

The project developed allowed to continue an underdevelopment project originally created by Rei [7]. With this in mind, the paper entitled "Assessment of an IoT Platform for Data Collection and Analysis for Medical Sensors" was accepted for publication and presentation on the 1st International Workshop on Technology Convergence for Smart Cities

(TeC4C 2018) in Philadelphia, Pennsylvania, USA. This conference is part of the Workshop of CIC 2018 IEEE 4th International Conference on Collaboration and Internet Computing. This project was also the basis for the paper "Applied deep learning for detection and classification of arrhythmias in an Healthcare IoT-based environment" which comprised the initial deep learning statements made, it was rejected when submitted for the Machine Learning for Healthcare 2018, an annual conference held on the Stanford University School of Medicine. Furthermore, two other papers are being prepared to be submitted to two different Conferences. The paper entitled "Electrocardiogram Beat-Classification based on a ResNet Network" will be submitted to The 17th World Congress of Medical and Health Informatics. The last article, "Assessment of Deep Learning Models with Apache Spark and Horovod" will also be submitted to The Thirtieth IEEE/CVF Conference on Computer Vision and Pattern Recognition.

1.4 Structure of the Document

This work is structured into 7 chapters, comprising this chapter as the first, in which the reader understands the context of the study and the motivations behind it as well as the proposed objectives. Adjacently, a theoretical chapter introduces fundamental information about concepts and technologies further used as a base in this work. This is a chapter that can be seen as a broad summary regarding Artificial Intelligence, especially Deep Learning, and open-source frameworks for data processing and streaming, therefore, is an important chapter for readers without any medical, AI or data processing background. Chapter 3 is depicted the research methodology adopted and in Chapter 4 the proposed solution and system architecture, from which the system was based, is exposed. Chapter 5 concerns the decisions made during the development of this project and its implementation processes for the deployment of the several steps. Chapter 6 portrays the evaluation metrics used for the assessment of the system while depicting the results and their respective discussion. At last, Chapter 7 ends with some conclusion notes and prospects for future work.

2

STATE OF THE ART

This chapter will focus on the theoretical foundations and scientific concepts concerning the development of this master thesis. Section 2.1 presents a brief introduction containing the major subjects of this chapter. Secondly, Section 2.2 presents the concept of ML and DL, exhibiting some of its challenges, applications and major architectures. The following section, Section 2.3, focus on the available frameworks to stream and process the data that is streamed initially from health sensors. In order to perform the visualization, Section 2.4 introduces some concepts about ElasticSearch and Kibana. Section 2.5 focus on the biological features of this master thesis, the Human Heart, its anatomy, the irregular heart rhythms and also the tests performed to diagnose these abnormal rhythms. Next, Section 2.6 presents a literature review of related works focused in classification and monitoring of Arrhythmias using ML and cloud-based systems for monitoring of electrocardiograms. Finally, Section 2.7 has a brief description of data protection in healthcare according to the guidelines applied by the European Union and the United States of America and the type of protection a cloud-based platform should have.

2.1 INTRODUCTION

As proposed initially, this master thesis will focus on data collected from health sensors which should become available as soon as is computationally possible. For this chapter, it is important to focus in several fields such as Data Science, Machine Learning and Deep Learning, Distributed Streaming and Processing Systems and evidentially it is essential to understand basic concepts of the biological organ that is being analyzed. Data Science has been one of the most talked topics in the later years once there was enough data to be analyzed, to find patterns or anomalies in it has become a daily basis of many scientists and engineers. Going further with this topic, big data[1] has become the major foundation of many fields and relying on it, data science has found unknown patterns and correlations, overpassing former results in this area.

[1] Large amounts of data which are defined by the 3 V's, volume, velocity and variety [8, 9].

On the other side, real-time monitoring is essential for fast action-reaction and in the medical field, it becomes vital to doact and react fast and have an outstanding performance since human lives depend on it.

With this in mind, it is necessary to understand several concepts and also the behavior of certain frameworks when dealing with real data. Going from ML to visualization of data, this Chapter explains major DL concepts, with a brief introduction to some available-frameworks and libraries. Without losing the focus on the subject, also the Human Heart is concisely explained, as is the ECG and the main arrhythmias. Three types of arrhythmias are further defined since they are targeted on the development of the deep learning stage of this master thesis.

With several stages, it became essential to understand what has been done when dealing with data from ECG's. Hence, after the definition of several concepts from DL and arrhythmias, a literature review focusing on feature extraction, pre-processing of ECG's and detection and classification of arrhythmias is done.

On the other side of the development of this master thesis is the data processing and streaming, the availability of frameworks to process and stream huge amounts of data is increasing. Further ahead, some data processing and streaming frameworks are expounded to comprehend somedecisions made when developing this project, emphasizing the advantages of each framework but also not neglecting its disadvantages.

Later, data visualization becomes the focus of the project, *ElasticSearch (ES)* and Kibana are the focal points of this subsection. The project concludes with a final visualization of the real-time streaming of the ECG data and the classification of each beat.

2.2 MACHINE LEARNING

The ideology of AI raised by 1950s as Alan Turing hurled the ideathat still haunts researchers nowadays [10]:

"Can machines think?"

Despite reaching further with this question, Alan Turing proposed that they understood this question as a simple "Imitation Game" where a fine line was made between an enigma for humans or an enigma for machines [10]. This question led to further investigations, however, AI did not rise easily since there was only theoretical evidence and there was not enough computational power.

AI was later defined by one of its godfathers as "the science and engineering of making intelligent machines... but AI does not confine itself to methods that are biologically observable." [11].

In the last few decades, making a machine learn some basic steps has become clearer and almost every day an algorithm is created or improved. Although some basic concepts of ML can be easily introduced, machine learning is built above some mathematic concepts which need intense knowledge such as multivariate probability densities or even the partial derivatives. ML can be described as a way for machines to learn and adapt according to its environment. Thus, ML, as a subfield of AI, was defined, by Arthur Samuel, as the field that "gives computers the ability to learn without being explicitly programmed" [12].

The learning algorithms go from classification to prediction, learning from examples and being tested with even more examples and later tested with unseen samples to see true results. All the algorithms used are highly sophisticated, however, machines still do not learn as humans. Decades of development had passed and more will pass until there is an exact remake of a human brain moreover because even the human brain is not yet fully understood [1].

As technology developed, machine learning has been able to use the computational power for its purposes and even further,machine learning has now the power to create its own artificial neural networks and do more computational demanding tasks.

2.2.1 *Artificial Neural Networks*

The first artificial neuron appeared in 1943 by Dr. Warren McCulloch and the logician Walter Pits [13]. *Artificial Neural Networks (ANNs)* arose as a machine learning technique much later in the mid-1980s and early 1990s [14] and were slowly adopted once there was not the amount of data needed neither the time for studying it deeply, their fundamental unit is a node.

Based on the biological neuron of the mammalian brain, ANNs develop over time, with training [1, 15]. This can be noticed when looking to a sequence of handwritten digits like the one in the Figure 1.

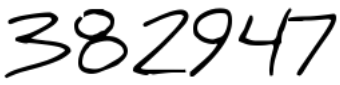


Figure 1.: Handwritten digits sample.

Our neurons easily perceive these digits, creating the sequence of **382947**. Using our visual cortex we can observe objects, numbers, landscapes, persons and get useful information if we already have seen something familiar or if we know something about it and its features. It is possible to say our brain is a supercomputer which has been tuned over the years. For this matter, perceiving a handful of handwritten digits has become a mere task

for our neurons. However, when dealing with a machine that has no brain of itself, making sense of some digits becomes increasingly difficult [15].

In some problems an algorithm, composed with some if statements, is enough but when working with numbers and visual recognition, simple algorithms do not overpass the problem itself. ANNs surpass the visual recognition problem once they take a dataset of handwritten digits and use it for training their system into perceiving the digits. Instead of creating an algorithm with almost infinite if statements, ANNs automatically infer rules to recognize the digits. And the larger the dataset, the better the results [1, 15].

An easy way to learn about neural networks starts with computer vision, however, they are already applied to speech, signal and natural language processing.

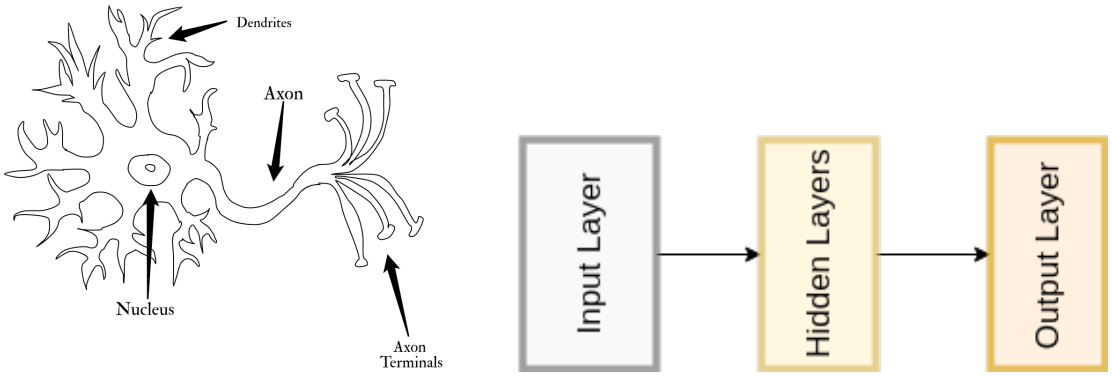
Besides all of this information, it is important to understand what is a neural network. One of the first definitions was granted by Dr. Robert Hecht-Nielsen as quoted below:

> *"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."* In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989

ANNs, as said before, are modeled after the biological neural networks, but the number of processing units does not come near the billions of neurons in a mammalian brain. Although they do not reach that number, ANNs are known for reacting to new inputs and variables as mammalian neurons do. Perceiving handwritten digits is part of a complex system, billions of neurons in several parts of our visual cortex work as one to understand which numbers are in the sequence, in ANNs there are several layers working as one to perceive the same thing [1, 15].

Figure 18 presents a comparison between the mammalian neuron and the artificial neural network. As seen in the Figure 2a, the unit of the biological neural network is the neuron and it is composed by **dendrites**, the **axon** and the **nucleus**. The **dendrites** are known for receiving the input signal, this signal is then processed and sent as an electric impulse via the axons to another neuron, creating a neural network which the output of the system is the response to the first stimulus. On figure 2b, an easier schematic is presented, however, the overall process is the same. ANNs are organized in layers, the first layer is the input layer, the second set of layers are the hidden layers and the last layer is the output layer. These layers are composed by interconnected nodes (the basic unit of ANNs) which usually contain an "activation function". The input layer is connected to the hidden layers where the true processing of information is done which then send their processed data to a final output layer [1, 15].

Our brain is programmed to use the necessaryneurons, and work alone, ANNs do not have that power. It is then essential to train and test the network until the number of hidden layers matches our needs of accuracy and sensibility (two terms that will be further



(a) Representation of the biological neuron. (b) Representation of an artificial neural network.

Figure 2.: Comparison between a biological neuron and an artificial neural network.

discussed in this master thesis). However, not only the number of hidden layers matter when creating a good neural network, as neural networks use a system of weighted connections, its weights can be modified according to the results of each training, this feature is called as a hyperparameter (the hyperparameters will be further discussed and explained in the next subsections) [1, 15].

Artificial Neural Networks can be seen as a remarkable system for finding meaning from imprecise data, extract patterns and detect trends impossible for humans to be aware of it or even other computer programs. It is important to emphasize the Algebra's role once these neurons work as correlations likewise when the input is **X** and the output **Y**, and the algorithm works to find the right function *f(x)=y* always assuming that **X** and **Y** are correlated. These concepts are the foundations of the first artificial neuron, the *perceptron*. The perceptron creates a correlation between the binary inputs it receives (x1,x2,x3, ...) and outputs a single binary (y equals to 0 or 1) [15].

2.2.2 *Deep Learning*

Deep Learning, also known as *Deep Neural Networks (DNNs)*, is a subfield of machine learning and has become one of the most studied fields in AI. Although machine learning as yet to become obsolete, it became essential to use neural nets as a pivotal resource to improve and reach other results in the learning mechanisms and it is now possible since there is the required computational power [1, 12]. The Figure 3 exhibits the relationship between AI and DL.

DL has yet to have a true definition, but it was early defined as a "neural network of more than two hidden-layers" (Figure 4). However, Deep Learning is much more than a neural network of more than two hidden-layers. DNNs have been used for speech recognition, image recognition and many other applications that can be applied to self-driving cars, to

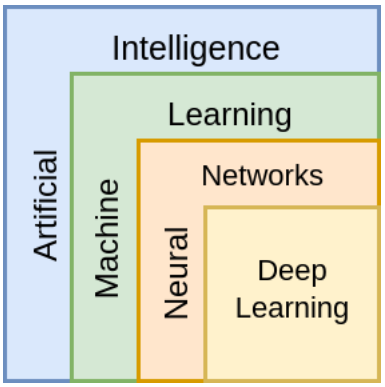


Figure 3.: Relationship between Deep Learning and Artificial Intelligence.

detecting cancer or neurodegenerative diseases or playing complex games and, in several of these fields, deep learning can now exceed human accuracy [12].

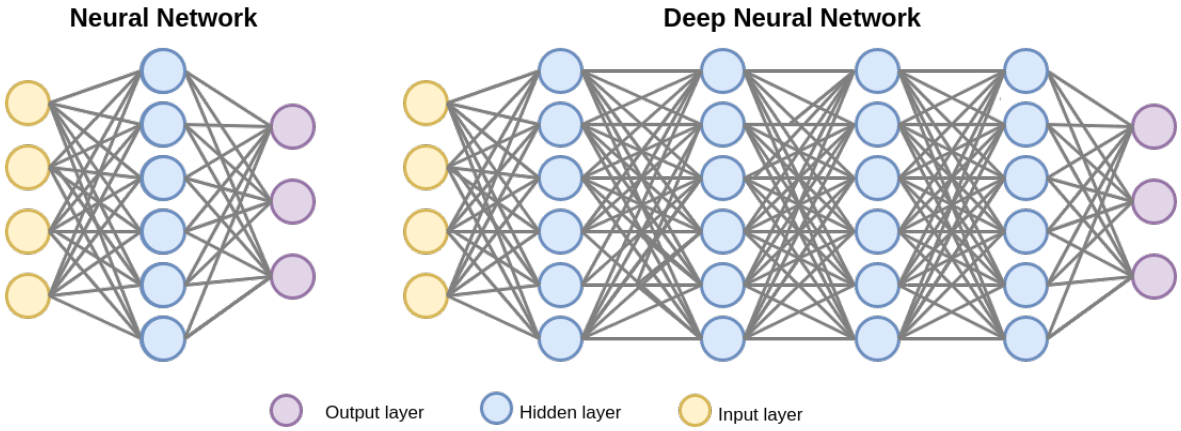


Figure 4.: Comparison between a Neural Network and a Deep Neural Network.

Although DL is a relatively new field, Schmidhuber [16] built a thorough work in reviewing the entire history of developments in this field which led to today's state while Arel et al. [17] emphasize the novelties from the last decade, both exhibiting all the promising results this field may present.

As ML, also DL relies on advance knowledge of mathematics, however, when creating the algorithms in machine learning, the mathematics are visible but in deep learning, all the frameworks and libraries available offer an easy way to work with it without knowing the mathematics behind the neurons [1].

However, it becomes fundamental to understand somecore concepts when dealing with deep networks such as [1, 18]:

- Parameters: These parameters are directly related with the weights on the connections of the network and each architecture may need different parameters/weights.
- Layers: Being a fundamental architecture unit in deep neural networks, layers can be of various types and are changed by the activation function applied to it.
- Activation Functions: An activation function propagates the output of one layer to the adjacent one, until the output layer. Each neuron in the hidden layers will have the defined activation function for that layer.
- Loss Functions: The loss functions exhibits how a neural network is doing when training, quantifying how ideal the model is. The model will only be ideal if the parameters are well chosen, to minimize the error.
- Optimizers: The optimizers are used to minimize the result from the loss function.
- Hyperparameters: Parameters which are tuned to make networks faster and well trained. The hyperparameters control the optimization functions, model selection and updaters (which are used to update the weights and minimize the error).

Feed-forward Neural Networks

Feed-forward neural networks are one of the easiest neural networks to understand. It has an input layer, many hidden layers, and one only output layer. Each of these layers is fully-connected to the forward layer and can have a different number of neurons. This network can represent any function if having enough neuron units and it is trained with the backpropagation learning algorithm [12].

The backpropagation learning algorithm uses the gradient descent [2] on the weights of the connections in orderto minimize the errors on the output of the network [21]. Although backpropagation has been considered slow throughout the years, the advances in using graphic processing units (GPUs) and parallel processing increased the computational power and therefore the ability to use the backpropagation and neural networks [1, 12].

The fully-connected layers can be summarized as:

$$y = f(x * w) \tag{1}$$

where x is the input vector, w represents the set of weights, f is the activation function and y is the output vector [18].

[2]Gradient descent is an algorithm that minimizes a function. [1, 19, 20]

Activation Functions

The activation functions are directly correlated with the existing types ofneurons. There are three major types of neurons that are commonly used, the Sigmoid, the Tahn and the ReLu neuron [1, 18].

The **Sigmoid** neuron, as implied by the name, uses the sigmoid function,

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2}$$

where $\sigma(z)$ represents the output of the layer. When diving into the Equation 2, it is noticeable that when z is very small, the output of the function tends to 0, on the opposite side, when z is high, the output tends to 1 (Figure 5a) [1, 18].

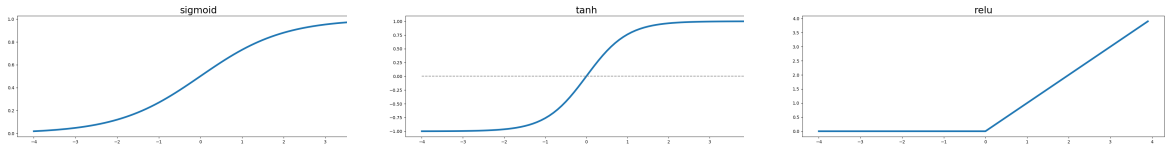
The **Tanh** neuron is also represented by an S-shaped graph but it has a range from -1 to 1 compared to the 0 to 1 range from the sigmoid function as seen in the Figure 5b. The Equation 3 exhibits how the function behaves and it can also be seen as a derivative of the sigmoid function [1, 18].

$$tanh(z) = \frac{sin(z)}{cos(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3}$$

The last of the three major types of activation function is the **ReLu** neuron or Rectified Linear Unit, which is becoming broadly used in computer vision and it is represented by the following equation:

$$f(x) = max(0, z) \tag{4}$$

where the function will choose the highest value between zero and the value of z, this equation can also be represented by $f(x) = x^{+}$ and its graph is delineated in the Figure 5c [1, 18].



(a) Graphic representation of the Sigmoid function [3].

(b) Graphic representation of the Tanh function[3].

(c) Graphic representation of the ReLu function[3].

Softmax, another activation function, can transform any score into probabilities. It is usually used as an output layer for multi-classification tasks. It normalizes the outputs of the previous tasks so its probabilities sum up to one, so the output portrays the probability of each class [1, 18].

[3]Designed with python.

Loss Functions and Optimizers

After the parameters are set, the learning algorithm tends to predict the target value for each training example and also outputs its loss function values. The loss function represents the measurement of the difference between the target value and the predicted value. Hence, it outputs a value for incorrect predictions. The learning algorithm will then search for the set of weights and biases parameters which can minimize the value of the loss function [1, 18].

There are loss functions more suitable than others for certain models (regression, classification, reconstruction) [1] although their descriptions are not under the scope ofthis master thesis it is important to reference the Cross-Entropy function.

The Cross-Entropy function, besides being one of the most used loss functions, it is used in multi-classification tasks although it is mainly based in the Binary Cross Entropy. The cross-entropy function quantifies the difference between two probability distributions [1].

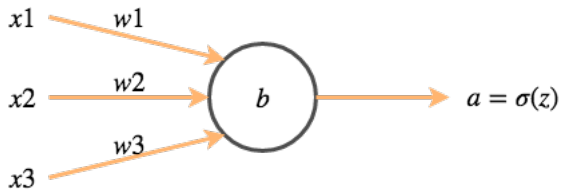


Figure 6.: Example of a neuron. X1, X2 and X3 represent the inputs; W1, W2, W3 represent the weights, b represent the biases; a is the output.

The Equation 5 represents the cross entropy function for the neuron in Figure 6, where n depicts the number of items of training data, the sum is done over the training inputs, x and y is the output we want to predict [15].

$$C = \frac{-1}{n}\sum_{x}[ylna + (1-y)ln(1-a)] \tag{5}$$

This function tends towards zero, in the positive range, as the performance of the neuron increases, the neuron improves its prediction of the targeted output [15]. When the loss reaches zero, the output is a perfect prediction of all the training examples.

To minimize the value of the loss function, an optimizer should be used. The *Stochastic Gradient Descent (SGD)*, which follows the same principle as gradient descent, finding the local minima or maxima by iteration [1, 18, 19], the *Adaptive Moment Estimation (Adam)* and *Root Mean Square Propagation (RMSprop)* are three of the most used optimizers. As the loss functions, these optimizers will not be further explained in this master thesis since a good explanation would encompass an explanation of statistics and differentiable functions.

2.2.3 *Deep Networks Architectures*

This subsection presents two of the most known architectures of DNNs, *Convolutional Neural Networks (CNNs)* and *Recurrent Neural Networks (RNNs)*, which are fundamental to create the future deep learning model. These architectures have been proving their value, surpassing former results got with the traditional machine learning algorithms/techniques [22].

Convolutional Neural Networks (CNNs)

CNNs use convolutions to learn features in data. They use images to object recognition or classification and it is possible to recognize faces, street signs, meerkats or boars, cars, boats and many other objects or individuals using visual data. The results already obtained by CNNs had promoted the deep learning and the good results in image recognition have brought acknowledgment to the power of deep learning [1, 18]

CNNs and its machine vision properties have been empowering solutions for self-driving cars, robotics and improving treatments for visually impaired patients. Although its focus is computer vision, CNNs have been used for sentiment analysis and natural language translation. It is inspired by the visual cortex and acts in accordance with it. The visual cortex has the *visual field*, which is covered by smaller subregions tiled together. These smaller subregions come from the sensitivity of the cells in the visual cortex [1, 18].

As said before, CNNs use the convolution to learn features in data, in other words, it applies several filters (known as kernels) to the object to perform the feature extraction; the outputs become the feature maps [1, 23]. Figure 7 exhibits how convolution works on an image. Dealing with images, the CNNs has to deal with three-dimensional features: width, height and depth. The width and the height represent the width and height in pixels and the depth represents the RGB channels.

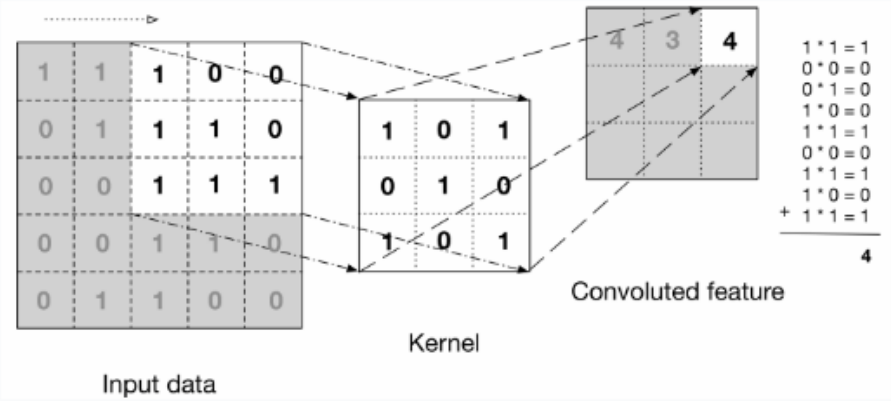


Figure 7.: The convolution operation. Obtained from [1]

A CNNs architecture is commonly depicted in three groups, the input layer, the feature extraction layers and the classification layers. The input layer gets the three-dimensional input, width, height and depth. The feature extraction layers are usually composed by a convolutional layer (commonly used with ReLu activation function) and a pooling layer. The pooling layer has the task ofreducing the dimensionality of the feature maps and increasingly build higher-order features. This layer splits each feature map into small equally sized subsets and creates a feature map with the higher-order features. The last block of layers, the classification layers, composed by one or more fully-connected layers, have as input the higher-order features and output the class probabilities or scores [1, 18, 21, 23].

Some commonly used CNNs architectures are [1, 23]:

- LeNet [20]
- AlexNet [24]
- ZF Net [25]
- GoogLeNet [26]
- VGGNet [27]
- ResNet [28]

A thorough review of convolutional neural networks can be found in Stutz's [29] seminar report.

Recurrent Neural Networks (RNNs)

The neural networks above mentioned are characterized by having no memory, there is no state saved between each layer input. When dealing with sequences or time series, these networks need to have access to the entire sequence at once. On the opposite side, when writing this sentence, each previous word was memorized to create a perceptible sentence. This happens when reading the same sentence also, keeping a memory of the previous words gives a meaning to the sentence, making it not just a group of meaningless words put altogether [23].

RNNs [30] are inspired by the biological intelligence that incrementally recalls past information and keeps the new information as it is introduced. Whilst using an eased version of this biological neuron, RNNs use the recurrence to forward information from the past. Hereupon, these neural networks are suitable to output predictions from sequences [18].

While CNNs need a fixed size input and output, RNNs deals with non-fixed inputs and outputs, non-fixed computation steps and it can even operate over sequences of vectors. These unconventional ways to deal with the input and the output are presented as an advantage of RNNs. These neural networks can operate on the input vectors with a method

of One-to-Many, where the input vector can be an image and the output the caption of that image; Many-to-One, where a sentence is given and the sentiment analysis is the output or even Many-to-Many, where a video is classified by frame [1, 31].

A deeper understanding of Recurrent Neural Networks can be found in the textbook by Graves [32].

Long Short-Term Memory (LSTM) Network

Long Short-Term Memory (LSTM), one of the most used variations of RNNs, are used to model time sequences and its long-range dependencies with more accuracy than conventional recurrent neural networks, as first proposed in the mid-90s [1, 33]. Thus, LSTMs are programmed to remember the information for long periods of time. These neural networks came to tackle the vanishing gradient[4] and the exploding gradient[5] problem for which RNNs are known when dealing with long input sequences [34].

Figure 8 presents a schematic of the LSTM's block. A LSTM layer is composed by the input data, which is fed to the cell but also to three gates, namely the input gate, the forget gate and the output gate. These gates will decide how the input will be treated.

