SAPIENZA
UNIVERSITÀ DI ROMA

Arrhythmia classification from ECG signals

Dipartimento di Informatica, Automazione e Gestionale

Corso di Laurea Magistrale in Engineering in Computer Science

Candidate

Angelo Catalani
ID number 1582230

Thesis Advisor

Prof. Aris Anagnostopoulos

Co-Advisor

Prof. Ioannis Chatzigiannakis

Academic Year 2018/2019

Thesis defended on 15 July 2019
in front of a Board of Examiners composed by:

Prof. Nome Cognome (chairman)

Prof. Nome Cognome

Dr. Nome Cognome

Arrhythmia classification from ECG signals
Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Version: July 3, 2019

Author's email: catalaniangelo@gmail.com

Dedicated to
my parents

Abstract

A programmer is a problem solver and its activity requires a high level of creativity, as well as multidisciplinary knowledge. This project, allowed me to implement innovative algorithms for medical purposes: we strongly believe in technological advancement as the key to improve the quality of life and the world around us.

Acknowledgments

we would like to express my special thanks of gratitude to my professors (Aris Anagnostopoulos and Ioannis Chatzigiannakis) who gave me the golden opportunity to do this wonderful project.

Secondly, we would also like to thank my parents and relatives who sustained me a lot over my studies.

Contents

1 Introduction **1**
1.1 Document Structure . . . 3

2 Principles of ECG Evaluation **5**
2.1 Heart's Anatomy . . . 5
2.1.1 Electrical Cell . . . 5
2.1.2 Heartbeat Cycle . . . 6
2.2 ECG tracing . . . 7
2.2.1 Heartbeat Shape . . . 8
2.2.2 Type of Arrhythmias . . . 9

3 Pre-Processing **11**
3.1 De-Noise . . . 11
3.1.1 Median Filter . . . 12
3.1.2 Discrete Wavelet Transform . . . 12
3.2 Data Augmentation . . . 14
3.2.1 Smote and Adasyn . . . 15

4 Classification Algorithms **17**
4.1 Support Vector Machine . . . 17
4.2 Large Margin Principle . . . 17
4.2.1 Kernel Trick . . . 19
4.3 Feedforward Neural Network . . . 20
4.3.1 Weighted Loss Function . . . 21
4.4 Convolutional Neural Network . . . 22
4.4.1 Convolutional Layer . . . 23
4.4.2 Pooling Layer . . . 24
4.4.3 Mathematical Interpretation . . . 24
4.5 Recurrent Neural Network . . . 26
4.5.1 Vanilla RNN . . . 27
4.5.2 Long Short Term Memory . . . 28
4.6 Attention . . . 31
4.6.1 Encoder-Decoder model . . . 31
4.6.2 Attention in Encoder-Decoder . . . 32
4.6.3 Multiplicative and Addititive Attention . . . 33
4.7 Transformer . . . 33

4.7.1 Transformer's Attention 34
4.8 Neural Ordinary Differential Equations 35
4.8.1 Residual Neural Network 35
4.8.2 NODE 37

5 Experiment **41**
5.1 Dataset 41
5.1.1 Training and Test Set 42
5.2 Performance Measures 43
5.3 Random Forest [17] 43
5.3.1 Custom Variation 44
5.4 Support Vector Machine [15] 46
5.5 Convolutional Neural Network [24] 47
5.6 Neural Ordinary Differential Equations 50
5.6.1 Test 1 51
5.6.2 Test 2 53
5.6.3 Test 3 53
5.7 LSTM 53
5.7.1 Test 1 55
5.7.2 Test 2 55
5.7.3 Test 3 56
5.8 LSTM/NODE with Attention 57
5.9 Transformers 59
5.10 Per Patient Dataset 60
5.10.1 Test 1 60
5.10.2 Test 2 62

6 Conclusions **67**

Chapter 1

Introduction

Cardiovascular diseases (CVDs) are the first cause of death worldwide: 18 million people died from CVDs in 2016, representing 31% of all global deaths ([29]). This project is focused on the cardiovascular diseases related to complications with the electrical system that regulates the steady heartbeat and generates abnormal heartbeats called arrhythmias. Many people, during life, can experience irregular heartbeats and generally, they are harmless and happen in healthy people free of heart disease. However, some abnormal heart rhythms can be even deadly.

A standard tool to diagnose arrhythmias is the electrocardiogram (ECG): measurement of the heart's electrical activity. Since it is a non-invasive and painless test for the patients, it can be employed to collect massive volumes of data, and subsequently analyze them, to automatically detect arrhythmias. The equipment to produce electrocardiograms is principally present in hospitals, where cardiologists use their knowledge and experience to carefully evaluate the ECG. However, this activity is time-consuming and prone to error: an automatic approach can support them during their decision.

In addition to this, accurate and low-cost diagnosis of arrhythmic heartbeats is highly desirable since it can detect early heart problems and prevent dangerous consequences. This is the reason it is not surprising recent literature is focused on this topic. In general, the ECG's analysis is constituted of two phases:

1. beat detection: extrapolate the temporal segment of the ECG related to the contraction of the heart
2. arrhythmia classification: detection of arrhythmic beats

This work is concentrated only on the latter, that is more challenging because until now, there is not any perfect solution to the problem. We propose a series of algorithms able to automatically detect arrhythmias, with the aim for the future, to implement them on wearable devices. In detail, several works have proposed a complete infrastructure to integrate that device in a fog-computing organization. Specifically, [32] and [10], describe an environment with:

1. Wearable Device. It collects data from the ECG sensor and detects arrhythmias with suitable algorithms

2. Edge Device. It is the patient's device with software to communicate with the wearable device and perform more complex computation

3. Cloud Services. They offer all the necessary features to facilitate authentication, notifications, data integration and other services relying on privacy-preserving data integration

The described fog computing approach extends the cloud computing paradigm by migrating data-processing closer to the production site, thus accelerating the system's responsiveness to events ([4],[3],[32]). This prototype will allow patients to share information to their physicians, monitor their health status independently and notify the authorities rapidly in emergencies. Historical data will also be available for further analysis, towards identifying patterns that may improve medical diagnoses in the foreseeable future ([5]). The performance of the system is evaluated concerning the accuracy of detecting abnormal events and the power consumption of the wearable device. Results indicate that a very high percentage of success can be achieved in terms of event detection ratio and the device being operative up to several days without the need for a recharge ([2], [11]).

We use the *MIT-BIH Arrhythmia Database dataset*: it is open source and consists of 47 patient's ECG recording. Initially, we have investigated the literature about the pre-processing techniques to remove noise from the ECG and deal with unbalanced classes in the dataset: Discrete Wavelet Transform and Median Filter for de-noise; Smote, Adasyn for weighted loss (data augmentation). Secondly, we have tried a simple model based on Random Forest, and subsequently more complex. In particular, we have re-implemented some of the best algorithms in the literature based on Support Vector Machine and Convolutional Neural Network.

The latter has an excellent accuracy we have attempted to improve with my neural network models. Specifically, we have adopted:

1. Neural-ODE. It is a novel approach presented in NeurIPS 2018, where the dynamic of a neural network is modeled using a first-order differential equation

2. Long Short Term Memory (LSTM). Since the ECG is a time series, LSTM is acknowledged to have good result since can capture long term dependencies without the vanishing gradient problem

3. Attention Layer. It is a layer widely used in Natural Language Processing (NLP) to explicitly force the networks to learn the portion of the input to pay attention.

4. Transformer. It is a model introduced by Google for NLP. we have re-adapted it for our task.

In the models above, the training and test set are obtained considering all the 37 patients in the dataset: this is a common practice in the literature. The described models reach the state of the art accuracy for arrhythmia detection.

Finally, we have removed from training data three patient and considered them as test set. In particular, we have first trained the LSTM model and then built a different classifier for each of the three patients in several ways:

1. Removing the last layer of the trained LSTM to get some features for each beat and then applied SVM
2. Retraining the LSTM completely
3. Retraining the last four layers of the LSTM

In this case, the performance is still good and this means our models can be employed in wearable devices and achieve high accuracy after a short period of personalized fine-tuning on the specific users.

1.1 Document Structure

In **Chapter 2**, we will discuss the heart's anatomy and the shape of beats and arrhythmias in an ECG.

Then in **Chapter 3** and **Chapter 4**, we will cover pre-processing and feature extraction techniques applied to the ECG, to improve:

1. information's quality: signal filters and discrete wavelet transform
2. classification performance: data augmentation

In **Chapter 5** we will describe the implemented algorithms, with the corresponding results, mainly based on Random Forest, Support Vector Machine and Neural Network.

Finally, in **Chapter 6**, we will compare some of the best models found in the literature with my approaches: the latter take advantage of the last advancement in Machine Learning and reach the state of art's performance.

Chapter 2

Principles of ECG Evaluation

This chapter explains the underlying mechanism behind the heart beating and how this process is visible on the ECG. The importance of this argument relies on the consequent ability to define ECG's key features for arrhythmias detection.

2.1 Heart's Anatomy

The human heart pumps blood to the body, as the result of electrical impulses and mechanical response, due to conductive and mechanical cells. The contraction is possible only in the presence of an electrical stimulus but in some pathological cases, the inverse is said to be true. This is the reason why to determine the health state of this muscle, it is of paramount importance to asses both the mechanical and electrical functions. The first, one can be monitored with blood pressure, pulses while the latter with the ECG. The latter gives a graphic display of the electrical activity in the heart so that it is possible to identify arrhythmias as recognizable patterns. For example, it is straightforward to presume the ECG of Fig. 2.1 contains 5 beats.

2.1.1 Electrical Cell

Electrical cells have the exclusive capability to discharge electrical current without the necessity of external stimuli: this property is called automaticity, and the underlying mechanism is the sodium pump. In a resting cardiac cell the electrical charges are equally balanced and there is no electrical flow: the cell is said to be polarized or in the ready state. The automaticity property, allows this cell to modify the membrane and pull sodium inside and potassium outside, so that the resulting difference of potential causes an electrical flow: this is the basic sodium pump functionality

Figure 2.1. Example of ECG trace

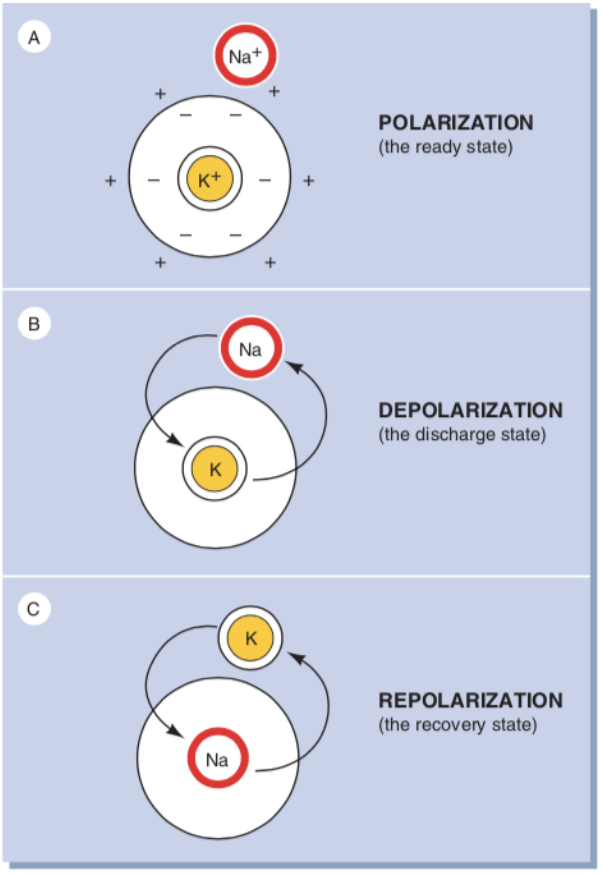

Figure 2.2. Electrical cell transactions: [37]

that makes the cell in a state called depolarized or discharge state. Finally, after this step, the reverse process to turn the depolarized state in the original restring one, is called repolarization and it consists of pulling out of the cell's membrane the sodium and inside the potassium, bringing, in the end, the cell to the original resting state. The whole process is described in Fig. 2.2. The heart is made of 4 chambers: two atriums in the upper part and two ventricles downside. The electrical cells conduct the current flow through a set of connections that overall constitute the conductive system and completely cover the heart's surface: this allows the consequent contraction of atriums and ventricles.

2.1.2 Heartbeat Cycle

Normally, the electrical impulse is produced in the sinoatrial (SA) node positioned in the upper part of the right atrium. Then, it propagates down to the right atrium with the internodal pathways to reach the atrioventricular node (AV) and left with the interatrial pathways to reach the left atrium. The entrance to the left and right ventricles is provided by the Bundle of His, through the left and right bundle branch, that terminates with Purkinje fibers who are in charge to contract the ventricles. It is interesting to note not all the portion of the heart have the same beat rhythm (beats per minute):

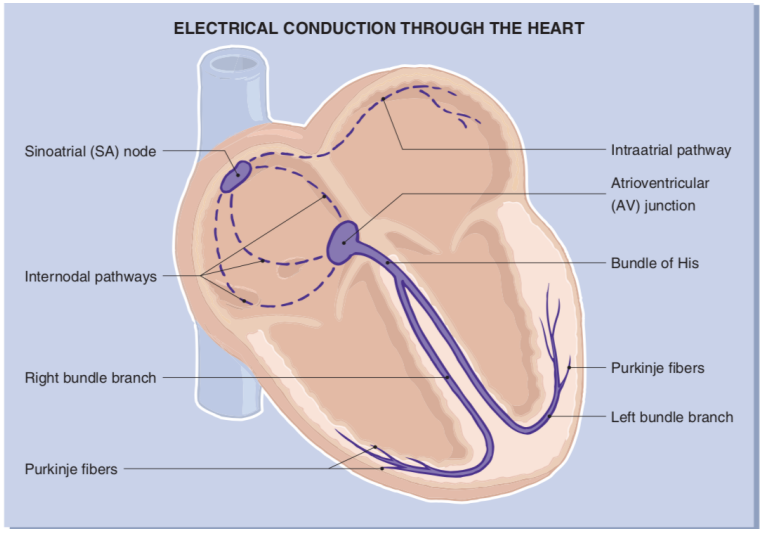

Figure 2.3. Electrical tissue of heart: [37]

1. SA node: 60-100
2. AV node: 40-60
3. Ventricles: 20-40

The discussed structure is depicted in Fig. 2.3. In the ECG, in normal condition, the visible beat rate is due to SA node because it has the highest frequency and overrides all the other: this is the reason why it is defined as the pacemaker. The above values are called inherent rates and can help us to infer the heart's dysfunctions visible in the ECG as a different SA rhythm. In a pathological context, for example, if the SA node fails, the pacemaker becomes the AV node and a slower rhythm is visible on the ECG: this callback safety mechanism is named escape.

2.2 ECG tracing

The acquired knowledge in the previous subsection is essential to investigate how the electrical activity is transferred to graph paper to be analyzed for our task. The ECG is obtained placing some electrodes on the patient skin and connecting it, to a machine that will display the electrical activity. The location of the electrodes is crucial for the final output and it is possible to obtain multiple views (or lead) of the same heart, just by varying their position. In particular, some arrhythmias are more visible on a given view, and in general, an effective diagnosis interpolates more leads. For example, in the dataset we consider, each ECG has two leads:

1. modified limb lead II (MLII) where the electrodes are on the chest.
2. modified chest lead V1 (MCL1) where the negative electrode is placed close to the left shoulder and the positive electrode to the right of the sternum

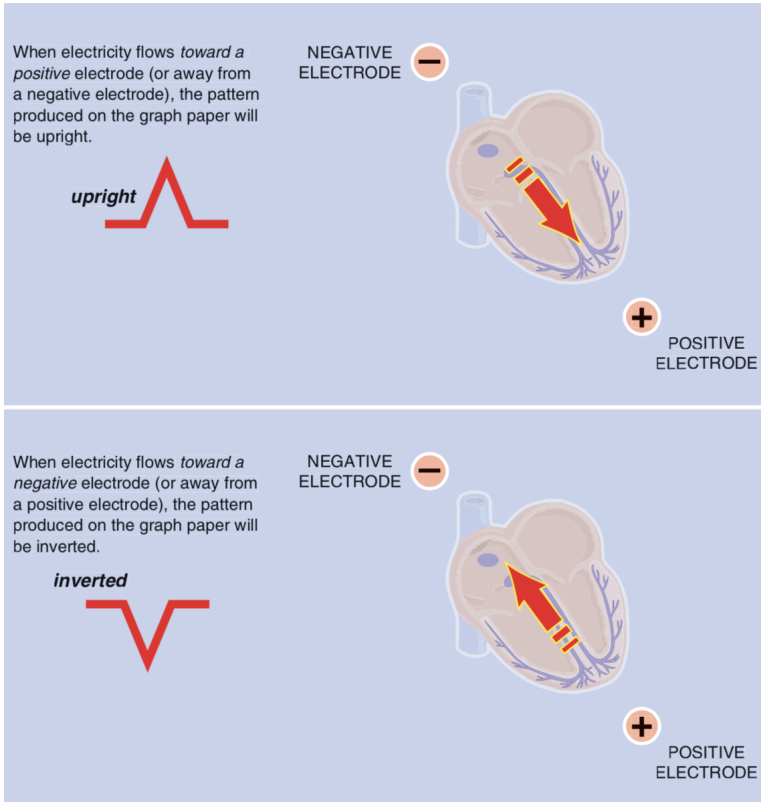

Figure 2.4. Lead II basic principles: [37]

For a long time, the most adopted lead has been the Lead II, and in the following, we will take into consideration this configuration. However, there are different leads with better effectiveness. In Lead II the negative electrode is placed to the right, above the heart, while the positive to the left, behind the heart. In this way, when the electrical impulse propagates from the atrium to the ventricle, the ECG device will display an upright wave, while the upside-down shape is generated in the opposite case (Fig. 2.4). Finally, the ECG is written on a conventional graph paper (Fig. 2.1) where the voltage levels are measured comparing the horizontal lines, and the time interval, considering the vertical lines: two consecutive vertical splits cover 0.04 seconds.