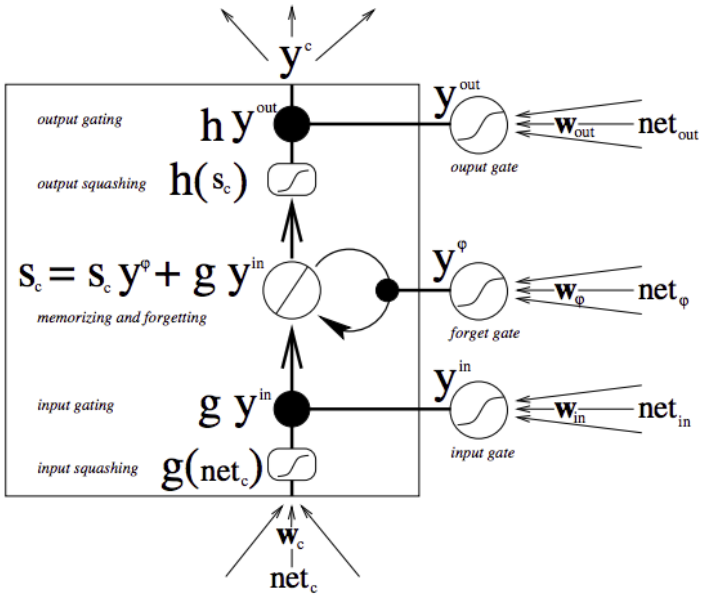


Figure 8.: Schematic of the Long Short-Term Memory block. Obtained from [1].

The three black dots after the gates will decide if the new input will get in or not, with this, the present cell state can be erased or not, impacting on the output at the present time step. Each gate can be open or shut and this combination can be changed at each step. As

[4] Vanishing Gradient: When the gradient is so small that the model learns too slow ending up vanishing.

[5] Exploding Gradient: when the values are above 1, the successive multiplication will make the model to learn too fast ending up not being successful, exploding.

for the cell, it can not only forget its own state, but it also can be overwritten and can be read at each step. Between the cell and the gates there are activation functions and, for input and output activation it is usually used the *Tanh* function and the *Softmax* function respectively [1, 18, 23, 31, 35].

Olah [35] developed an article that fully reviews the long short-term memory networks and works as a reference for Google Brain's framework, Tensorflow.

2.2.4 *Deep Learning Applications*

Deep Learning has been broadly used in several fields and tasks of classification, prediction and clustering are commonly employed in the healthcare field.

Classification

When dealing with classification tasks, the neural net uses labeled data, known as *supervised learning*, to learn the correlation between those labels and the data. So, in supervised learning, the classification task is done by taking an input with a label and classify it with that same label. When performing this task, each sample is classified into a class [1, 36].

The binary classification, based on the first artificial neural neuron, the perceptron, relies on a single output, 0 or 1. This type of classification is used in a context where something must be classified as "is" or "not is", such as if a person has a disease or does not have a disease [1, 18].

On the other side, multi-classification, where we have N labels, the output will the label with the highest score, that is, the sample is labeled with all the available labels and each of these labels is scored, the label with the highest score is the true output [1, 18, 36].

Some examples of deep learning applications using classification are [37]:

- Face detection and recognition in static images;
- Objects identification;
- Gestures recognition in video;
- Sentiment recognition, voice detection;
- Classification of text/emails as spam.

In the healthcare field, classification has also been widely used. From the classification of physiological signals to medical imaging and the creation of decision support systems [34, 38], the results are promising and are expected to reach new heights.

Although there are other applications, the classification task is the fundamental basis since when something can be classified, it becomes easier to detect or rank something.

Regression

The task of regression is based on the assumption that it is possible to predict a value or dependent variable by knowing the independent variable. This task has many ways to be performed and developed and one of the most common ways is the linear regression, which is supported by $y = a + Bx$, where y is the predicted value and x the parameter vector [1].

With regression, DL can find correlations between past, present and future events, where the future events are the predicted values. According to this, regression can be applied in several fields to predict different outcomes, outputting the result that is most likely to occur [1]. So, regression can be applied to any system which deals with time series, without actually caring with the time series.

This task can be performed for predicting hardware breakdowns, as predicting if the system can somehow fail in the future, or for predicting if a customer will leave the company based on its activity and other metadata [37].

Regression can be performed in the healthcare environment not only to predict the hardware breakdowns but to predict major conditions as strokes, heart attacks, and others based on the vital signs and data wearables [39].

Clustering

On the opposite side of classification, the clustering task is commonly performed with unsupervised data. [34, 37]. This task groups data that presents the same features or are similar between them. This grouping is done with a distance measure algorithm, where the output is usually represented by the items densely grouped around the n centroid and these items are classified in that class [1] however, when the classes are not specified, clustering groups data which seem to be similar and "fall naturally together" [40].

Typically, clustering is used for [37]:

- Images, sounds or documents comparison to find similarities;
- Anomaly detection, where the search for similarities is reversed.

In healthcare, clustering has been used from the classification of electrocardiograms [41] to the prediction of the risk of a disease [42].

2.2.5 *Deep Learning Frameworks*

Theano

Theano was the first Python library to address Neural Networks as a mathematical problem involving multi-dimensional arrays efficiently. Also being able to use GPUs and per-

forming efficient symbolic differentiation. It was created in 2007 and was the first widely used deep learning framework [43, 44].

Known as the fundamental basis of deep learning, Theano was still alive in the year of 2017 yet its life support was shut down later that year since other frameworks and libraries arose, however, maintenance is still on the table by its developers [43].

Tensorflow

Google Brain Team built a framework for doing research in machine and deep learning and in the year of 2015 that framework was made public as an open-source framework called Tensorflow. It is written in C++ and Python and has a large community and easy to read the documentation. Tensorflow is described by its developers as "an open-source machine learning framework for everyone". Following Theano library description, also Tensorflow is described as a library for numerical computation using data flow graphs. The multi-dimensional arrays are called tensors and are the connection between the graph nodes [45, 46].

Like other frameworks, Tensorflow can be deployed to one or more CPUs or GPUs to be more efficient and faster. It is known that Tensorflow lack of speed makes some developers avert from this framework however, it is the most widely used framework in the DL field and it will stay that way for the next few years. One of the latest releases, Tensorflow1.5 came to bridge the lack of speed and it now offers developers a faster programming style. Big companies such as Uber, eBay and Coca-Cola are known to have adopted Tensorflow as its predilect framework for AI [45, 46].

Torch & Pytorch

Not as known as Theano, Torch was used by several major companies such as Google, Twitter and Facebook. It is still widely used, and it relies upon LUA as its scripting language although it can alsobe used with C [34, 47]. And it is the foundation of another framework, Pytorch which has an increasing popularity. As indicated by its name, Pytorch is a python version of Torch framework that was released by Facebook in 2017 and it came as a new way to analyze unstructured data using dynamic computational graphs [46, 48, 49].

Using GPU allocators, Pytorch makes DL models more memory efficient and is presented as a high speed and flexible framework [18, 46, 48]. It is being used by major companies still there isn't as much community support as other frameworks that have been on this run for a longer time. Yet, this framework may be a strong opponent of Tensorflow if its growth continues as it is.

Caffe & Caffe2

Focused on giving priority to expression, speed and modularity, a team from Berkeley Artificial Intelligence Research created Caffe and it keeps growing thanks to its community contributors. Mostly centered in Convolutional Neural Networks, Caffe is a reliable framework when modeling CNN's [50]. Considering its success, in 2017, Facebook presented Caffe2 which is known as the successor of Caffe. Caffe2 introduced Caffe users' pre-trained models to build demo applications with little trouble [51].

Caffe is presented as one of the fastest deep learning frameworks when dealing with CNN's and its expressive architecture abets for its successful application and posteriorly innovation, which are shown as major contributions of this framework and also its advantages [52].

As of May of this year, the developers of Caffe2 wrote a statement in which they understand that since Facebook relies on Pytorch and Caffe2, the best solution was to merge both frameworks [53].

Deeplearning4j (DL4J)

The Skilmind company created the first commercial oriented, open-source, distributed deep learning library for Java and Scala. Deeplearning4j has been in the AI field from 2014 and it can be run on top of some big data tools as Apache Spark or Apache Kafka. Although the DL4J community is not as large as Theano's, Tensorflow's or Torch and Pytorch, the framework developers' use Gitterto be connected all the time with its users and to help the development of its models when the documentation is not enough. With an easy-to-read documentation and also a book in Deep Learning and the framework, DL4J came to incite Java developers into the AI field. Without neglecting some great work already made in the neural networks fields, DL4J can import neural models from other major libraries [54, 55].

SystemML

SystemML was created by IBM and recently went from an Apache Incubator Project to an Apache Top Level Project. This open-source framework is a scalable and flexible machine learning system with recent deep learning capabilities that can be run on top of Apache Spark or MapReduce [56, 57].

ML algorithms are usually written in R although it now has a Python built-in integration. One of its major advantages is how it can be executed, as standalone, Spark Batch, Hadoop Batch or even with Java ML Connector (JMLC) [56, 57].

MXNet

MXNet is one the Apache Incubators deep learning libraries. It is presented as flexible and efficient when dealing with deep learning and is supported by Microsoft and Amazon. This framework supports several programming languages, such as JavaScript, R, Go, Python and C++, becoming one of the most global frameworks existing. However, its community is very limited and there isn't a vast easy-to-use documentation [58, 59].

To increase its popularity, but also simplify its use with easier coding and more flexibility, Amazon and Microsoft launched a new deep learning interface. Gluon was implemented in MXNet and came to revolutionize Tensorflow's monopoly. Gluon is, as told by its developers, an interface that simplifies the development of deep learning models without discarding the training speed. Dynamic graphs and high performance were two of the main features developed and delivered[58, 59].

CNTK

The Microsoft Cognitive Toolkit is a Microsoft's open-source deep learning framework. Known previously as CNTK, this framework has refined algorithms to deal with big datasets and grants high scalability and performance [60].

Microsoft's own deep learning framework provides an easier way to use and combine convolutional neural networks (CNN), feed-forward deep neural networks, recurrent neural networks and long short-term memory networks as it implements the Stochastic Gradient Descent (SGD) in a standalone cluster or across multiple GPUs and servers. Speed and scalability are some advantages of this toolkit [60].

BigDL

Intel developed BigDL as a distributed deep learning framework for Spark and can run directly on top of Spark or Hadoop clusters. This framework uses Scala or Python as languages for writing deep learning applications [61–63].

BigDL is being used because of its three major advantages, the community support, it is modeled after Torch and can load pre-trained models from it and Caffe and both of this mature frameworks have huge community support. Its direct connection with Spark promotes the efficient scalability of data analytics as well as its implementation of SGD. The third reason is its high performance once it uses high processing tools like Intel's Math Kernel Library and Intel's Xeon processors [61, 63].

Keras

Keras was launched in the middle half of 2015 and is an open-source API made to enable fast experimentation of neural networks. Built-in Python, its focus was to ease the struggle

of doing good AI research. As a deep learning library, Keras is able to run on top of Tensorflow, CNTK or Theano [64, 65]. Although it has been in the scenery for a while, Google has made notice that Keras will be included in the upcoming Tensorflow releases.

Summary

All these frameworks exhibit advantages compared to the others, being that the reason why they were developed. Yet, choosing the most suitable framework depends mostly on what is intended to be built and if the system's scalability is important or not. So, it is crucial to know beforehand the type of system that will be built since it gives another perception when choosing the right framework. Table 1 summarizes the comparison of the above frameworks.

Many data scientists, engineers and big companies have created comprehensive comparisons between these frameworks [52, 66–72]. According to several of these articles, the popularity of a framework causes a huge impact when choosing the developing tool, however, how fast it performs and how it can be deployed can also be an impediment for choosing such framework.

Nevertheless, in matters of visualization, due to the easy integration of some libraries, it is not a huge barrier. Although Tensorflow (Tensorboard [73]) and DL4J (DL4J Training UI [55]) have their own visualization modules likewise Torch (Neural Network Graph Package [47]) and Theano (d3viz [74]), some of the frameworks presented can integrate some type of visualization or are currently developing their own.

Multi-language support can be an important factor when dealing with a diverse developing team, allowing to not narrow the choice of a framework based only on its programming language.

On the other side, one of the most significant factors derives from the use of GPU's since it provides a faster and lighter processing of deep learning models, promoting parallel processing and cheaper computational power [75].

[6]Supports integration with a visualizer, does not have its own.
[7]On the roadmap for next release.
[8]Only works on Intel Chips.

Table 1.: Comparison of Deep Learning Frameworks based on Licenses, CPU and GPU Support, Multi-Language Support, Supported Neural Networks and Visualization Integration.

Name	Creator	License	CPU and GPU Support	Multi Language	Supported Neural Networks		Visualization Integration
					RNN	CNN	
Theano	Université de Montréal	BSD	✓	X	✓	✓	✓
Torch	Facebook	BSD	✓	X	✓	✓	✓
Pytorch	Facebook	BSD	✓	X	✓	✓	✓[6]
Tensorflow	Google Brain	Apache 2.0	✓	X	✓	✓	✓
SystemML	IBM	Apache 2.0	X[7]	✓	✓	✓	✓[6]
BigDL	Intel	Apache 2.0	X [8]	X	✓	✓	✓[6]
MxNet	Chen et al. [59]	Apache 2.0	✓	✓	✓	✓	✓
Caffe	Berkeley Center	FreeBSD	✓	✓	✓	✓	✓[6]
DL4J	Skymind	Apache 2.0	✓	✓	✓	✓	✓
CNTK	Microsoft	MIT	✓	X	✓	✓	✓[6]
Keras	Chollet et al. [64]	MIT License	✓	X	✓	✓	✓[6]

2.3 DISTRIBUTED DATA PROCESSING AND STREAMING

2.3.1 *Overview*

Real-time distributed streaming and processing has become a hot topic in recent years, with many open-source frameworks surfacing. Such frameworks are usually being applied to applications focused on sensor-based monitoring, stock trading, web traffic processing, network monitoring and even mobile devices. All the produced data is typically sent as tuples or as a stream of events and as the volume of this data is exponentially increasing, batch processing in distributed systems, almost introduced by MapReduce, is being overpassed by these recent frameworks. Promoting the real-time analysis, these frameworks came to fill the void in the big data world. Real-time/near-real-time processing is a demanding task but in the long run, it is intended to reduce the time consuming of posterior data processing [76–81].

The frameworks depicted below can be decomposed in three major groups, batch-only frameworks, stream-only frameworks and the last group which can be called hybrid frameworks, although some of these frameworks show also the ability for processing, streaming tools are present too.

2.3.2 *Frameworks*

Apache Kafka

When dealing with data streaming, it is, sometimes essential to creating a real-time data pipeline or even a real-time application that reacts to streams of data, Apache Kafka provides these two features. As a distributed streaming platform, Kafka can publish and subscribe to streams of records, similar to a message queue, the ability to store streams of records in a fault-tolerant manner and process streams of records as they occur makes Kafka one of the most suitable frameworks when dealing with high volumes of real-time data. Written in Scala, Kafka has four core APIs, Producer and Consumer API, Streams API and Connect API. When working in a cluster environment, each node or server is known as a broker [82–84].

Kafka's use the producers (publishers) to send messages and the consumers (subscribers) to receive them. The producer is a process that publishes those messages to a **topic**, on its side, the consumer is a process that subscribes to these topics and receives its messages [82, 85].

Kafka Topics

A Kafka topic is a category to which the messages are published. These topics can have zero, one or many consumers which had subscribed to its data. Topics also introduce the concept of **logs** in Kafka, as for each topic, the Kafka cluster keeps a partitioned log. Figure 9 presents the anatomy of a topic where each partition is an immutable sequence of messages appended to a structured commit log. Each message has a unique id appended to it called offset to identify each message in its partition [82, 83, 85].

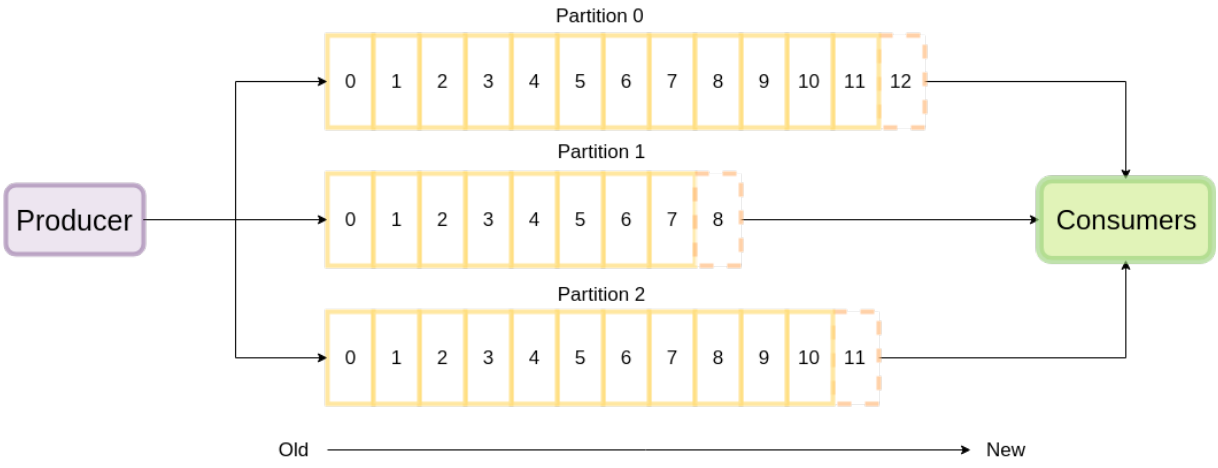


Figure 9.: Anatomy of a Kafka Topic with Producers and Consumers.

Kafka has several properties and its persistence is shown in topics where the messages can be durably persisted, making the messages available for consuming for the duration previously set and after that, the data is discarded. Although Kafka discards data once it is consumed, it is not a problem to store data for a long time as it does not affect its performance. As for the metadata retained, only the offset or position of a consumer in the log is saved. However, a consumer can reset to an older offset or move to the newer record, without causing much disturbance on the cluster or other consumers [82–85].

Kafka Streams

The Streams API provides an application acting as stream processor, which consumes from one or more topics and produces a streaming pipeline. Kafka Streams is built on the Kafka Producer and Consumer libraries which facilitates its development. Working on top of these features also allows taking advantage of the data parallelism, fault tolerance and distributed coordination of native Kafka [82, 83].

When dealing with stream processing, the processing guarantees become of major importance and Kafka Streams is being developed to deliver an end-to-end exactly-once processing. It can guarantee the processing results from any record that is read from the source Kafka topics, are reflected exactly once in the output topic and also in the state stores for stateful operations. This also means that records will be processed even if a Stream client or a Kafka broker fails. In order to achieve its low latency (millisecond processing latency), Kafka Streams applies one-record-at-a-time processing [82, 83].

This API partitions data for processing it which enables data elasticity, scalability, fault tolerance and high performance. Using both the concepts of partitions and tasks, Kafka Streams creates a parallelism with Kafka itself and its topics. As so, a stream partition maps to a Kafka topic partition, totally ordered and the keys of data records establish the partitioning of data in both sides, Kafka and Kafka Streams [82, 83].

Kafka Connect

Kafka has the ability to copy data from a location to another location using connectors as so, the connectors move the data into and out of Kafka. Kafka Connect can take an entire database and send it to one of its topics, making it available to be later streamed with low latency. Focused on scalability and reliability, Kafka Connect provides a tool for streaming data between Apache Kafka and other systems like ElasticSearch [82, 85]. Figure 10 presents how Kafka Connect is applied.

Kafka Connect provides a commons framework for Kafka connectors, simplifying its development and further deployment and management. Like Kafka, this tool can be scaled up to support an entire organization or can be run only in standalone mode to development, testing or small deployments. In order to manage its available and connected connectors,

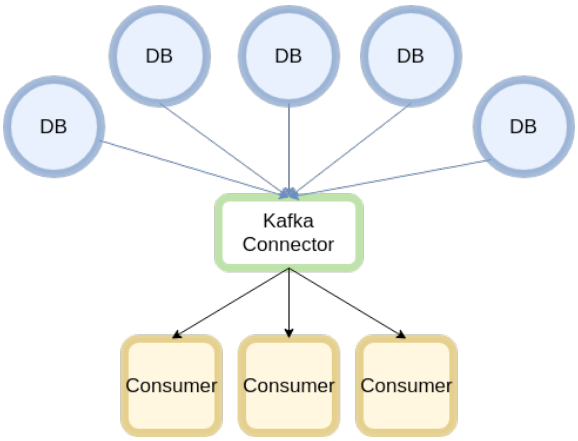


Figure 10.: Example of how Kafka Connect is applied.

Kafka Connect relies on a Rest API. Besides the distributed and standalone modes, Kafka Connect can also add more workers to its cluster in order to scale up. Relying upon the commodities of Kafka, Kafka Connect can be used to connect batch and streaming data systems [82, 85].

In overall, it is essential to understand Kafka Connect as a moving tool, which takes data and moves it to another place, these connectors pull data from a source and push them to a sink or an application [82, 85].

Apache Hadoop

On the first subset, Apache Hadoop stands out as the batch processing framework whereas MapReduce is its processing engine. Hadoop is the oldest big data framework that stood out in the open-source community. Hadoop revolutionized the way the algorithms were implemented to make dealing with large amounts of data more accessible [76–79, 86–88].

During the years, Hadoop has been improved and new components and layers were introduced like HDFS and YARN. As so, Hadoop is now composed by [86, 89]:

- **Common**: Hadoop Common is the layer that supports all the other Hadoop modules;
- **HDFS**: Known as the distributed filesystem layer, HDFS coordinates the storage and replication in the clusters, assuring the availability of the data in case of failures. It is used as a data source and to store intermediate processing results and also the final results. HDFS surpasses other filesystems once it is highly fault-tolerant and is intended to be deployed on low-cost hardware, HDFS also delivers high throughput access to application data and is suitable for large data sets;
- **YARN**: Hadoop YARN, Yet Another Resource Negotiator, manages the cluster resources and schedules jobs to be run. This management allows running diverse

workloads in the cluster. YARN was built on the guiding principle of splitting the functionalities of resource management and job scheduling/monitoring;

- **MapReduce**: MapReduce is nothing more than the batch processing engine of Hadoop, which is also seen as the YARN-based system for parallel processing of large data sets. Hadoop's MapReduce provides an easy manner of writing applications to process great amounts of data on large clusters in a reliable and fault-tolerant way.

As the batch engine, MapReduce works according to a simple procedure [79, 86, 87, 89, 90]:

1. Read the data set from the HDFS filesystem;
2. Divide the data set into segments and distribute among the present nodes;
3. Apply the computation technique on each node to the subset of data (the intermediate results are stored back to HDFS);
4. Redistribute the intermediate results to group by key;
5. Summarize and combine the results calculated by the individual nodes;
6. Write the final results to HDFS.

Figure 11 presents a schematic of the above procedure [86].

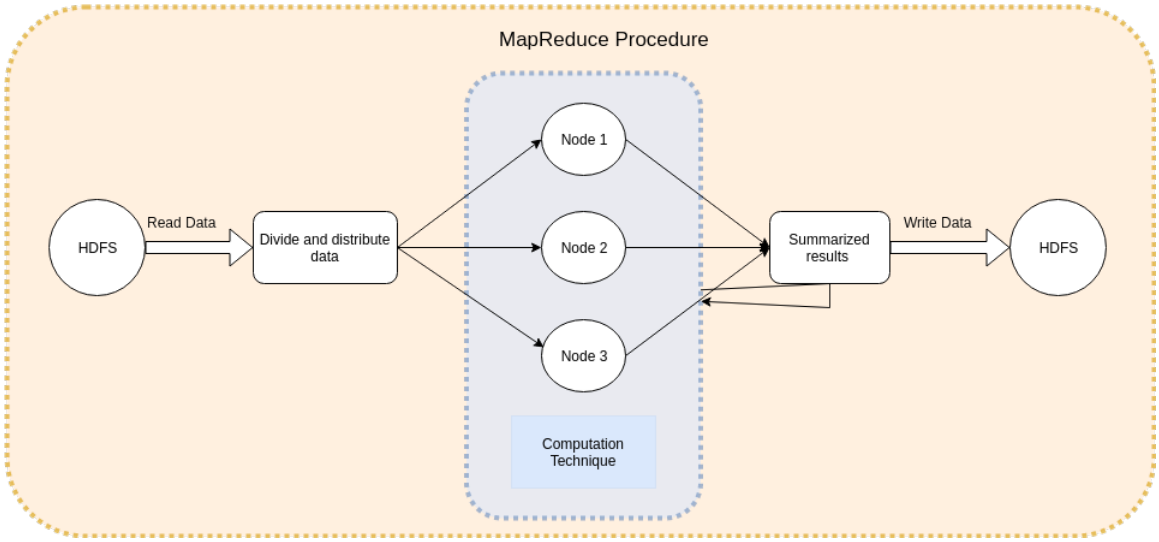


Figure 11.: Schematic of a MapReduce job.

Despite the emerging of other frameworks, MapReduce is still broadly used on not time-sensitive data as it has proven its value as a batch processing model for handling large data

sets. A well-maintained and functioning Hadoop cluster is considered inexpensive and effective for many use cases which promotes it to be widely applied.

Apache Storm

Storm, which is also used under the Apache License, is a real-time processing and streaming framework. As Apache Spark, Storm is a distributed computation system for big data, focusing on velocity. The ability to process millions of records per second per node on a cluster emphasizes the speed which is a fundamental part of Storm [91, 92].

This framework allows for preventive or even optimization use cases such as real-time customer service management, operational dashboards or cybersecurity analytics. Apache Storm is shown as a fast, scalable, fault-tolerant, reliable and easy to operate framework which are some of the conditions that are becoming elementary for processing in real time the amount of data daily produced [76, 79, 91, 92].

Storm applications can be written in many programming languages like Java, Python, Scala, etc [91].

One of the most relevant characteristics of Storm rests in the guaranty that each unit of data is processed and if a unit is lost, Storm has the ability to replay data to make that each unit is processed at least once [76, 79, 89, 91, 92].

On the opposite side of Hadoop's MapReduce jobs, Storm runs on top of topologies. The main difference between a MapReduce job and a topology relies on the duration of the event, a MapReduce job will eventually finish while a topology will run forever until something or someone kills it [89, 91, 92].

A topology can be seen as a direct acyclic graph (DAG) with multiple components where data flows between the components of the graph. These graphs are composed by spouts and bolts and each represents sources of streaming data to be operated on and the processing step that consumes streams and outputs the result as a stream, respectively. Storm was built to do small and discrete operations using spouts and bolts composing a topology [89, 91–93].

Figure 12 presents an example of a Storm topology.

Apache Storm rests on the second subset as a streaming framework that ensures an at-least-once processing, however, it does not ensure that the messages will be processed in the supposed order. Storm can have layers on top of it, as so, for an exactly-once processing it can use Trident, which, besides the exactly-once processing, also allows "transactional" datastore persistence and stream analytics operations. Another layer is the Streams API, an alternative interface that delivers a typed API for expressing streaming computations and supports functional style operations, although, it is still an experimental feature. As for the last layer, Storm SQL provides users the ability to run SQL queries over streaming data, as

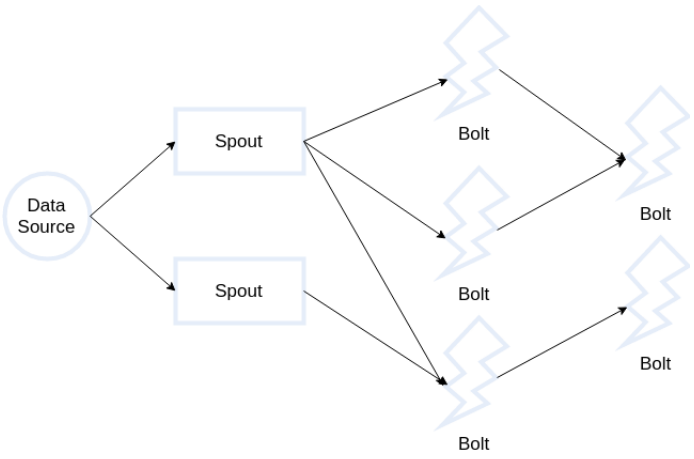


Figure 12.: Storm topology example

Streams API, also Storm SQL is an experimental feature and is subject to change over time [76, 77, 79, 80, 87, 91, 92, 94].

Apache Samza

Lying on the streaming framework, Apache Samza is a distributed stream processing framework. In order to provide fault tolerance, processor isolation, security and resource management, Samza uses Hadoop Yarn. Samza also employs Apache Kafka for messaging, benefiting from its architecture and guarantees. As some of the frameworks mentioned, Samza provides fault tolerance, durability and scalability too (Figure 13) [76, 78, 89, 95, 96].

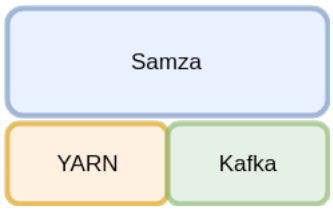


Figure 13.: Samza architecture

As Samza depends on Kafka's architecture, the Kafka's concepts are important for dealing with data. Kafka has five major concepts, topics, partitions, brokers, producers and consumers as previously mentioned, these five components are essential for Samza since Kafka shows an immutable log and so Samza deals with immutable streams. Any transformations create new streams that are consumed separately without interfering with the other streams [76, 78, 81, 89, 94, 95, 97, 98].

Samza also relies on Kafka's replicated storage of data which can be accessed with low latency. An arbitrary number of subscribers can be added to the output of any step, be-

ing helpful for teams that need to access similar data without adding additional stress to databases. Having a direct connection with Kafka, Samza have minimum problems with backpressure[6] once Kafka can store data for long periods of time and therefore the components have time to do the processing stage [76, 78, 81, 89, 94, 95, 97].

Samza delivers high-level abstractions that are more easy to work than Storm yet, spouts in Storm are similar to stream consumers in Samza and bolts are similar with tasks.

Samza prevails as one of the best frameworks to work with when Hadoop and Kafka are already available or are easy to implement. Samza is also a good option when the platform is used by multiple teams needing to use the data streams in different stages of processing. The low latency performance of Samza is presented as an advantage of its use however, Samza does not perform well for systems that need extremely low latency.

Apache Spark

Apache Spark is known for being an open-source distributed cluster computing framework, it uses most commonly in-memory data processing engines that can do ETL, analytics, machine learning and graph processing on large volumes of data as batch processing or streaming processing, using Java, Scala, Python, R and SQL as it main languages although it is mainly written in Scala. Spark can be deployed as a standalone cluster or in Hadoop YARN, Apache Mesos, Kubernetes or even in the cloud as it can, as well, access several data sources (Figure 14) [76–80, 96, 99].

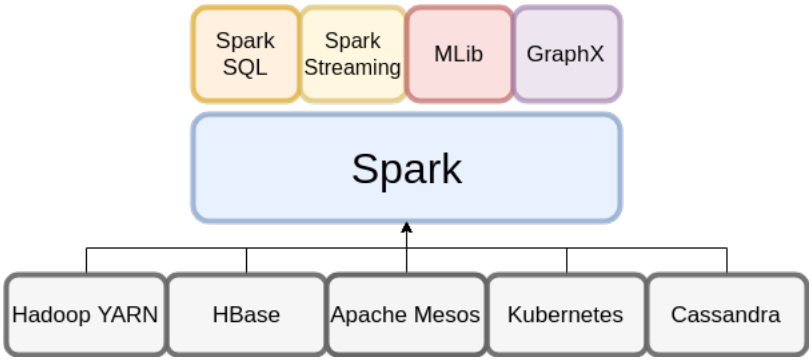


Figure 14.: The Apache Spark architecture platform

Spark is also commonly compared to Hadoop's two-stage disk-based MapReduce computation engine, however, once Spark has a multi-stage in-memory computing engine, it is allowed to run most computation in memory and it promotes better performance for certain applications [76–80, 99].

[6]The problem of backpressure happens when the influx data rate is greater than the rate at which the system can process the data in real time [89].

Spark was built in order to reach better results in speed, ease of use, extensibility and interactive analytics. Using Spark API's can simplify machine learning and predictive analytics scalability [76–80, 99, 100].

This framework is typically used as an alternative to MapReduce when dealing with data that requires low latency processing which is not provided by Hadoop's MapReduce. However, Spark does not overtake Hadoop once it does not have its own filesystem and often uses Hadoop's HDFS [76–80, 94, 99, 100].

For implementing the in-memory batch computation, Spark uses a data abstraction called Resilient Distributed Datasets, known as RDD. RDD's are immutable structures that exist within memory that represent collections of data. When operating in an RDD, it will output new RDDs. RDDs give the fault-tolerance feature to Spark since it creates a lineage graph, allowing to know the parent RDD and also the data on disk. Sparks' inherent speed is also provided by RDDs, being known as lazy, a sequence of alterations can be performed on RDDs without spending any computing time on them [77–80, 89, 99].

On the streaming side, Spark Streaming processes data streams as a series of small batches. As so, it works by sending the data as small fixed datasets for batch processing. Although this process usually works well, it has discrepancies when compared with real stream processing frameworks [76–80, 94, 99, 100].

Sparks' deployment versatility and large ecosystem promote the use of Spark nevertheless when streaming with Spark, it is important to disclose the buffer which is not an advantage when low latency is truly needed. Although in-memory computing may come costly once RAM can be more expensive than disk space, profit can come from the systems' speed [76, 77, 79, 97, 99].

Showing both streaming and processing capabilities, Spark belongs to the hybrid frameworks subset and it should be considered when the throughput value is more important than latency.

Apache Flink

Apache Flink is another open-source framework for distributed stream processing. This framework divides its features by the type of data to be treated, unbounded[7] data sets are dealt with the DataStream API and the DataSet API is used for working with bounded[8] data sets [76, 78–80, 94, 102, 103].

For unbounded data sets, Flink uses a streaming execution model that executes continuously, an event-at-a-time. When dealing with bounded data, Flink treats it as a special case of unbounded data, making it possible to use all the streaming concepts used previously on unbounded data. Since Flink has two distinct APIs for both types of data, it can be

[7] Infinite data sets that are appended to continuously [101]
[8] Finite and unchanging data sets [101]

run on the same distributed streaming engine creating a powerful and simple architecture [76, 78–80, 89, 94, 102].

Flink has four APIs on top of it, DataStream API, DataSet API, Table API - a SQL-like expression language for relational stream and batch processing that can be applied on top of the two previous APIs - and Streaming SQL -an API that enables SQL queries to be executed on streaming and batch tables - and three libraries that can be applied, CEP - for event processing -, FlinkML - for machine learning - and Gelly - for graph processing. Figure 15 presents the Apache Flink architecture [76, 78–81, 89, 94, 97, 102].

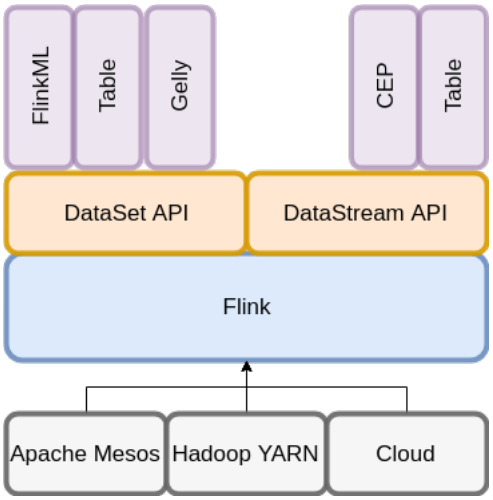


Figure 15.: The Apache Flink architecture

Flink's processing method, an event-at-a-time, enables Flink's resilience and, like Spark, Flink can run in a standalone cluster node but also provides support for Hadoop YARN and Apache Mesos. As well as other Apache frameworks, Flink can integrate with HDFS and Kafka and can even run tasks written for other frameworks like Storm using compatibility packages [76, 78–80, 89, 94, 102].

Apache Flink framework provides a low latency, high throughput and real-time processing, which makes it more suitable than Spark for many more use cases. Flink has an autonomous management of resources and optimization since it does not rely on the Java garbage collector mechanisms for its memory management neither it requires manual optimization when data changes, Flink automatically adjusts its features according to the data at the moment. This shows that Flink is best suited for projects with a heavy stream processing demands while still needing some tasks using batch processing [76, 78–81, 89, 94, 97, 102].

2.3.3 *Summary*

Although only seven frameworks were thoroughly analyzed, there are still some alternatives for processing big data.

Hadoop still excels when dealing with not time-sensitive data and once it is less expensive than other systems, it keeps being extensively used. Onto the stream-only frameworks, Storm and Samza have many advantages each but the lack of guarantee in ordering in Storm prevails as a major disadvantage while Samza allows a good state management and replication and has a tight connection with YARN and Kafka.

Spark is a suitable framework when the system demands speed and high throughput but not low latency. Without being a young framework when compared with Flink, Spark has a good community for support, several available libraries and integrations. On the other side, Flink provides high throughput and low latency using stream processing with batch processing support. Flink also delivers automatic optimizations of the system yet, being a recent framework it is not yet widely applied neither has a huge support community.

Having so many options, it is fundamental to know the data to which these frameworks shall be applied, being this the biggest variable when choosing any of these frameworks. Understanding if the system shall have a high throughput or a low latency will simplify the choice of a framework and moreover, it will help create a first proposal of the system.

Furthermore, it is essential to emphasize that the interoperability between some of these frameworks promote the flexibility of big data systems. These frameworks can sometimes work alone but in order to enhance the performance of the system, it sometimes becomes essential to hook several frameworks. As so, Spark does not overpower Hadoop but, correctly used, can replace MapReduce and also work as a streaming framework.

The Table 2 summarizes the features of each framework here presented except for Apache Hadoop and Apache Kafka[97].

Table 2.: Comparison of Data Streaming Frameworks based on Streaming Model, Guarantees, Fault Tolerance Mechanism, Latency, Throughput and Maturity.

	Storm	Storm -Trident API	Spark Streaming	Samza	Flink
Streaming Model	Native	Micro-batching	Micro-Batching	Native	Native
Guarantees	At-least-once	Exactly-once	Exactly-once	At-least-once	Exactly-once
Fault Tolerance	Record ACKs		RDD checkpointing	Log-based	Checkpointing
Latency	Very low	Medium	Medium/High	Low	Low
Throughput	Low	Medium	High	High	High
Maturity	High		High	Medium	Low

2.4 DATA VISUALIZER

2.4.1 *ElasticSearch (ES)*

ES appears as an additional framework for monitoring the streaming data, where it is possible to perform search and analytic operations. ES is based in another Apache framework, *Apache Lucene*, it is focused in scalability and therefore it presents a distributed architecture. Being a near real-time search engine, ES receives the data and soon it becomes researchable, this latency usually takes just one second [104, 105].

There are several core concepts in ElasticSearch such as [104, 105]:

- **Cluster**: As a general definition, a cluster is a set of one or more nodes (or servers). In ES, a cluster also holds the entire data and allow indexing and searching in all nodes. The cluster is identified by a name and it is this name that will be the connection between all the nodes. It is possible to have a cluster with a single node as well it is possible to have several clusters if each one owns a unique name.

- **Node**: The description of a node is as simple as it is a single server, and in this scenario, it stores data and is a fundamental part of the indexing and searching abilities of a cluster. As said before, a cluster with a single node can be enough when the system complexity is low, however, when the system should have an higher fault tolerance, a multi-node cluster should be implemented.

- **Index**: An Index is a collection of documents which have similar attributes. It is identified by a name which is used to refer to it when indexing, searching, updating or deleting documents. Built on top of Apache Lucene, an index can be compared to a database in a relational database, yet, an index in ES can be several indexes in Apache Lucene when using shards or replicas.

- **Document**: A Document is a basic unit to be indexed to an index and also the main entity in ES. In a SQL world, it can be seen as a row in a database table. These documents, from the point of view of the client, are JSON objects, likewise, the ES documents are stored as Lucene documents. Inside an Index, it is possible to store as many documents as the client wants.

- **Shards**: Shards allow to scale horizontally, that is, ES stores data in several physical Lucene indices, which are called shards and this is possible since the cluster allows to store volumes of information that overpass the single server processing abilities. Shards also provide distributed/parallelization of operations which increases the performance/throughput of the system. The number of shards is defined when creating the index and cannot be changed posteriorly. All this process is called sharding.

- **Replicas**: Although sharding allows a horizontal scalability and distributed processing, any shard still is susceptible to failure or even disappear. So, replicas of the shards are used to create a failover mechanism, these replicas are just copies of the index's shards. Replication is used as a failover mechanism so, a replica is never allocated in the same node as the shard. Replicas also increase the throughput since the searches can be executed in all replicas in parallel.

On the last update, the concept of Type was deprecated. An Index was seen as a database in a SQL world and a Type was seen as a table, however, in a relational database, a column with a certain name could not be associated with another column in another table with the same name. In ES, the same field name in different types must have the same mapping/definition. So, the type concept became deprecated and indices became completely independent from each other [105, 106].

This search-engine presents some major properties which make it suitable to be implemented in distributed systems, *per se*:

- Speed: A fast indexing allows to understand the changes in data almost as they occur. It is possible to leverage and access all of the data as it is indexed. The full-text-search and inverted index which are provided by ES increases the searching operation speed.
- Scalability: ES can work in single node mode or with hundreds of nodes being used to store huge amounts of data.
- High Availability: The ELK Stack (composed by ElasticSearch, Logstash and Kibana) was built to offer scalability but also to be always available, being able to scale vertically but more important horizontally, promoting the high availability of the whole stack.
- Resiliency: Sharding and Replication provide a failover mechanism if needed nevertheless ES provides an architecture without a SPOF (single point of failure) which detects failures to maintain the cluster and its data intact and available.

2.4.2 *Kibana*

Kibana is an open-source visualization interface which provides a visualization layer to the Elastic Stack. Kibana is used to search, view and analyze data and, more recently, Kibana allows to create certain alerts about the data. The interface is web-based and has a huge variety of graphs, charts, histograms or maps to visualize the indexed data. This web UI provides easy configuration settings to set a dynamic dashboard which will enable a visualization in real-time of changes in data or in the ES queries [107, 108].

Kibana has also analytic abilities, that is, it can perform mathematical operations on the datasets. The X-pack, a recent update to the top layers of the Elastic Stack, can be integrated into Kibana and has machine learning capabilities, promoting the lookout for patterns in the data, adding even more features to the interface [107, 108].

The near real-time indexing and distributed search engine, integrated with a powerful visualization interface as Kibana, encourages the creation of a real-time decision system.

2.5 THE HUMAN HEART

2.5.1 *Hearts' Anatomy*

The human heart has been widely studied and is considered the main human organ, it can be decomposed in two separate pumps. The left side of the heart pumps enough blood to the peripheral organs while the right side of the heart pumps the blood through the lungs. The overall anatomy of the heart is composed of four cavities, each of the left and right pumps have two cavities, the atrium (top cavity) and the ventricle (bottom cavity). The atrium is the primer pump however it is weaker than the ventricle, it makes the blood flow through itself and the ventricle. On the other side, the ventricle has a harder job, it needs to have the pumping force to expel the blood to the pulmonary circulation (by the right ventricle) or the peripheral circulation (by the left ventricle) as seen in the Figure 16 [2, 6, 109, 110].

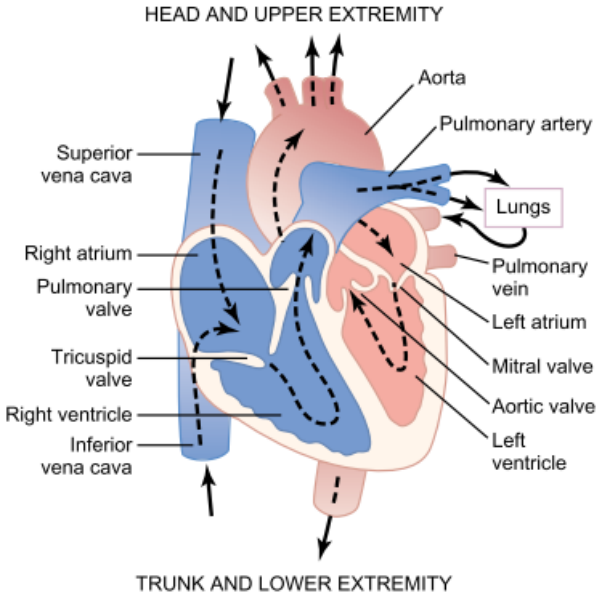


Figure 16.: Structure of the heart. Obtained from [2]

Heart contractions' create the cardiac rhythm, for which it is sent action potentials throughout the heart muscles. The heart has three major cardiac muscles: atrial muscle, ventricular muscle and specialized excitatory and conductive muscle fibers. The first two muscles work as the skeletal muscles except for the duration of the contractions which are longer. The specialized excitatory and conductive muscle fibers have a lack of contractile fibrils and therefore they act differently from the other two muscles, however, they present an automatic rhythmical electrical discharge providing an excitatory system which controls the rhythmical beating of the heart [2, 6, 109].

The Cardiac Cycle represents all the cardiac events that occur from one heartbeat to another heartbeat. This cycle starts with the generation of a potential action in the *sinoatrial node*. This node is placed on the right atria near the superior vena cava and, from here, the action potential travels to the atria and then to the atrioventricular valves and in the end to the ventricle, the transference of the action potential between the atria to the ventricles has a delay of more than 0.1 second which allows the atria to contract earlier than the ventricle making the blood to flow to the ventricles before the ventricular contraction starts [2, 6, 109].

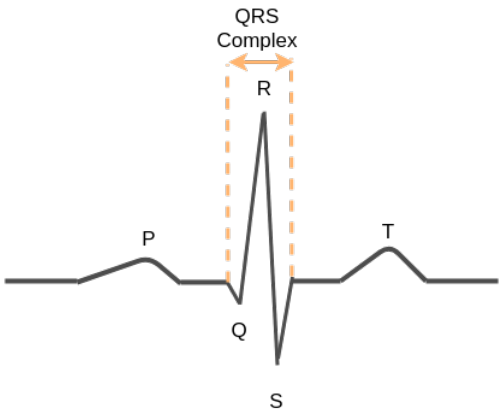
In order to monitor the heart, there are several tests that can be performed, such as echocardiograms, electrocardiograms, blood tests and blood pressure monitoring, stress tests, cardiac catheterization and even cardiac computerized tomographies or cardiac magnetic resonance imaging. This master thesis will only focus on electrocardiograms [111, 112].

2.5.2 *Electrocardiogram (ECG)*

The ECG is a record done with the aid of electrodes to monitoring the bioelectric signals. It is the most common test for heart conditions and it shows the heart rhythm and provides enough information to know if the heart muscle is damaged or short of oxygen. The electrodes are placed in opposite sides of the heart and record the action potential that spreads to the adjacent tissues surrounding the heart. Typically, ECG's are done with 12 electrodes carefully placed [2, 3].

The Figure 17a presents a schematic representation of a normal sinus rhythm present in a ECG. A normal sinus rhythm has a P wave, a QRS complex and a T wave, often the QRS complex can be seen as three different waves, the Q wave, the R wave and the S wave (Figure 17b). These waves are formed based on the continuous depolarization and repolarization of heart's cavities which are represented by the contraction and relaxation of the heart [2, 3, 113].

Although many conditions can be found in an ECG, this thesis will focus on the classification of arrhythmias which can be diagnosed by an ECG.



(a)

P wave	Depolarization of the atria
QRS complex	Depolarization of the ventricular myocardium
T wave	Rapid ventricular repolarization

(b)

Figure 17.: Ideal ECG with key features; (a) Schematic representation of normal sinus rhythm; (b) Electrocardiogram features, how each feature is generated [3].

2.5.3 *Arrhythmia*

Heart malfunction can be induced by the presence of abnormal heart muscle, however, some of the most distressing malfunctions occur because of the abnormal rhythm of the heart. There are three general types of abnormal sinus rhythm, *Tachycardia*, *Bradycardia* and *Sinus Arrhythmia* [2], however, there are many other specific types of arrhythmias which depend on where the arrhythmia occurs as well as why. The list presented below exhibits the specific groups to which specific arrhythmias belong [2, 113]. It is based on Guyton and Hall [2], it may diverge from others authors.

- Arrhythmias originated by blocks on the conduction pathways;
- Premature Contractions;
- Paroxysmal Tachycardia;
- Ventricular Fibrillation;
- Atrial Fibrillation;
- Atrial Flutter.

Tachycardia

When the heart rate of an adult person is faster than 100 beats per minute, its rhythm is defined as tachycardia[3, 6, 114]. There are several causes of tachycardia such as [2, 113]:

- Increased body temperature: for each degree Celsius, the heart's rhythm increases 18 beats per minute until the temperature reaches 40.5°C, after this temperature, the

heart rate tends to decrease since the heart muscle starts to be debilitated. This increase happens because temperature increases the metabolism of the sinoatrial node and consequently the heart rate.

- Stimulation of the heart by the sympathetic nerves: there are many motives for the sympathetic nervous system stimulate the heart, blood loss induces the stimulation of the heart and thus, the increase of its rhythm.
- Toxic conditions of the heart: the heart can have defects and these defects tend to influence the heart rhythm. When the myocardium weakens and the heart does not pump enough blood, the sympathetic nerves reflex to increase the heart rate.

Bradycardia

Bradycardia reflects a slow heart rate, in an adult, this means a heart rate below 60 beats per minute. This condition is analyzed accordingly to the adult background since athletes have a larger heart and stronger than a normal adult, as result, when at rest, their heart rate is usually below 60 beats per minute. One of the main causes of bradycardia is the Vagal Stimulation, any reflex that stimulates the vagus nerves tend to give a parasympathetic effect to the heart. This reflex causes the release of acetylcholine at the vagal ending in the heart and this reflex can be so powerful inducing the heart to stop for 5 to 10 seconds [2, 6, 113–115].

Sinus Arrhythmia

Sinus arrhythmia is diagnosed when the ECG presents an irregular cardiac rhythm which increases during inspiration and decreases during expiration. It may be caused by circulatory conditions that affect the sympathetic and parasympathetic nerve signal to the sinoatrial node. Usually, sinus arrhythmia is detected using a cardiotachometer [9], where is exhibited the difference in heart rate when the patient is breathing normally and when the patient is breathing deeply [2, 6, 113–115].

Premature Contractions

The contraction of the heart before its normal contraction time is known as *Premature Contraction*. It is usually due to *ectopic foci*[10] in the heart, sending abnormal electrical impulses at different times during the cardiac cycle [2, 6].

[9] In 1929, Ernst P. Boas designed the cardiotachometer to "take continuous records of the heart rate for periods of hours or days"[116].

[10] *Ectopic Foci*: abnormal conduction sites which led to abnormal conduction pathways within the heart, usually are masked by the higher electrical activity from the sinoatrial node [117].

According to Guyton and Hall [2], inside the premature contractions are the Premature Atrial Contractions, the Premature Ventricular Contractions and also the A-V Nodal or A-V Bundle Premature Contractions (the last one also known as Supraventricular premature contraction).

For the purpose of this master thesis only these three specific arrhythmias will be further explained.

- **Atrial Premature Contractions**: These contractions are characterized by the P-R interval, caused by the P wave which starts sooner than normal. After this beat, the next interval (between the premature beat and the normal sinus beat) is slightly prolonged. This type of premature contraction can frequently occur in healthy people, occurring usually in athletes, however, bad habits such as smoking, alcoholism, lack of sleep and even ingestion of more of the normal dose of caffeine can lead to atrial premature contractions (Figure 18a) [2, 6, 113–115].

- **A-V Nodal/A-V Bundle or Supraventricular Premature Contractions**: In this type of contraction the P wave disappears from the electrocardiography record, overlapping the QRS-T complex. This occurs since the electrical impulse traveled to the backward of the atria at the same time it was traveling forward to the ventricle. Its causes are similar to the Atrial Premature Contractions (Figure 18b) [2, 113].

- **Premature Ventricular Contractions (PVC)**: These premature contractions alter visibly the ECG. The QRS complex becomes longer and has a high voltage, the T wave presents the exact opposite potential of the QRS complex as it can be noticed in Figure 18c. This condition can happen due to many reasons similar to the other premature contractions but can also be a result of emotional irritability, heart attack or ischaemic areas in the heart. The presence of PVCs can induce the development of Ventricular Fibrillation, which can lead to death [2, 6, 113–115].

2.6 Related Work

The way big data is being handled in healthcare is not as efficient as expected. However, some progress is being made towards the efficiency of those systems. As days go by, it becomes clearer the role of big data in healthcare. From personal medical records to clinical trial data, radiology images to biometric sensor readings, data is everywhere and must be dealt with great caution. Virtualization and cloud computing is helping the development of new systems for capturing, storing and manipulating these enormous volumes of data [118]. As data keeps increasing, so are the systems and the complexity of the analytic

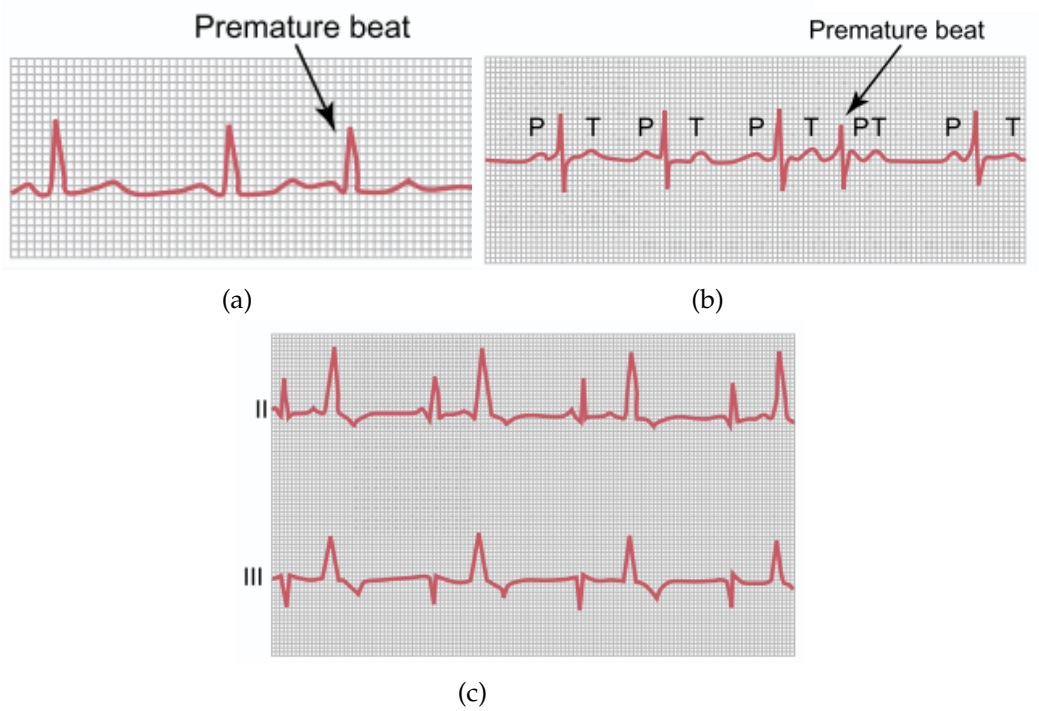


Figure 18.: Premature Contractions present in an ECG; (a) Atrial Premature Contraction (lead I); (b) Supraventricular Premature Contraction (lead III); (c) Premature ventricular Contraction (lead I and II). Obtained from Guyton and Hall [2].

techniques. The mention of unstructured or semi-structured data keeps surfacing in healthcare, the limitations of treating this type of data are diminishing but it still keeps this field challenging and of major importance [119].