2.2.1 Heartbeat Shape

During a normal beat, the heart first enters in the atria and then into the ventricles: this requires a synchronized mechanism so that while the atria fill, the ventricles contract and vice-versa. During each phase of the cardiac cycle, a distinct pattern is produced on the ECG graph paper. This pattern describes instantly what is happening in the heart and it has the shape depicted in Fig. 2.5. The beat shape starts with a deflection: the P wave and it is generated by the SA node. Subsequently, the impulse leaves the atria to reach the AV node and it encounters a slight delay because the encountered electrical tissues are slower. This is translated into a short period of electrical inactivity called the PR segment. The subsequent wave is the

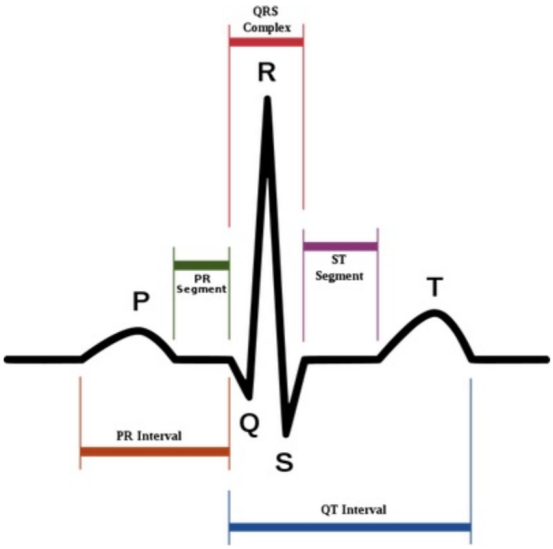

Figure 2.5. Heartbeat in the ECG

QRS complex, due to ventricular depolarization and it is significantly larger than the P wave because to ventricles are greater than atria, so that more potential energy is required. The Q wave is defined as the first negative deflection following the P wave but before the R wave. This wave flows immediately into the R wave, which is the first positive deflection following the P wave. Next comes the S wave, which is defined as the second negative deflection following the P wave, or the first negative deflection following the R wave.

After the ventricles depolarize, they begin their repolarization phase, which results in another wave in the ECG. The T wave is indicative of ventricular repolarization. The atria also repolarize, but this event is not significant enough to show up in the ECG. Between the S and the T wave is a section called the ST segment. The ST-segment is the flat, isoelectric section of the ECG between the end of the S wave and the beginning of the T wave. It represents the interval between ventricular depolarization and repolarization, and anomalies in this segment are a symptom of infarction.

2.2.2 Type of Arrhythmias

Arrhythmia analysis is a complex task due to the high variability of the heart's mechanism inside each patient. Arrhythmias are classified into two classes:

1. rhythmic: constituted of a series of irregular beats
2. morphological: single abnormal beat

This work is concentrated on the latter. Morphological arrhythmias are classified into 4 macro classes by the Association for the Advancement of Medical Instrumentation (AAMI) standard, with respect to the location where the anomaly starts:

Figure 2.6. Examples of Arrhytmias

CLASSES	N	S	V	F
DEFINITIONS	Normal Beat	Atrial Premature Beat	Premature Ventricular Contraction	Fusion of Ventricular and Normal Beat
	Left Bundle Branch Block Beat	Aberrated Atrial Premature Beat		
	Right Bundle Branch Block Beat	Nodal (junctional) Premature Beat	Ventricular Escape Beat	
	Atrial Escape Beat	Supraventricular Premature Beat		
	Nodal (junctional) Escape Beat			

Figure 2.7. Detailed classification of arrhythmias

1. N-class. It happens when the beat regularly starts at the SA node but can be blocked in the left or right bundle branch. It includes a regular beat. For example, the left bundle branch block happens when the impulse does not reach the left ventricle because the left bundle is compromised: in this situation, first, the right ventricle will depolarize and then also the left one. This results in a QRS complex longer than 120 ms.

2. S-class. It happens when the impulse starts below the SA node: in this situation the impulse will begin at the bottom and subsequently will reach the upper part, resulting in an upside-down P wave.

3. V-class. It happens when the impulse originates in the ventricles and it shows up in the ECG as a tall and wide QRS

4. F-class. It happens when there are multiple sources of depolarization: for example in Fig. 2.6 the P wave is modified by an additional source of depolarization

A more detailed arrangement of the above classes are depicted in the figure below.

Chapter 3

Pre-Processing

As we have previously, discussed, the ECG tracing is an empirical process prone to noise measurements whose removal is of high importance to guarantee the detection of the relevant features without incurring into overfitting issues. The principal methods to remove noise are based on frequencies domain analysis : in the following we will describe Median Filter and Discrete Wavelet Transform. In addition to this, the dataset we are taking into consideration is highly overbalanced: one class constitutes more than 80% of all the elements: this can lead to inconsistencies because an algorithm could reach high accuracy, merely predicting only that class without any generalization competence. For the purpose of handling the latter, we have used several approaches based on data augmentation and weighted loss function.

3.1 De-Noise

The ECG can be affected by several sources of noise during the recording, attributable to :

1. Baseline wanders caused by the patient's movements.It is visible in the signal with frequencies below 0.5 Hz (Fig. 3.1.a).

2. Power line interference caused by electromagnetic external devices. It appears in the signal as 50/60 Hz sinusoids (Fig. 3.1.b).

3. Muscle artifact caused by muscle activity near the electrode for the contraction of the heart's patient. It is difficult to be removed since it is strictly related to the electrical impulse of the ventricles that determines the QRS complex on the ECG. Its removal can result in unwanted distortion of the signal (Fig. 3.1.c).

It is crucial to note the role of de-noise algorithms is to clean the signals without losing relevant information: after the processing, the filtered signal must contain the original key features of a beat in terms of P, Q, R, S, T wave. In this project we have considered 2 different ways of noise removal: median filter and discrete wavelet transform (DWT).

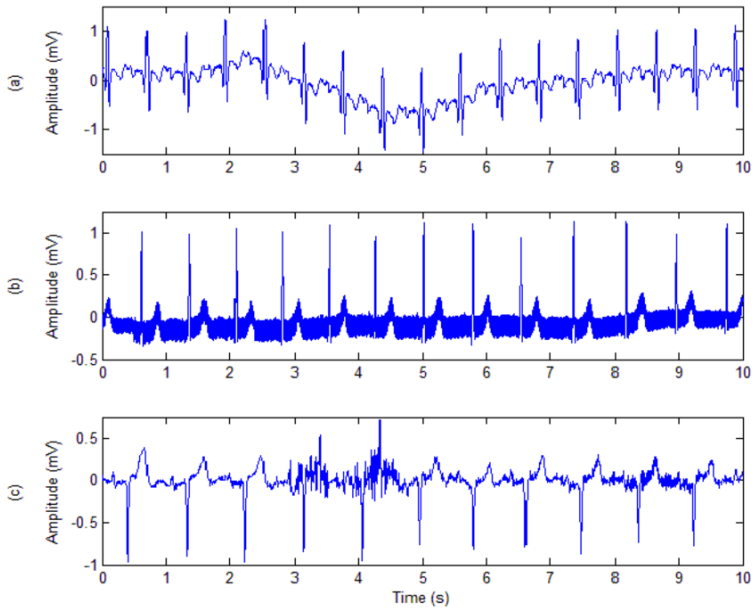

Figure 3.1. Example of noise in ECG recordings. (a) Baseline wander, (b) Power line interference, and (c) Muscle artifact. [9]

3.1.1 Median Filter

A median filter takes as input a signal $S = [x_1, \ldots, x_n]$ and a window of size: w. It returns a converted signal where for every sample : $x_i \in S$, it is computed the median of the elements : $[x_i, x_{i+1}, \ldots, x_{(we+w) \ mod \ n}]$. The the intuition behind a median filter is that noisy samples tend to have too high or too low values, with respect to the neighbor samples: replacing them with the median in a given range can be a good approach. This the technique is used in [18] to mitigate the noise due to baseline wander.

3.1.2 Discrete Wavelet Transform

The ECG signal is non-stationary because beats are generated by the heart under influences of physiological factors that depend on the brain and change continuously over time. In order to characterize it, a simple approach could consist of a time-domain analysis, taking into consideration factors like the duration of QRS complex or ST segment. This approach does not fit with the subtle changes in the amplitude and duration in the ECG that are specific for any patient.Another the approach is to perform a frequency domain analysis with Fourier Transform. However, the latter is not perfectly applicable because it is not accurate to assume that a given frequency is equally distributed in the signal: this is only true for a periodic, stationary signal. In particular, to represent the ECG signal we are interested not only in the frequency contained but also in the portion of the signal exhibits a given frequency.

With the aim of having a better resolution in time, it is possible to study the frequencies of the signal present in a temporal window of the signal: for example we can study the frequencies of the signal from $t = 0$ to $t = 0.1$, so that we can infer

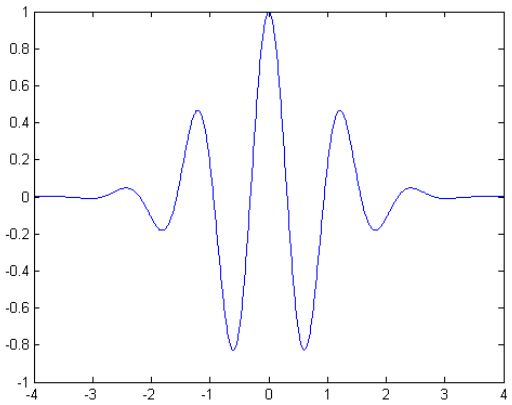

Figure 3.2. Wavelet named Meyer

the frequencies localized in the interval.

However, the drawback of this approach is that it is not possible to detect all the frequencies because in that window we can only find the frequencies greater than $\frac{1}{0.1} = 10Hz$, since for frequencies greater than $10Hz$ we need to consider a window larger than $0.1s$. The explained tradeoff between frequency and time resolution is the application of the Heisenberg's Uncertainty Principle. In order to mitigate this issues: the optimum approach is to consider a small window for high frequencies and large window for low frequencies. In fact, when a signal component is rapidly changing it is essential to detect the instant when this happens. On the contrary, when a component repeats slowly it is necessarily a larger window to detect it.

The Discrete Wavelet Transform (DWT) takes advantage of this latter consideration. DWT considers as basis function a wavelet instead of sinusoid employed by Fourier Transform. A wavelet is a function with a limited duration: it has an amplitude that begins at zero, increases, and then decreases back to zero. A wavelet is more localized in time than a sinusoid because it exists only in a restricted interval (Fig. 3.2). The main parameters of a wavelet are:

1. scaling.The scale of a wavelet is a measure of how it is expanded, so that the higher the scale the more it is spread and the less it is localized in time (Fig. 3.3).

2. shifting.The wavelet is shifted across the input signal to align it with the features we are interested in, for example the QRS complex.

In general, the idea of DWT is to analyze the correlation between the signal and a given wavelet shifted over the signal with increasing scale: initially it is analyzed high-frequency behavior and at each successive step the scale is doubled and the signal downsampled for the Nyquist criterion. In conclusion, DWT outputs the signal at different bandwidth, such that at each level of analysis, the output is high, if there is a strong correlation of the signal at that frequency with the wavelet at

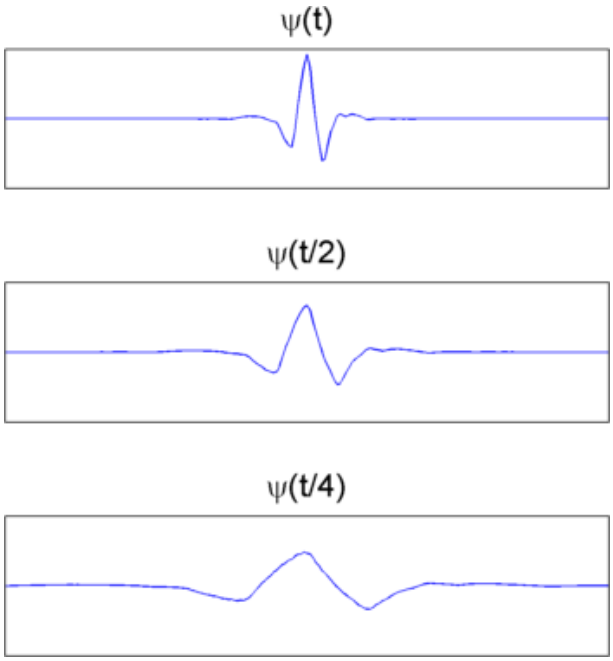

Figure 3.3. Wavelet with scaling factor of 1,2,4: [27]

that scale.

There are several families of wavelet, and the choice of the correct one, relies on the similarity of shape with the portion of ECG we are interested (Fig. 3.4) . For example, a common choice is the Meyer wavelet (Fig. 3.2) because it is comparable to the QRS complex. DWT is mainly used as :

1. a filter ignoring the coefficients at a given frequency level while reconstructing the signal ([16]).
2. feature selection considering the coefficients with high value, or with high correlation with the original signal ([18],[16]).

3.2 Data Augmentation

The dataset's details will be discussed later: so far it is important to note it is overbalanced. In particular, the N class is more than 80% of the whole dataset. Models trained on imbalanced dataset do not acquire any generalization power: they are biased toward the most populated class and fail to detect the distinguishing features of the others. For example in our case, a model could reach a high accuracy during the train just by learning to always predict the same class.

Class imbalance is typical of biological dataset, where the interest is towards minority samples that are linked to rare events ([21]). We have tried to solve this issue using the following methods: Smote, Adasyn and weighted loss function for neural networks. In the following, we will use the following notation:

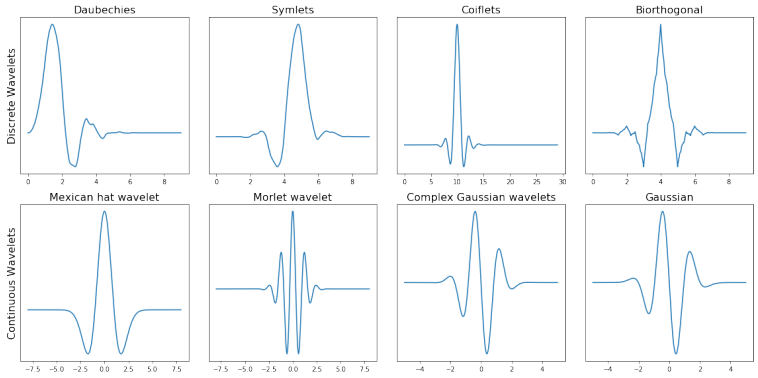

Figure 3.4. Wavelet families: [35]

1. Dataset: D, with n samples: $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
2. Classes: $Y = \{1, -1\}$ such that $|y_i \in Y$. For now we consider only two classes, but the same considerations can be generalize to N classes
3. m_s and m_l is the number of samples belonging respectively to the under-balanced and over-balanced class: $m_s \leq m_l$ and $m_s + m_l = m$

3.2.1 Smote and Adasyn

To build a consistent dataset, it is possible to add new elements to the less populated class (over-sampling) or to remove existing samples of the most populated class (undersampling). We have adopted the first one, to take advantage of all the information contained in the dataset even for unbalanced classes. The main oversampling algorithms are Smote ([8]) and Adasyn ([21]). The latter is an improved version of the first one. In general, Smote takes into consideration a given class and generates N additional synthetic examples for each original element. For example, if N is 1, the size of the class is doubled. Specifically, this algorithm performs the following steps:

1. take a sample: x_i, and consider the difference vectors $z_1, \ldots, z_n$ between x_i and the K closest neighbors of the same class: $x_j, \ldots, x_{j+K}$. In the end, $z_j = x_i - x_j$ where $1 \leq j, we \leq n$
2. for every $N \leq K$, randomly chosen, difference vectors: z_j, returns a new sample: $s_j = x_i + r \cdot d_j$ where $r \in [0, 1]$ is a random value.

In the end, a new point is randomly positioned in the segment whose extremities are the original sample and its neighbor. This approach effectively forces the decision region of the minority class to become more general ([8]). However, Smote does not create representative samples when the under-balanced class has extreme values: in this case, the created element will be centered in the dataset space even if the under-balanced samples will probably lie on the boundary.

Adasyn is an advanced version of Smote: the main difference is the first automatically detect the number of samples to generate: g_i, with respect to the element

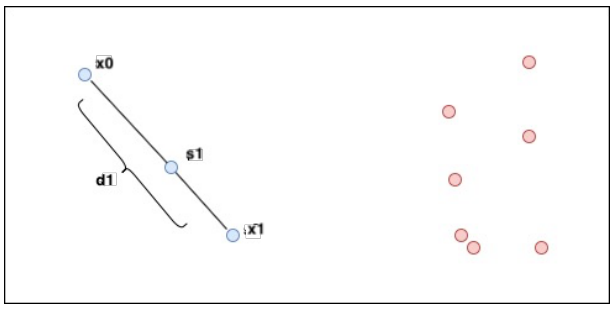

Figure 3.5. The blue class is underbalanced: Smote generates a new sample: s_1 between x_0 and x_1

considered in the under-balanced class: x_i.In general, more samples will be created for samples whose neighbors belong to another class. The idea is that missing points should concentrate in the dataset space where a given element is under-represented. Specifically, it performs the following steps:

1. $d = \frac{m_s}{m_l} \in [0, 1]$ is a measurement of class imbalance. We define a threshold value: d_{th} as limit to the maximum class unbalance: if $d_i \leq d_{th}$ the algorithm terminates.

2. $G = (m_l - m_s) \cdot \beta$, where $\beta \in [0, 1]$. G is the total number of new samples to generate. β is user defined, for example if $\beta = 1$, the resulting dataset is completely balanced.

3. $r_i = \frac{\Delta_i}{K}$ $we \in \{1, \ldots, m_s\}$, where:

   (a) Δ_i is the number of the K closest neighbors to x_i, belonging to the other class: this is different from Smote where we consider only the samples of the same class as neighbors

   (b) $r_i \in [0, 1]$ and it indicates how x_i is isolated from the rest of points belonging to the same class

4. $0 \leq \hat{r_i} = \frac{r_i}{\sum_{i=0}^{m_s} r_i} \leq 1$. In this way, r_i is normalized: $\sum_{i=0}^{m_s} \hat{r_i} = 1$

5. $g_i = \hat{r_i} \cdot G$. It is the number of samples to generate for each minority sample: $\sum_{i=0}^{m_s} g_i = G$

6. for each minority sample creates g_i samples s_j, in the same way of Smote: $s_j = x_i + r \cdot d_j$

The idea is that $\hat{r_i}$ measures how difficult it is, to learn the features of x_i: greater values of r_i are indicative that x_i is an outlier and hard to generalize it: the more a sample is difficult to learn, the more Adasyn will generate new elements related to it. Finally, it is possible to generalize Adasyn to the multi-class case just by varying the value of Δ_i: it can be computed as the number the samples belonging to the most populated class or to any different class.