In their article, Najafabadi et al. [120] addressed all the challenges that deep learning may encounter, but the authors also accentuated the advantages of using deep learning into massive amounts of unsupervised data for learning. DL can be then used for some major questions in Big Data such as extracting complex patterns from data or even simplifying discriminative tasks.

Furthermore, arrhythmia detection and classification have been studied for several years and electrocardiograms became the main data source for these studies and it is becoming extremely important to address these conditions once they are becoming more dominant as death factors.

2.6.1 *Arrhythmias: feature extraction, detection and classification*

The feature extraction of ECG signals is one of the most important tasks in automatic analysis of an ECG whereas it can lead to the detection of abnormalities and efficient pro-

gnosis. Therefore it is essential to understand how this process works and which features have demonstrated to be of major prominence.

Karpagachelvi et al. [121] article features several other papers and authors whose main goal was to create an algorithm or a method to extract the main features of ECG's in which almost all the methodologies are studied. This survey advocate that most of ECG's information that can be used clinically is generated from the intervals and amplitudes defined by some characteristic wave peaks and their time duration. It is also mentioned that time domain methods weren't as successful as expected once it wasn't the most correct method to analyze all the features. Since some important features weren't being handled it became important to analyze these bio-signals in their frequency representation.

As Karpagachelvi et al. [121] described in their article also Umer et al. [122] perceived that ECG's most significant features are the P, Q, R, S and T waves. Umer et al. [122] present a feature extraction methodology based on *Windowing*. In which the authors use this method to select the previous waves and the windows are based in the RR-intervals. The ECGs used for this work were obtained in the MIT-BIH Arrhythmia Database, from where the Lead I was chosen and it was also used an ECG data simulator. All the features not directly retrieved from the signal were calculated using the formulas proposed in the paper. These features are, namely, the RR-interval, QRS duration, QT interval, QT corrected interval and vent rate. Umer et al. [122] noticed that the QT corrected intervals enhances the detection of patients with risk of ventricular arrhythmia. The authors were able to obtain good results in both databases where the sensitivity reached for P, Q, R, S and T waves were, respectively, 96.72%, 97.12%, 96.04%, 97.32% and 97.56% and for specificity the results were 83.47%, 99.61%, 99.80%, 99.68% and 80.41%. The process of feature extraction has a bigger problem that begins with the detection of the P and T waves since these waves are known to have low amplitudes, low signal-to-noise ratio, an amplitude and morphology which may vary and there is even the possibility of the overlapping of the P wave with the QRS complex. These conditions are shown in the results obtained since the values of specificity are the lower for these waves.

Another methodology is described by Peshave and Shastri [123], the authors present the pivotal points of an ECG as the RR-interval and the amplitude of P, R and T waves. Using wavelet transforms and threshold method they discuss the detection of three types of arrhythmia, namely bradycardia, tachycardia and premature ventricular contraction. For this study, the authors used the MIT-BIH Arrhythmia Database using all the leads in which the signals were recorded. This article promotes the use of wavelet transforms [124], which have the ability to analyze non-stationary signals (signals that change over time) such as ECG's. After testing 5 wavelet transforms to recognize which was the proper wavelet to use the authors found that the Daubechies of order 4 performed better than all of the other wavelets. On the other side, Peshave and Shastri [123] used also a thresholding method,

where it was used a combination of filtering and threshold. On the first stage was applied a high pass filter, based on the fact that QRS complexes are high-frequency components. After that, a threshold is applied performing as a denoising filter. The threshold used was a global thresholding and it showed great accuracy in detecting the QRS complexes and their amplitudes. The authors claim that global thresholding can be used to detect missing beats or even inverted QRS complex, which may help in diagnosing premature ventricular contraction (PVC). Although the results weren't as great as the ones achieved by Umer et al. [122], the authors obtained a sensitivity of 80% and 85% using both methods, wavelet transforms and thresholding.

Rodríguez et al. [125] article's, "Feature extraction of electrocardiogram signals by applying an adaptive threshold and principal component analysis", is based on a thresholding method allied to Principal Component Analysis [126]. After ECG filtering, the author use differentiation to find the minimum slope of the ECG signal and the high slope points, after this stage, a Hilbert transform is applied which relies on Fast Fourier Transform [127]. In this step, adaptive thresholding can be employed for detecting the R wave peak, during all the iterations, the minimum and maximum limits are updated. To withdraw the relevant features it is then used a PCA technique. With this work, the authors were able to obtain a sensitivity of 96.28% and a precision of 99.71%.

Additional studies based in Wavelet Transform were made by Wang and Smieee [128] which presented a robust real-time algorithm for feature detection. In the preprocessing stage, the signal is divided into sections of 20 seconds. The authors chose the Daubechies Wavelet to transform the signal since its shape is similar to QRS complex and can preserve its energy which makes possible to find the P and T wave since their frequencies are the lowest frequencies found in the signal. The wave detection is made using the wavelet but also a moving window centered with the wavelet detection. The results were promising since the overall sensitivity was higher than 99% and it became obvious that this methodology could be comparable with other reports using Pan Tompkins algorithm [129].

Another approach is presented in the article *"Cardiac arrhythmia detection using deep learning"* [130], using deep convolutional neural networks, the authors proposed a deep learning framework to classify three different cardiac conditions. In order to do this, three different conditions in ECG waveforms from MIT-BIH Arrhythmia Database were used, where an AlexNet architecture [24] is used to extract features and these features are then fed to a back propagation neural network to classify the beats. Firstly, the QRS detection was performed with the help of the Pan-Tompkins algorithm. Then, 200 sample points after the R-peaks are extracted and converted to a 256x256 binary images and later in RGB images to use as an input of an AlexNet architecture. The framework obtained high-performance results, exhibiting a 98.51% correct recognition, highest value, and an accuracy of 92%. These res-

ults proved that the framework proposed can be an efficient automatic cardiac arrhythmia detector.

Moreover, the paper *"Cardiac Arrhythmia Classification Using Neural Networks with Selected Features"* [131], shows a novel approach to classification of arrhythmias applying Neural Networks. For this work, the authors have chosen the UCI arrhythmia database which presents 452 instances and 279 attributes. To shrink the feature vector provided by the data set, to reduce the database and speed up the inference procedure, the authors used the algorithm of Correlation-based feature subset selection (CFS)[132]. The classification phase was made recurring to an incremental back propagation neural networks and the Levenberg-Marquardt (LM) algorithm [133]. The CFS algorithm was implemented using the WEKA software [134] whereas the classification algorithms were implemented using Alyuda Neurointelligence software [135]. The results obtained in this paper showed greater outcomes when using CFS and LM algorithms and the authors conclude that greater results could be accomplished using the set of reduced features with a final decision made by the doctors. This paper concludes that well-chosen features may decide the outcome of the overall project.

A great amount of these papers refers to Feature Extractions as one of the most important steps to reach good results with machine learning or deep learning. This stage gives the correct input to properly analyze an ECG record. Although, when dealing with deep learning this process may be intrinsic to the whole process as stated by Isin and Ozdalili [130]. In [136] the authors have created a mechanism to detect atrial fibrillation using a Mexican Hat Wavelet to extract the R-peaks and therefore, to extract the features from ECG's.

One of the latest advances in the DL field was made by a team in Stanford [137], in which the authors proposed a 34-layer deep learning model to detect arrhythmias in 30-second length ECGs and to classify 12 types of arrhythmias. The dataset used consisted of a single-lead ECG signal with a sampling frequency of 200Hz and a sequence of annotations for every second of the ECG. For optimization of the deep learning model, the authors used residual connections and batch normalization. The dataset adopted was created from scratch and consists of records of 30,000 unique patients who had to use the Zio Patch monitor[138]. As for test set, they used 336 examples, annotated by board-certified cardiologists. The results obtained surpassed the cardiologists' results, the size of the dataset and the depth of the deep learning model are the focal points of these results.

Two different network architectures were proposed by Zihlmann et al. [139] for classification of Electrocardiograms. A Convolutional Neural Network (CNN) and a Convolutional Recurrent Neural Network (CRNN) were built having as inspiration the works of Lipton et al. [140], Bashivan et al. [141] and Cakir et al. [142]. The main focus of this article was the classification of Atrial Fibrillation using the data set provided by *Physionet/Cinc Challenge 2017*. When training these networks, the authors were faced with the overfitting prob-

lem. In order to reverse it, they performed data augmentation which, according to Simard et al. [143], can be used as a regularizer to prevent the overfitting. The classification was performed under the assumption of four major labels, ”Normal rhythm”, ” AF rhythm”, ”other rhythm” and ”noisy recording”. The results were promising and data augmentation proved to be able to increase the accuracy of both architectures, with more emphasis on the CRNN architecture which reached an accuracy of 82.3%.

As other articles here presented, Pyakillya et al. [144] proposed a deep learning solution for classifying electrocardiograms, where the first layers of the network architecture are employed as feature extractors and the last layers are fully-connected layers to reach a final decision. The authors, in comparison with the latter article, also have only four classes, ”normal sinus rhythm”, ”arrhythmic”, ”other rhythm” and ”noisy”. Although the network built did not present much of computational complexity, architecture or optimization methods, it did outperform other results and reached an overall accuracy of 86%. These results are encouraging since one of the main goals of this article was to employ machine learning algorithms without manual feature extraction.

Both articles from Warrick and Nabhan Homsi [145, 146] rely on an approach of a Convolutional plus Long Short-Term Memory (LSTM) Neural Network. In the first article [145], the authors used a model composed of two main components, a representation learning mechanism done by the CNN and a sequence learning mechanism provided by LSTMs. The first convolutional layer is used to extract local and discriminative features and then three LSTM layers are used to encode the output from the latter layer. The output of the whole model comes from a dense layer applied after the LSTMs. Since these articles were done on the *Physionet/Cinc Challenge 2017*, they used the same nomenclature as the two latter articles [139, 144]. The evaluation of the model was based on the F1-score [147] and the overall results reached the 80% of effectiveness classifying the data. The second article from these authors presents the same neural network model, however, the model suffered experiments in order to improve its performance. Another upgrade to the first model was the creation of an ensemble classifier. According to Javadi et al. [148] and Ju et al. [149], the ensemble classifiers mix all the outputs from the classifiers to reach the higher accuracy and to get a reliable estimation of the generalization error. With these updates to the model and the overall system, the authors were able to overpass their old results and got an overall result of 82%.

Another approach to classifying the atrial fibrillation was built by Limam et al. [150]. As Zihlmann et al. [139] and Warrick and Nabhan Homsi [145], also Limam et al. [150] used a Convolutional Recurrent Neural Network (CRNN) to classify electrocardiograms. Using the same classification methodology, the authors decided to solve the problem with multiple binary classification methodologies, that is, each model classifies a class against all the other classes and the classified samples are discarded from the next model up until

the last model, in this scenario, three models were built. The results were obtained model by model and against another methodology where a Support Vector Machine was added to the neural network. The CRNN plus SVM outperformed the results obtained only with CRNN.

Swapna et al. [151] proposed the classification of cardiac arrhythmia using ECGs with minimal data pre-processing. Following the same pattern as the latter articles, the authors presented some hybrid neural models, composed by convolutional layers, recurrent layers, LSTMs layers and GRU[11] (gated recurrent unit). The authors performed a beat-to-beat classification of the MIT-BIH arrhythmia database, from which each beat had 361 data points and the beats classified as other than normal or some type of arrhythmia were discarded. Using a five-fold cross-validation technique, the authors were able to obtain an accuracy of 83.4% in classifying the signal as normal or abnormal (arrhythmic) with the CNN-LSTM model.

The MIT-BIH arrhythmia database was also used in the works of Savalia and Emamian [153] and Jun et al. [154]. The first article proposes the classification of electrocardiograms using multi-layer perceptrons (MLP) and convolutional neural networks. Besides using the MIT-BIH dataset, it was also used data from *kaggle.com*. The MIT-BIH data was used to train and test the MLP model and the Kaggle dataset for the CNN model, reaching accuracy values of 88.7% and 83.5% respectively. Both models classify the ECGs differently, while the MLP model only classifies as normal sinus and arrhythmia, the CNNs model classifies the signals as "arrhythmia, normal sinus, second degree AV block, first degree AV block, atrial flutter, atrial fibrillation, malignant ventricular, ventricular tachycardia, and ventricular bigeminy" [153]. On the other side, Jun et al. [154] pretended to detect premature ventricular contractions present in electrocardiograms using deep neural networks. In this work, the authors were able to reach an accuracy rate of 99.41% and a sensitivity of 96.08%.

Besides all of the methodologies described it is important to emphasize the first phase of analyzing an ECG, the *Preprocessing*, which is described below.

2.6.2 *Preprocessing of an Electrocardiogram*

Before being able to extract features from an ECG it is fundamental to preprocess the signal. Since ECG is recorded by placing surface electrodes on the human chest as previously mentioned, there are several interferences that may occur such as coughing which can lead to the displacement of the leads and the signal becomes noisy [123].

Rodríguez et al. [125] used a band-pass Butterworth filter of order 6 to maximize the QRS complex and also to remove muscular noise from the ECG signal. The band-stop frequencies were set from 5 to 15 Hz, where 5 Hz and 15 Hz are respectively the starting

[11] The Gated Recurrent Units are an altered version of LSTMs, with fewer parameters [152].

and stopping frequencies. The method used enables the removal of the high frequencies and baseline wander but it also inhibits the P and T waves and enlarges the QRS complex.

To process the ECG's, Wang and Smieee [128] removed the baseline drift using median filters. This procedure implies to remove each wave from the ECG leaving it with only the baseline, which is then removed from the signal itself.

Vishwa et al. [155] knew that the first step to truly analyze an ECG was to remove the baseline noise. In this article, the authors appealed to the Fast Fourier Transforms and also its inverted transform to remove the low frequencies and to restore the ECG record in order to remove the baseline noise.

Isin and Ozdalili [130] proposed a Deep Learning technique to feature extraction, however, the preprocessing technique was still hand-made in which the DC noise was removed by the subtraction of the mean of the ECG signal. To remove the high frequencies it was used a low-pass filter and the low frequencies are mostly caused by the respiration of the patient which is filtered using a high-pass filter. It was also important for the authors to suppress the power line interference to which they used a band-pass filter to remove the 60Hz frequency.

2.6.3 *Cloud-based and IoT systems applied to Healthcare and Electrocardiograms*

Xia et al. [156] presented a cloud-based system for ECG monitoring and analysis in real-time. The system was built for mobile devices and web browsers and was designed to unravel the time-consumption problems of ECG collected data. As for the monitoring and analysis, the authors implemented algorithms to evaluate and enhance the ECG data, as well as to extract the most relevant features. The proposed system use case was built over a mobile phone sending the ECG data to a web server where the analysis was performed in real-time. The authors also mention the importance of these systems to health in general and the simplicity of their system, which could be used by patients before sending their records for the physicians for assessment and by physicians to help in the diagnosis of the data and the extraction of useful features.

Using Cloud services and an API, the SenseMyHeart application [157] aimed to tackle the difficulties of mobile monitoring and use of Linux dedicated tools. Focusing on the integration of tools already available, as PhysioToolkit [158] - a library for processing and analysis of physiological signals that is used via command line requiring basic knowledge of Linux-based systems' bash interface - the authors proposed a computing infrastructure, web and mobile based, providing flexibility to researchers and app-developers. The overall goal was to contribute with a "standardized computational backbone" to other projects that intend to study physiological stress or measure Heart Rate or Heart Rate Variability. Although a Windows application was created to be used as an off-line analysis system, the

Android prototype application was built to work in real-time as a cardiovascular ambulatory monitoring. Connecting directly with the sensor via Bluetooth, the data was then passed to the web service where it is analyzed and several indicators plotted. The use of a cloud-based service allows the system to present good performance results on the mobile settings. Relying on a stable internet connection, the server-side infrastructure needs to be cautiously created.

Ahmed [159] tried to create a system using IoT sensors to predict the number of patients that may suffer a heart attack. Adopting machine learning as the technology to be used for prediction, the author separated the data into three types: fixed, such as sex and age; periodic, like blood sugar or cholesterol level, which is measured by a period of a day or week; and live data, which encompasses blood pressure and heart rate. The first type of data was collected via a web-based platform, while the other two were collected with wearable and IoT devices for home monitoring that can then sent the data to the remote cloud server. On the other side, for machine learning purposes, the author used an example dataset from UCI Machine Learning Repository, which was restructured to a format that could be obtained from the previously explained healthcare cloud platform. After applying the k-Nearest Neighbor algorithm, the results seemed to be hopeful, although the accuracy obtained reached 96%, the number of false negatives were still higher than expected. The author also highlights that, in order to increase the accuracy of the model, the number of false negatives should be reduced.

Other works are worth noting, such as the one developed by Jimenez and Torres [160], which presents an IoT-aware system for healthcare using the Clayster IoT platform and Android as a gateway for transforming data. Nigam et al. [161] created a system that would monitor ECG waves and alert the doctor or the relative for normal or concerning arrhythmias, using the concept of IoT. Onto the same subject, Dineshkumar [162] proposed an IoT system for healthcare where blood pressure, heart rate and humidity sensors were added to the IoT agent. Afterward, data is sent to the cloud where Hadoop's MapReduce is used for its processing. The alert system is based only on a threshold and once the arrhythmia is detected it sends an alert. The connection between Apache Spark and Deep Learning is discussed by Gupta et al. [163] and the results were promising. The cascade learning, that created the true connection between the two tools showed improvements in the accuracy results and enhanced the proposed methodology. On the other side, Pandey et al. [164] presented an autonomous system for analyzing ECG data based on mobile and Cloud computing that showed also promising results, yet, the authors were compromised to reduce the costs of Cloud Computing.

2.7 PERSONAL DATA PROTECTION IN HEALTHCARE

In an era of Big Data, the threats to data protection continue to increase. On the year of 2016, the European Union approved the General Data Protection Regulation (GDPR) [165], which is applied to all sectors, to all "personal data" and to all the companies who control or process this data, also providing the protection of this data throughout its lifespan. While the GDPR is applied horizontally to several sectors, in the United States the laws to protect healthcare data are regulated by the Health Insurance Portability and Accountability Act (HIPAA)[166] which sets the standard for sensitive patient data protection [167]. Canada, United Kingdom, Morocco, Russia, India, Brazil and Angola have also their own guidelines to secure healthcare data [168].

The compliance with these standards is done by improving the security around data but also by giving the patients and/or customers all the information about how their data is being collected and to what ending.

According to the Article 5 of the GDPR, personal data in healthcare can only be collected for specified and legitimate purposes, has to be processed in a transparent manner, lawfully and fairly and must ensure appropriate security of this data. This data must be relevant and limited to what is necessary for the overall purpose for which it is processed and it also needs to be accurate and to be up to date. Still, according to this article, the identification of the subject should only be kept while needed. It is also stated that all this data should be controlled by a controller which is responsible for the data and must comply with all this guidelines [169].

Dealing with personal data in cloud environments can be a tricky task since it raises issues concerning privacy and security breaches. It is also important to understand which type of data needs to be protected and its stages in these environments. To this end, data should be protected in the three stages it may be found such as at rest, in use and in motion.

- Data at rest: data is known to be at rest when it is already stored. In this stage, data can be secured by conventional defenses such as firewalls. Nevertheless, since this data is sensitive other additional steps must be considered, likewise, the encryption of hard drives or the individual storing of data are two of the available approaches [170, 171].

- Data in Use: When using data, it should be accessible to those who want to use it. This stage is vulnerable since the more clients/servers need to access the data the more exposed the data might be. The protection of this data must be done through some authentication mechanisms and to control the access provided to the users [170]

- Data in Motion: When in motion, the data is considered to be the most vulnerable. To protect this data is important to implement network security solutions on top of all

the measures already implemented on the other two stages. Encryption and masking of received and sent messages is also vital to protect data in motion [170, 171]

In accordance with Abouelmehdi et al. [168], authentication, encryption, data masking and access control are four of the most broadly used mechanisms to ensure the safety of healthcare data. The authors advocate the use of a proactive and preventive approach to address all these security issues and breaches in healthcare preventing conundrums.

2.8 SUMMARY

Data in healthcare is being generated faster than it can be processed and analyzed. It begins to be of major importance to disclose and understand this data in a visible and accurate way. Cardiology, in particular, presents a high death rate and one major condition are arrhythmias. Arrhythmias can develop into difficult situations and result in the patient death if not carefully diagnose and the sooner the patient gets the right treatment, the sooner the disease is controlled.

Deep learning has been greatly improved over these last years since GPUs have proven their value in this area. With this, many frameworks have emerged and owned their space as Deep Learning frameworks. Their open-source approach has exceedingly increase their performance with the integration of many features from several developers.

With the same approach, a large spectrum of data processing and streaming frameworks are available. The scalability and fault-tolerance of such frameworks are presented as one of their biggest achievements and the `at-least-once` and `exactly-once` mechanism gives the guarantee that data will not be lost and will be processed, which in the medical field is highly important.

The real-time monitoring of electrocardiograms seems to fill the void regarding this medical field. Moreover, Deep Learning came to fulfill the accurate classification of data providing a weight-lifting from this human task. When combining both areas, it is possible to create a highly scalable decision support system, delivering a solid system to the healthcare environment. It is important to emphasize the importance of securing these systems, the healthcare environment has been the target of many security breaches, ergo, these systems must be secured for as long as the data must be kept safe.

3

RESEARCH METHODOLOGY

This chapter presents a description of the research methodology employed onto the development of this master thesis. For so, the first section, 3.1, focuses on the concept of research methodologies and its advantages for the benefit of a project. The adjacent sections, 3.2 and 3.3, present the chosen methodology and the practical application of the methodology.

3.1 INTRODUCTION

Research is known as the activity that allows the understanding of a theory. In which the theory is an abstract or generalizing thinking that becomes interesting for the researchers [172]. Research is recognized as *"defining and redefining problems, formulating hypothesis or suggested solutions"* or even *"the pursuit of truth with the help of study, observation, comparison and experiment; the search for knowledge through objective and systematic method of finding solution to a problem"* [173].

In Science and Engineering but also *Computer Science (CS)* and *Information Technologies (IT)*, it is commonly used the *Design Science Research Methodology (DSRM)*. This methodology is based on the identification of a problem, the design and development of a solution and its evaluation [174, 175].The DSRM technique also made its way through the healthcare systems as proposed by [176], which proves that a well followed DSRM can be of major support for a well built Business Intelligent System for an Hospital. This approach seemed of significant value for this project and therefore, this methodology was adopted to reach the main objectives of this master thesis.

3.2 DESIGN SCIENCE RESEARCH METHODOLOGY - DSRM

The DSRM is in general a methodology that brings together the creative process used for creating new products and the evaluation of its benefit and usefulness, it can be defined as the action that leads to the design and creation of useful artifacts [172].

As proposed by [174], the DSRM follows a five step procedure, more recently discussed by [177]:

1. **Awareness of the Problem**: the researcher becomes aware of the surrounding problem, from reading about it or reaching other people facing the same problem. Having the perception of the problem will output a Proposal to solve the problem found and it will lead to a new research step.

2. **Suggestion**: In this step, the researcher has to create a *Tentative Design* or at least an overall idea of a solution for the problem. Therefore, this phase is known as the creative one, where a new functionality is imagined based on the available elements or on recent features or configurations. Since this step is mostly based on human creativity with the theoretical foundations in mind, it is expected that different researchers present different ideas to solve the problem as so, different Tentatives Design will be proposed.

3. **Development**: The Development phase is when the Tentative Design will be developed and implemented. An artifact should be developed, which can be an algorithm or a software, yet, the methods used for the development of the artifact do not need to be groundbreaking, the singularity should be in the design.

4. **Evaluation**: The artifact developed should be assessed according to the initial proposal although it may not specify the evaluation criteria it is usually implicit. The artifact must behave as the it was designed, and if not, the researcher should detail any discrepancy or how it truly behaves.

5. **Conclusion**: As a typical conclusion, this step presents the obtained results. Positive results or less positive results should be documented and conclusions should be retained from it. Facts or ”loose ends” can come from it but this knowledge can be applied by researchers in further researches.

Peffers et al. [175] use a similar approach of the DSRM in which the authors try to also summarize some of the reported literature. Drawing a parallelism with the above methodology, the article divides the *Awareness of the Problem* phase into two different steps, *Problem identification and motivation* and *Objectives of a solution*; on the other side, they join the *Suggestion* and *Development* phases into one step only, *Design and Development*; the proposed methodology also breaks the *Evaluation* phase into two sets, *Demonstration* and *Evaluation*; and for the *Conclusion* step, the authors rename it as *Communication*.

3.3 PRACTICAL APPLICATION

The application of the DSRM in this master thesis was very clear on the overall development and the several steps were recognized during the project's development.

The *Awareness of the Problem* became more evident after some research on the Cardiology field and also on the medical informatics field, where the lack of scalable, available and low-cost systems for monitoring and analysis of arrhythmias were identified as major problems. Without disregarding the first research, further researching was done to reach an *Tentative Design* where it was suggest the creation of a real-time monitoring system using deep learning for the analysis. The *development* step encompasses the integration of Kafka with a Kaa cluster and then the integration between Kafka and ElasticSearch and, on the other side, Keras on top of Tensorflow while integrating with Apache Spark. The Kaa IoT already supports HBase and Cassandra so, data can be stored for testing purposes in any of the integrated NoSQL databases. After having the architectures built it was time to test it and assess the results. To concluded the project, all the results were deeply analyzed. All the conclusions and steps performed until the final architecture were registered in this master thesis.

4

SYSTEM ARCHITECTURE

This chapter discloses the overall problem addressed by this master thesis and its challenges, proposing an approach to undertake these challenges and presenting the overall system architecture. Firstly, Section 4.1 presents an introduction to the problem. Section 4.2 presents the overview of the proposed approach to tackle the main issue. Section 4.3 exhibits the system architecture with all its parts and its stages, the retrieval and analysis architecture in Section 4.3.1 and the deep learning architecture in Section 4.3.2.

4.1 INTRODUCTION

As stated in Chapter 2, Big Data and Healthcare are two hot fields. The amount of data created each day in healthcare is increasing in volume, variety, and velocity. This data, when thoroughly studied allows extracting useful insights for the management of healthcare facilities and more important the wellbeing of its patients.

Health monitoring has been used since Boas et al. [116] created the first cardiotachometer and over time the use of new and improved sensing techniques only increased. As so, biosignals became of major importance to detect several health conditions with an outstanding accuracy. Using IoT technologies in this field can only improve this accuracy since it can collect all vital signals from different sensors such as electrocardiograms, blood pressure, body temperature and oxygen saturation sensors in real-time. IoT also allows analyzing this information and extract even more data from the sources. These processes can help in early diagnosis and it can even help to reduce medical cost [119, 178].

The possibility of building an IoT system based on big data technologies and on exponentially data analytics such as deep learning presents a remarkable opportunity for improving the quality of patient-based care and faster and reliable diagnosis.

On the other side, Deep Learning has been showing its potential in the healthcare field as previously reported and in the context of an IoT system it can boost further the analytic step, incrementing accurate data analysis and also providing insightful information about the patients' conditions faster than other ordinary techniques.

Notwithstanding, the main concerns to create a reliable system are focused on: i) the quick storage of all this data, commonly unstructured, from different sources; ii) the accuracy of the deep learning model; iii) its real-time capabilities, which should be humanly possible to visualize.

4.2 PROPOSED APPROACH

This master thesis is focused on the collection, processing, and analysis of bio-signals in a healthcare facility. The proposed solution encompasses three blocks: Data Acquisition, IoT Platform and Storage and Data Analytics and Visualization (Figure 19). These are the components the system must implement to be functional.

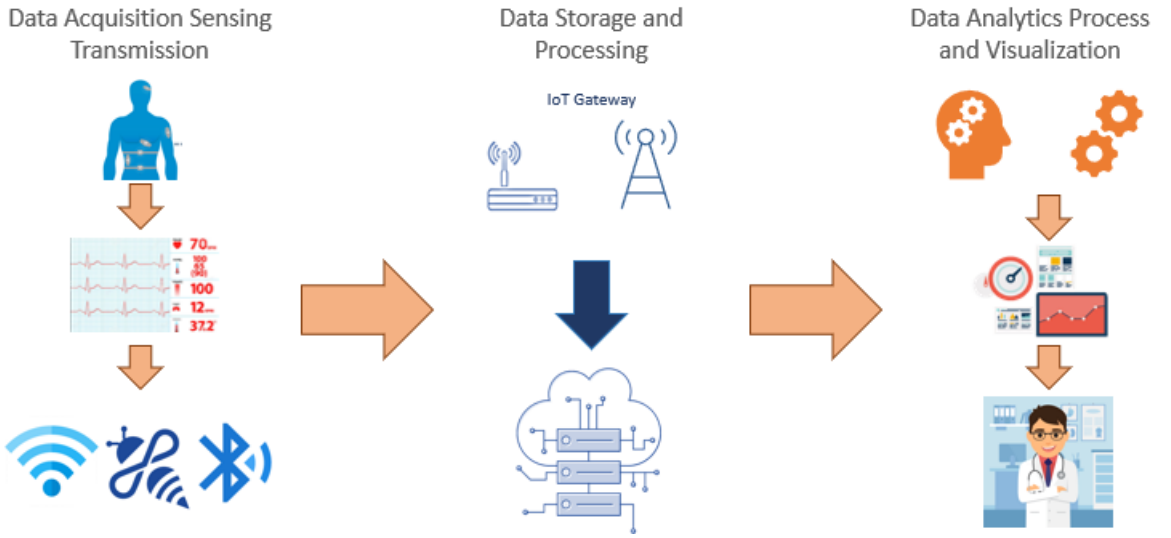


Figure 19.: Proposed prototype architecture represented by three blocks. The first block represents the data sources, the acquisition of the bio-signals. Secondly, the middle block represents the IoT gateway and platform as well as the cloud storage. The third and last block represents the analytic stage and the dashboard visualization of the obtained results.

Data Acquisition

The first block represents the bio-signals collected from the patients in the healthcare environment. In this architecture, the data source has the data in a format which can be rightly processed. However, normally there would be the development of an adapter for transforming the incoming data in a format that the IoT platform admits. On the proposed approach the data is sent directly from the sensors to an IoT gateway.

IoT Platform and Storage

The middle block portrays the IoT gateway as a middleware, establishing the connection between the sensors and the processing components. It should also manage all the tasks in the system and secure all the data that is passing through. As a middleware, the platform also creates the connection between the storage and the sensors, ensuring all data is fully saved. Since this block will be used for Big Data, a large-scale data storage is needed.

Data Analytics and Visualization

The last block should comprise the data analytics and also the visualization of both the results from data analytics and the real-time visualization of the data that is being received by the IoT platform. So, this block represents the deep learning for analytics and the data streaming for real-time visualization of the data.

4.3 System Architecture

The development of the whole project was based on the abstract architecture presented on Figure 19. Figure 20 represents the final architecture which is based on the frameworks previously analysed and also on the Kaa Project as IoT platform and Apache Hbase as the storage unit. Ergo, this architecture comprises the Kaa Project as the IoT platform; Apache HBase for storing the collected data; a deep learning stack with Apache Spark for data parallelism; Apache Kafka as a message broker between the IoT platform and Spark Streaming as well as between the Deep Learning Stack and ElasticSearch. Finally, Kibana is used as a dashboard visualizer for ElasticSearch.

The above architecture was designed using uniquely open-source frameworks, although there are proprietary frameworks the development of this whole project was focused in frameworks easily adapted for the purpose needed.

The Kaa IoT platform has proved to be suitable for a increasing number of sensors as well as users and has also proved to be able to provide some data protection using authentication and encryption mechanisms [7].

Regarding the use of Apache Kafka as a message broker, Section 2.3.2, presented all the advantages of a message broker like Kafka. Its scalability, fault-tolerance, low latency and exactly-once processing are presented as the major contributions this framework provides, being ideal to create pipelines between the rest of the components.

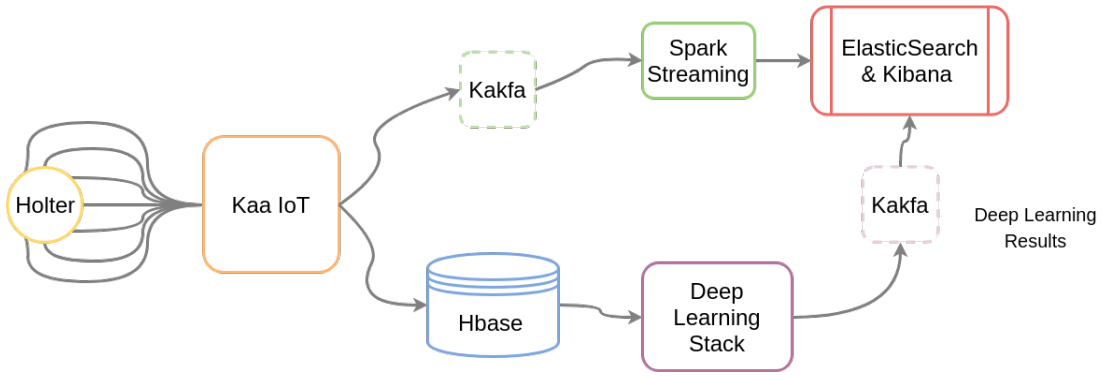


Figure 20.: System architecture for data collection, processing, analytics and monitoring. It comprises the Kaa IoT, HBase, Kafka, Spark, a Deep Learning Stack and ElasticSearch and Kibana for the last block.

4.3.1 *Retrieval and Storage Architecture*

The retrieval and storage blocks are based on Rei [7] which proposed the Kaa IoT Project as an IoT middleware, and Hbase as storage. This work has created the log appender for connecting the Apache Hbase to the Kaa IoT project and the benchmarking developed exhibited better results for HBase when compared to Apache Cassandra. Since the Kaa Project is based on SDK applications, the main concern in this stage was to develop an application for integrating on the data sources, here depicted as Holters, for sending the data to the IoT platform.

4.3.2 *Deep Learning Architecture*

The Deep Learning Stack block encompasses the fundamental components of this master thesis which are the classification and prediction of arrhythmias. This block shall comprise the best models for these tasks and should train the models with the correctly classified and predicted instances.

4.3.3 *Analysis and Visualization Architecture*

The last block comprises ElasticSearch and Kibana. ElasticSearch has proven its value along the years and the easiness of querying this framework for later visualize the data in Kibana is shown as an advantage. Moreover, ElasticSearch has a near real-time indexing of documents which in 1 second all data become accessible to be queried. ElasticSearch will receive both the results from the deep learning and the data streamed from Spark.

5

DEVELOPMENT AND IMPLEMENTATION

This chapter presents the implementation of the developed prototype and the motives that led to the decisions made during the development of this master thesis. Section 5.1 discloses all the decisions made throughout the course of this project, this section is divided in three subsections. Section 5.1.1 presents all the stages of creating the deep learning architecture and testing the models. Section 5.1.2 presents the decisions made for data streaming for each stage of the final architecture and the decisions for visualizing the biosignals. Moreover, the implementation of the whole system is presented in Section 5.2.

5.1 DECISIONS

5.1.1 *Deep Learning*

Data Selection

MIT-BIH Arrhythmia Database presents 48 records, where the last 23 records became online only in 2005. The 48 records were chosen from a set of over 4000 long-term Holters recorded in the laboratories of the late Boston's Beth Israel Hospital, now known as Beth Israel Medical Center, between 1975 and 1979. The first 23 records were chosen randomly from inpatients and the 25 other records were chosen from the same set yet to contain a diversity of uncommon but clinically important conditions. The two groups have different purposes since the first one is to serve as a representative sample of waveforms and artifacts which most of the arrhythmia detectors might encounter normally. While the second set of records presents more complex arrhythmias and other conduction abnormalities [179, 180].

The signals were sampled at 360Hz and recorded with a two-channel recorder. The annotations were made based on a simple slope-sensitive QRS detector and by two cardiologists, who added additional beat labels missed by the QRS detector and changed all the labels of abnormal beats. Yet, during the following years many records had their beats relabeled by users who reported errors in the annotations [179, 180]. These records are composed by three files, an `.hea` format file, `.dat` format file and a `.atr` format file.

In the terms of data for testing the models and the whole system, it was used data of 128 patients from the Hospital of Braga. All the data was carefully treated and all sensible data was discarded. The data used consisted in 128 records from long-term Holters recorded during 24 hours in March of 2018, 5 records were discarded due to misclassification and therefore the lack of annotations, ten plus records were later dismissed because of other errors. The signals were sampled at 125Hz and recorded with a three-channel recorder, the annotations were made based on the Mars Software from GE Healthcare [181] and by technicians who added additional beat labels and changed the incorrectly classified beats. These records are composed by three files, an `.HEA` format file, a `.SIG` format file and an `.ANN` format file.

Data Extraction and Treatment

The first step to analyze an ECG record begins with the load of the record and posterior filtering. These two processes were performed with the help of the **CardIO**[182] library, an open-source Python library which was built to create "end-to-end machine learning models for deep research of electrocardiograms" [182].

The CardIO library relies in the WFDB[158] package to read and load files in the MIT-BIH format, this library can be executed via command line and also as a python library (the code is publicly available on Github [1]). Using the capabilities of CardIO library, the signals were resampled to the same frequency, making all the signals, for training and testing, being sampled at 125Hz. Then the signal was filtered, the filtering used in this preprocessing was based on several works that proved to be efficient. Thus, it uses a band-pass filter, a finite impulse response filter (FIR filter [127]) which uses a frequency of 0.5Hz and 60Hz, in accordance with the theoretical foundations. The wandering baseline in Portugal, due to the electric current, is present at 50Hz, so the frequency of the FIR was changed when dealing with the testing dataset.

As literature emphasizes, depending on how deep learning is used, there might not be required to previously select the features because inside the *DL* model there might be a way to deal with these features without further filtration of the signals.

For the evaluation of the whole system, after the extraction and treatment of the data, the final data should be compressed for creating the final datasets.

Nevertheless, in order to build the final datasets for training and testing, after these transformations, the ECG is sliced by beats and labeled with the annotations available creating the datasets for training and testing.

The annotations were carefully reviewed and converted accordingly to the types of arrhythmias that were intended to classify. Therefore, the annotations were converted into

[1] https://github.com/MIT-LCP/wfdb-python

five classes: Normal beats as 0, Other Rhythms as 1, Atrial Premature Contractions as 2, Premature Ventricular Contractions as 3 and Supraventricular Premature Contractions as 4.

Data Preparation

The records were split based on the correct detection of the R-Peak and since it is sampled at 125Hz, it was decided to use a window with a size of 120 data points. In order to find the most accurate R-peak detector three approaches were used.

First, it was tested a script built according to the article "An Efficient R-peak Detection Based on New Nonlinear Transformation and First-Order Gaussian Differentiator" by [183]. This script is publicly available on Github[2] and is here referenced as R-Peak Script. The script detects the "midpoint of the QRS complex" and its accuracy may shift from 1 to 2 data points.

The second approach was the use of the BiospPy [184] python library, which allows to segment the record according to several models to find the R-peaks. This library provides five scripts to segment the ECG:

1. `biosppy.signals.ecg.christov_segmenter(signal=None,sampling_rate=1000.0)` based on Christov [185].
2. `biosppy.signals.ecg.engzee_segmenter(signal=None, sampling_rate=1000.0, threshold=0.48)` based on A. H. Engelse and Zeelenberg [186].
3. `biosppy.signals.ecg.gamboa_segmenter(signal=None, sampling_rate=1000.0, tol=0.002)` based on Gamboa [187].
4. `biosppy.signals.ecg.hamilton_segmenter(signal=None, sampling_rate=1000.0)` based on Hamilton [188].
5. `biosppy.signals.ecg.ssf_segmenter(signal=None, sampling_rate=1000.0, threshold=20, before=0.03, after=0.01)` based on Zong et al. [189].

These five algorithms were scrutinized in Canento [190], the creators of this python library.

The last script was created attending the functionalities from the WFDB library [158], in which the processing abilities were tested[3].

According to this, it was possible to disclose the most accurate algorithm to find the R-Peak. The performance of each methodology was studied and the results can be observed on the Table 6 in Appendix A1. The first script above-mentioned obtained the best results, ergo, the script used for finding the QRS complex on unforeseen data was the R-Peak Script.

[2]https://github.com/tru-hy/rpeakdetect
[3]These two last scripts are available on Github. https://github.com/claudiavmbrito/RPeakDetection

Models

One of the most important aspects when developing a model is to carefully attend the data specificities. For this task it was decided to use Keras as the main framework with Tensorflow as backend. In accordance with the related work presented in Section 2.6.1 for the task of classification several model architectures were tested. As per, a fully convolutional neural network, a Resnet architecture and a multilayer perceptron were tested. Focusing on the article by Rajpurkar et al. [137], the ResNet architecture was the one chosen to prevail for fully training and testing the data.

Inside the creation of the deep learning block, there was the opportunity to deal with data parallelism and model parallelism, when dealing with large amounts of data these additional steps become essential. This step was made with Apache Spark, using the Elephas library [191].

Data parallelism promotes the parallelization of the optimization process, the learning of the model, while model parallelism happens when a single model is distributed to multiple machines. The model parallelism mechanism increases the performance of models with several parameters which benefit from accessing more CPU's and memory however, it is not yet implemented on the library. On the other hand, data parallelism uses several replicas of a model distributed to several machines, for which it is needed at least one parameter server[4], this will improve the usage of the resources on the cluster and parallelize the training. The data parallelism technique can be done asynchronously or synchronously. on the first methodology, the workers collect the center variable and update the parameter server with the computed gradient whenever a worker is ready. On the other side, all workers must end their mini-batch task before continuing to the next iteration [191].

Although, sometimes, this may not be the right solution, the Apache Spark Ecosystem is already implemented in this project and the amount of workers were chosen having in mind the existing amount of data.

For the task of arrhythmia prediction, it was created an architecture based on LSTM layers. This task is performed to know how long in time an arrhythmia can be predicted with an high accuracy rank for which the LSTM are well suitable. Being the ECG considered a biometric authentication factor [192–194], forecasting an arrhythmia must be a patient-specific task. However, this task can further be used for deepen the general knowledge about this topic, i.e, although being a patient-specific task, it should be tested to disclose patterns in order to generalize the forecasting.

[4]Parameter Server: a thread that aggregates the new gradients coming from the workers into a center variable [191].

5.1.2 *Data Streaming and Visualization*

The block of Data Streaming in the architecture proposed on Chapter 4.3.3 must receive the data directly from the IoT platform and send it to the visualization block. This process must be done quickly and in a fault-tolerant way. Therefore, as implicit in the system architecture described in Section 4.3, this issue was addressed with the implementation of Apache Kafka as message broker and Apache Spark Streaming.

Apache Kafka and Apache Spark are integrated as well as Apache Spark with Elastic-Search, all three frameworks present an increasingly good performance when dealing with Big Data which encourages this approach. On the overall, Apache Kafka will work as a message broker, sending messages from the Kaa IoT platform, and ElasticSearch as a consumer, while Apache Spark will work as data pipeline, performing the streaming of the data.

5.2 IMPLEMENTATION

Based on the proposed approach and the decisions made, the implementation of the whole system was subdivided in the same blocks as the architecture of the system. So, the implementation was divided into the IoT Platform, the Deep Learning block and, at last, the Data Streaming and Visualization.

5.2.1 *IoT Platform*

Following the methodology proposed by Rei [7], the IoT platform was built based on Kaa Platform, which was deployed in a single node for each virtual machine used. The cluster implemented comprises also the necessary third party software such as Zookeeper for coordination purposes, MariaDB as SQL database once Kaa does not work without a SQL database attached to it, Hadoop's HDFS and Apache HBase. All these components were configured accordingly to the environment to which they were being employed. The configuration schemas, needed to be implemented inside the Kaa platform were built in accordance to the documentation of Kaa Project.

5.2.2 *Deep Learning*

According to Section 5.1.1, it was possible to disclose the most accurate algorithm to find the R-Peak. The performance of each methodology was studied and the results can be observed on the Table 6 in Appendix A1. The first script above-mentioned obtained the best results, ergo, the script used for finding the QRS complex was the R-Peak Script.

The implementation of this block was done by installing all the necessary Python libraries in the server to run this models. All these libraries were accessible by the PIP package. After the first instances of training, the model that surpassed the others was the model with a ResNet architecture [28], becoming the one chosen to perform all the tests. Figure 21 presents the high-level architecture of the network. It takes as input a time-series of 120 data points that represents the beat and outputs its label prediction. The model is composed by an initial input in a BatchNormalization layer, followed by four blocks of Convolution1D layers with BatchNormalization and ReLU as activation function, the model could also use a Dropout schema but it was considered that for the complexity of the model the BatchNormalization would obtained good results. ResNet architectures use shortcut connections that allow information to propagate well in very deep neural networks, with it, this network comprises 3 residual blocks with 4 convolutional layers per block.

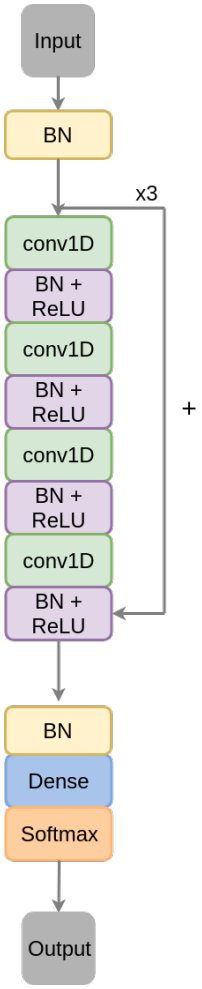


Figure 21.: Network Structure of the model for the classification task. Overall the network contains 12 layers of convolution followed by a fully-connected layer and a softmax.

This model was compiled with a predefined loss function and an optimizer. As loss function it was used the Categorical Cross Entropy (`categorical_crossentropy`) and as optimizer it was tested the Adam (Adaptive Moment Estimation) function and the Stochastic Gradient Descent (SGD).

The model can be saved in .h5 format using h5py Python package [195], after concluding the training. The metrics from model's history are saved along with other useful train metrics and can be accessed later and run as a trained model.

When creating the model with Spark integration it was used the Elephas package. The major difference on the model instantiation is the addition of a SparkContext, since this was implemented in Python, it was created a PySpark Context.

```
conf = SparkConf().setAppName('Elephas_App').setMaster('local[4]')
sc = SparkContext(conf=conf)
```

As well as the transformation of the training data to RDDs, then the model is run with the `Spark_model` function to which the Keras model is used as input as well as the frequency (epochs) and the mode it should be used, asynchronous or synchronous. This model is run with the `spark-submit` script, using the default configurations, however, because of the large dataset, both the `networkTimeout` and `heartbeatInterval` had to be increased from the default values.

Hereon, the Deep Learning model developed with Spark will be known as "Keras with Spark".

The model developed for the forecasting of arrhythmias followed the architecture on Figure 22. Where four LSTM layers with two Dense layers as output were put together. The last layer uses a Softmax activation.

This model was compiled with the Mean Squared Error (`mse`) as loss function and tested with SGD as optimizer. Since this task was considered as patient-specific, the saving model mechanism is discarded. This model was created to forecast arrhythmias while using a time-window with 35 data points, representing approximately 30 seconds of records. As it was briefly explained in the Chapter 2, Section 2.2.3, the LSTM forecast the next step, as so, in a time window with 35 data points, the network will predict the 36^{o} data point and so on, the prediction of arrhythmia will occur once a value of 1, 2 or 3 is predicted.

5.2.3 *Data Streaming and Visualization*

The Data Streaming and Visualization block, as presented in Section 4.3, comprises four different components, Apache Kafka, Apache Spark, ElasticSearch and Kibana, all deployed outside the IoT cluster. All these four components were implemented in local mode al-

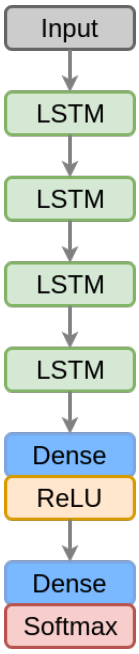


Figure 22.: Network Structure of the model for the prediction task based on LSTMs.

though both Apache Kafka and Apache Spark can be deployed in a cluster environment, potentiating their features.

As a proof of concept, a Kafka Producer, which read the incoming record, was created which sent the messages to the topic `test-elasticsearch-stream`. Afterwards, a Spark Streaming application was also developed for streaming the messages from the topic `test-elasticsearch-stream` to the type `ecg` inside the index `patient-info` on ElasticSearch.

ElasticSearch receives the data as tuple messages, `{elapsed-time, mlii-lead, leadi}`, hence it was needed to create a mapping function and a pipeline function. The mapping function configures how the data on the index `patient-info` should be indexed. The pipeline function reads the message and converts it to the format of the index, removing the type `ecg`, where data was momentarily stored and slicing the message in the three types mapped as `elapsed-time`, `mlii-lead` and `leadi`.

Kibana, after the storing of the messages, can be used to create visualizations of the data as well as creating entire dashboards with information. The scalability of this integration of Kibana and ElasticSearch motivates the use of these two systems for a medical environment, where usually in Intensive Care Units a large number of inpatients need to be monitored in real-time. Figure 23 presents an Electrocardiogram visualization created in Kibana after passing the data through the IoT platform and being ingested on ElasticSearch.

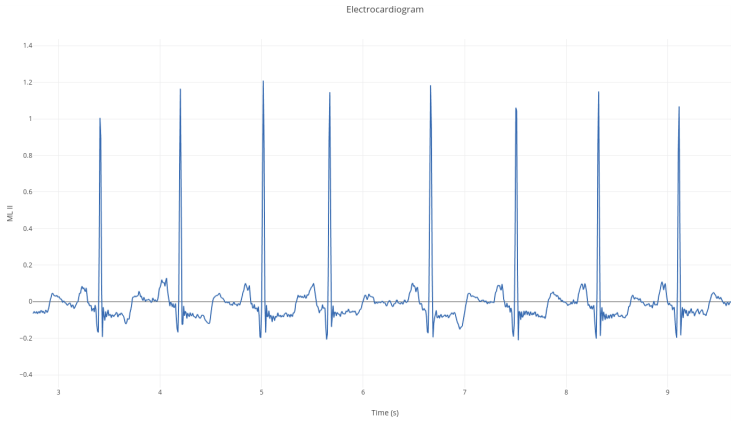


Figure 23.: Visualization of one Electrocardiogram after passing through the IoT platform. As seen, this electrocardiogram has not suffer the filtration it needs and therefore it is a noisy record.

5.3 SUMMARY

This chapter intended to present the chosen approach to develop the system architecture proposed on Chapter 4.3.3, addressing the main issues of the development of the system and the decisions taken to overcome these conditions.

The decision-making process allowed to understand the difficulties of treating this type of data and all the stages this data must go through to be used in a Deep Learning environment and also for data streaming and indexing. It is important to refer the need to develop each block in its own cluster or as standalone mode in order to not overload the overall system.

The implementation process is grouped into three stages, the deployment of the IoT platform, the development of the Deep Learning Stack, and the development of the Data Streaming and Visualization block. The Deep Learning block brings another approach for classifying arrhythmias but also a new approach to predict in a time-window when another arrhythmic beat may occur. Fast and scalable tools such as Apache Kafka, Apache Spark and ElasticSearch are ideal to deal with medical data, either for visualization or the real-time streaming, making this data highly searchable. These tools have a good documentation support which helps to create even more components that might be integrated in the future for data correlations or other analytic processes. After the implementation stage, the system must be evaluated.

6

RESULTS

This Chapter portrays the evaluation of the whole system and the results obtained. On Section 6.1 the environment where the system was built is thoroughly explained, while on Section 6.3 the results and evaluation mechanisms of the system are depicted as well as a discussion of the obtained results.

6.1 EXPERIMENT SETUP

The implementation of the whole system was executed in an experimental environment with virtual machines in a local area network. The Kaa IoT platform was implemented in a cluster composed of three virtual machines, each running Ubuntu 16.04 with 2 CPU cores, 4 GB of RAM memory and 20 GB of available disk space. The deep learning block was implemented in a virtual machine also running Ubuntu 16.04, with 250GB of disk space, 20 CPU cores and 32GB of RAM memory. The last block, the visualization block was implemented in a virtual machine running Ubuntu 16.04, with 4 CPU cores, 120 GB of disk space and 8GB of RAM memory. Other processes such as the conversion of the records from the MIT-BIH format to a workable format and the resampling and filtering of the records from the MIT-BIH Arrhythmia Database were performed on the local server, which runs Fedora 28, with 4 CPU cores, 100GB of disk space and 8GB of RAM memory as seen in Figure 24.

Since this platform has several blocks, the most prominent block to be disclosed is the Deep Learning block, without disregarding the Data Streaming and Visualization. The Kaa IoT block was previously analyzed by Rei [7] and although the Kaa Application had to be built, the analysis of throughput times are not depicted neither thoroughly analyzed.