Chapter 4

Classification Algorithms

4.1 Support Vector Machine

Support vector machines (SVMs) belongs to the class of supervised learning algorithms and can be used both for classification and regression. We will discuss SVM for classification task because arrhythmia detection belongs to the latter class. The goal in classification is to take an input vector x and to assign it to one of K discrete classes: C_k where $k = \{1, \ldots, K\}$: a discriminant is a function that performs this classification.In the following we will assume for simplicity $K = 2$ and in the end we will show how to generalize this approach.

In SVM, the discriminant function is linear in the weights and it has the form: $y(x) = w^T\phi(x) + w_0$ where w is called a weight vector, and w_0 is a bias. In the simple case where ϕ is the identity function, if x is D-dimensional, the discriminant function is in the form of: $w^T x + w_0$ and it defines hyperplanes in the $D - 1$ space. Specifically, the input vector x is assigned to class C_1 if $y(x) \leq 0$ and to class C_2 otherwise. In the general case where ϕ can be any function, the same consideration holds, except that the input space is not D but it depends on the specific input transformation performed by ϕ.

4.2 Large Margin Principle

In this section, we will describe how SVM determines the surface boundary: in order to simplify the discussion we will assume there are only two classes: $y_i = \{1, -1\}$ and ϕ is the identity function. Given a generic point: x_i in a set of size: N, and a discriminant function: $y(x) = w^T x + w_0$, we have that:

1. $y(x) = 0$ is the decision boundary
2. $r = \frac{f(x_0)}{||w||}$ is the distance between x_0 and the decision boundary

The goal of SVM is to maximize the distance between the closest point to the decision boundary and the latter, while ensuring all points are correctly classified. Specifically, we define the following optimization problem:

1. $max_{w,w_0} min_{i=0}^{N} \frac{y_i(w^T x_i + w_0)}{||w||}$

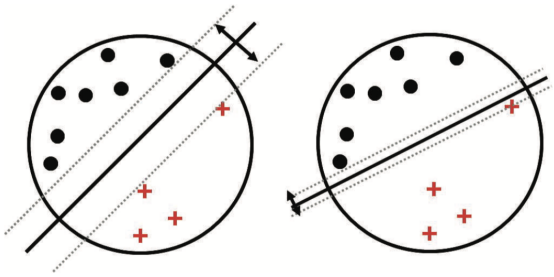

Figure 4.1. Soft Margin principle: left has a wider margin than right [7]

2. s.t $y_i \cdot f(x_i) > 0$

The same problem can be rescaled so that the minimum distance is set to one and it becomes:

1. $min_w \frac{1}{2} \cdot ||w||^2$
2. s.t $y_i \cdot f(x_i) \geq 1 \; \forall we \in \{1, \ldots, N\}$

The second constraint says that we want all points to be on the correct side of the decision boundary with a margin of at least 1. For this reason, we say that an SVM is an example of a large margin classifier.

If the data is not linearly separable, there will be no feasible solution in which $y_i \cdot f(x_i) > 0$. We can introduce slack variables $\xi_i \geq 0$ such that $\xi_i = 0$ if the point is on or inside the correct margin boundary, and $\xi_i = |y_i - f(x_i)|$ otherwise. If $0 \leq \xi_i \leq 1$ the point lies inside the margin, but on the correct side of the decision boundary. If $\xi_i > 1$, the point lies on the wrong side of the decision boundary (Fig. 4.1). We replace the hard constraints that $y_i \cdot f(x_i) \geq 1$ with the soft margin constraints that $y_i \cdot f(x_i) \geq 1 - \xi_i$. The new objective becomes:

1. $min_w \frac{1}{2} \cdot ||w||^2 + C \sum_{i=0}^{N} \xi_i$
2. s.t $\xi_i \geq 0$
3. s.t $y_i \cdot f(x_i) \geq 1 - \xi_i \; \forall we \in \{1, \ldots, N\}$

The parameter C is a regularization parameter that controls the number of errors we are willing to tolerate on the training set: the higher the C, the less we tolerate. The term $\sum_{i=0}^{N} \xi_i$ can be interpreted as an upper bound to the total number of misclassified samples because when $\xi_i > 1$ the sample is misclassified. The given optimization problem is quadratic program in $N + D + 1$ variables:

1. N for the ξ_i
2. D for the parameters of w
3. 1 for C

It is possible to show that the solution has the form: $\hat{w} = \sum_i \alpha_i x_i$ where α_i are always zero, except for some values x_i called support vector. The latter are the points which are either incorrectly classified, or are classified correctly but are on or inside the margin.

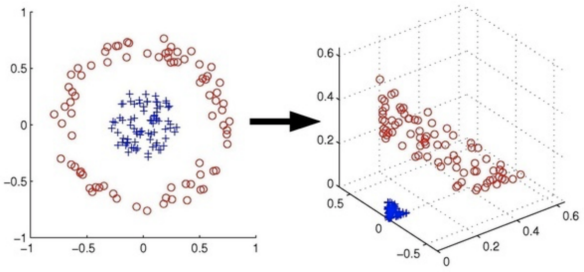

Figure 4.2. The original point in 2D are not linearly separable (left). After the kernel transformation they are reported in a 3D space and becomes linearly separable (right)

4.2.1 Kernel Trick

A kernel function is a function: k takes two arguments (points):$x, x^{'}$ and returns a positive value: $k(x, x^{'}) \geq 0$ that can be interpreted as a similarity score. An important type of kernel that we will use is the radius basis function (RBF kernel): $k(x, x^{'}) = exp^{-\gamma||x-x^{'}||^2}$. A kernel is said to be positive definitive when the relative Gram matrix is positive definite. The important thing is the RBF kernel has such property.

It can be shown, the latter requirement is sufficient to write the kernel function as: $k(x, x^{'}) = \phi(x)^T\phi(x^{'})$ where ϕ is a mapping from the original dimension of x to an arbitrary space of size: D. For example given: $x, x^{'}$ are bi-dimensional points and a mercer kernel: $k(x, x^{'}) = (1 + x_1x_1^{'} + x_2x_2^{'})^2$, we can rewrite the latter as a dot product between the same points translated to the six dimensional feature space: $\phi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2]^T$. The utility to represent points in higher dimensional space is represented by the Cover's theorem: *A complex pattern-classification problem, cast in a high-dimensional space nonlinearly, is more likely to be linearly separable than in a low-dimensional space, provided that the space is not densely populated.* (Fig. 4.2).

In detail, if we consider the solution to the SVM: $\hat{w} = \sum_i \alpha_i x_i$, we have that the prediction is computed as: $\widehat{f(x)} = \sum_i \alpha_i x_i^T x$ and we can plug in the kernel so that we obtain: $\widehat{f(x)} = \sum_i \alpha_i \phi(x, x_i)$. In this way we have executed the same algorithm of SVM but on a higher dimensional space making the points represented in a higher dimensional space where they are more linearly separable. As we said before, α_i are zero except for same x_i that are called support vector. This means, $\phi(x, x_i)$ is computed only for the latter.

In RBF the γ parameter measures how strong is the influence of a support vector on a given point. The greater is γ and less is the influence of support vector and no amount of regularization with C will be able to prevent overfitting. When γ is small, the model tends to be too simple and have a high bias.

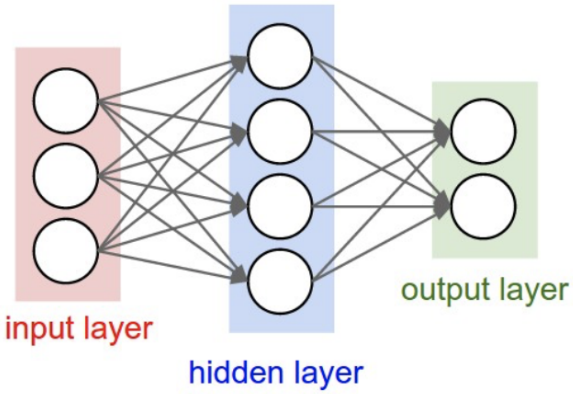

Figure 4.3. A neural network with 3 neurons in input, 4 in the fully connected layer, and 2 in the output [19]

4.3 Feedforward Neural Network

A Feedforward Neural Network (FNN) defines a mapping $f(x;\theta)$ and it adaptively learns the parameters: θ to best approximate a target function: f^* with respect a loss measure. It has a network structure because its internal operations can be represented through a direct acyclic graph. It is called neural because it is inspired by the human brain. In detail, a neural networks is constituted of stacked layers, each of them made of multiple neurons. The first one is called input layer; the last one is the output layer; in the middle there are hidden layers.

For regular neural networks, the most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections (Fig. 4.3). If we take into consideration a single neuron in the hidden layer of Fig. 4.3, and we define:

1. x: the neuron's output
2. w_i: the weights associated to its connections with the previous layer
3. σ: the activation function

we have that $x = \sigma(\sum_i w_i \cdot x_i)$.

Neural Networks with at least one hidden layer are demonstrated to be universal approximators: the can can approximate any continuous function. This is a powerful results but it does not tell how to build that hidden layer neither how to train the network and this is why in general we need to stack multiple layers.

As mentioned before the output of a neuron depends on the activation function. The latter is a non linear function that gives the model the ability to approximate functions that are not linear or to classify non linearly separable data. The main types of activation functions are:

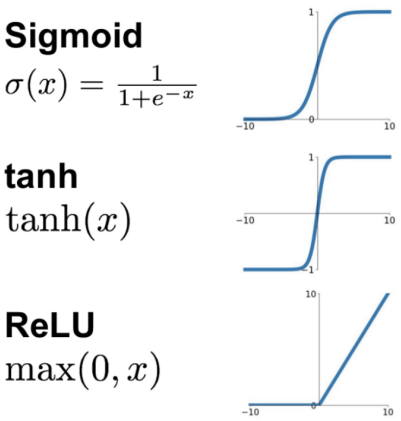

Figure 4.4. Activation function

1. $Sigmoid(x) = \frac{1}{1+e^{-x}}$. It squashes the input to the range: $[0, 1]$ and since it can be interpreted as a probability measure it is often used in the output layer. The first drawback is that the gradient of points near to 0 or 1, is close to zero causing the vanishing gradient problem. Secondly, the output is always positive and this results in gradients that are always positive or negative for the weights of the layer. For example if at some layer we have a 2-dimensional output: x_1, x_2 that is positive because it is the result of a *sigmoid* and a loss function: L, that depends on $f = w_1x_1 + w_2x_2$, we have that $\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial w_i} = \frac{\partial L}{\partial f}x_i$ that has the same sign of $\frac{\partial L}{\partial f}$. This means w_i needs to be both updated of the same positive of negative values, determining a slower convergence and the zig-zag update dynamics.

2. $Tanh(x)$. It suffers from the vanishing gradient but since it is zero centered the convergence is faster.

3. $ReLU(x) = max(0, x)$. It does not suffer from vanishing gradient since its derivative does not saturate. It is quicker to compute because it does not require exponential functions. However it suffers from the dying neurons: when a weight is updated after a great gradient and becomes a big negative number, it will outputs only zero values since that weight will difficult be updated for a large positive value.

4.3.1 Weighted Loss Function

In this section we will talk about weighted loss function. This technique can be used by all machine learning algorithms, but we have presented in this section because for our project, we have adopted it only with neural networks. Neural networks cast the task of learning as an optimization problem and solve it through the application of specific algorithms. In particular we define distinctively a *loss function* that measure the error and the *optimizer* to minimize it. The latter can be: Stochastic Gradient Descent (SGD), Adaptive Moment Estimation (Adam) and others, but in the following we will only talk about the loss function.

Formally, the objective function measure the deviation between the predicted value: $\hat{y}$ and the true label: y. If we have a dataset: $D = \{x_i, y_i\}^N$, the loss function: $\mathcal{L}$ is defined as: $\mathcal{L}(\theta) = \frac{1}{N}\sum_i^N L(y_i, f(x_i, \theta))$, where θ are the model's learnable parameters and f is the neural network. The term $\mathcal{L}$ depends on the specific task and for example it can be:

1. Mean Squared Error (MSE): $\mathcal{L} = \frac{1}{N}\sum_i^N (y_i - \hat{y}_i)^2$. It is used for regression.

2. Cross Entropy : $\frac{1}{n}\sum_i^N (y_i \cdot log(\hat{y}_i) + (1 - y_i) \cdot log(1 - \hat{y}_i))$. It is used for binary classification

The above loss formulas consider each element equally: the error due to a given sample: $L(y_i, f(x_i, \theta)$ accounts the same of all the others. When we have overbalanced dataset, we could leverage the possibility to give greater weights to low represented classes so that we implicitly assign them more importance. In this way the loss function becomes: $\mathcal{L}(\theta) = \frac{1}{N}\sum_i^N w_i \cdot L(y_i, f(x_i, \theta))$ where w_i is the weight associated to x_i. If we set a high value of w_i, the neural networks will train its parameters to profoundly capture the features of x_i. There are several ways to set that weights and the best approach we have found is basically trial and error. We have initially assigned to all the samples of class: C_j, a weight calculated as: $\frac{|x_i \in C_j|}{N}$ and then increased until an optimal value was reached.

4.4 Convolutional Neural Network

Convolutional Neural Networks (CNN) is a deep model that performs well with a variety of tasks such as image classification, natural language processing, and signal processing. CNNs are explicitly designed to deal with multi-dimensional input and overcome the high number of parameters that are requested by standard FNN. For example, a single RGB image of size 64x64, in an FNN would require: $64 \cdot 64 \cdot 3 = 12288$ neurons as input. The issues that arise when the FNN is over parametrized are the following:

1. a huge number of input neurons will require more layers at a high computation cost and time required for training

2. over-parametrization is a symptom of overfitting: in the specific case of an image, the FNN would behave too meticulous since it will take into account each single pixel

In order to take into account the multidimensional input, CNN's neurons are organized in three dimensions and to reduce the overall complexity, the neurons in each layer are connected only to a portion of the previous one. This is the opposite of what happens in fully connected neural networks. The main types of layers used in CNN are:

1. convolutional: consists of groups of neurons apply a scalar product with the connecting portions of the input.

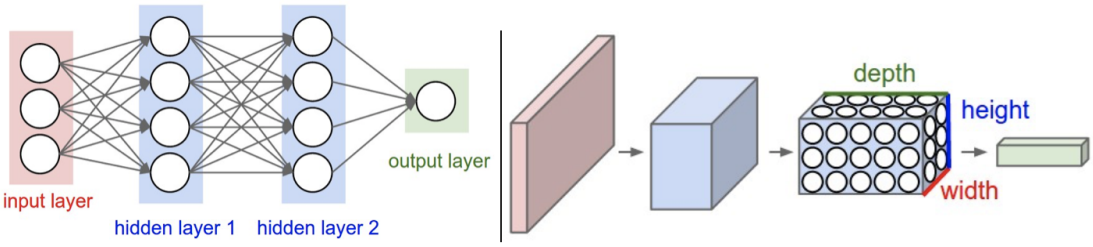

Figure 4.5. Comparison of FNN versus CNN. In CNN each layer has 3 dimension: depth, height, length.[19]

2. pooling: downsampling to reduce the dimensionally

3. fully connected: produces the final predictions. Generally, it is preceded by a flattening operation since the convolutional outputs are always 3-dimensional

4.4.1 Convolutional Layer

The convolutional layer is the building block of CNNs. It consists of a set of filters (or kernel), whose parameters are learned during the training. A a single filter is a multidimensional matrix with height, length and the same width of the input layer. For example, if we have an RGB image of size: $64 \cdot 64 \cdot 3$ and filer of size: $6x6x3$, the total number of learnable parameters of the latter are: $6 \cdot 6 \cdot 3 = 108$. It is immediate to note the crucial reduction of parameters with respect to a classical feed-forward neural network, that would have required $64 \cdot 64 \cdot 3 = 12288$ weights in the input layer.

During forward pass, each filter is shifted through the input and performs a convolution that in practice is a simple dot product. The filter can be interpreted as a particular feature and the convolution asses how much that feature is present in the portion of the input. During the training, CNN learns the weights of such filters, so that they can extract the actual features to discriminate the final prediction. In the end, each filter outputs a 2-dimensional matrix that for what we said before it is called feature map.

As we said before, each filter has a limited area called receptive field that is connected only to a small portion of the input to reduce the number of parameters. This technique as the name suggests is called parameter sharing and is based on the fact the same feature is present in multiple regions of the input: for example if we want to detect the border of an image, we could use a filter that properly activates when it is convoluted with them. In any case, the filter is applied to all the input and at the end of any convolutional layer is always present an activation function that is generally a ReLU. The parameters of a convolutional layer are:

1. spatial extent of the filter (F)

2. depth (K): the number of filters that determines the depth of the output

3. stride (S): the number of pixels to skip every time the filter is convoluted

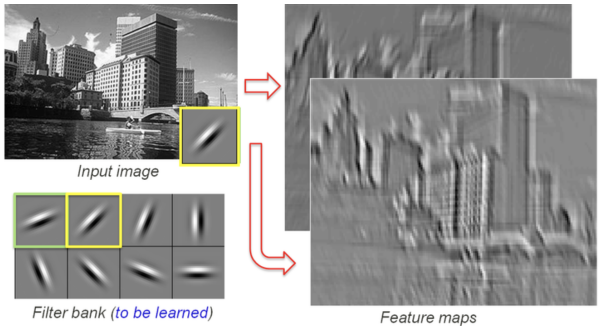

Figure 4.6. 2 different filters are applied to the input image, resulting in 2 different feature maps [34]

4. zero-padding: consists of adding some zeros to the image border to have control on the output dimension. Specifically if $P = 1$ each border has one zero and so on.

With the notation above, we can deduct the output dimension of a convolutional layer. Given an image of size: $W_1 \cdot H_1 \cdot D_1$, the output is:

1. $W_2 = 1 + (W_1 - F - 2P)/S$
2. $H_2 = 1 + (H_1 - F - 2P)/S$
3. $D_2 = K$

The number of parameters for a single filter are: $F \cdot F \cdot D_1$ and globally: $F \cdot F \cdot D_1 \cdot K$

4.4.2 Pooling Layer

The pooling layer is used to reduce the dimensionality of the given output's layer and consequently to obtain a reduced number of parameters at a less computational cost. Generally, it has a local extension of 2x2 ($F = 2$) and it outputs the maximum in the region it is applied (max pooling) with a given stride: S. It performs a downsampling only through the width and height: the original depth is left unchanged. Also in this case it is possible to define the output dimension as:

1. $W_2 = 1 + (W_1 - F)/S$
2. $H_2 = 1 + (H_1 - F)/S$
3. $D_2 = D_1$

4.4.3 Mathematical Interpretation

In this section, we will describe CNNs from a geometric point of view, so that it is possible to understand the necessity of an activation function at the end of each layer and the advantages of multiple stacked convolutional layers. In [25], the CNN is interpreted as a RECOS model: set of operation that performs Rectified Operation On a Spere. Specifically, each filter is interpreted as an anchor vector in a sphere,

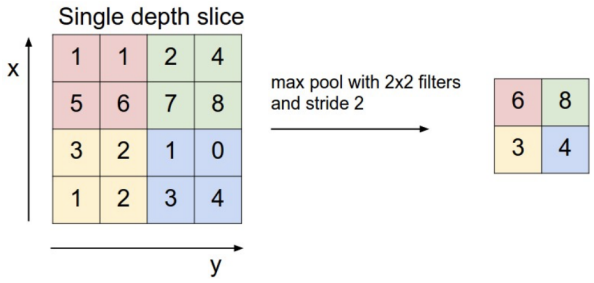

Figure 4.7. Max pool with stride 2.(Left: original input, Right: pooling output) [19]

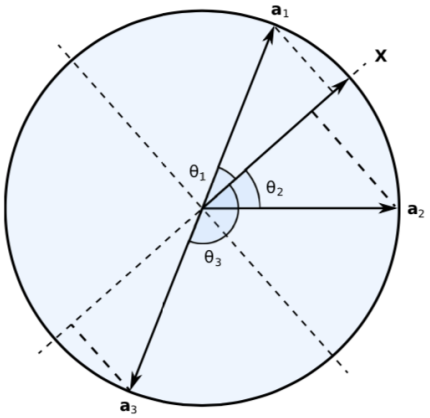

Figure 4.8. An input vector $\boldsymbol{x}$ with three anchor vectors: $\boldsymbol{a_1}, \boldsymbol{a_2}, \boldsymbol{a_3}$, in the sphere S: [25]

because they serve as reference signals. Each convolution between a local region and filter is translated as a similarity score between the anchor vector and the vector representation of that region.

In the RECOS model, a negative correlation is set to zero, and the reason will be described later. Given a image's region in a vectorial form: $x = (x_1, \ldots, x_N)^T$, and a sphere centered in the origin of unitary size: $S = \{\boldsymbol{x} \mid ||x|| = 1\}$, we analyze the geodesic distance between x and an anchor vector: $\boldsymbol{a_1}$.

In particular, the geodesic similarity: d is the output of the convolutional step between a filter and the portion of input it is applied, that is a simple dot product. Specifically, $d = \boldsymbol{a_1}^T \boldsymbol{x} = cos(\theta)$ where θ is the angle between the two vectors (Fig. 4.8). The longer the correlation, the shorter the distance. The problems with this similarity score, happen when $\theta > 90$, because $cos(\theta) < 0$. Even if we could state a negative value correctly indicates high distances, issues arise when an additional anchor vector is negatively convoluted on the previous output producing a positive outcome. In this case, low correlated anchor vectors have outputted a positive score. Similarly, the model is not able to distinguish with two anchor vectors, between positive/negative response and vice-versa. To solve this problem, we apply a Rectified Linear Unit (ReLU) so that negative correlation is set unambiguously to zero.

Figure 4.9. Amazon stock price in 6 months. To predict its value in August, the RNN could consider the fact in June has a positive and the same could happen in August

The paper ([25]), further discuss the need to multiple convolutional layers. The point is that each layer operates at different pattern level detection: from more general to more specific patterns. For example, the paper tested on MNIST dataset that by changing the background's images, the first layer's filters change while the second layer is unaltered. This is due to the first layer is more general and capture the general picture so also the background, while the following ones tend to concentrate on more specific invariant features.

Finally, we will conclude with the role of the fully connected layers. Generally, the convolutional layers perform feature selection, while the last dense layers are used to detect the surface boundary required for the final classification. In details, the latter need to cluster the input vector to the specified dimension. For example with an FC layer of size 100 and 4, the features are first clustered to 100 clusters and then to 4.

4.5 Recurrent Neural Network

Recurrent Neural Networks (RNN) have been introduced in the '80s for modeling time series because traditional approaches based on FNN were not appropriate. The latter operates under the assumption each element of the input data is independent of the others. Since the information only flows in the forward direction in every layer of the network, it does not take into account the information contained in the order of the input.

The latter is crucial for tasks such as time series regression, word sense disambiguation and in general any problem where the input is an ordered sequence. For example, if we want to predict the Amazon's stock price in August 2019, we need to take into consideration the sequence of stock prices in July, June and so on: temporal dependency (Fig. 4.9). If we need to disambiguate the sense of a word, it is necessary to evaluate the position of that word in the phrase: spatial dependency. In the specific case of arrhythmia detection, it is required to capture the spatial dependency among samples, to detect features like the QRS complex, ST segment.

The same dependencies the RNN aims to capture, characterize also the human learning process. As a human being, we learn the sequence not the symbols alone:

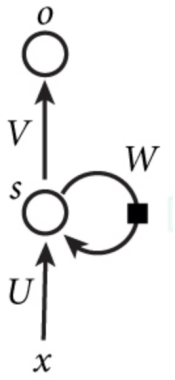

Figure 4.10. RNN general structure [38]

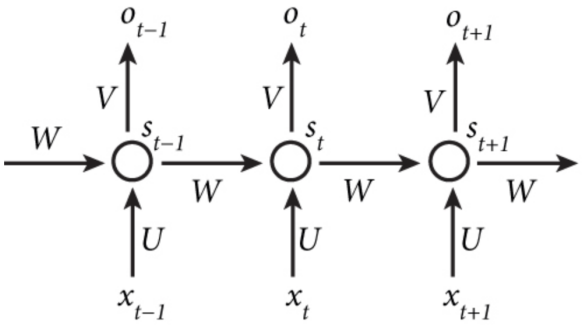

Figure 4.11. RNN unrolled [38]

for us, it is immediate to say the alphabet from A to Z, but in the opposite direction is not straight forward. The same is true for our telephone's number. To model dependencies, RNN has an internal memory that is continuously updated through the input sequence. In the following, we will first describe Vanilla RNN, the vanishing gradient problem and how LSTM solves the latter.

4.5.1 Vanilla RNN

The intuition to make explicit the sequence dependency in an RNN is to feed each layer with the previous one through a retroactive loop (Fig. 4.10). In this way, a sample $x_t \in S = [x_1, \ldots, x_T]$, is fed into the RNN to produce the output o and a hidden state: s_t that will be taken into consideration by the network for the next sample: x_{t+1}. This structure can be unrolled in multiple blocks so that the loop is deleted, as shown in Fig. 4.11: in this way it is clear how the output o_t depends on the current input: x_t and previous state: s_{t-1} Formally, we have:

1. x_t is the t_{th} input of a sequence

2. s_t is the hidden state of the network, continuously updated in this way: $s_t = \phi\left(Ux_t + Ws_{t-1}\right)$ where ϕ is a generic activation function(ReLU, softmax,

tanh). Note the dependency with the previous state s_{t-1}: this is where the sequence dependency is made explicit.

3. $\hat{y}_t = o_t$ is the final output computed as: $\hat{y}_t = o_t = \phi(Vs_t)$

It is important to note the matrixes: U, V, W are the same for every unrolled block. It is different from classical FNN, where each layer has its own parameters. This results in less parameters for the network but also, some issues with the gradient computation.

Vanishing Gradient Problem

In order to asses the quality of a Vanilla RNN, we define a loss function: L, so that for a given sequence of size: T, the total error is: $E = \sum_{t=0}^{T} L(\hat{y}_t)$ that is the sum of the errors over all the predictions in the sequence.

The gradient of the error with respect to any of the matrix parameters such as W is: $\frac{\partial E}{\partial W} = \sum_{t=0}^{T} \frac{\partial L(\hat{y}_t)}{\partial W}$.

If we concentrate on $\frac{\partial L(\hat{y}_t)}{\partial W}$, we have that $\hat{y}_t = f(s_t)$ and $s_t = g(W, s_{t-1})$, we conclude that $\frac{\partial L(\hat{y}_t)}{\partial W} = \sum_{k=0}^{t} \frac{\partial L(\hat{y}_t)}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial s_t} \frac{\partial s_t}{\partial s_k} \frac{\partial s_k}{\partial W}$.

In the following, we will assume $s_j \in D^1$ and $W \in D^1$ namely the RNN is 1 dimensional, so that it is easier to proceed. However, the same consideration can be taken in multi dimensional case.

Given that, $s_j = \phi(Ws_{j-1} + Ux_t)$ we have: $\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial (Ws_{j-1}+Ux_t)}{\partial s_{j-1}} \frac{\partial (\phi(Ws_{j-1}+Ux_t)}{\partial (Ws_{j-1}+Ux_t)} = W\phi^{'}(Ws_{j-1} + Ux_t)$.

The term $\frac{\partial s_t}{\partial s_k}$, can be written using the chain rule as: $\frac{\partial s_t}{\partial s_k} = \prod_{j=k+1}^{t} \frac{\partial s_j}{\partial s_{j-1}} = W^{t-k} \prod_{j=k+1}^{t} \phi^{'}(Ws_{j-1} + Ux_t)$, and this is where the vanishing problem is ([30]). With respect to the fact $|W| > 1$ or $|W| < 1$, W^{t-k} will explode or vanish exponentially. Equally, $\left|\phi^{'}(Ws_{j-1} + Ux_t)\right| \leq 1$ as it is the case of ReLU, tanh, softmax.

These last two considerations, allows us to conclude the more two states are distant the higher will be $t - k$ and $\frac{\partial s_t}{\partial s_k}$ will become extremely great (exploding gradient) or small (vanishing gradient). In the first case the weights will become so large as to overflow and result in NaN values, the model will be unstable and not able to generalize anything from the data. In the second case the contribute of a previous state to the current will be irrelevant: the model can not learn long term dependency.

4.5.2 Long Short Term Memory

LSTMs are a type of RNN, introduce in 1997 ([23]), to solve the vanishing gradient problem. They are constituted of a cell state: C_t that is updated through structures called gates. The latter consists of a sigmoid that decides the amount of information to pass to the cell: zero values means nothing and one everything. Overall, an LSTM

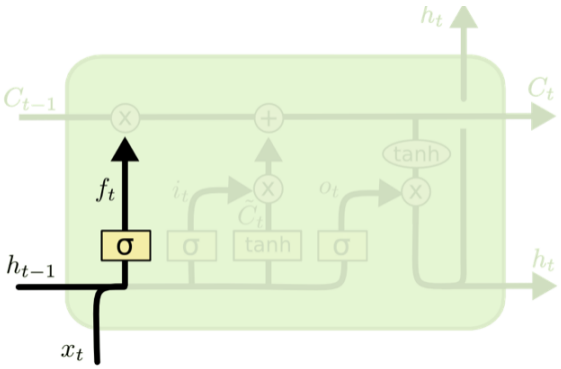

Figure 4.12. Forget layer in LSTM [14]

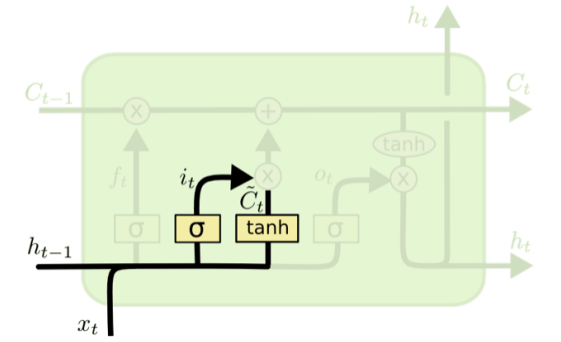

Figure 4.13. Input layer in LSTM [14]

contains three gates in this order: forget, input and output gate. In addition to this, each unrolled LSTM cell takes as input the actual sample: x_t,the previous cell state: c_{t-1} and hidden state: h_{t-1}.

Initially, the forget gates decides what portion of x_t and h_{t-1} we need to discard for updating the cell state. Specifically, it computes $f_t = \sigma\left(W_f[h_{t-1}; x_t] + b_f\right)$ so that f_t is a vector of values in the range: [0,1], that indicates the amount of information, related to the previous hidden state and current input, not to consider (Fig. 4.12). In a symmetric way, the input gate defines the values of the same concatenated vector to keep: $i_t = \sigma\left(W_i[h_{t-1}; x_t] + b_i\right)$.

Secondly, always at this level, a candidate cell state is built: $\widetilde{C_t} = tanh\left(W_c[h_{t-1}; x_t] + b_c\right)$ (Fig. 4.13). It is clear now how to update the cell state: $C_t = f_t \cdot C_{t-1} + i_t \cdot \widetilde{C_t}$ (Fig. 4.12). Finally, the output gate will produce the output: $o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$ and the hidden state: $h_t = o_t \cdot tanh(C_t)$ (Fig. 4.15). The discussed structure is a valid solution to overcome the vanishing gradient problem. In fact, if we compute the recursive gradient in this case we have: $\frac{\partial C_t}{\partial C_{t-1}} = f_t + K$, where K is the result of the other partial derivatives, that for our concern are not of interest.

The crucial point here, is that even in case of long term dependency where we multiply the derivative previously computed several times, the term: f_t can always

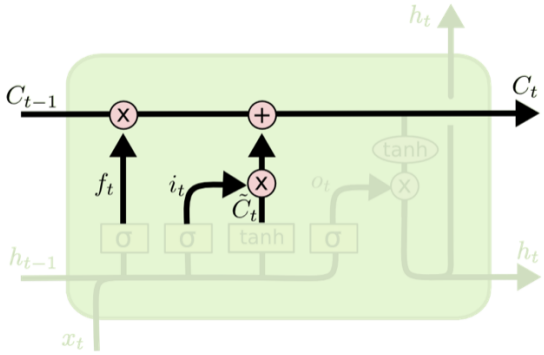

Figure 4.14. Cell update in LSTM [14]

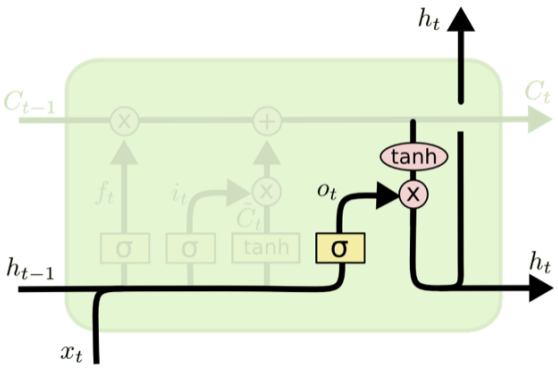

Figure 4.15. Hidden state and output in LSTM [14]

assume any values in the range [0,1] and it does not exponentially converge to zero.

In addition to this the presence of gates with internal parameters, instead of simple activation functions, allows the model to adaptively set the corresponding parameters with respect to the current input and hidden state. In this way the gates can actively update their state when the gradient is too low or too high.

Finally, it is worth to note there are several variation on the described architectures. In 2002, it has been proposed a LSTM model ([20]) where all the three gates:G_1, G_2, G_3 are conditioned by the cell state as well as hidden state: $G_i = \sigma(W_i[C_{t-1}; h_{t-1}; x_t] + b_i)$. Another variation that recently gained popularity is the Gated Recurrent Unit (GRU) ([13]). In the latter, the input gate is also used as forget gate: $f_t = 1 - i_t$. In addition to this, the cell and hidden states are merged together. Overall, we have:

1. $i_t = \sigma(W_f[h_{t-1}; x_t])$: what to remember about the previous hidden state and input

2. $r_t = \sigma(W_r[h_{t-1}; x_t])$: what to keep from the previous hidden state

3. $\widetilde{C_t} = tanh(W[r_t \cdot h_{t-1}; x_t])$: candidate hidden state (different from the candidate cell state in regular LSTM)

4. $h_t = (1 - i_t) \cdot h_{t-1} + i_t \cdot \widetilde{C_t}$: the hidden state is built by forgetting about the previous hidden state and adding what to remember about the candidate hidden state

4.6 Attention

Attention in neural networks aims to replicate the same human cognitive process. The latter is defined in [31], as the behavioral process of concentrating on a portion of the total information while ignoring the rest: it is the allocation of limited cognitive resources. In the context of neural networks and predictions, attention allows detecting significant features during the training. Namely, it learns the filter to distinguish between irrelevant, noisy inputs and a remarkable one. Generally, attention is applied to Encoder-Decoder neural networks, to make explicitly the decoder knows where to look for the final prediction.

4.6.1 Encoder-Decoder model

Encoder-Decoder model is a general architecture constituted of two sequential blocks:

1. encoder that converts the input into a context vector. Basically, the encoder performs feature extraction.

2. decoder that takes the context vector and the previous output, to produce a final prediction. This is why it is autoregressive.

Specifically, the layers in the encoder and decoder can be of any nature: for example, image auto-encoders have Convolutional Neural Network (CNN) in the encoder and the analogous deconvolution in the decoder. Sequence to sequence tasks such as translation can have an encoder and decoder RNN or LSTM. It is essential to note the encoder turns the raw input into a suitable representation for the decoder.

However, as the input's size increases the more difficult it is for the encoder to correctly summarize it into a fixed-length vector. In addition to this, as it is the case of sequence to sequence task such as translation, the output should be conditioned not to the whole input sequence but only to a smaller useful portion.

With attention the encoder does not need to produce a fixed-length vector, but it encodes the input into a sequence of vectors and adaptively chooses which of them are essential. Formally, the encoder-decoder model can be formulated as ([6]):

1. input sequence: $\mathbf{x} = (x_1, \ldots, x_{T_x})$, where T is the length of the x_{th} input sequence
2. hidden states at time t: $h_t = f(x_t, h_{t-1})$ where f can be for example an LSTM or CNN
3. context vector: $c = q(h_1, \ldots, h_T)$ where for LSTM $q = h_T$. In fact, with a recurrent encoder-decoder the context vector for the decoder input is the last encoder's hidden state
4. decoder's prediction at time: t' is: $y_{t'}$ and it is conditioned to the previous output: $y_{t'-1}$ and context vector: c.