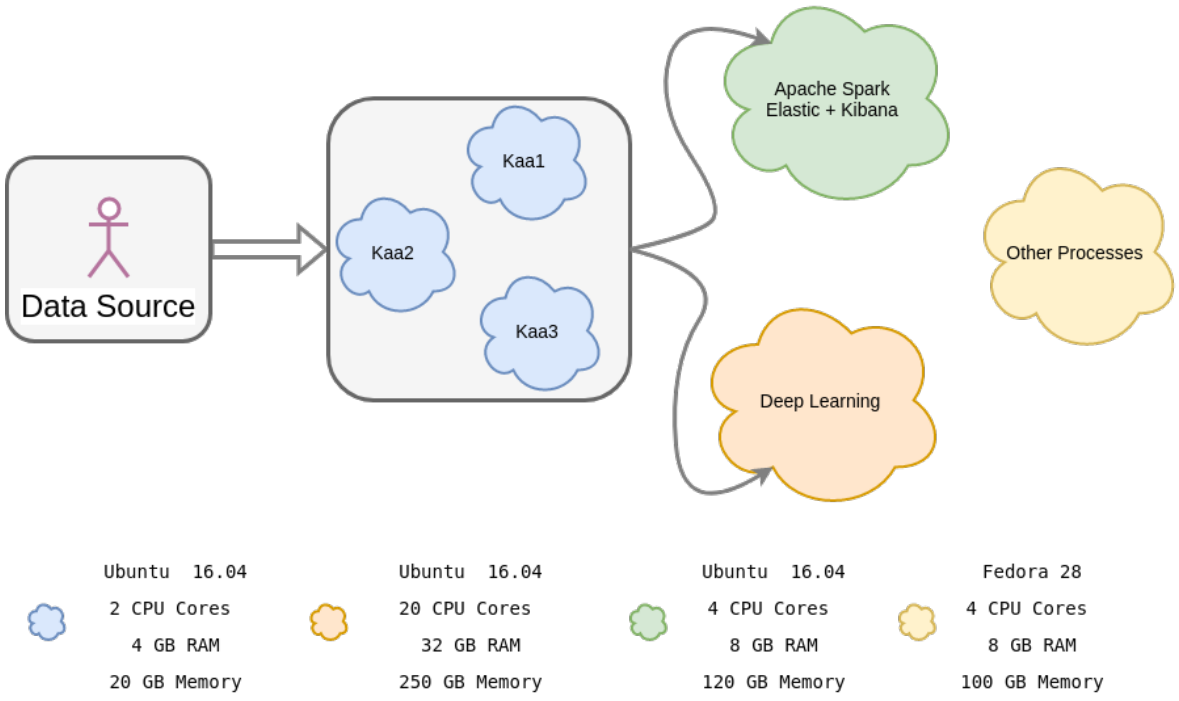


Figure 24.: Environmental setup. This setup comprises six machines, five of them run Ubuntu 16.04. The Kaa Cluster has three machines with 2 CPU cores, 4 GB RAM, 20 GB Memory. The visualization machine has 4 CPU cores, 120 GB of disk space and 8GB of RAM memory. The deep learning machine has 250GB of disk space, 20 CPU cores and 32GB of RAM memory. The other processes were performed on a machine running Fedora 28, with 4 CPU cores, 100GB of disk space and 8GB of RAM memory.

6.2 DEEP LEARNING

6.2.1 *Dataset*

Initially, the data used for testing relied on 20% of the data from the MIT-BIH Arrhythmia Database, who suffered the same pre-processing stages as the training dataset. As so, all the data from this database was sliced in 80% for training and 20% for testing, while the training data was further sliced in training and validation, 65% and 35% (Figure 25).

After all the pre-processing stages and after excluding the records from which the beats were not correctly found, this data ended up consisting of 43 records and 97737 beats. The Table 3 summarizes the number of beats of each class used for classification.

These first results showed that Supraventricular Premature Contractions classification should be discarded since the number of samples were not enough to be considered meaningful. However, these results also encourage the data augmentation of the beats representing the two other arrhythmic classifications, balancing the input data. Taking this into

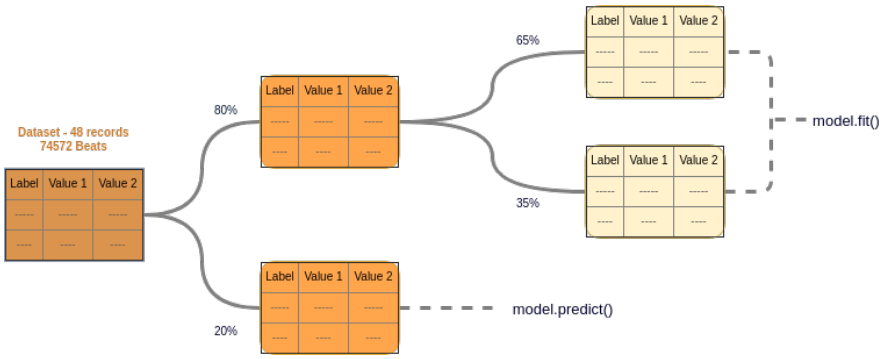


Figure 25.: Splitting the dataset. The splitting is done in two steps: first, 20% for test subset and, from the 80% left, 65% is reserved for training subset and 35% for validation.

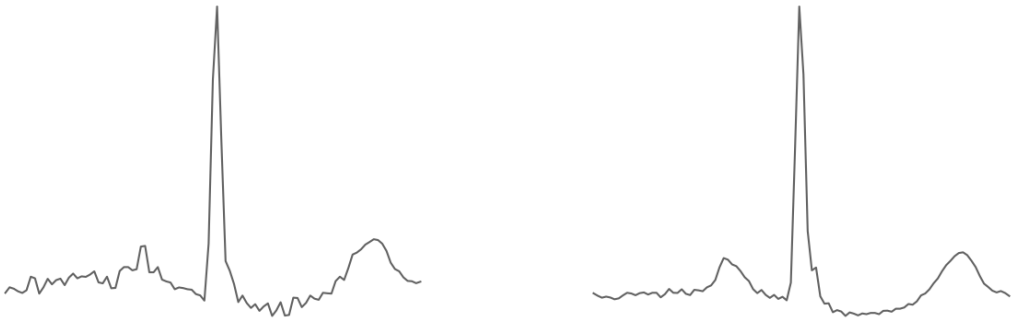
Table 3.: Summary of number of beats of each class.

Class	Number of beats	Label
Normal	69457	0
Others	19001	1
Atrial Premature Contraction	2534	2
Premature Ventricular Contraction	6261	3
Supraventricular Premature Contraction	484	4

account, the number of Atrial Premature Contraction beats became nine times the initial number and the number of Premature Ventricular Contractions increased by four times. When performing data augmentation it is important to not fall in overfitting, where the machine learning algorithm tries to overfit the oversampled examples.

Regarding the dataset for the prediction task, each record was sampled on the same way, 80% for training and 20% for testing, while the 80% data training was further sliced in training and validation. However, the data in use was the elapsed time at which a beat occurred and its classification.

It is important to emphasize the step of signal filtration. This step allows minimizing the errors on the input data. Figure 26 presents a comparison between one beat before and after filtering, although the filtering smooths some values, more filtering should be added in order to reach the optimal performance of the classification model. However, these signals date from 1975 to 1979, so the filtering done on the microprocessors of the sensors has evolved.



(a) Beat before filtering. (b) Beat after filtering.

Figure 26.: Comparison between a before and after filtering of one beat from file 101 of MIT-BIH Database Arrhythmia.

6.2.2 *Evaluation Metrics*

The effectiveness of a deep learning model for classification is usually measured by several parameters. Four of the most common are accuracy, specificity, precision, and recall, as well as the F1-score [154, 155, 196]. To calculate these parameters it is important to understand the following expressions:

1. True Positive (TP): correctly classified positive instances, in an ambiance of *Yes* and *No*, TP happens when the predicted output is equal to the actual output, both *Yes*;
2. True Negative (TN): correctly classified negative instances, in an ambiance of *Yes* and *No*, TN appears when the predicted output is *No* but the actual output is an *Yes*;
3. False Positive (FP): incorrectly classified as positive, in a binary classification paradigm, FP represents the instances predicted as *Yes* while the actual output is *No*;
4. False Negative (FN): correctly classified as negative, which mean the instances predicted as *No* and are actually *No*.

From this perspective, the below equations relate respectively to Accuracy, Specificity, Precision and Recall.

$$Accuracy = \frac{TN + TP}{TP + TN + FP + FN} * 100 \tag{6}$$

The Accuracy (Equation 6) measures the rate of True Negatives and True Positives on all classified instances. This result may induce in error or can hide important details since

it does not distinguish between the number of correct labels of different classes. It can be understood as the number of correct predictions on top of the total number of prediction. The problems of misclassification may arise even though the accuracy results are high. In the medical field, the misdiagnose of an unhealthy subject may cost their life.

$$Specificity = \frac{TN}{TN + FP} * 100 \quad (7) \qquad Precision = \frac{TP}{TP + FP} * 100 \quad (8)$$

Specificity (Equation 7) is used to calculate the rate of True Negatives, or, in a medical context, it represents the proportion of patients correctly classified negative for the disease, in other other, patients who do not have the disease and are classified as not having it. While Precision (Equation 8) equalizes to the rate of True Positive instances in the correctly classified instances.

$$Recall = \frac{TP}{TP + FN} * 100 \quad (9)$$

Recall or Sensitivity (Equation 9) gives higher scores when a high number of True Positives is achieved while avoiding False Negatives, this defines the True Positive Rate [197, 198].

The TP, FN, FP and TN parameters are obtained from the Confusion Matrix, which records the correctly and incorrectly classified instances for each class. These metrics are individually calculated for each class [197, 198].

The F1-score is another general metric that is broadly used for evaluating these systems, the Equation 10 combines the precision and recall into a single number.

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall} * 100 \quad (10)$$

6.2.3 *Training Methodology - Classification*

The tests performed for training the models were optimized with SGD with a learning rate of 0.1 and Adam with a learning rate of 0.1. To understand the evolution of the models with the number of epochs, both systems were tested with 10, 25, 50, 100 and 150 epochs, the maximum number of epochs was defined after the first tests, where any test over 150 epochs demonstrated that the model stopped learning.

First Scenario - Model without Apache Spark

The first tests were performed with the MIT-BIH Arrhythmia Database, where the data was randomly sliced in 65% for training and 35% was used for testing. Several tests were

performed for tuning the hyperparameters and adapting the learning rates to the behavior the model was exhibiting. The main objective of this test is to acquire the best model for later test it in the unknown testing dataset in order to understand how the model behaves when testing samples it has not previously seen. Since this is a multi-classification task, the metrics for each class are calculated.

Second Scenario - Model with Apache Spark

The same process of the first batch of tests was applied to the model implemented with Elephas for data parallelism. This test intended to evaluate the performance of the system, regarding its accuracy and also to compare the elapsed time for all the process with the model without Spark. Following the same methodology of the first scenario, also these tests were performed with the MIT-BIH Arrhythmia Database which was sliced, 65% for training and 35% for testing.

6.2.4 *Testing Methodology - Classification*

The third batch of results comprises the results from the best model previously obtained, tested with the dataset from the Hospital of Braga, real data with no further pre-processing neither additional annotations except for the ones already obtained. This data was used for testing the model.

Since each patient has a 24-hour record, for the real-time purpose of this master thesis, the results were evaluated with a time window of one minute. Therefore, the beats present in one minute were processed and classified. For the sole purpose of simplifying this process, it was admitted that for all the patients the time window of one minute comprised between 60 to 80 beats.

6.2.5 *Training and Testing Methodology - Prediction*

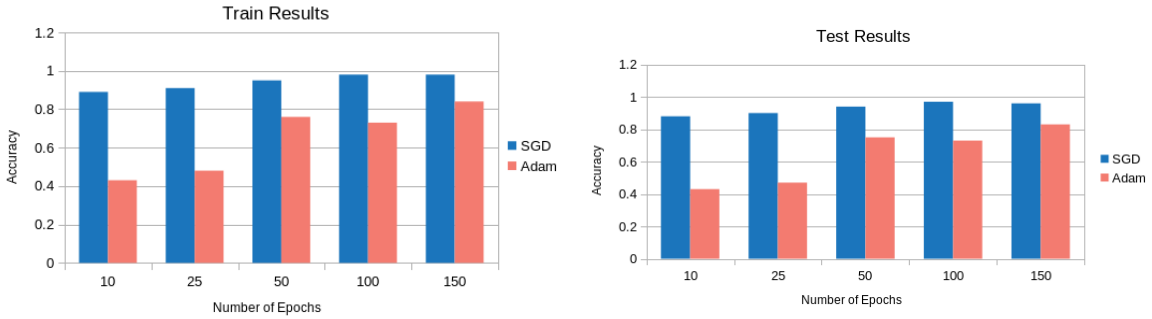
The task of forecasting an arrhythmia is a hazardous one. The methodology used for doing this task was based on the annotations from the dataset from the MIT-BIH Arrhythmia Database. The data was composed by the elapsed time between each beat, it was represented as `Time` and `Class`, where the class represents the type of beat found. To this end, the same methodology of data slicing was performed, however, it was done for each record. The number of real beats used can be seen on the table 6 from Appendix A.1. The nomenclature regarding the classes to each beat belongs was the same proposed before.

6.3 Results and Discussion

6.3.1 *Training Results - Classification*

First Scenario

The Figure 27a and Figure 27b present the overall accuracy of the models with a different number of epochs, a batch size of 2000 beats and as dealing with multi-classification, it is used the categorical cross-entropy as loss function.



(a) (b)

Figure 27.: Training and testing results regarding the accuracy of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Train Results on model without Apache Spark integration; b) Test Results on model without Apache Spark integration.

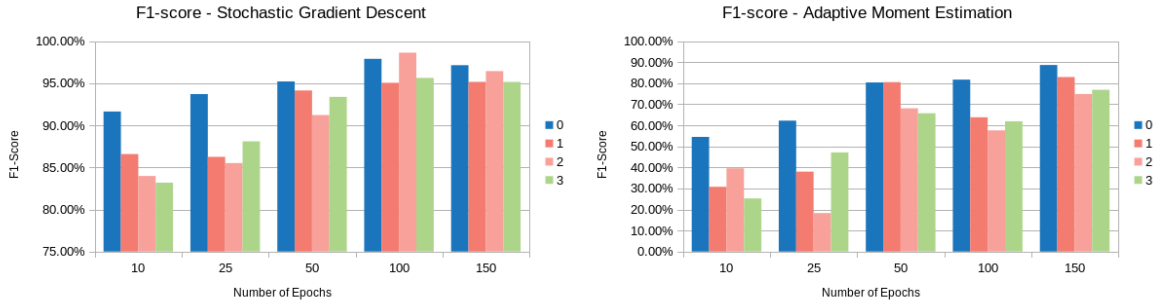
On the other hand, Figures 28a and 28b rely on the F1-score regarding the test results with the 20% of the dataset created. The calculations performed to find the F1-score are presented in the Appendix A.2.1.

The results obtained present no hint of overfitting, the error was minimal and the test set obtained good results both in accuracy and F1-score. The model was able to classify all four classes and these results increased with the number of epochs.

Second Scenario

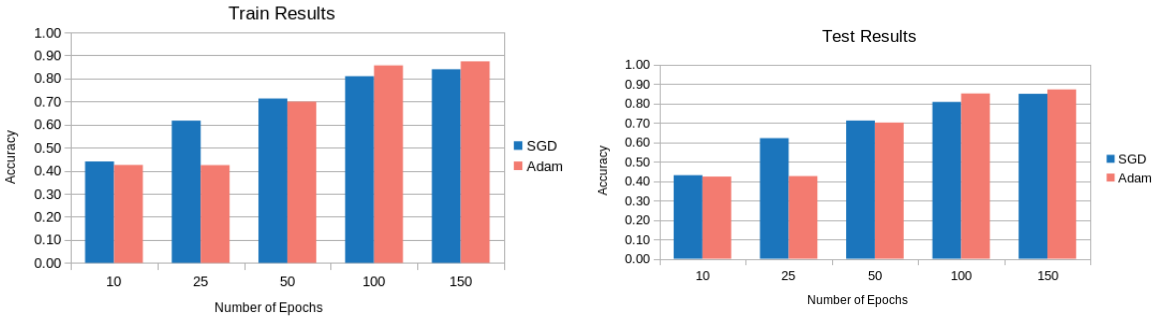
The Figure 29a and Figure 29b exhibit the performance of the model with the Spark integration in local mode with 4 workers, which can be monitored in the Spark Web UI on port 4040 (Figure 35 on Appendix A.3).

Figures 30a and 30b present the values of F1-scores obtained for the test dataset with respect to each class classified with the different optimizers. The calculations performed to find the F1-score are presented in the Appendix A.2.2.



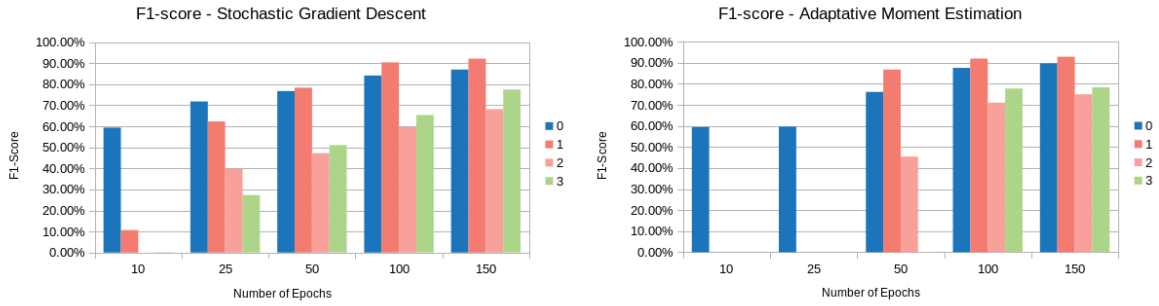
(a) (b)

Figure 28.: Testing results regarding the F1-score of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Test Results on model without Apache Spark integration using Stochastic Gradient Descent for each class; b) Test Results on model without Apache Spark integration using Adaptive Moment Estimation for each class.



(a) (b)

Figure 29.: Training results regarding the accuracy of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Train Results on model with Apache Spark integration; b) Test Results on model with Apache Spark integration.

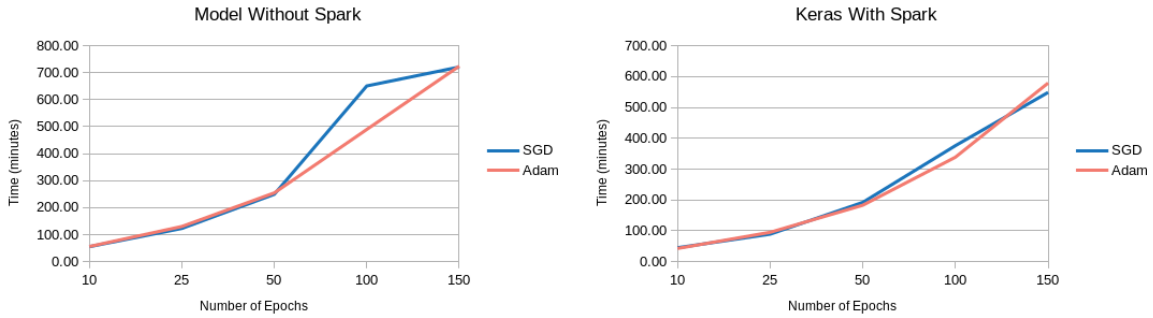


(a) (b)

Figure 30.: Testing results regarding the F1-score of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Test Results on model with Apache Spark integration using Stochastic Gradient Descent for each class; b) Test Results on model without Apache Spark integration using Adaptive Moment Estimation for each class.

First and Second Scenario - Time Related Results

It is important to notice the elapsed time for both models and optimizers while classifying the test data. Figure 31a and Figure 31b present a comparison between the elapsed times when increasing the number of epochs for the model with Spark integration and the model without Spark integration.



(a) (b)

Figure 31.: Training results regarding the elapsed time of the model using Stochastic Gradient Descent and Adaptive Moment Estimation as optimizers. a) Test Results on model without Apache Spark integration; b) Test Results on model with Apache Spark integration.

Before data augmentation, the dataset obtained from MIT-BIH Arrhythmia Database was not as big as the final dataset later used and therefore it did not use all the potential from Apache Spark and the time spent performing these tasks was longer than it was intended. Although the results obtained regarding the accuracy and F1-score did improve over time, the model without Spark integration reached better results, however, the model with Spark was able to reduce the training time in 2 hours only reducing the accuracy for the model with the Stochastic Gradient Descent in about 11% but also improving the accuracy of the model with the Adam optimizer by 4%.

The major difference between these two approaches relies on how the Stochastic Gradient Descent and the Adaptive Moment Estimation do behave, as above-mentioned, a good explanation of these two gradient descents would need a good explanation of derivatives[1], despite that, it is important to understand that when parallelizing the SGD, according to Elephas and Hogwild![2] the processors are allowed to access shared memory without locking the parameters, however, in practice, some parameters can be ignored and therefore are not optimized.

The decrease of computing time when using the Apache Spark technology without the large decrease on accuracy and F1-score provides a good foundation for the continued

[1]Ruder [199] and Wilson et al. [200] present both an overview and a comparison of adaptive gradient descent optimizers which would help in further understanding of these optimizers.

[2]Elephas was built under the assumption of the approach used in Hogwild! to parallelize Stochastic Gradient Descent [201].

employment of this processing framework for deep learning in a real-time environment. Nevertheless it is important to emphasize that deep learning relies primarily on the CPU (in CPU-only environments) while when adding the Apache Spark both the CPU and RAM are used, as so the computational power of the hardware has to be carefully chosen.

6.3.2 *Testing Results - Classification*

When dealing with a single dataset for training and testing it is possible to reach good results, however, it is very noticeable the lack of accuracy of the once best model when dealing with real clinical data collected. Figure 32 presents the accuracy obtained for each record when classifying the beats (F1-scores and confusion matrices are present on the Appendix A.2.3).

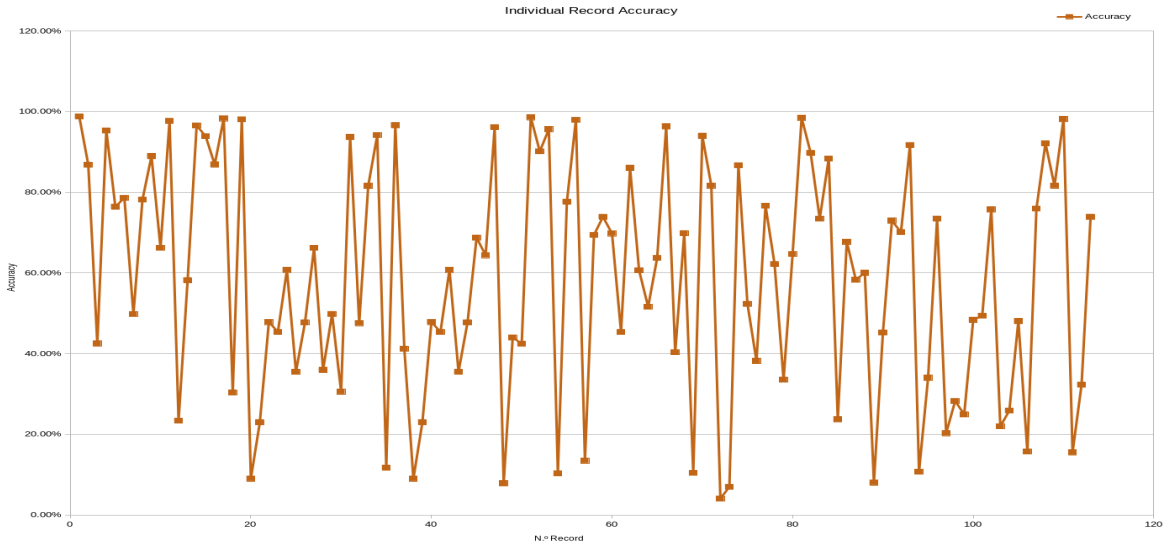


Figure 32.: Testing results regarding the classification of arrhythmias. This graphic representation presents the accuracy of the classification for each record.

Class 2 (Atrial Premature Contraction) is one of the misclassified classes since there is no evidence of Atrial Premature Contraction in any of these records. In light of this, it is possible to observe that the model understands the training data and since the oversampling does not give different data it does not learn to generalize the information taken and so performs badly in certain classes with unforeseen data. It is important to remember that class 2 was oversampled 10 times, there were only 2534 different beats classified as class 2.

The results obtained encouraged to create a different dataset for training, validating and testing the results with only the data from the hospital since the bigger the dataset, the less noisy the accuracy will be and these records are forty-eight times bigger than the first records used.

Due to processing capabilities, two new datasets were created but they did not comprise the 113 records, the training dataset comprised 11 records and the test dataset was built with 5 records. The selection criteria for the test dataset was no more than the 5 worst results obtained previously. The training dataset comprised 1007596 beats as seen in Table 4.

Table 4.: Summary of number of beats of each class for the new train dataset.

Label	Number of Beats
0	929068
1	61591
3	16937
Total	1007596

Since these datasets became bigger, it was impossible to conclude the training of the model with Apache Spark integration, however, it was possible to calculate the learning time for 100 epochs, which would be 50 hours with also the testing of the other dataset. Regarding the real model tested, the ResNet with 14 layers was used for training this new dataset and took an overall of 83 hours and achieved an accuracy of 96% for training and 81% for testing. Table 5 presents the precision, recall and F1-score values for the test dataset obtained for the dataset later proposed.

Table 5.: Results obtained for the new dataset created based on 11 records from the hospital for training and the five worst records from the hospital for testing.

Precision	
Class 0	98.9%
Class 1	8.8%
Class 3	4.1%

Recall	
Class 0	82.3%
Class 1	47%
Class 3	35.3%

F1 Score	
Class 0	89.8%
Class 1	14.9%
Class 3	7.4%

In order to improve the performance of the model, it is possible to use cross-validation[3], despite this fact, due to the lack of resources, the last model was just validated with 35% of the training dataset while the remaining 65% were used for the training itself.

The task of classification should be done in real-time and for that, the supervised learning classification task presented would become a semi-supervised learning task. Hereupon, after the model is trained in large volumes of data, it becomes able to classify new data without the need of it to be previously labeled. Furthermore, this newly classified data should be later used for the training dataset, to make the model continuously learn new information and be able to improve its performance.

[3]The cross-validation technique uses the dataset repeatedly to train the algorithm N times and tests 1/N of the training examples[202].

6.3.3 *Training and Testing Results - Prediction*

The Figure 33 presents the results regarding the accuracy of the model for arrhythmia prediction, following the same work ethic concerning the data splicing.

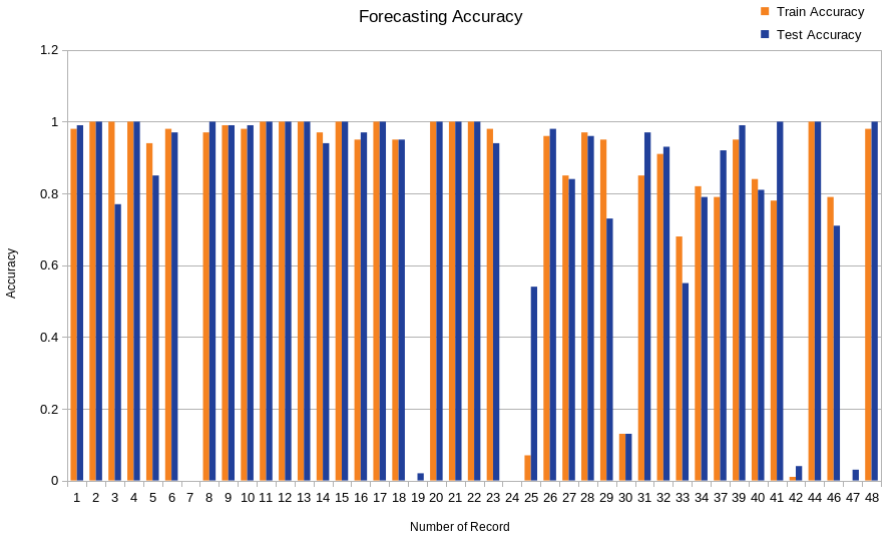


Figure 33.: Training and testing results regarding the forecasting of arrhythmias. This graphic representation presents the accuracy of the forecasting for each record on the MIT-BIH Arrhythmia Database.