It is important to note that t' is the output sequence length that is not necessarily equal to the input length: T. Practically, the decoder defines the prediction as: $p\left(y_{t'}|(y_{t'-1}, c, s_{t'}\right) = g\left(y_{t'-1}, c, s_{t'}\right)$ where g is the function implements the decoder block and s_t is its internal state.

Finally, the encoder part, before producing the context vector for the decoder can traverse the sequence in both ways (left/right and right/left) so that now each annotation is the concatenation of both hidden states : $h_j = \left[\overrightarrow{h_j}, \overleftarrow{h_j}\right]$: Bidirectional RNN. In this way, the context vectors model the relationship of a given word with the previous and following positions.

4.6.2 Attention in Encoder-Decoder

The previously discussed formulation can be extended to attention, by conditioning the output at time t': $y_{t'}$ not to a global context vector: c, but to a context vector that is specific for the given instant: $c_{t'}$. In particular we have that: $p(y_{t'}|(y_{t'-1}, c_{t'}, s_{t'}) = g(y_{t'-1}, c_{t'}, s_{t'})$.

Generally, c_i is a weighted sum over all the hidden states of the encoder for a given sequence: $c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$, where $\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$. Finally, the crucial step

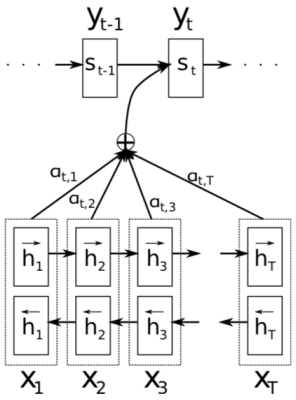

Figure 4.16. Graphic representation of attention in RNN ecoder-decoder: [31]

is how the alignment scores are computed: different attention algorithms mainly differs for this computation: $e_{ij} = a(s_{i-1}, h_j)$. The latter measures how the previous hidden state of the decoder: s_{i-1} is relevant to the encoder's hidden state: h_j.

Basically, the context vector is a weighted sum over all the encoder's hidden states: c_i and α_{ij} is a probability measure of how much h_j is relevant for the previous decoder's internal state: s_{i-1} for the following state and final prediction: s_i and y_i (4.16).

At the beginning of each sequence, the decoder state is initialized with the last hidden state of the encoder. In this way, the decoder select which part of the input hidden states to pay attention to. The encoder-decoder models performs this type of filtering also without attention but with a latent mechanism: on the opposite attention explicitly forces the decoder to focalize on relevant features. The attention vector that is the final output is obtained applying a dense layer to the concatenation of c_t and h_t: $a_i = tanh(W[c_i; h_i])$

4.6.3 Multiplicative and Addititive Attention

There are two main mechanism to computer the attention scores:

1. Luong's multiplicative ([26]): $e_{ij} = h_i^T W h_j$
2. Bahdanau's additive style ([31]): $e_{ij} = v_a^T \cdot tanh(W_a s_{i-1} + U_a h_j)$

In the above definitions, W, U, v_a are matrixes to be learned during the training: practically they are fully connected layers. The two methods have the same computational complexity but the first one is more efficient since it only requires matrix multiplications for which the GPU hardware is highly specialized.

4.7 Transformer

Transformer ([36]) is a neural network architecture introduced in 2017 by Google research, for Natural Language Processing (NLP). A series of experiments in NLP

task have shown Transformer is better than classical approaches based on Recurrent Neural Networks (RNN), in terms of accuracy and computational efficiency. We have adopted this structure for arrhythmia detection since like NLP tasks, the input data is a collection of sequences (heartbeats) and the information about the output is concentrated in the relationships between the single sequence's components (millivolts sample).

The main issue for NLP tasks such as translation is word sense disambiguation, namely the semantics of a term given its context in a phrase. RNN approaches this problem considering for each word, the previous hidden state. However, this sequential characteristic precludes parallelization within training samples because before predicting the actual hidden state we need to know the previous one. In addition to this, RNN has difficulties to learn dependencies between distant words.

The Transformer solves these problems using nonrecurrent layers and self-attention. It has an encoder-decoder structure. Specifically, the encoder is made of 6 identical layers. Each of them consists of two consecutive residual blocks where the first one is the self-attention layer, while the second one is a fully connected layer: they are both followed by a normalization layer. The decoder is equal to the encoder, except for the presence of an additional self-attention layer (4.18).

4.7.1 Transformer's Attention

The essential transformer's layer is multi-head attention. we will first describe the low dimensional case and secondly the multi-head approach. As we said before, the input is a sequence where each component has in total: d_k features because as it is the case of NLP, the input is followed by an Embedding Layer. Initially, it applies to each component three different dense layers to obtain respectively: Query (Q), Key (K), Value (V) vectors. Secondly, for each position in the sequence, it compares the associated component against each other to obtain a score of how much a given input is expressed in that position: $A = softmax\left(\frac{QK^T}{\sqrt{(d_k)}}\right)$. Finally, all the value vectors are weighted summed considering the latter score, to obtain the output vector in that position: $Attention(Q, K, V) = AV$.

This type of attention is dot-product attention and it is faster than additive attention because the described multiplications can be computed with the optimized vectorial operation. The role of d_k is required to avoid the vanishing gradient, caused by high values in the $softmax$ argument: without d_k, the longer input sequences the higher is: QK^T, and the lower will be the gradient of the $softmax$. The drawback of this approach is related to the fact that when computing the scores of a component with all the others, the higher score will be presumably related to the actual input itself.

To overcame this issue, the paper introduces the multi-head attention, where basically the attention vectors are computed h times (or head) with a different triple of matrixes W_i^Q, W_i^K, W_i^V $\forall 0 \leq i \leq h$. In this way, the model learns the importance of a given component from different representation subspaces at different positions,

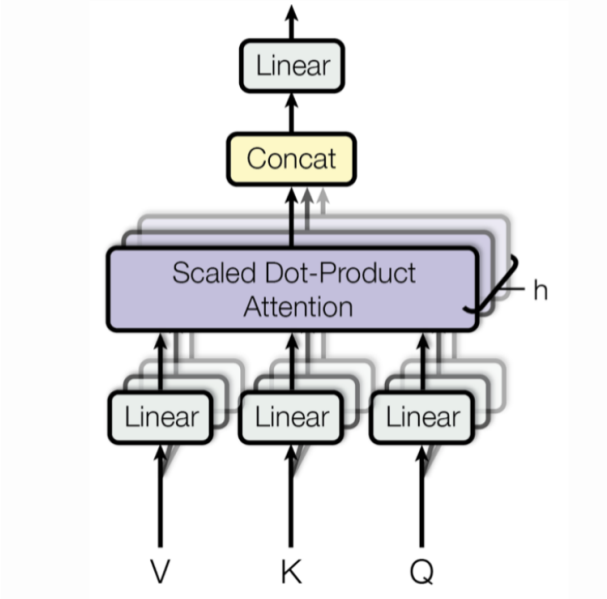

Figure 4.17. Multi Head Attention: [36]

because each head can learn different relationships. Specifically, it performs the following operation: $MultiHead(Q, K, V) = Concat\left(head_1, \ldots, head_h\right) W^O$ where $head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)$ (Fig. 4.17). The above operations are independent of the sequence length so that the computational cost is constant while RNN needs to iterate over all the sequence's components. It can be parallelized since each single attention vector can be computed independently. Finally, the scores between sequence's components does not depend on the distance while RNN fails to model relationships between distant input.

4.8 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations (NODE) ([12]) have been awarded as best paper in NeurIPS 2018 . In the following, we will describe first the Residual Neural Networks (ResNet), since NODE are the direct successor.

4.8.1 Residual Neural Network

ResNets ([22]) have been introduced after verifying the consequences on neural network performance when more layers are added. The interest in this argument is justified by the presence of very deep neural networks that perform well on challenging datasets such as the ImageNet dataset ([33]). Basically, the argument is if increasing the depth of a neural network, always the optimal choice to improve its performance. Vanishing gradient is an issue related to deep models, but it can be solved using normalization techniques. The paper's response to the above problem is negative.

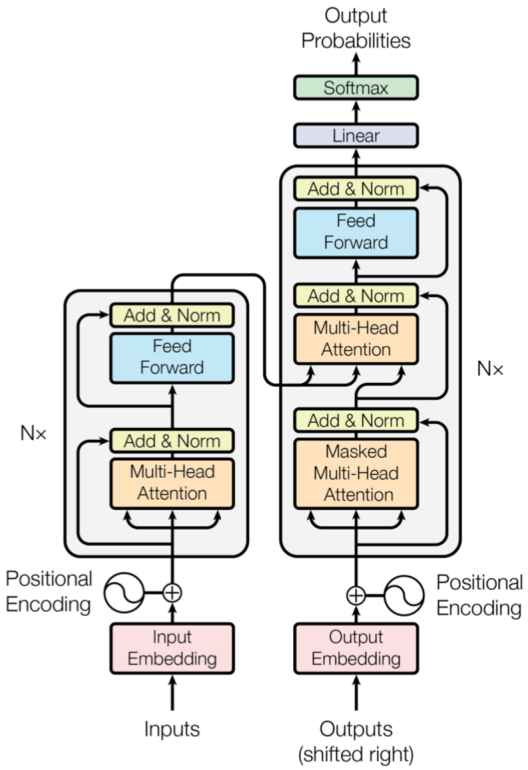

Figure 4.18. Transformer model: [36]

The authors made a series of experiments and verified deep neural networks experience the degradation problem: with the depth increasing, accuracy on the test set does not improve and the culprit is not overfitting. The degradation problem shows not all systems are similarly easy to optimize ([22]): deeper model can lead to higher error both on training and test set. An example of the above issue is visible if we add identity layers to a neural network. We should except no loss in accuracy than the shallower counterpart, but in practice the opposite is true. ResNets are introduced to solve the degradation problem.

A single residual block turns the input: x into the output: Y: through the formula: $Y = x + F(x)$: F is a generic function implemented as multiple stacked neural network layers (4.19). In this context, a ResNet with N residual blocks can be modeled as : $Y_{i+1} = Y_i + F(Y_i, \theta_i),\ \forall i \in [1, .., N]$, where θ_i are the weights of the i_{th} layer to be learned. ResNet are different from Feed Forward Network (FNN) because: FNN maps the input: x, to the output Y, with multiple stacked hidden layers: $h_1, \ldots, h_n$, so that the objective of the learning is a direct mapping from input to the output : $Y = h_n(\ldots(h_1(x))$. On the opposite a single residual layer maps the input to the output specifying a function F that instead of learning a direct maps between input and output, models the difference between them because the hypothesis is that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping ([22]).

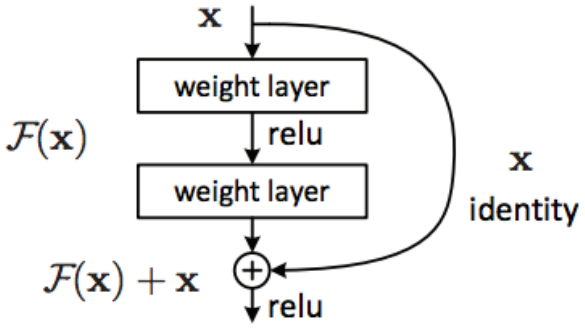

Figure 4.19. Graphic representation of a residual block

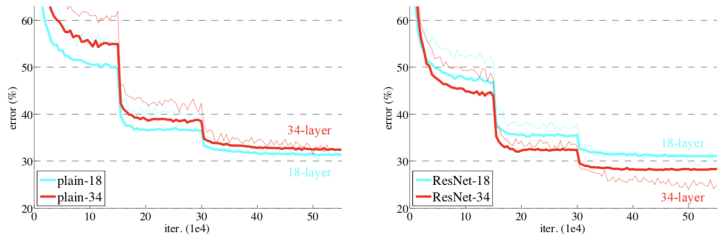

Figure 4.20. Comparison of plain and residual neural networks ([22])

The reason why ResNet solves this problem has been verified in [22]. and the explanation is that for the model it would be easier to set to zero the residual function, rather than learning the identity mapping through nonlinear activation functions. In practice, the identity mapping is not the optimal one. However, it should be more feasible for the solver to find the perturbations with reference to an identity mapping, than to learn the function like a new one ([22]). This last consideration is a crucial similarity between ResNet and NODE: David Duvenaud (author of [12]), explains in a forum that the idea behind NODE is the intuition it is easier to model a small change to an almost-correct answer than the whole improved answer at once.

In Fig. 4.20, thin and bold curves denote respectively the training and validation error on ImageNet dataset. It is visible how on a plain neural network (on the left), more layers cause higher training/validation error, while deeper ResNet (on the right) have better performances.

4.8.2 NODE

The connection between ResNet and NODE is the interpretation of ResNet as a differential equation. In particular, from the definition of ResNet previously discussed, we can relax the constraint that i is a discrete value, by gradually reducing the step size: $[i, i+1]$. For example we could take half of the original step and rewrite the relationship as: $Y_{i+0.5} = Y_i + F(Y_i, \theta_i$, so that: $Y_{i+1} = Y_{i+0.5} + F(Y_{i+0.5}, \theta_{i+0.5}) = Y_i + F(Y_i, \theta_i + F(Y_{i+0.5}, \theta_{i+0.5})$. The latter equation shows reducing the step-size to the half is equivalent to double the number of residual layers in the discrete case. In

the limit, if we take smaller steps and add more layers, it is possible to describe the dynamic of the ResNet as an initial value problem where:

1. $F(Y(t), \theta_t, t) = \frac{\partial Y(t)}{\partial t}$ is an ordinary differential equation
2. $Y(0) = Y_0$ is the input data, namely the initial condition
3. T is the evaluation time

In particular, the NODE output at time T is obtained using an ODE-solver, where it is possible to specify the maximum approximation in terms of a scalar: e. To summarize, in a NODE block we specify the respective output as the solution of an ODE problem. The main difficulty to train this structure is that performing back-propagation through the ODE solver requires high memory cost and introduces additional numerical error ([12]). To solve this issue the authors propose the Adjoint method that scales linearly with problem size, has low memory cost, and explicitly controls numerical error, whose implementation is written in PyTorch and available on GitHub.

It is important to note that the depth of a NODE is not a clear concept because t is now real value, not an integer as it is the case of ResNet.However, the number of function evaluations to solve the differential equation during the forward pass of the NODE can be considered as an estimation of the complexity of the model. The benefits of NODE are the following :

1. Memory efficiency because to train the model it is not required to store the intermediate forward output at every layer, that normally it is used by the back-propagation algorithm to compute the gradients of the weights of each layer.
2. Adaptive computation because the time complexity of the ODE solver depends on the error accuracy specified and the ODE solver can adapt their strategy on the fly on the basis of that accuracy. In addition to this after the model has been trained with a given tolerance, it can be tested and used in production with a lower error tolerance so that it is faster and less intensive (useful for embedded device)
3. Parameter efficiency because parameters of nearby layers are automatically tied together ([12])

We will conclude this section describing the connection between NODE and ResNet in terms of ODE approximation. The characteristic equation of a residual block is: $Y_{i+1} = Y_i + F(Y_i)$ and it can be interpreted as the Euler approximation of $\frac{\partial Y}{\partial i} = F(i)$ at time i with unitary step size. NODE instead approximate the same function using more accurate ODE solver than the Euler method (Fig. 4.22).

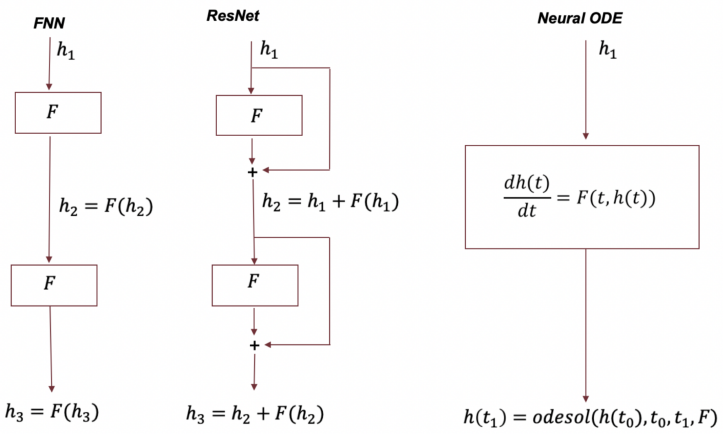

Figure 4.21. Comparison of FNN, ResNet, NODE

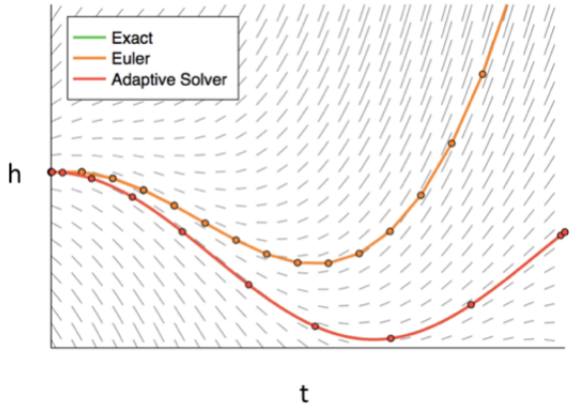

Figure 4.22. Comparison of Euler method (orange), NODE solver (red) to approximate $\frac{\partial Y}{\partial i} = F(i)$. The latter coincides with the NODE solution

Chapter 5

Experiment

5.1 Dataset

The database considered in this study is a collection of annotated ECG recordings obtained by the Arrhythmia Laboratory of Boston's Beth Israel Hospital, known as the *MIT-BIH Arrhythmia Database* ([28]). It contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, obtained from 47 subjects studied by the BIH Arrhythmia Laboratory between 1975 and 1979. The recordings are obtained with a frequency of 360 Hz: this means each recording approximately consists of $30 \cdot 360 = 10800$ samples. Two or more cardiologists independently annotated each record for a total of 110,000 annotations included with the database: it is is worth to mention, each annotation is placed in correspondence to the R peak of a single beat so that we have implicitly solved for this specific case the beat detection problem.

According to the Association for the Advancement of Medical Instrumentation (AAMI) standard ([1]) for testing and reporting performance results of cardiac rhythm and segment measurement algorithms, the evaluation of an arrhythmia detector algorithm should be made on the classification performance on five major categories of heartbeats:

1. N: includes beats originating in the sinus node (normal and bundle branch block beat types)
2. S: includes supraventricular ectopic beats (SVEBs);
3. V: includes ventricular ectopic beats (VEBs);
4. F: includes beats that result from fusing normal and VEBs;
5. Q: includes any heartbeat not in the N, S, V or F categories, essentially the undefined heartbeats as well as beats resulting from the use of a pacemaker.

The last category includes beats that are a result of a pacemaker operating under the skin of the patient. These beats are regulated and it is not meaningful to use them for arrhythmia classification: this is why we excluded all these beats from our study. Please note also that this is a common practice followed by other studies in the state-of-the-art([32]). Therefore, we are left with four-beat categories. A

Categories	N	S	V	F
Definitions	Normal beat	Atrial premature beat	Premature ventricular contraction	Fusion of ventricular and normal beat
	Left bundle branch block beat	Aberrated atrial premature beat		
	Right bundle branch block beat	Nodal (junctional) premature beat	Ventricular escape beat	
	Atrial escape beats	Supraventricular premature beat		
	Nodal (junctional) escape beat			
Annotations	90,585	2,781	7,235	802

Figure 5.1. AAMI 5 categories of heartbeats ([1])

more detailed topology of the dataset is depicted in Fig. 5.1. It is clear how the N class is more populated than the others: it constitutes the 89% of the whole dataset. This overbalancing can lead to inconsistent training and poor quality algorithms. In order to mitigate the latter, we have adopted data augmentation techniques such as ADASYN and SMOTE and a weighted loss function. The latter is intended to give higher weight and consequently penalize more the misclassification error due to underrepresented samples.

5.1.1 Training and Test Set

We have considered two dataset configuration. In the first one, we have built the test set, by randomly choosing from the dataset the following samples:

1. N: 900
2. S: 900
3. V: 900
4. F: 250

In a second configuration, we wanted to test the robustness of our algorithms: we decided to create personalized classifiers for the patients: 222,223,233. In detail, we first trained a model on the whole dataset without the latter three patients. Secondly, for every single patient, we have built the test set in the following way. Patient 222:

1. N: 50 from 2273
2. S: 20 from 209
3. V: 0 (this patient does not have beats labeled as V)
4. F: 0 (this patient does not have beats labeled as F)

Patient 223:

1. N: 50 from 2045
2. S: 20 from 73
3. V: 100 from 473
4. F: 5 from 14

Patient 233:

1. N: 500 from 2229
2. S: 2 from 7
3. V: 250 from 830
4. F: 2 from 11

5.2 Performance Measures

In a classification problem with N classes: $C_1, \ldots, C_n$, it is possible to determine the quality of the classifier, in terms of $C_{i,j}$, that is the number of dataset's samples belonging to C_i , but predicted as C_j, where $1 \leq i, j \leq n$. In particular given a generic class: C_i, we define:

1. True Positive (TP): $C_{i,i}$
2. False Positive (FP): $\sum_{j \neq we} C_{j,i}$
3. False Negative (FN): $\sum_{j \neq we} C_{i,j}$

The performance metrics we have used are:

1. Precision: $\frac{TP}{TP+FP}$
2. Recall: $\frac{TP}{TP+FN}$
3. F-score: $2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$

It is significant to note the trade-off between precision and recall: if the model predicts all the samples as belonging to a given class it will reach a recall of 100% because no sample will be classified with a different label. Consequently, the false negative will be zero. On the other hand in this situation, the number of wrongly classified samples (FP) will be high and consequently the precision low. In order to plot precision and recall for all the classes, it is possible to use the confusion matrix: M: it is $n \cdot n$ and $M_{i,j} = C_{i,j}$

5.3 Random Forest [17]

The paper ([17]) performs a different classification task since it does not predict the 5 classes specified by the (AAMI) standard, but the following different labels: N, L, R, V, P, described in the MIT-BIH annotations (Fig. 2.7). Even if this is a different task, we have decided to replicate it, because the approach is straightforward and performs well. In this way, we were able to concentrate on dataset building and fix some bug in the beat extraction. In addition to this, we could immediately re-adapt it to the original problem and infer the main issues of our dataset, and find more appropriate solutions. The algorithm consists of three steps:

1. pre-processing: remove noise from the ECG signal

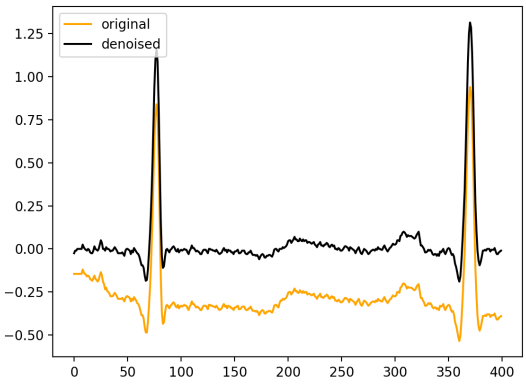

Figure 5.2. A fragment of ECG before and after the median filter

2. processing: heartbeat detection and feature extraction with DWT

3. classification: this is the first approach in literature to use Random Forest for arrhythmia classification

The ECG signal is first de-noised with a median filter, to remove baseline wandering. Recall that the latter is noise due to patient movement during the ECG recording. In Fig. 5.2, there is the application of the median filter: it is visible how the filtered signal is translated along a common axis. Each beat is defined as a sequence of 256 samples in the RR interval. To extract relevant features, they applied DWT and precisely Daubechies db2 wavelet because it gives better accuracy than other wavelets such as Symmlet sym6 and Symmlet sym10. Specifically, for each beat, the DWT is applied considering the first 4 levels of details coefficients (129+66+34+18 coefficients) and the fourth level of approximation coefficients (18 coefficients), for a total of 265 features per beat.

The training set is made of 240 samples for each class and the test set of 120 for each. The algorithm as said before is Random Forest. In Fig. 5.3, the confusion matrix plots the performance of this approach: it is almost perfect and this justified our attempt to use this approach for the original task: discussed in the following section

5.3.1 Custom Variation

Initially, we have applied the same technique to the original problem with N,S,V,F classes. In Fig. 5.4, it is possible to see the accuracy metrics. It is clear how this modification in the dataset, determined a consistent drop in accuracy with respect to the previous problem. Primarily, this is due to the fact, our dataset is overbalanced and this is why we applied SMOTE and ADASYN (Fig. 5.5) to make it more uniform. The benefits of SMOTE and ADASYN for the model are completely the same and in general the overall performance are still low.

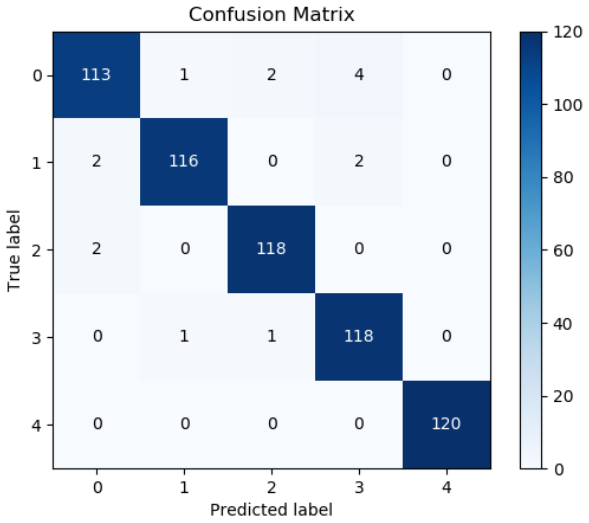

Figure 5.3. Confusion Matrix : random forest for the N,L,R,V,P classification

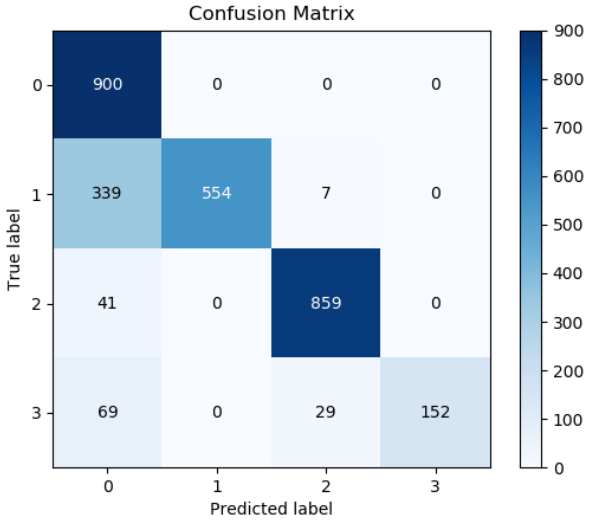

Figure 5.4. Confusion Matrix : random forest without data augmentation for the N,S,V,F classification

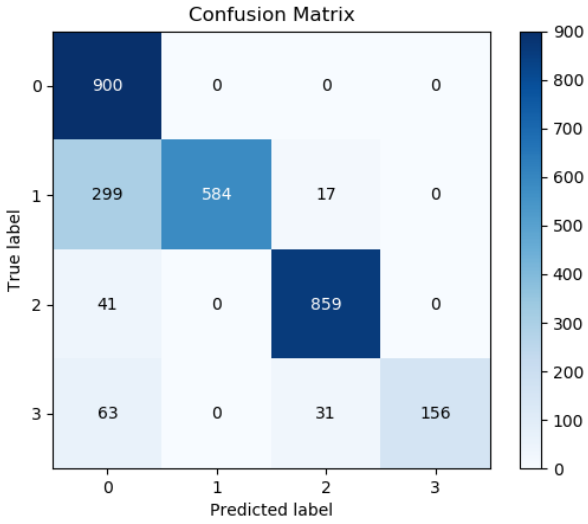

Figure 5.5. Confusion Matrix : random forest after data augmentation for the N,S,V,F classification

This is caused by the fact the N, S,V, F classes are macro-categories that consist of multiple single beat types: in the original paper the classes were single type so that it was easier for Random Forest to detect the relevant features. In order to solve the latter, we have opted for more complex algorithms, described in the following sections.

5.4 Support Vector Machine [15]

This paper ([15]) describes a valid solution to our classification problem with the four AAMI classes. As it is the case of the previous algorithm ([17]), from a high-level perspective it is divided into pre-processing, processing and classification. In this case, the paper performs an initial phase of QRS-complex detection using the Pan-Tompkins algorithm. However, we have not implemented this step, because as we said in the introduction, we will directly consider the R peak annotation already present in the dataset.

During the pre-processing phase, the de-noise is realized using the DWT: Daubechies db6 with nine levels: the signal is reconstructed considering only the details coefficients from level three to nine. In this way, the frequency ranges from 0 to 0.351 Hz and 45-180 Hz are not considered because the first one is linked to baseline wander and the latter does not contain useful information for arrhythmia detection.

Each the beat is a sequence of 200 samples centered in the R peak location. In Fig. 5.6, it is visible how the DWT makes the original ECG signal more smooth and less affected by noise. During the processing step, DWT, PCA, HOS and ICA are used.

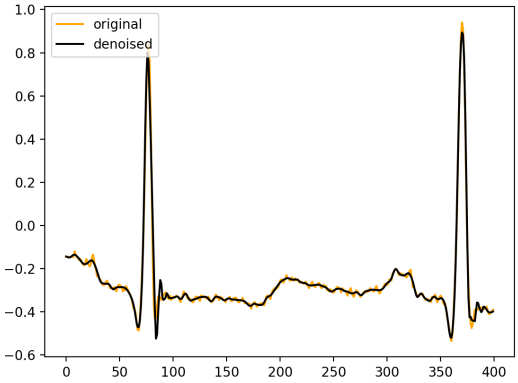

Figure 5.6. A fragment of ECG before and after the DWT

Initially, the DWT: Mayer with 4 levels is applied on every single beat to obtain the approximation coefficients in the range: 0 to 11.25 Hz. From that values, it considers only the third and fourth sub-bands for all the beats and consequently it creates respectively two matrices. Secondly, we apply PCA on each of the two matrices with 6 components so that now we have 12 features per beat.

Next, from each original beat of 200 samples, it applies 4_{th} and 5_{th} moment to every single beat to obtain two more features per beat. Finally, ICA with 14 components is applied to the matrix of beats, for a total of 28 features per beat. For the classification, it is used Support Vector Machine with the following parameters:

1. C: 70
2. Gamma: 0.7
3. Kernel: radial basis function

The results, we have obtained are described in Fig. 5.7: we have comparable performances with respect to the previous algorithm based on Random Forest. We have tested the same approach applying data augmentation with ADASYN, obtaining slightly better results (Fig. 5.8).

5.5 Convolutional Neural Network [24]

This paper ([24]) describes an effective approach to solve two problems: arrhythmia classification and myocardial infarction. In particular, the idea is to use a neural network to solve the first task and then apply transfer learning for the latter. Since we are only interested in arrhythmia detection, we will discuss only the first part. The main difference of this solution with respect to the previous two is that it does require a simple pre-processing phase without any de-noise procedure. Specifically, the latter consists of a first resample of the signal from 360Hz to 125Hz. Then, in order to find the R-peaks, it considers a temporal window of 10 seconds, normalizes

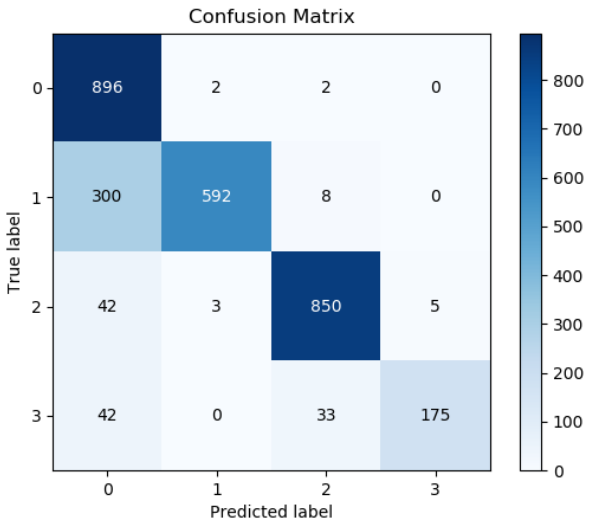

Figure 5.7. Confusion Matrix : svm for the N,S,V,F classification

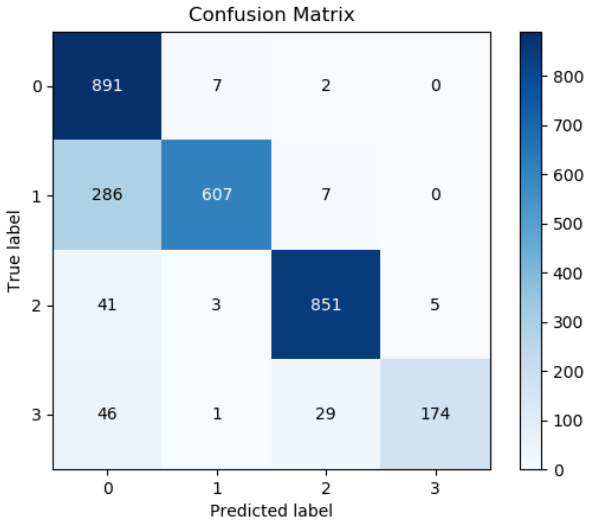

Figure 5.8. Confusion Matrix : SVM with Adasyn for the N,S,V,F classification

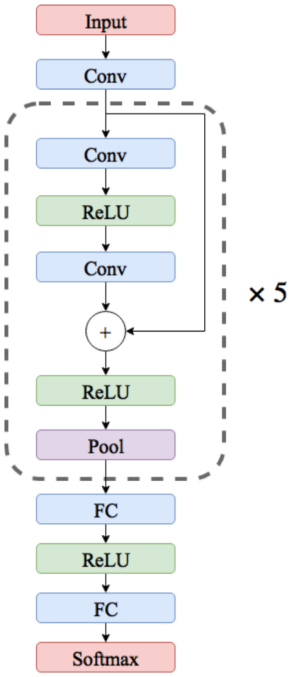

Figure 5.9. Architecture of the model in [24]

it and finally detects the R peaks as local maximum points above a threshold of 0.9. Secondly, it considers T as the median of the RR peak distances in time for a given signal. Finally, each beat consists of R peak and the samples in the temporal segment of $1.2 \cdot T$. Since T is specific for a given signal, to make all beats of the same size, shorter beats are padded with zeros.

The the whole pre-processed dataset is available on Kaggle by the same author's paper, so we have not directly implemented this pre-processing. The algorithm for the classification is a neural network with one-dimensional convolutional layers each with 32 kernels of size 5. In addition to this, it has max polling layers of size: 5 and stride: 2. The latter layers are arranged in a residual block the is repeated five times. In the end, there are two fully connected layers followed by a $softmax$ (Fig. 5.7).(Fig. 5.9). As loss function, it is used cross entropy with logits since the network's output is obtained with a $softmax$ and it can be interpreted as a probability distribution. The optimization method is Adam with the following parameters:

1. learning rate: 0.001
2. beta-1: 0.9
3. beta-2: 0.999

It is employed an exponential learning rate decay, where the learning rate is decayed of 0.75 every 10000 iterations. We have implemented it in TensorFlow with Colab and trained it for 70 epochs. In Fig. 5.10, we have plotted my results: they are not

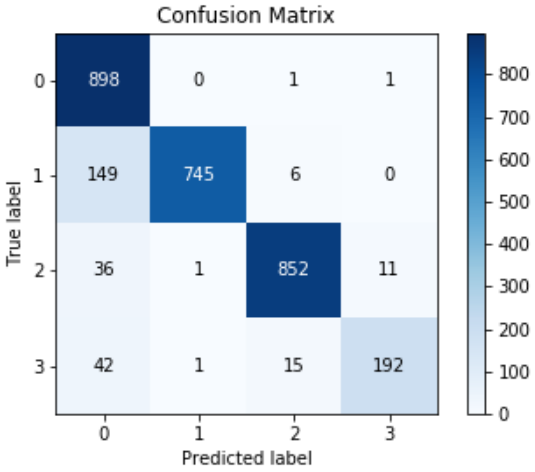

Figure 5.10. Confusion Matrix : cnn for the N,S,V,F classification

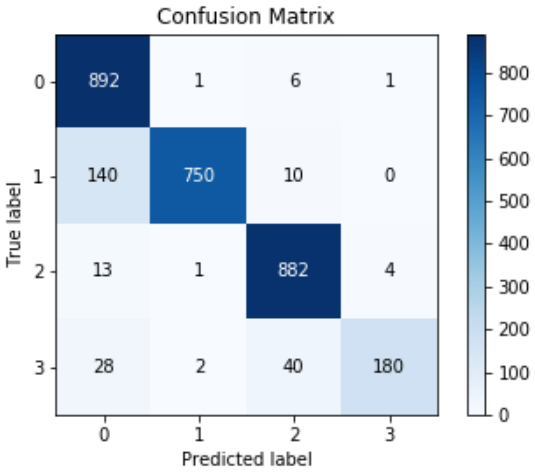

Figure 5.11. Confusion Matrix : CNN with Adasyn for the N,S,V,F classification

as good as the one described in the paper: we think this is due to the fact the latter does not describe the data augmentation technique used. We have tested the same approach using ADASYN as data augmentation (Fig. 5.11) This approach has the benefit of having a simple pre-processing phase and it is conventional in the use of a neural network model that can be deployed in an embedded device with the support of libraries such as Tensorflow-Lite and sensor. Finally, it has a better performance than the approach with Random Forest and Supports Vector Machine.