The records from the MIT-BIH Arrhythmia Database were relabeled from 100-234 to 1-48. As seen, for records 7, 19, 24, 30, 42 and 47 the model was not able to predict accurately, in fact, it was not able to predict at all. Which can be seen as a result of no existing pattern in these records, the 30-minute record is not able to give enough information about the patient and therefore is not able to correctly predict the type of the next beat.

Dealing with time-series forecasting is a perilous task, with some unique quirks and idiosyncrasies such as the retraining of the model for each new prediction or the importance of the uncertainty of the forecast, both characteristics are on the opposite side of what is done in supervised learning for classification, where the model can be trained once and then be used for classifying new data or the certainty of the classification when the training data comes from the same distribution of the test data. The same process on the Hospital data was performed, however, since it represents 24 hours of recording, it was sliced by a ratio of 60% train - 40% test. Figure 34 presents a sample of the results of this dataset.

On the first sight these results seem promising, being able to predict 40% of the records with major accuracy and a low mean squared error, however, these results, according to the previous classifications do not increase the knowledge about the signal since there are few arrhythmic signals with a long duration. One of the least accurate results is present in

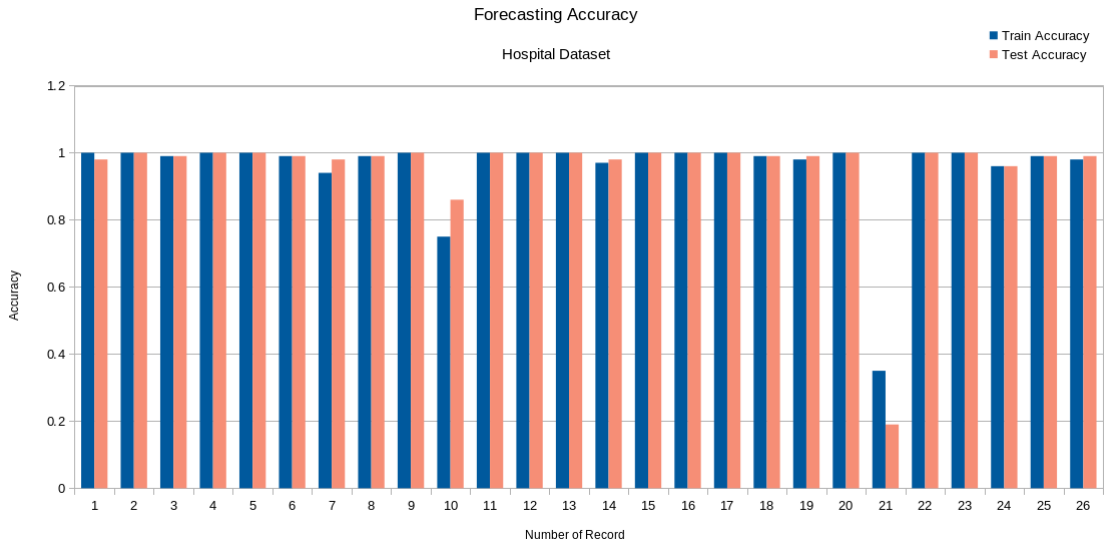


Figure 34.: Training and testing results regarding the forecasting of arrhythmias. This graphic representation presents the accuracy of the forecasting for each record on the collected records from the Hospital of Braga.

the record 21, which compared with the results obtained for its classification it is possible to see that this is one of the few records with a great number of arrhythmic beats. These predictions were based on the whole signal, for a real-time prediction of arrhythmias, nonetheless, in a real environment, when the signal is processed in real-time, this prediction could be done based on early records obtained from the same patients.

7

CONCLUSION

This master thesis used a wide variety of technologies and frameworks, for which different development stages were performed. The following Section, Section 7.1, presents some conclusion notes on the different chapters presented and Section 7.2 presents several ideas for proceeding with this study.

7.1 CONCLUSIONS

The goal of this project was the development of a real-time analytic platform based on IoT technologies in a Cloud environment. Scrutinizing this problematic, the end goal of this master thesis was to develop the analytic block while integrating the other frameworks. While developing the system, several frameworks were analyzed and revised to disclose the most suitable tool for the project. The amount of open-source technologies for the development of such a wide system allowed the opportunity to enlarge the vertical knowledge about these frameworks but also to understand that sometimes only one tool is not enough for the chosen task and the existing integration between them is of major importance for a fast deployment.

On the other hand, focusing on the field of Cardiology, the abnormal rhythm of the heart was further analyzed in order to diagnose the types of beats present in an Electrocardiogram. Focusing in three types of arrhythmia, disregarding one of them after the first stages, the results obtained for this classification task are promising. Without falling on the rabbit-hole of overfitting, these results reached over 90% of accuracy and F1-scores for the four classes proposed. Despite this, when dealing with real clinical data without further pre-processing, the results did not obtained the expected output since one of the classes of the trained model did not exist on the new records. The dataset later created for training and testing a new model could then accurately classify the instances, reaching an accuracy of 96% on the training set and an accuracy of 81% for the testing set. It is important to emphasize that only a sum of 16 records with 100000 ± 20000 beats each was used for creating these datasets making a total of1007596 beats for the training set and a total of414245 beats for the testing dataset.

The time series prediction is a precarious task and although being studied it is a growing field with new discoveries each day, for this matter and regarding the tests performed in the Hospital data, the results are indeed promising since almost every record was successfully predicted, however, time series prediction is still not very reliable for such small predictions, therefore, although reaching good results it should be further studied and with the addition of other variables, such as medical history, medication, blood samples or other time series variables like CO_2 and O_2 monitoring among others. For a real-time prediction of arrhythmias and following the assumption that electrocardiograms are bio-markers, it would be interesting to create a database of these bio-markers, and create a mechanism to detect rapidly an anonymous patient based on its previous records and predict the arrhythmic beats.

Although no further evaluation of the systems performance was made, it was possible to observe that the stack comprising Apache Kafka, Apache Spark and ElasticSearch was successfully implemented and can be further used for other time series data or unstructured data making it searchable and, with the new X-Pack of ElasticSearch, a new machine learning for finding patterns can be employed.

In sum, the combination of classification and prediction tasks revealed to be promising in research and development of Deep Learning models in the Cardiology field. While the classification task was successfully accomplished both for the MIT-BIH database as well as for the clinical data obtained from the Hospital of Braga, the prediction task leaves a lot to learn about the features of the electrocardiograms and the question about being a bio-marker.

The infrastructure implemented supports the continuation of studies with other time series data or other unstructured data and allows the integration of other analytic tools to create decision support systems with high accuracy and demand.

7.2 Prospect for future work

The IoT platform which was used as a foundation for this master thesis should be thoroughly tested both in terms of security and scalability in order to be applied in a real scenario were these two parameters are not a need but a requirement. Despite the good results achieved with the models, there is still a large space for further researching. One of the first steps could pass by enlarging the types of arrhythmia and use this data to detect other conditions such as strokes and heart attacks as well as to create patient-dedicated full-diagnosis. It would be of major importance to understand the results regarding the correlation and generalization of data, i.e., arrhythmia forecasting for a not patient-specific approach. Moreover, the process of detecting the R-peak of a signal should also be op-

timized, an anomaly detection approach could be somewhat used for detecting the R-peak which would diminish the pre-processing stages and improve the overall performance.

Enlarging the spectrum of work, with more time series data and more types of records, such as Electroencephalograms, could also help to understand other data correlations, with this in mind, it becomes even more essential to retrieve non-time series data from patients and combine all the data.

The use of Apache Spark in the Deep Learning world should be thoroughly analyzed. The results presented did not bring attention to this integration, however, Apache Spark has shown incredible results for advanced in-memory programming and its scalability for Machine Learning should be benchmarked. Apache Spark can even be used for hyper-optimization, providing the tuning of the hyperparameters by finding the best set, enabling to discard a hyperparameter focused study.

Last but not least, it is important to parallelize all the work, preventing overload of the system, create a load balancing system and, it is crucial to increase the automation of the system, easing the deployment of the clusters with other open-source frameworks.

BIBLIOGRAPHY

[1] Josh Patterson and Adam Gibson. Deep learning, A Practitioner's Approach. *O'Reilly*, 2015.

[2] Arthur C. Guyton and John E. Hall. *TextBook in Medical Physiology*. Elsevier Saunders, 2006.

[3] Daniel E Becker. Fundamentals of electrocardiography interpretation. *Anesthesia progress*, 53(2):53–63; quiz 64, 2006.

[4] Benjamin EJ. Heart Disease and Stroke Statistics - 2017 Update. *Circulation*, 135: 2017–2018, 2017.

[5] Dr. João Lopes Gomes. A Morte Súbita Cardíaca - Fundação Portuguesa Cardiologia. http://www.fpcardiologia.pt/a-morte-subita-cardiaca/. Accessed: 2018-08-09.

[6] Gerard J. Tortora and Bryan Derrickson. *Principles of Anatomy & Physiology 14th Edition*. 2014.

[7] João P. N. Rei. Study, Selection and Evaluation of an IoT Platform for Data Collection and Analysis for medical sensors, 2018.

[8] Sebastian Salas-Vega, Adria Haimann, and Elias Mossialos. Big data and health care: challenges and opportunities for coordinated policy development in the EU. *Health Systems & Reform*, 1(4):285–300, 2015.

[9] K Hammad, M Fakharaldien, J Zain, and M Majid. Big Data Analysis and Storage. In *International Conference on Operations Excellence and Service Engineering*, pages 10–11, 2015.

[10] Alan M. Turing. Computing machinery and intelligence. *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer*, pages 23–65, 2009.

[11] John McCarthy. What is Artificial Intelligence? pages 1–15, 2007.

[12] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. 105(12):2295–2329, 2017.

[13] Warren S Mcculloch and Walter Pitts. A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mothemnticnl Biology*, 52(l2):99–115, 1990.

[14] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[15] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[16] Juergen Schmidhuber. Deep Learning in Neural Networks: An Overview. *Expert Systems with Applications*, 72:221–230, apr 2014.

[17] Itamar Arel, Derek C Rose, and Thomas P Karnowski. Deep Machine Learning - A New Frontier in Artificial Intelligence Research. *IEEE Computational Intelligence Magazine*, 5(4):13–18, 2010.

[18] Nikhil Ketkar. *Deep Learning with Python*, volume 1. Apress, Berkeley, CA, nov 2017.

[19] Matt Nedrich. An Introduction to Gradient Descent and Linear Regression. https://spin.atomicobject.com/2014/06/24/gradient-descent-linear-regression/, 2014. Accessed: 2018-02-03.

[20] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] John Cristian Borges Gamboa. Deep Learning for Time-Series Analysis. 2017.

[22] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

[23] F. Chollet. *Deep Learning with Python*. Manning Publications Company, 2017.

[24] Alex Krizhevsky, Ilya Sutskever, and Hinton Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25 (NIPS2012)*, pages 1–9, 2012.

[25] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. *Computer Vision–ECCV 2014*, 8689:818–833, 2014.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Chapel Hill, and Ann Arbor.

[27] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. pages 1–14, 2014.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 7, 2015.

[29] D Stutz. Understanding Convolutional Neural Networks. *Nips 2016*, (3):1–23, 2014.

[30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back Propagating Errors. *Nature*, 323:533–536, 1986.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[32] Alex Graves. *Supervised Sequence Labeling with Recurrent Neural Networks*, volume 12. 2013.

[33] Sepp Hochreiter and Jurgen Schmidhuber. Long Short-term Memory. *Computation, Neural*, (December 1997), 1997.

[34] Daniele Ravi, Charence Wong, Fani Deligianni, Melissa Berthelot, Javier Andreu-Perez, Benny Lo, and Guang Zhong Yang. Deep Learning for Health Informatics. *IEEE Journal of Biomedical and Health Informatics*, 21(1):4–21, 2017.

[35] Christopher Olah. Understanding LSTM Networks – colah's blog. 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, Accessed: 2018-04-03.

[36] Christopher M Bishop. *Pattern Recognition and Machine Learning*, volume 53. 2013.

[37] Skymind. A Beginner's Guide to Neural Networks and Deep Learning — Skymind. https://skymind.ai/wiki/neural-network. Accessed: 2018-07-31.

[38] Oliver Faust, Yuki Hagiwara, Tan Jen Hong, Oh Shu Lih, and U. Rajendra Acharya. Deep learning for healthcare applications based on physiological signals: A review, 2018.

[39] Hrayr Harutyunyan, Hrant Khachatrian, David C. Kale, and Aram Galstyan. Multitask Learning and Benchmarking with Clinical Time Series Data. *CEUR Workshop Proceedings*, 1953:39–42, mar 2017. ISSN 16130073.

[40] Ian H Witten, Eibe Frank, and Mark a Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. 2011.

[41] J. H. Abawajy, A. V. Kelarev, and M. Chowdhury. Multistage approach for clustering and classification of ECG data. *Computer Methods and Programs in Biomedicine*, 112(3): 720–730, 2013.

[42] Mohammed A. Khalila. *Relational Data Clustering Algorithms with Biomedical*. PhD thesis, University of Missouri-Columbia, 2014.

[43] Theano Development Team. Theano. http://www.deeplearning.net/software/theano/, 2010. Accessed: 2018-02-25.

[44] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[45] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

[46] Sebastian Raschka. Introduction to Artificial Neural Networks and Deep Learning A Practical Guide with Applications in Python. 2017.

[47] Torch Development Team. Torch. http://torch.ch/. Accessed: 2018-02-25.

[48] Pytorch Development Team. Pytorch. https://pytorch.org/. Accessed: 2018-02-25.

[49] Akarsh Singh. Deep Learning Tensorflow vs Keras vs PyTorch. http://codeinpython.com/tutorials/deep-learning-tensorflow-keras-pytorch/, . Accessed: 2018-02-25.

[50] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[51] Caffe2. A New Lightweight, Modular, and Scalable Deep Learning Framework. https://caffe2.ai/. Accessed: 2018-03-30.

[52] Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. Comparative Study of Deep Learning Software Frameworks. 2015.

[53] Caffe2. Caffe2 and PyTorch join forces to create a Research + Production platform PyTorch 1.0. https://caffe2.ai/blog/, 2018. Accessed: 2018-06-30.

[54] Nisha Muktewar and Seth Hendrickson. Deep learning on Apache Spark and Apache Hadoop with Deeplearning4j - Cloudera Engineering Blog. https://blog.cloudera.com/blog/2017/06/deep-learning-on-apache-spark-and-hadoop-with-deeplearning4j/, 2017. Accessed: 2018-02-22.

[55] Eclipse Deeplearning4j Development Team. Deeplearning4j: Open-source, Distributed Deep Learning for the JVM, Apache Software Foundation License 2.0. https://deeplearning4j.org/. Accessed: 2018-02-23.

[56] Matthias Boehm. *Apache SystemML*, pages 1–6. Springer International Publishing, Cham, 2018.

[57] Apache SystemML - Declarative Large-Scale Machine Learning. https://systemml.apache.org/. Accessed: 2018-02-21.

[58] MXNet: A Scalable Deep Learning Framework. https://mxnet.apache.org/. Accessed: 2018-02-21.

[59] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. *eprint arXiv:1512.01274*, pages 1–6, 2015.

[60] Microsoft. Microsoft Cognitive Toolkit. https://www.microsoft.com/en-us/cognitive-toolkit/. Accessed: 2018-02-23.

[61] Nisha Muktewar. Deep Learning with Intel's BigDL and Apache Spark. https://blog.cloudera.com/blog/2017/09/deep-learning-with-intels-bigdl-and-apache-spark/, 2016. Accessed: 2018-02-19.

[62] Sergey E. BigDL: Distributed Deep Learning on Apache Spark* — Intel® Software. https://software.intel.com/en-us/articles/bigdl-distributed-deep-learning-on-apache-spark, 2017. Accessed: 2018-03-03.

[63] Intel. BigDL — Intel® Software. https://software.intel.com/en-us/ai-academy/frameworks/bigdl. Accessed: 2018-03-14.

[64] François Chollet et al. Keras. `https://keras.io`, 2015.

[65] François Chollet. What are some important engineering and design decisions you made in creating Keras? - Quora. https://www.quora.com/What-are-some-important-engineering-and-design- decisions-you-made-in-creating-Keras, 2016. Accessed: 2018-03-30.

[66] Exastax. A Comparison of Deep Learning Frameworks — Exastax. https://www.exastax.com/deep-learning/a-comparison-of-deep-learning-frameworks/, 2018. Accessed: 2018-02-28.

[67] Nuzhi Meyen. A Survey of Deep Learning Frameworks – Towards Data Science. https://towardsdatascience.com/a-survey-of-deep-learning-frameworks-43b88b11af34, 2017. Accessed: 2018-03-11.

[68] Martin Heller. Review: The best frameworks for machine learning and deep learning — InfoWorld. https://www.infoworld.com/article/3163525/analytics/review-the-best-frameworks-for-machine-learning-and-deep-learning.html, 2017. Accessed: 2018-03-12.

[69] Gopi Kumar, Josée Martens, Paige Bailey, Evgeny Grigorenko, Paul Shealy, Ross McAllister, Marktab, and Kristine Toliver. Deep Learning and AI frameworks - Azure — Microsoft Docs. https://docs.microsoft.com/en-us/azure/machine-learning/data-science-virtual-machine/dsvm-deep-learning-ai-frameworks, 2017. Accessed: 2018-03-13.

[70] Intel. Hands-On AI Part 5: Select a Deep Learning Framework — Intel® Software. https://software.intel.com/en-us/articles/hands-on-ai-part-5-select-a-deep-learning-framework, 2017. Accessed: 2018-07-30.

[71] Timothy Wesley Molter. Deep Learning Frameworks Hands-on Review. https://knowm.org/deep-learning-frameworks-hands-on-review/, dec 2017. Accessed: 2018-03-13.

[72] Indra Bakker. Battle of the Deep Learning frameworks - Part I. https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cffoe3841750, 2017. Accessed: 2018-03-11.

[73] Tensorflow Development Team. TensorBoard: Visualizing Learning — TensorFlow. https://www.tensorflow.org/guide/summaries_and_tensorboard. Accessed: 2018-02-03.

[74] LISA Lab. d3viz – d3viz: Interactive visualization of Theano compute graphs — Theano 1.0.0 documentation. http://deeplearning.net/software/theano/library/d3viz/index.html. Accessed: 2018-05-01.

[75] Michael V. Copeland. DEEP LEARNING EXPLAINED. *NVIDIA*.

[76] Supun Kamburugamuve, G Fox, D Leake, and J Qiu. Survey of Distributed Stream Processing for Large Stream Sources. *Grids.Ucs.Indiana.Edu*, (March), 2013.

[77] Ahmed Oussous, Fatima Zahra Benjelloun, Ayoub Ait Lahcen, and Samir Belfkih. Big Data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, 2017.

[78] Praveen Kumar Singh. Survey of Big Data Frameworks for Different Application Characteristics. .

[79] Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, and Alexander Jung. An Experimental Survey on Big Data Frameworks. *Future Generation Computer Systems*, 2016.

[80] Ziya Karakaya, Ali Yazici, and Mohammed Alayyoub. A Comparison of Stream Processing Frameworks. *2017 International Conference on Computer and Applications, ICCA 2017*, pages 1–12, 2017.

[81] Petr Zapletal. Introduction Into Distributed Real-Time Stream Processing. *Cakesolutions*, pages 1–9, 2015.

[82] Apache Kafka. Apache Kafka - Documentation. https://kafka.apache.org/documentation/, Accessed: 2018-05-10.

[83] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: a Distributed Messaging System for Log Processing. *ACM SIGMOD Workshop on Networking Meets Databases*, page 6, 2011.

[84] Matthias J. Sax. *Apache Kafka*, pages 1–8. Springer International Publishing, Cham, 2018.

[85] N. Garg. *Apache Kafka*. Packt Publishing, 2013.

[86] Apache Hadoop. Hadoop – Apache Hadoop 2.9.1. http://hadoop.apache.org/docs/current/, Accessed: 2018-03-05.

[87] Saeed Shahrivari. Beyond Batch Processing: Towards Real-Time and Streaming Big Data. *Computers*, 3(4):117–129, 2014.

[88] Kai Wähner. Real-Time Stream Processing as Game Changer in a Big Data World with Hadoop and Data Warehouse. *InfoQ*, pages 1–9, 2014.

[89] Justin Ellingwood. Hadoop, Storm, Samza, Spark, and Flink: Big Data Frameworks Compared — DigitalOcean. https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared, Accessed: 2018-03-26.

[90] Ashwin Belle, Raghuram Thiagarajan, S M Reza Soroushmehr, Fatemeh Navidi, Daniel A Beard, and Kayvan Najarian. Big Data Analytics in Healthcare. *Hindawi Publishing Corporation*, 2015:1–16, 2015.

[91] Apache Storm. Apache Storm - Documentation. https://storm.apache.org/releases/2.0.0-SNAPSHOT/index.html, Accessed: 2018-03-07.

[92] Microsoft Docs. What is Apache Storm - Azure HDInsight — Microsoft Docs. https://docs.microsoft.com/en-us/azure/hdinsight/storm/apache-storm-overview, Accessed: 2018-03-22.

[93] Hortonworks. Apache Storm - Hortonworks. https://hortonworks.com/apache/storm/, Accessed: 2018-04-06.

[94] Ravi Jain. Real-Time Data Streaming Tools And Technologies – An Overview - Algoworks. http://www.algoworks.com/blog/real-time-data-streaming-tools-and-technologies/, Accessed: 2018-04-12.

[95] Apache Samza. Samza - Documentation. https://samza.apache.org/learn/documentation/0.14/, Accessed: 2018-03-06.

[96] Alexandre da Silva Veith and Marcos Dias de Assunção. *Apache Spark*, pages 1–5. Springer International Publishing, Cham, 2018.

[97] Petr Zapletal. Comparison of Apache Stream Processing Frameworks: Part 2. *Cakesolutions*, 2016.

[98] Ben Lorica. Stream processing and mining just got more interesting, Accessed: 2018-02-06.

[99] Apache Spark. Overview - Spark 2.3.0 Documentation. https://spark.apache.org/docs/latest/, Accessed: 2018-02-17a.

[100] Jacek Laskowski. *Mastering apache Spark*. Number 14. 2012.

[101] Sridhar Alla. *Big Data Analytics with Hadoop 3*. Packt Publishing, 2018.

[102] Apache Flink. Apache Flink 1.5 Documentation: Apache Flink Documentation. https://ci.apache.org/projects/flink/flink-docs-release-1.5/, Accessed: 2018-03-07.

[103] Fabian Hueske and Timo Walther. *Apache Flink*, pages 1–8. Springer International Publishing, Cham, 2018.

[104] Rafał Kuć and Marek Rogoziński. *Mastering Elasticsearch*. 2015.

[105] ElasticSearch. Basic Concepts — Elasticsearch Reference [6.3] — Elastic. https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html, Accessed: 2018-05-17.

[106] ElasticSearch. Removal of mapping types — Elasticsearch Reference [6.3] — Elastic. www.elastic.co/guide/en/elasticsearch/reference/current/removal-of-types.html, Accessed: 2018-06-17.

[107] Yuvraj Gupta. *Kibana Essentials*. Packt Publishing, 2015.

[108] Elastic. Getting Started — Kibana User Guide [6.3] — Elastic. https://www.elastic.co/guide/en/kibana/current/getting-started.html. Accessed: 2018-05-10.

[109] Richard L. Drake, A. Wayne Vogl, and Adam W. M. Mitchell. *Gray's Anatomy For Students, Third Edition*. Elsevier Inc, 2015.

[110] Tony Curran and Gill Sheppard. Cardiology Self Learning Package. *Anatomy and Physiology of the Heart*, (October):2–22, 2011.

[111] Mayo Clinic. Heart disease - Diagnosis and treatment - Mayo Clinic. https://www.mayoclinic.org/diseases-conditions/heart-disease/diagnosis-treatment/drc-20353124, Accessed: 2018-03-30.

[112] Heart Foundation. Heart tests — ECG, echo and other tests. https://www.heartfoundation.org.nz/your-heart/heart-tests, Accessed: 2018-03-30.

[113] Gail Walraven. *Basic Arrhythmias*. 2006.

[114] John A Kastor. Cardiac Arrhythmias. *Encyclopedia of Life Sciences*, pages 1–11, 2002.

[115] Elizabeth Borgelt. Basic Cardiac Rhythms – Identification and Response. 2014.

[116] Norman F Boas, Donald A Rowley, Seymour Glagov, and Peter Stoner. Cardiotachometer. *Science*, 131(3399):564–565, 1960.

[117] Richard E. Klabunde. CV Physiology — Ectopic Foci. https://www.cvphysiology.com/Arrhythmias/A017, 2016. Accessed: 2018-01-03.

[118] Bonnie Feldman and Ellen M Martin. Big Data in Healthcare Hype and Hope. (October), 2012.

[119] Wullianallur Raghupathi and Viju Raghupathi. Big data analytics in healthcare: promise and potential. *Health Information Science and Systems*, 2(1):3, 2014.

[120] Maryam M. Najafabadi, Flavio Villanustre, Taghi M. Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1–21, 2015.

[121] S. Karpagachelvi, M. Arthanari, and M. Sivakumar. ECG Feature Extraction Techniques - A Survey Approach. 8(1), 2010.

[122] Muhammad Umer, Bilal Ahmed Bhatti, Muhammad Hammad Tariq, Muhammad Zia-Ul-Hassan, Muhammad Yaqub Khan, and Tahir Zaidi. Electrocardiogram Feature Extraction and Pattern Recognition Using a Novel Windowing Algorithm. *Advances in Bioscience and Biotechnology*, 5(October):886–894, 2014.

[123] Juie D Peshave and Rajveer Shastri. Feature Extraction of ECG Signal. pages 1864–1868, 2014.

[124] R.M. Rao and A.S. Bopardikar. *Wavelet Transforms: Introduction to Theory and Applications*. Number vol. 1 in Wavelet Transforms: Introduction to Theory and Applications. Addison-Wesley, 1998.

[125] R. Rodríguez, A. Mexicano, J. Bila, S. Cervantes, and R. Ponce. Feature extraction of electrocardiogram signals by applying adaptive threshold and principal component analysis. *Journal of Applied Research and Technology*, 13(2):261–269, 2015.

[126] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern classification*. Pattern Classification and Scene Analysis: Pattern Classification. Wiley, 2001.

[127] A.V. Oppenheim, A.S. Willsky, and S.H. Nawab. *Signals and Systems*. Prentice-Hall signal processing series. Prentice Hall, 1997.

[128] Xinqi Louis Wang and J Mikael Eklund Smieee. A real-time ECG feature detection algorithm. *Ieee*, pages 5–7, 2014.

[129] Jiapu Pan and Willis J. Tompkins. A Real-Time QRS Detection Algorithm. *IEEE Transactions on Biomedical Engineering*, BME-32(3):230–236, 1985.

[130] Ali Isin and Selen Ozdalili. Cardiac arrhythmia detection using deep learning. *Procedia Computer Science Procedia Computer Science Procedia Computer Science*, 120(00):268–275, 2017.

[131] Malay Mitra and R.K. Samanta. Cardiac Arrhythmia Classification Using Neural Networks with Selected Features. *Procedia Technology*, 10:76–84, 2013.

[132] Mark A Hall and Lloyd A Smith. *Feature subset selection: A correlation based filter approach*. 1998.

[133] Ananth Ranganathan. The Levenberg-Marquardt Algorithm. 142(June):1–5, 2004.