5.6 Neural Ordinary Differential Equations

In this section we will demonstrate the application of NODE ([12]) for arrhythmia detection: this is an innovative approach never used before in the literature. In order to implement it, we have adopted the PyTorch library provided by the same authors.

The latter not only provides (ODE) solvers but also the optimized back-propagation using the adjoint method. In addition to this, it has fully GPU support. We have conducted several tests, but all of them have in common the following choice:

1. the ODE solver has a tolerance of 0.001
2. loss function: categorical cross-entropy
3. learning rate: 0.001
4. training epochs: 100
5. Adam optimizer

We have tested both data augmentation and weighted loss function to mitigate the class imbalance. It is worth to note that we will describe only ADASYN because it gives better results than SMOTE. When we adopt the weighted loss function, we have used the following weights: 1,40,20,100, respectively for N, S, V, F classes. This choice is justified by the single class size proportion and after several trial and error tests. The input: x is a sequence of 200 samples. Finally, the models below have in common a general structure in this order: downsampling layer, feature layer, and fully connected layers. The NODE block is present in the second block.

5.6.1 Test 1

The downsampling layer consists of a residual block with these layers:

1. P_1: pooling layer of size 5 and stride 2
2. C_1,C_2: two 1-dimensional convolutional layers with 32 filters, each with padding 5 and stride 2
3. ReLU as activation function
4. the output is: $y_1 = P_1\left(x + z\right)$, where $z = ReLU\left(C_2\left(ReLU\left(C_1\left(x\right)\right)\right)\right)$

The feature layer is constituted of:

1. C_3 and C_4 are two convolutional layers with the same parameters of C_2
2. the output is the ODE function: $y_2 = ReLU\left(C_4\left(ReLU\left(C_3\left(y_1\right)\right)\right)\right)$

The fully connected block is made of:

1. F: flattening layer:
2. D_2 and D_1: two convolutional layers with respectively 32 and 4 neurons
3. the output is: $y_3 = F(D_2(ReLU(D_1(y_2))))$

In Fig. 5.12 there is the confusion matrix related to this approach using the weighted loss function to mitigate class imbalance. In Fig. 5.13 the same model is tested without loss function but using Adasyn. It is clear that the weighted loss function performs better than Adasyn: this is due to the first does not create any new samples that may be unrepresentative of the given class.

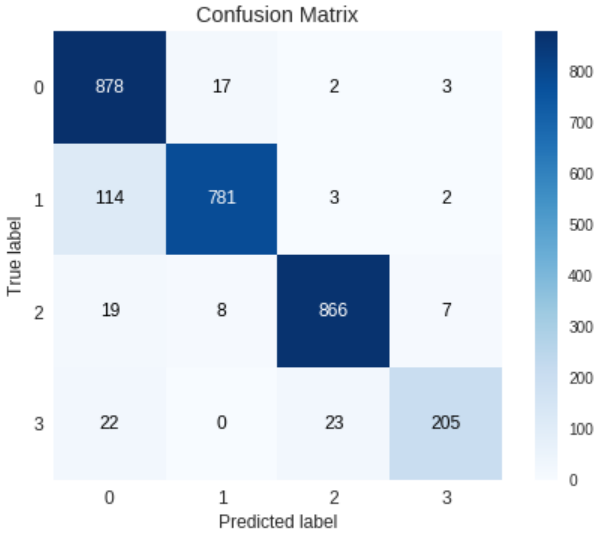

Figure 5.12. Confusion Matrix : first Node model with weighted loss function

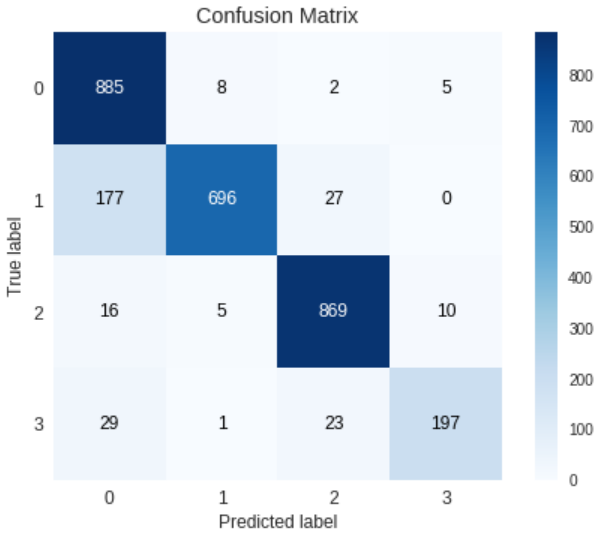

Figure 5.13. Confusion Matrix: first Node model with Adasyn

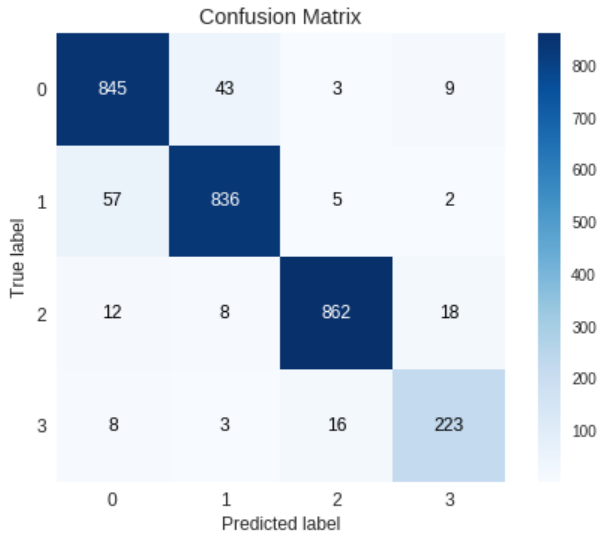

Figure 5.14. Confusion Matrix: second Node model with less parameter

5.6.2 Test 2

In this model, we have tried to reduce the number of parameters to take full advantage of the parameter efficiency offered by NODE. In the previous one, the high number of parameters: 116000, is due to the last fully connected layer that has 32 neurons with an input of size: 3136, for a total of $3136 \cdot 32 = 100352$ parameters. In order to reduce the size of the input to this layer that is the output of the ODE block, we have applied a pooling layer in between.

Specifically, it has a size of 10 and as stride: 5. In this case, we have used the weighted loss function since it generally gives better results than Adasyn. The number of total parameters is reduced to 34244 which is also less than the CNN of [24] that has 50000 parameters. The results are depicted in fig. 5.14 This model has the best accuracy so far and it is indicative of how a less complex neural network can have better performance.

5.6.3 Test 3

In this case, we wanted to check the effect of a de-noise procedure before training the neural network. Specifically, we have used the same filtering described in [15], which consists of the application of DWT to first decompose and later reconstruct the signal. In this case, we have used the same model as the first test. The results are depicted in Fig. 5.15 and there is an improvement in the accuracy concerning the latter (Fig. 5.12)

5.7 LSTM

In this section, we will describe the application of LSTM for arrhythmia detection. The intuition behind this approach is to leverage the sequential temporal structure

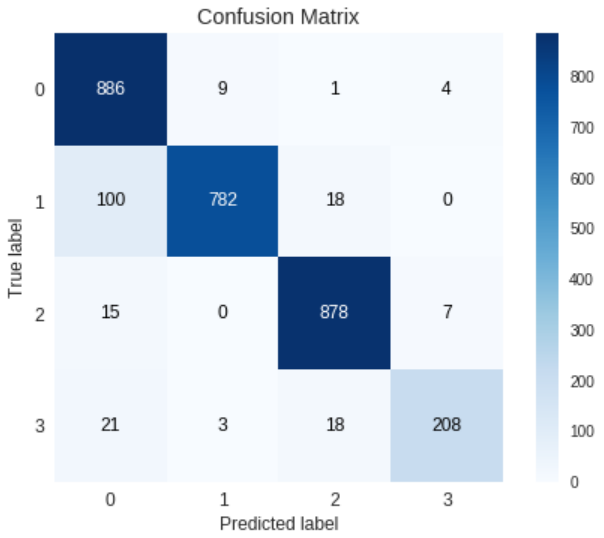

Figure 5.15. Confusion Matrix: third model with DWT denoise

of each beat. In fact, the LSTMs have shown great performance for time series, since they can model long term dependency. Specifically, for our case, we have that the sequential nature of each beat is crucial for the arrhythmia prediction because it captures the P wave, QRS complex, ST segment that are essential for the diagnosis. We have conducted two main tests, that differ with the type of dataset:

1. sequences of 200 samples (single beat).
2. sequences of 3/4 beats.

We have tested several models, and in all the circumstances the first configuration had superior performances, and the second one did not reach comparable results either with the initial algorithm: Random Forest and SVM. This is why in the following we will describe only the first case. We have implemented all the test below in Keras/TensorFlow, with the following common parameters:

1. loss function: categorical cross-entropy
2. learning rate: 0.001 with exponential decay
3. training epochs: 100
4. Adam optimizer

Finally, the LSTM's implementation in all the main Python frameworks (Keras, Tensorflow, PyTorch), allows obtaining not only the final hidden state but also the intermediate ones that are crucial if we want to apply attention techniques. Specifically, in Keras it is possible to specify an LSTM layer with the following parameters:

1. `cell`: each LSTM layer can have multiple cells and each of them outputs a hidden state, for every single element in the input sequence. If no other

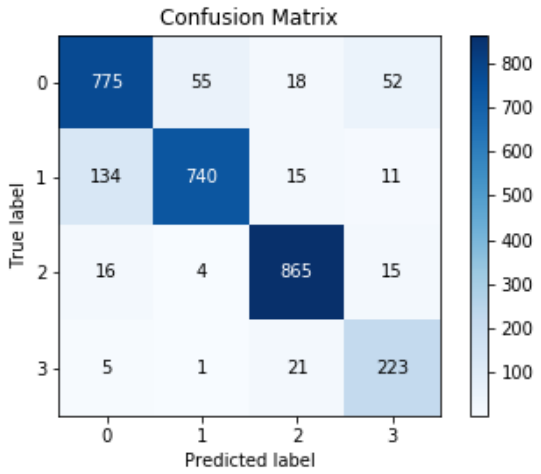

Figure 5.16. Confusion Matrix: simple model with LSTM

parameters are specified, the final output of this layer is the last sequence of cell hidden state.

2. `return_sequences`: if set to `True` allows to obtain all the hidden states for each input in the sequence. It is mainly used for applying attention.

The input needs to be three-dimensional with this structure: `(batch_size, timesteps, input_dim)`. The `timesteps` is always set to 200, while the `input_dim` is 1 or 2 concerning the number of channels we are considering in the original dataset: the ECG signals are recorded with 2 channels. In Test 1 and Test 2 we have considered only one channel, while in all the other cases both.

5.7.1 Test 1

The neural network has the following structure:

1. LSTM layer with 32 cells and `return_sequences` set to `True`: the output's shape is `(batch_size, timesteps, 32)`

2. LSTM layer with 32 cells but `return_sequences` set to `False`: the output's shape is `(batch_size, 32)`

3. Dense layer with 4 neurons followed by $softmax$ for the final classification

In Fig. 5.16, there is the confusion matrix associated with this model. In the following, we will describe more complex approaches.

5.7.2 Test 2

In this case we wanted to build a more elaborated model, with the aim to get better results. We succeed in the latter. However, the drawback is that it has an enormous number of parameters: 1.913.284 The model is the following:

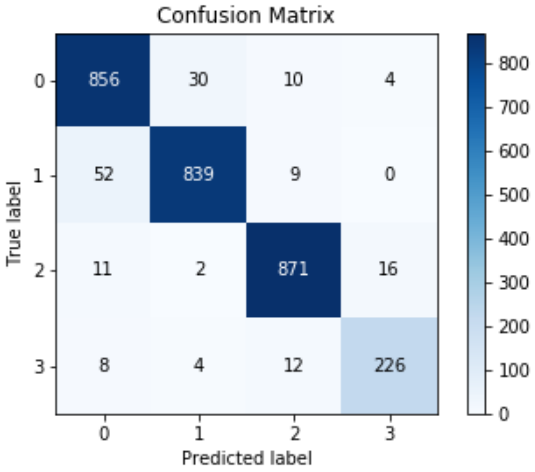

Figure 5.17. Confusion Matrix: complex model with LSTM

1. LSTM layer with 256 cells and `return_sequences` set to `True`: the output's shape is `(batch_size, timesteps, 256)`
2. LSTM layer with 512 cells but `return_sequences` set to `False`: the output's shape is `(batch_size, 512)`
3. Dense layer with 128 neurons followed by *ReLU*
4. Dense layer with 64 neurons followed by *ReLU*
5. Dense layer with 4 neurons followed by $softmax$

In Fig. 5.17, it is visible the quality of this model in terms of accuracy with respect the previous one, at the expense of a greta number of parameters.

5.7.3 Test 3

In this case we used both channels so that the input to the model has the shape: texttt(batch_size, timesteps, 2). We have simplified the structure so that it has 822.916 parameter. The model is:

1. LSTM layer with 256 cells and `return_sequences` set to `True`: the output's shape is `(batch_size, timesteps, 256)`
2. LSTM layer with 256 cells but `return_sequences` set to `False`: the output's shape is `(batch_size, 512)`
3. Dense layer with 128 neurons followed by *ReLU*
4. Dense layer with 4 neurons followed by $softmax$

In Fig. 5.18 we have plotted the results that are better than all the models tried so far.

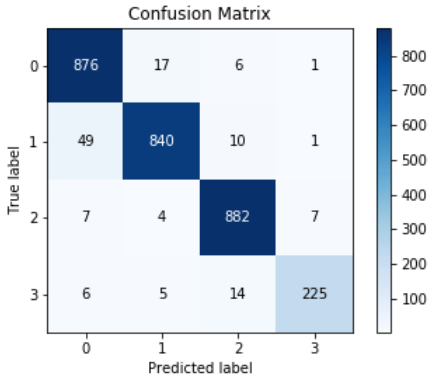

Figure 5.18. Confusion Matrix: third model with LSTM and 2-D input data

5.8 LSTM/NODE with Attention

In this section, we will describe how powerful is the application of Attention on the previous models. Since our task is a classification problem, namely sequence to one, the decoder state is not present. To compute the attention score we have taken into consideration the hidden states of the LSTM and as decoder state the last of one, that generally is the value which the decoder state is initialized. Finally, in the following test, we have considered 2 channels for the ECG recording. Initially, we have adopted Bahdanau's attention on LSTM. The neural network is the following:

1. LSTM layer with 256 cells and `return_sequences` set to `True`: the output's shape is `(batch_size, timesteps, 256)`

2. Bahdanau attention: the output's shape is `(batch_size, 256)`

3. Dense layer with 4 neurons followed by $softmax$

The number of parameters is 462.597. The results are depicted in Fig. 5.19 Secondly, we have used Luong's attention instead of Bahdanau's.

The results are depicted in Fig. 5.20. Next, we have tested the latter model on the dataset with the pre-processing of [24], with the result in Fig. 5.21. It is interesting to note the pre-processing does not help our models. Finally, we have adopted Luong's attention to the following model that has both a NODE block and LSTM layer. The latter is composed of four principal components: downsampling layers, feature layers, attention layers and fully connected layers.The first one is implemented as:

1. P_1: pooling layer of size 5 and stride 2

2. C_1,C_2: two 1-dimensional convolutional layers with 32 filters, each with padding 5 and stride 2

3. ReLU as activation function

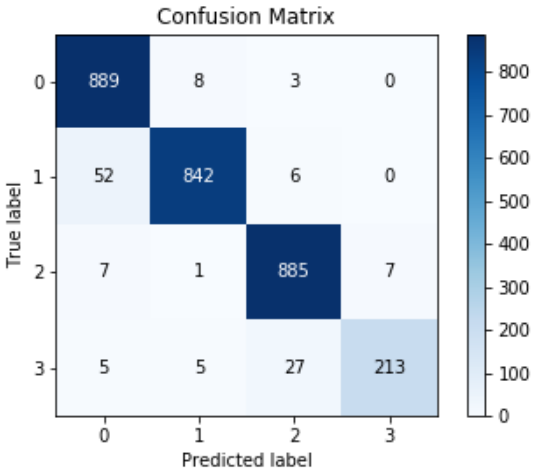

Figure 5.19. Confusion Matrix: Bahdanau's attention with LSTM

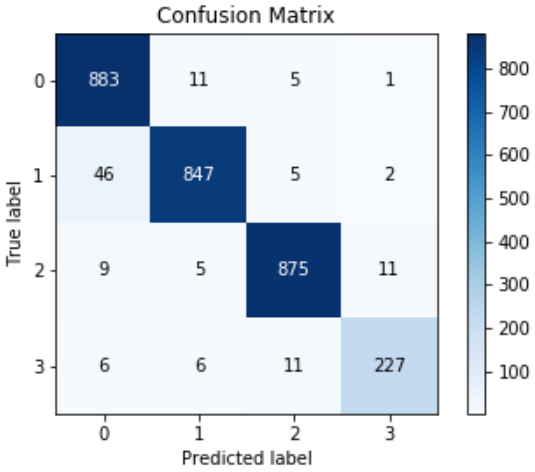

Figure 5.20. Confusion Matrix: Luong's attention with LSTM

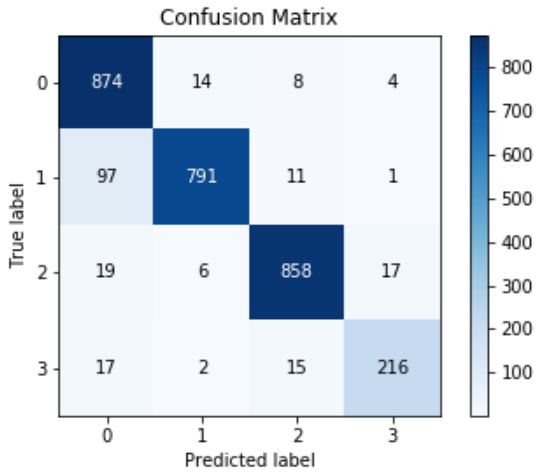

Figure 5.21. Confusion Matrix: Luong's attention with LSTM on the pre-processed dataset of [24]

4. the output is: $y_1 = P_1(z)$, where $z = ReLU(C_2(ReLU(C_1(x))))$ where x is the original input with 2 channels

The second one is and ODE block with the following parameters:

1. C_3 and C_4 are two convolutional layers with the same parameters of C_2
2. the output is the ODE function: $y_2 = ReLU(C_4(ReLU(C_3(y_1))))$

The third one is where the Luong's attention is applied:

1. LSTM layer with 128 cells and `return_sequences` set to `True`
2. Luong's attention: the output's shape is `(batch_size, 128)`

Finally the last block is dense layer with 4 neurons followed by $softmax$. The results are shown below (Fig. 5.22).