[134] The University of Waikato Weka. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. https://www.cs.waikato.ac.nz/ml/weka/, Accessed: 2018-02-01.

[135] Alyuda Research. Neural Network Software, Forecasting Software, Neural Networks System — Alyuda. http://www.alyuda.com/, Acessed: 2018-04-17.

[136] Chan Yuan, Yan Yan, Lin Zhou, Jingwen Bai, and Lei Wang. Automated atrial fibrillation detection based on deep learning network. *2016 IEEE International Conference on Information and Automation (ICIA)*, (August):1159–1164, 2016.

[137] Pranav Rajpurkar, Awni Y. Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y. Ng. Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks. 2017.

[138] IRhythm. Zio Patch Cardiac Monitoring: iRhythm Technologies. https://www.irhythmtech.com/professionals/why-zio. Accessed: 2018-09-08.

[139] Martin Zihlmann, Dmytro Perekrestenko, and Michael Tschannen. Convolutional Recurrent Neural Networks for Electrocardiogram Classification. pages 1–4, 2017.

[140] Zachary C. Lipton, David C. Kale, Charles Elkan, and Randall Wetzel. Learning to Diagnose with LSTM Recurrent Neural Networks. pages 1–18, 2015.

[141] Pouya Bashivan, Irina Rish, Mohammed Yeasin, and Noel Codella. Learning Representations from EEG with Deep Recurrent-Convolutional Neural Networks. pages 1–15, 2015.

[142] Emre Cakir, Giambattista Parascandolo, Toni Heittola, Heikki Huttunen, and Tuomas Virtanen. Convolutional Recurrent Neural Networks for Polyphonic Sound Event Detection. *IEEE/ACM Transactions on Audio Speech and Language Processing*, 25(6):1291–1303, 2017.

[143] P Y Simard, D Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 958–963, 2003.

[144] B. Pyakillya, N. Kazachenko, and N. Mikhailovsky. Deep Learning for ECG Classification. *Journal of Physics: Conference Series*, 913:012004, oct 2017.

[145] Philip Warrick and Masun Nabhan Homsi. Cardiac Arrhythmia Detection from ECG Combining Convolutional and Long Short-Term Memory Networks. 44:1–4, 2017.

[146] Philip A Warrick and Masun Nabhan Homsi. Ensembling convolutional and long short-term memory networks for electrocardiogram arrhythmia detection. *Physiological Measurement*, pages 0–20, jul 2018.

[147] Koo Ping Shung. Accuracy, Precision, Recall or F1? – Towards Data Science. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9, 2018. Accessed: 2018-07-29.

[148] Mehrdad Javadi, Reza Ebrahimpour, Atena Sajedin, Soheil Faridi, and Shokoufeh Zakernejad. Improving ECG classification accuracy using an ensemble of neural network modules. *PLoS ONE*, 6(10), 2011.

[149] Cheng Ju, Aurélien Bibaut, and Mark J. van der Laan. The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification. pages 1–20, 2017.

[150] Mohamed Limam, Frederic Precioso, and Université Côte. Atrial Fibrillation Detection and ECG Classification based on Convolutional Recurrent Neural Network. *Computing in Cardiology*, 44:1–4, 2017.

[151] G Swapna, K P Soman, and R Vinayakumar. Automated detection of cardiac arrhythmia using deep learning techniques. *Procedia Computer Science*, 132:1192–1201, 2018.

[152] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. pages 1–9, 2014.

[153] Shalin Savalia and Vahid Emamian. Cardiac Arrhythmia Classification by Multi-Layer Perceptron and Convolution Neural Networks. 2018.

[154] Tae Joon Jun, Hyun Ji Park, Nguyen Hoang Minh, Daeyoung Kim, and Young Hak Kim. Premature ventricular contraction beat detection with deep neural networks. *Proceedings - 2016 15th IEEE International Conference on Machine Learning and Applications, ICMLA 2016*, pages 859–864, 2017.

[155] Abhinav Vishwa, Mohit K. Lal, Sharad Dixit, and Pritish Vardwaj. Clasification Of Arrhythmic ECG Data Using Machine Learning Techniques. *International Journal of Interactive Multimedia and Artificial Intelligence*, 1(4):67, 2011.

[156] Henian Xia, Irfan Asif, and Xiaopeng Zhao. Cloud-ECG for real time ECG monitoring and analysis. *Computer Methods and Programs in Biomedicine*, 110(3):253–259, 2013.

[157] P. M. Pinto Silva and J. P. Silva Cunha. SenseMyHeart: A cloud service and API for wearable heart monitors. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 2015-November:4986–4989, 2015.

[158] Physionet. The WFDB Software Package. https://physionet.org/physiotools/wfdb.shtml, Accessed: 2018-02-01.

[159] Fizar Ahmed. An Internet of Things (IoT) Application for Predicting the Quantity of Future Heart Attack Patients. 164(6):36–40, 2017.

[160] Freddy Jimenez and Romina Torres. Building an IoT-aware healthcare monitoring system. *Proceedings - International Conference of the Chilean Computer Science Society, SCCC*, 2016-Febru:5–8, 2016.

[161] Kinshuk U. Nigam, Abhinee A. Chavan, Sumit S. Ghatule, and Vaishali M. Barkade. IOT-BEAT: An intelligent nurse for the cardiac patient. *International Conference on Communication and Signal Processing, ICCSP 2016*, (Cvd):976–982, 2016.

[162] P Dineshkumar. Big data Analytics of IoT based Health Care Monitoring System. pages 55–60, 2016.

[163] Anand Gupta, Hardeo Thakur, Ritvik Shrivastava, Pulkit Kumar, and Sreyashi Nag. A Big Data Analysis Framework Using Apache Spark and Deep Learning. (1), 2017.

[164] Suraj Pandey, William Voorsluys, Sheng Niu, Ahsan Khandoker, and Rajkumar Buyya. An autonomic cloud environment for hosting ECG data analysis services. *Future Generation Computer Systems*, 28(1):147–154, 2012.

[165] European Comission. The GDPR: new opportunities, new obligations. 2018.

[166] US Congress. Health Insurance Portability and Accountability Act of 1996. *Privacy Rule. 45 CFR 164*, 2002.

[167] N. Terry. Des défis existentiels pour la protection des données de santé aux États-Unis. *Ethics, Medicine and Public Health*, 3(1):19–27, 2017.

[168] Karim Abouelmehdi, Abderrahim Beni-Hssane, Hayat Khaloufi, and Mostafa Saadi. Big data security and privacy in healthcare: A Review. *Procedia Computer Science*, 113: 73–80, 2017.

[169] Nicholas Vollmer. Article 5 EU General Data Protection Regulation (EU-GDPR). jun 2018.

[170] Janacek. Bob. Best Practices: Securing Data at Rest, in Use, and in Motion. https://www.datamotion.com/2015/12/best-practices-securing-data-at-rest-in-use-and-in-motion/, 2015. Accessed: 2018-08-17.

[171] Nate Lord. Data Protection: Data In transit vs. Data At Rest. https://digitalguardian.com/blog/data-protection-data-in-transit-vs-data-at-rest, 2018. Accessed: 2018-08-19.

[172] N J Manson. Is operations research really research ? *Orion*, 22(2):155–180, 2006.

[173] Krishnan Nallaperumal and Annam Krishnan. Engineering Research Methodology A Computer Science and Engineering and Information and Communication Technologies Perspective. (December 2013), 2013.

[174] Vijay Vaishnavi and B Kuechler. Design Science Research in Information Systems. *Association for Information Systems*, 2004.

[175] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[176] Hao Yun Kao, Min Chun Yu, Mehedi Masud, Wen Hsiung Wu, Li Ju Chen, and Yen Chun Jim Wu. Design and evaluation of hospital-based business intelligence system (HBIS): A foundation for design science research methodology. *Computers in Human Behavior*, 62:495–505, 2016.

[177] Bill Kuechler and Stacie Petter. Design Science Research in Information Systems. *Design Science Research in Information Systems:*, pages 1–66, 2017.

[178] YIN Yuehong, Yan Zeng, Xing Chen, and Yuanjie Fan. The internet of things in healthcare: An overview. *Journal of Industrial Information Integration*, 1:3–13, 2016.

[179] G B Moody and R G Mark. The impact of the MIT-BIH arrhythmia database. *IEEE engineering in medicine and biology magazine : the quarterly magazine of the Engineering in Medicine & Biology Society*, 20(3):45–50, 2001.

[180] Ary L Goldberger, Luis A N Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-kang Peng, and H Eugene Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals. 2014.

[181] G E Healthcare. MARS Ambulatory Prevention starts with knowledge.

[182] Kuvaev A. Podvyaznikov D. Khudorozhkov R., Illarionov E. Cardio library for deep research of heart signals, 2017.

[183] P Kathirvel, M Sabarimalai Manikandan, S R M Prasanna, and K P Soman. An Efficient R-peak Detection Based on New Nonlinear Transformation and First-Order Gaussian Differentiator. *Cardiovascular Engineering and Technology*, 2(4):408–425, dec 2011.

[184] Carlos Carreiras, Ana Priscila Alves, André Lourenço, Filipe Canento, Hugo Silva, Ana Fred, et al. BioSPPy: Biosignal processing in Python. https://github.com/PIA-Group/BioSPPy/, 2015. Accessed: 2018-01-30.

[185] Ivaylo I. Christov. Real time electrocardiogram QRS detection using combined adaptive threshold. *BioMedical Engineering Online*, 3:1–9, 2004.

[186] W A. H. Engelse and C Zeelenberg. A single scan algorithm for qrs-detection and feature extraction. 6, 11 1978.

[187] Hugo Filipe Silveira Gamboa. Multi-modal behavioral biometrics based on HCI and electrophysiology. (April 2008):1–216, 2008.

[188] Patrick S Hamilton. Open Source ECG Analysis Software Documentation. *Computers in Cardiology*, 2002, IEEE:101–104, 2002.

[189] W. Zong, T. Heldt, G.B. Moody, and R.G. Mark. An open-source algorithm to detect onset of arterial blood pressure pulses. *Computers in Cardiology, 2003*, pages 259–262, 2003.

[190] Filipe Canento. Review and Comparison of Real Time Electrocardiogram Segmentation Algorithms for Biometric Applications. (i), 2005.

[191] Max Pumperla. Elephas: Distributed Deep Learning with Keras and Spark. http://maxpumperla.com/elephas/elephas-distributed-deep-learning-with-keras-spark, 2018. Accessed: 2018-07-19.

[192] Juan S. Arteaga-Falconi, Hussein Al Osman, and Abdulmotaleb El Saddik. ECG and fingerprint bimodal authentication. *Sustainable Cities and Society*, 40(December 2017): 274–283, 2018.

[193] Adrian Condon and Grace Willatt. ECG biometrics: the heart of data-driven disruption? *Biometric Technology Today*, 2018(1):7–9, 2018.

[194] Manjunathswamy B E, Appaji M Abhishek, and Venugopal K R. Multimodal Biometric Authentication using ECG and Fingerprint. *International Journal of Computer Applications*, 111(13):975–8887, 2015.

[195] Andrew Collette and And Contributers. HDF5 for Python. https://www.h5py.org/. Accessed: 2018-06-02.

[196] Shweta H. Jambukia, Vipul K. Dabhi, and Harshadkumar B. Prajapati. Classification of ECG signals using machine learning techniques: A survey. *2015 International Conference on Advances in Computer Engineering and Applications*, (December):714–721, 2015.

[197] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond Accuracy, F-Score and ROC: A Family of Discriminant Measures for Performance Evaluation. pages 1015–1021, 2006. ISSN 0302-9743.

[198] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45(4):427–437, 2009.

[199] Sebastian Ruder. An overview of gradient descent optimization algorithms. pages 1–14, 2016.

[200] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebro, and Benjamin Recht. The Marginal Value of Adaptive Gradient Methods in Machine Learning. (Nips), 2017.

[201] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent. pages 1–22, 2011.

[202] Y Bengio and Y Grandvalet. No unbiased estimator of the variancee of k-fold cross-valudation. *J Mach Learn Res*, 5:1089–1105, 2004.

A

SUPPORT MATERIAL

A.1 DETECTION OF THE QRS COMPLEX

Table 6.: Representation of the effectiveness of each approach to detect the R-peak in the electrocardiogram signals.

Number of sample	Real values	R-Peak Script	BiospPy	WFDB
100	2273	2272	2272	2272
101	1865	1866	1866	1865
102	2187	2187	2187	2187
103	2084	2084	2081	2083
104	2229	2228	2227	2224
105	2572	2573	2592	2579
106	2027	2010	1926	2029
107	2137	2136	2128	2133
108	1774	1784	1789	1755
109	2532	2532	2522	2528
111	2124	2124	2122	2113
112	2539	2539	2538	2539
113	1795	1794	1794	1805
114	1879	1879	1843	1852
115	1953	1952	1952	1953
116	2412	2392	2390	2397
117	1535	1535	1534	1535
118	2288	2280	2277	2277
119	1987	1987	1987	1988
121	1863	1863	1861	1863
122	2476	2476	2475	2475
123	1518	1518	1514	1517
124	1619	1619	1601	1617
200	2601	2600	2597	2598
201	2000	1951	1889	1933
202	2136	2134	2113	2129
203	2980	2857	2900	2902
205	2656	2646	2644	2640
207	2332	2037	2068	2082
208	2955	2934	2140	2936
209	3005	3005	3004	3005
210	2650	2581	2572	2604
212	2748	2748	2746	2748
213	3251	3249	3237	3248
214	2262	2257	2255	2260
215	3363	3354	3359	3356
217	2208	2208	2202	2207
219	2287	2153	2145	2159
220	2048	2047	2046	2047
221	2427	2423	2140	2425
222	2483	2484	2481	2463
223	2605	2605	2568	2604
228	2053	2074	1993	2058
230	2256	2256	2256	2255
231	1573	1571	1569	1571
232	1780	1791	1782	1788
233	3079	3070	3064	3076
234	2753	2752	2749	2752

A.2 EVALUATION METRICS RESULTS

This Appendix presents the calculations performed to find the values of Precision, Recall and F1-Score. Using the values from the confusion matrices the following equations were performed for each class.

As example the values of Precision, Recall and F1-Score for the class 0, SGD optimizer and 150 epochs are here presented:

$$Precision = \frac{1337}{1332 + 1337 + 178 + 112} = 0.9694 * 100 = 96.94\% \tag{11}$$

$$Recall = \frac{1337}{1337 + 94 + 32 + 237} = 0.9736 * 100 = 97.36\% \tag{12}$$

$$F1 - score = 2 * \frac{0.9694 * 0.9736}{0.9694 + 0.9736} * 100 = 97.15\% \tag{13}$$

A.2.1 *Keras without Apache Spark*

Table 7.: Confusion Matrix for each number of epochs tested on the model without Apache Spark integration and with SGD and Adam as optimizers.

Stochastic Gradient Descent				
10 epochs				
	0	1	2	3
0	13046	111	126	627
1	275	3324	45	234
2	652	293	3431	150
3	589	72	43	4242
25 epochs				
	0	1	2	3
0	13092	202	47	526
1	137	3517	21	71
2	422	595	3506	99
3	422	95	4	4504
50 epochs				
	0	1	2	3
0	13640	77	19	104
1	150	3562	34	45
2	582	92	3866	15
3	439	45	2	4588
100 epochs				
	0	1	2	3
0	13693	22	37	243
1	185	3506	56	78
2	11	0	4361	15
3	89	24	2	4938
150 epochs				
	0	1	2	3
0	13371	94	32	237
1	132	3550	36	76
2	178	13	4420	66
3	112	10	1	4932

Adam				
10 epochs				
	0	1	2	3
0	5907	25	7863	126
1	280	728	2739	65
2	103	0	4375	5
3	1468	168	2651	757
25 epochs				
	0	1	2	3
0	6832	4497	725	1814
1	119	3175	40	396
2	677	3257	566	133
3	463	2065	229	2272
50 epochs				
	0	1	2	3
0	10336	429	2064	985
1	200	3240	215	204
2	332	250	3848	163
3	1036	269	581	3108
100 epochs				
	0	1	2	3
0	13162	20	3	710
1	1175	1919	12	747
2	2131	158	1843	401
3	1843	64	6	3066
150 epochs				
	0	1	2	3
0	12171	53	17	1611
1	318	3163	22	272
2	754	565	2752	486
3	356	70	1	4649

Table 8.: Precision results obtained from the tests of the deep learning model without Apache Spark integration regarding each class and with SGD and Adam as optimizers.

Precision

		Number of Epochs				
Optimizer	Class	10	25	50	100	150
SGD	0	89.59%	93.03%	92.09%	97.96%	96.94%
	1	87.47%	79.77%	94.33%	98.70%	96.81%
	2	94.13%	97.99%	98.60%	97.87%	98.46%
	3	80.75%	86.62%	96.55%	93.63%	92.86%
Adam	0	76.14%	84.44%	86.83%	71.88%	89.50%
	1	79.04%	24.43%	77.36%	88.80%	82.13%
	2	24.82%	36.28%	57.36%	98.87%	98.57%
	3	79.43%	49.23%	69.69%	62.27%	66.24%

Table 9.: Recall results obtained from the tests of the deep learning model without Apache Spark integration regarding each class and with SGD and Adam as optimizers.

Recall

		Number of Epochs				
Optimizer	Class	10	25	50	100	150
SGD	0	93.79%	94.41%	98.55%	97.84%	97.36%
	1	85.71%	93.89%	93.96%	91.66%	93.57%
	2	75.81%	75.85%	84.87%	99.41%	94.51%
	3	85.77%	89.63%	90.42%	97.72%	97.57%
Adam	0	42.43%	49.26%	74.82%	94.72%	87.86%
	1	19.10%	85.12%	83.96%	49.81%	83.79%
	2	97.59%	12.22%	83.78%	40.66%	60.39%
	3	15.01%	45.18%	62.23%	61.58%	91.59%

Table 10.: F1-Score results obtained from the tests of the deep learning model without Apache Spark integration regarding each class and with SGD and Adam as optimizers.

F1 Score

		Number of Epochs				
Optimizer	Class	10	25	50	100	150
SGD	0	91.64%	93.72%	95.21%	97.90%	97.15%
	1	86.59%	86.25%	94.15%	95.05%	95.16%
	2	83.98%	85.51%	91.22%	98.63%	96.44%
	3	83.18%	88.10%	93.38%	95.63%	95.16%
Adam	0	54.50%	62.23%	80.38%	81.74%	88.67%
	1	30.76%	37.97%	80.53%	63.82%	82.95%
	2	39.57%	18.28%	68.10%	57.62%	74.89%
	3	25.25%	47.12%	65.75%	61.92%	76.88%

A.2.2 *Keras with Apache Spark*

The same process was used for the evaluation of the model with the Apache Spark integration, following the same steps of calculations regarding the Precision, Recall and F1-score.

Table 11.: Confusion Matrix for each number of epochs tested on the model with Apache Spark integration and with SGD as optimizer.

Stochastic Gradient Descent				
10 epochs				
	0	1	2	3
0	5974	14	0	2
1	3783	230	0	0
2	1990	4	0	0
3	2379	51	0	2
25 epochs				
	0	1	2	3
0	5916	113	21	113
1	1692	2120	128	51
2	1210	133	515	23
3	1493	446	47	408
50 epochs				
	0	1	2	3
0	5756	98	40	224
1	904	2932	19	78
2	1082	214	634	47
3	1137	307	14	943

Stochastic Gradient Descent				
100 epochs				
	0	1	2	3
0	5652	106	110	276
1	127	3757	15	73
2	781	193	874	72
3	727	284	16	1366
150 epochs				
	0	1	2	3
0	5402	65	169	423
1	99	3880	27	64
2	575	165	1092	72
3	283	240	11	1880

Table 12.: Confusion Matrix for each number of epochs tested on the model with Apache Spark integration and with Adam as optimizer.

Adam				
10 epochs				
	0	1	2	3
0	6103	0	0	0
1	3954	0	0	0
2	2000	0	0	0
3	2372	0	0	0
25 epochs				
	0	1	2	3
0	6139	1	0	0
1	3923	0	0	0
2	2003	0	0	0
3	2363	0	0	0
50 epochs				
	0	1	2	3
0	6097	71	5	2
1	478	3445	0	0
2	1161	209	573	2
3	2094	290	0	2

Adam				
100 epochs				
	0	1	2	3
0	5446	41	254	353
1	113	3655	89	67
2	474	108	1299	94
3	305	216	37	1878
150 epochs				
	0	1	2	3
0	5775	47	119	176
1	89	3801	37	30
2	456	152	1281	75
3	429	227	13	1722

Table 13.: Precision results obtained from the tests of the deep learning model with Apache Spark integration regarding each class and with SGD and Adam as optimizers.

Precision

		Number of Epochs				
Optimizer	class	10	25	50	100	150
SGD	0	42.29%	57.38%	64.83%	77.56%	84.95%
	1	76.92%	75.39%	82.57%	86.57%	89.20%
	2	0.00%	72.43%	89.67%	86.11%	84.06%
	3	50.00%	68.57%	72.99%	76.44%	77.08%
Adam	0	42.30%	42.55%	62.02%	85.93%	85.57%
	1	0.00%	0.00%	85.80%	90.92%	89.92%
	2	0.00%	0.00%	99.13%	77.37%	88.34%
	3	0.00%	0.00%	33.33%	78.51%	85.97%

Table 14.: Recall results obtained from the tests of the deep learning model with Apache Spark integration regarding each class and with SGD and Adam as optimizers.

Recall

		Number of Epochs				
Optimizer	class	10	25	50	100	150
SGD	0	99.73%	95.99%	94.08%	91.99%	89.16%
	1	5.73%	53.12%	74.55%	94.59%	95.33%
	2	0.00%	27.38%	32.07%	45.52%	57.35%
	3	0.08%	17.04%	39.28%	57.08%	77.88%
Adam	0	100.00%	99.98%	98.74%	89.37%	94.41%
	1	0.00%	0.00%	87.82%	93.14%	96.06%
	2	0.00%	0.00%	29.46%	65.77%	65.22%
	3	0.00%	0.00%	0.08%	77.09%	72.02%

Table 15.: F1-Score results obtained from the tests of the deep learning model with Apache Spark integration regarding each class and with SGD and Adam as optimizers.

F1 Score

		Number of Epochs				
Optimizer	class	10	25	50	100	150
SGD	0	59.40%	71.82%	76.76%	84.16%	87.00%
	1	10.67%	62.33%	78.35%	90.40%	92.16%
	2	0.00%	39.74%	47.24%	59.56%	68.19%
	3	0.16%	27.30%	51.07%	65.36%	77.48%
Adam	0	59.45%	59.69%	76.19%	87.61%	89.77%
	1	0.00%	0.00%	86.80%	92.02%	92.89%
	2	0.00%	0.00%	45.42%	71.10%	75.04%
	3	0.00%	0.00%	0.17%	77.80%	78.38%

A.2.3 *Test Dataset Results*

The next two tables present a sample of the results obtained from the data collected from the Hospital of Braga.

Table 16.: Confusion Matrix from the 8 records obtained from the Hospital of Braga and classified with the best model obtained previously.

1				
	0	1	2	3
0	122527	120	93	429
1	119	0	0	8
2	0	0	0	0
3	713	13	11	238
2				
	0	1	2	3
0	65837	142	4364	4203
1	24	1	2	2
2	0	0	0	0
3	1	0	0	0
3				
	0	1	2	3
0	38240	15782	9284	26294
1	193	90	92	127
2	0	0	0	0
3	83	25	84	39
4				
	0	1	2	3
0	94499	1111	1231	2300
1	0	0	0	0
2	0	0	0	0
3	9	0	0	0

9				
	0	1	2	3
0	88843	835	93	6206
1	819	35	0	118
2	0	0	0	0
3	2977	33	0	342
10				
	0	1	2	3
0	54818	1631	9115	16660
1	264	5	40	115
2	0	0	0	0
3	179	4	40	101
11				
	0	1	2	3
0	70507	43	456	1149
1	0	0	0	0
2	0	0	0	0
3	7	0	0	1
12				
	0	1	2	3
0	4470	76114	17	5702
1	556	20945	3	812
2	0	0	0	0
3	36	238	0	10

Table 17.: Precision from the 8 records above-mentioned obtained from the Hospital of Braga and classified with the best model obtained previously for each class.

Precision

	Number of Record							
Class	1	2	3	4	9	10	11	12
0	0.99	1.00	0.99	1.00	0.959	0.992	1.000	0.883
1	0.00	0.01	0.01	0.00	0.039	0.003	0.000	0.215
2	0.00	0.00	0.00	0.00	0	0	0	0
3	0.35	0.00	0.00	0.00	0.051	0.0060	0.001	0.002

Table 18.: Recall from the 8 records above-mentioned obtained from the Hospital of Braga and classified with the best model obtained previously for each class.

Recall

	Number of Record							
Class	1	2	3	4	9	10	11	12
0	0.99	0.88	0.43	0.95	0.93	0.67	0.98	0.05
1	0.00	0.03	0.18	0.00	0.04	0.01	0.00	0.94
2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
3	0.24	0.00	0.17	0.00	0.10	0.31	0.13	0.04

Table 19.: F1-Score from the 8 records above-mentioned obtained from the Hospital of Braga and classified with the best model obtained previously for each class.

F1-Score

	Number of Record							
Class	1	2	3	4	9	10	11	12
0	0.99	0.94	0.60	0.98	0.942	0.797	0.988	0.10
1	0.00	0.01	0.01	0.00	0.037	0.005	0.000	0.00
2	0.00	0	0.00	0.00	0.0000	0.000	0.000	0.000
3	0.29	0	0.003	0.00	0.068	0.012	0.002	0.00

A.3 APACHE SPARK JOBS

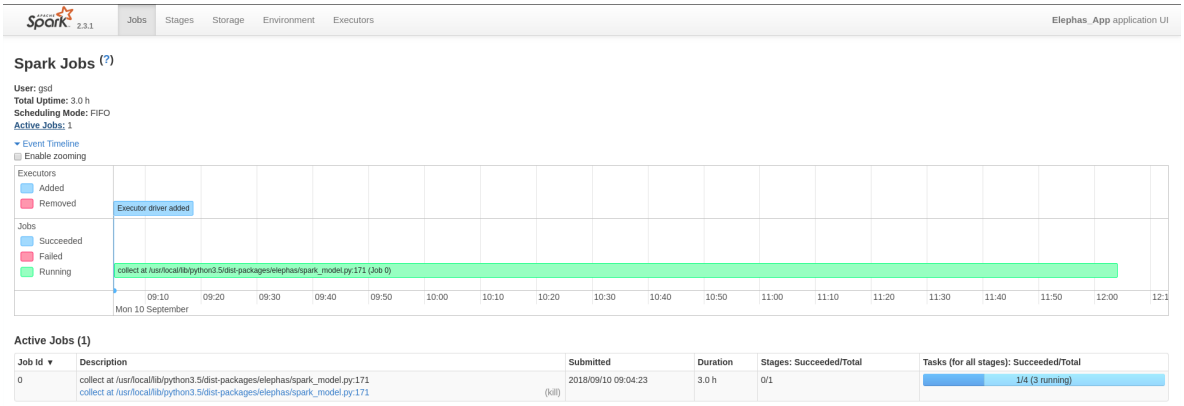


Figure 35.: Apache Spark Job with four workers running locally.

NB: Schrodinger's Cat