5.9 Transformers

We have adopted only the encoder part of the Transformers without the decoder because our task is a sequence to one while the latter is used for the sequence to many problems. The first block is made of:

1. LSTM layer with 128 cells and `return_sequences` set to `True`
2. Dropout layer with rate: 0.1

Then 4 identical layers implement Multi-Head Attention with 4 heads and constitute the Transformer's Encoder. Finally, it is applied, in order:

1. Dropout layer with rate: 0.1

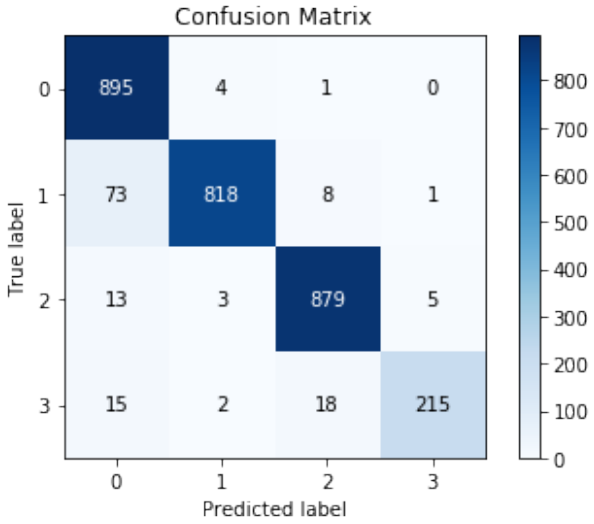

Figure 5.22. Confusion Matrix: Luong's attention with NODE model

2. Dense layer with 512 neurons
3. Flatten layer
4. 2 Dense layers with respectively 256 neurons and 4 neurons

It is worth to mention the LSTM layer is necessary to replace the Embedding layer that is originally present in the Transformer. However, in our case, we could not use the latter because it requires a dictionary of possible values for the input as it is the case of a word in input. We have also used CNN and the results are the same. The results are shown below (Fig. 5.23).

5.10 Per Patient Dataset

In this part, we wanted to check the robustness of the previous models in a more arduous scenario. As we explained in the section about the Dataset Configuration, we have removed from the whole dataset: D the following patient's data: D_{222},D_{223},D_{233}. We have re-trained the LSTM with Loung's attention: M on the remaining data: D_{train}, because it is one the best model we have implemented. Then we have built a personalized classifier for each of them using two different ways of transfer learning described respectively in Test 1 and Test 2

5.10.1 Test 1

This approach consists of removing the last layer of M so that the latter model can be used as feature extractor. In particular, now the output is a 128 feature vector. We have obtained for each patient's dataset (D_i) the corresponding features to train Support Vector Machine with the following parameters:

1. `gamma`: 'scale'

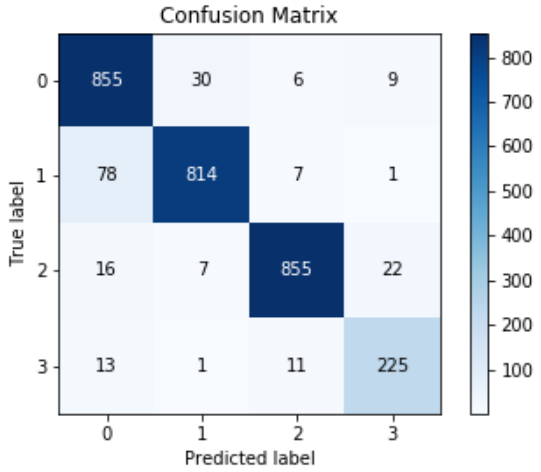

Figure 5.23. Confusion Matrix: Transfomers based model

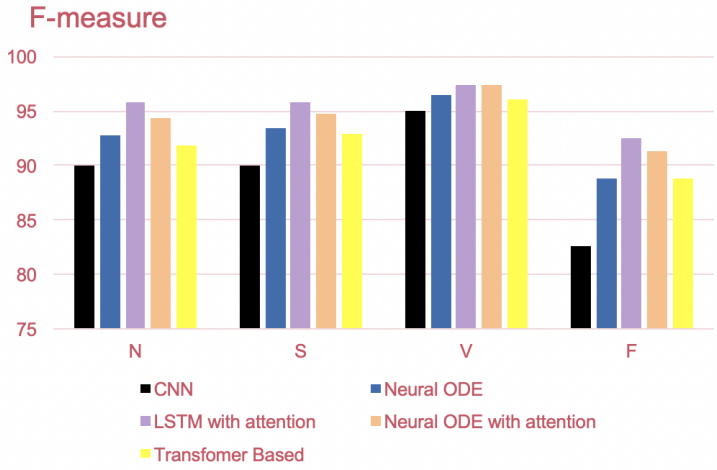

Figure 5.24. F score for the best models

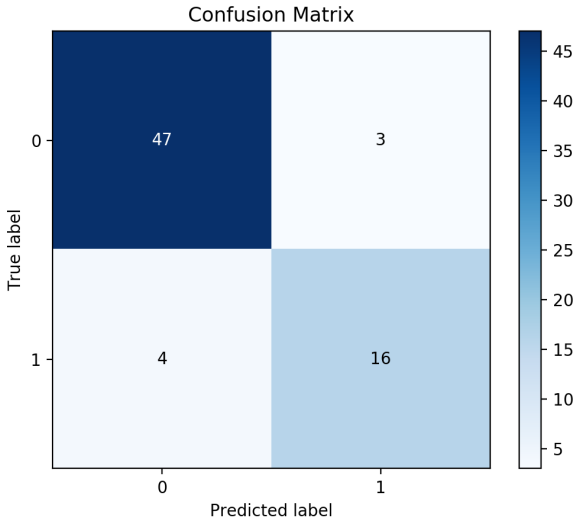

Figure 5.25. Confusion Matrix for Test 1: Patient 222

2. `decision_function_shape`: 'ovo'

We have used a weighted loss function, but with different weights because the dataset topology, in this case, is completely different from the previous one. In details we have tested several configuration and the best one is provided by `sklearn`, with the function: `compute_class_weight` and parameters: `'balanced'`. Basically, the weight for each class is assigned in this way: $w_i = \frac{N}{C \cdot f_i}$ where:

1. N: number of samples
2. C: number of classes (4)
3. f_i: number of beats belong to the i_{th} class

Fig. 5.25, Fig. 5.26, Fig. 5.27 , describe respectively the confusion matrix associated to the patients 222,223 and 233.

5.10.2 Test 2

In this part we have fine tuned the original model on each single patient. Initially, we have fine tuned all the network's layer. Fig. 5.28, Fig. 5.29, Fig. 5.30 , describe the results. Finally, we have fine-tuned only the last four layers because the idea is the the initial layers capture general feature about the beat nature. Fig. 5.31, Fig. 5.32, Fig. 5.33 , describe the results.

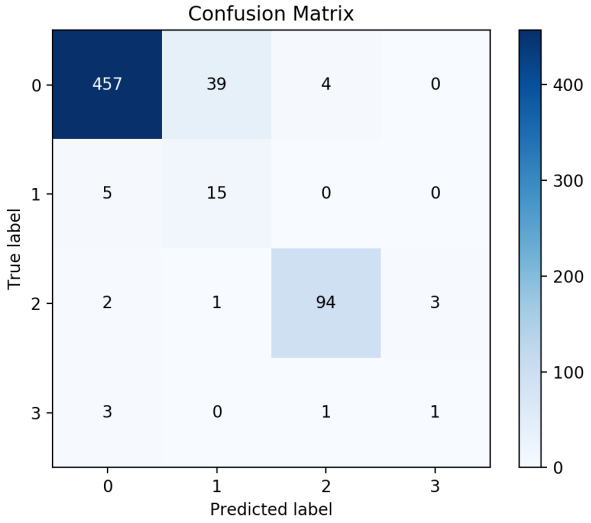

Figure 5.26. Confusion Matrix for Test 1: Patient 223

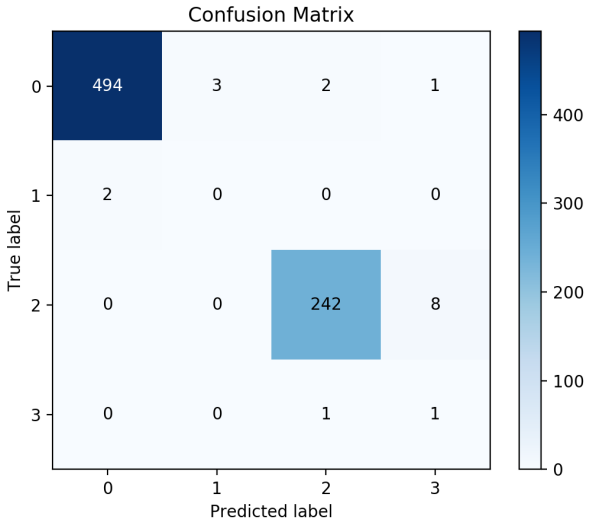

Figure 5.27. Confusion Matrix for Test 1: Patient 233

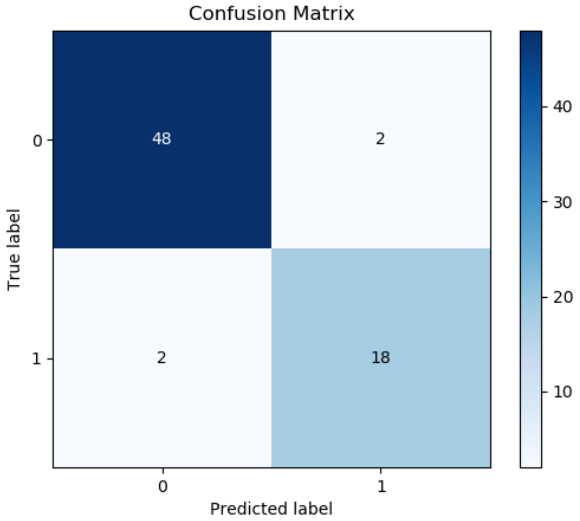

Figure 5.28. Confusion Matrix for Test 2 with complete training: Patient 222

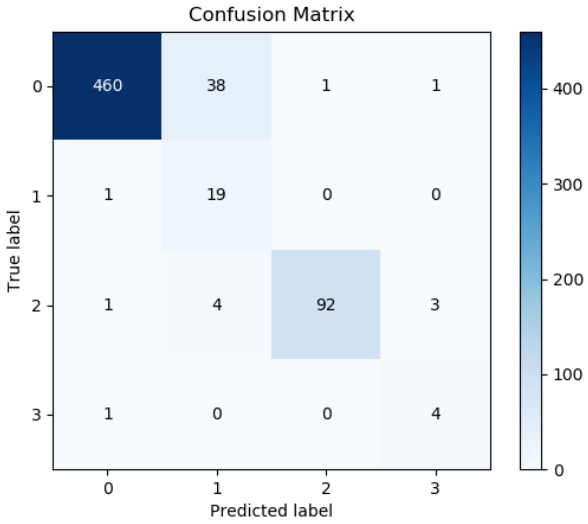

Figure 5.29. Confusion Matrix for Test 2 with complete training: Patient 223

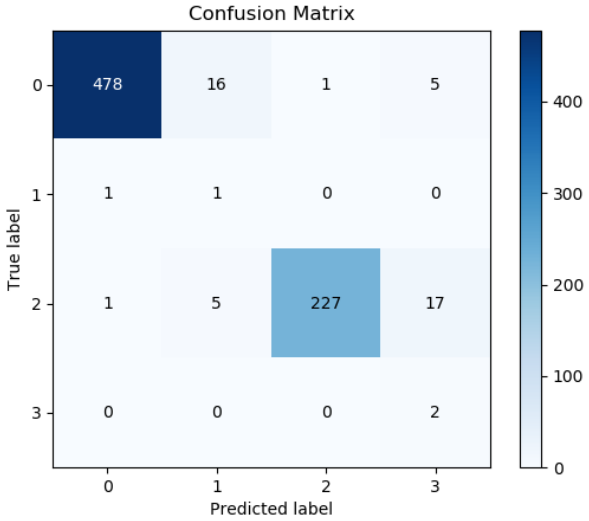

Figure 5.30. Confusion Matrix for Test 2 with complete training: Patient 233

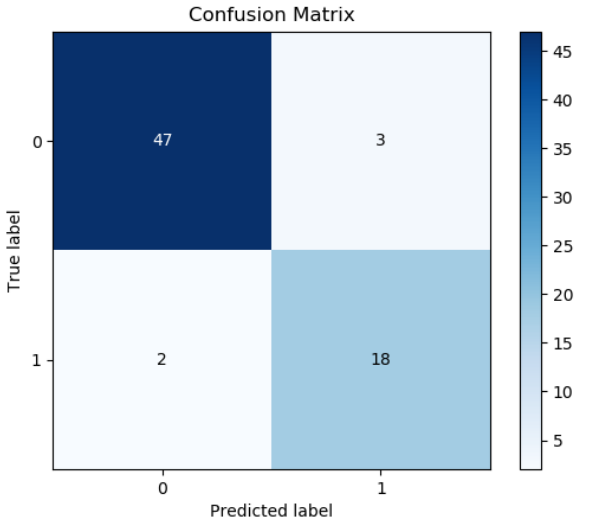

Figure 5.31. Confusion Matrix for Test 2 with partial training: Patient 222

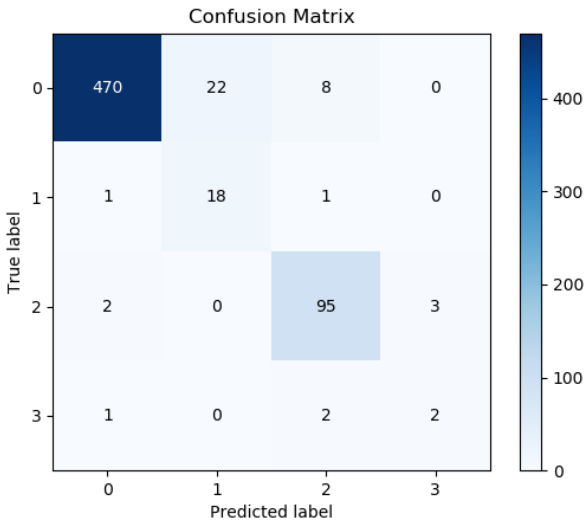

Figure 5.32. Confusion Matrix for Test 2 with partial training: Patient 223

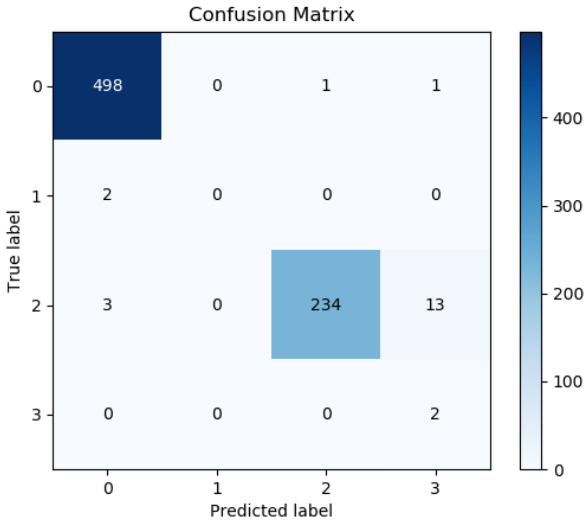

Figure 5.33. Confusion Matrix for Test 2 with partial training: Patient 233

Chapter 6

Conclusions

When we started working on this project, we considered the problem of arrhythmia detection as a simple classification task with only four classes. This is why at the beginning, we have approached it with elementary algorithms such as Random Forest. Secondly, we realized the results were not good and we thought the problem was due to the noise in the signal and a more accurate feature selection, would have provided better accuracy. In fact, we experimented with Discrete Wavelet Transform since it was a popular choice in the literature. However, we did not get a satisfactory outcome and decided to move to more complex models while keeping the pre-processing step. The reasons for this failure are due to the fact Random Forest applies the begging technique: it trains multiple decision trees on random partitions of the dataset and it computes the most probable outcome on the basis of each single decision tree's output. In our case, we have an overbalancing problem so that without any form of data augmentation, the various decision's trees will fail to learn the relevant features of rare classes. When we perform data augmentation, the synthetic samples fail to perfectly represent the low represented classes making weak classifiers for such labels. Even if we perform a pre-processing step to remove noise, the Random Forest can not generalize from synthetic samples.

At this point, we searched in the literature and the model described in [15] was exactly what we intended to implement: Support Vector Machine with a complex pre-processing. The results were slightly better than the first model but still, there was a lot to improve. The reasons for such low performance are probably the same as random forest since data augmentation is not able to create new authentic samples.

At this moment we realized the difficulties of the arrhythmia detection since it was challenging to implement an algorithm able to generalize the relevant features of each beat even for low represented class. We decided to use Neural Networks that are acknowledged to outperform the previous algorithms for complex tasks. The best model we found in the literature was [24] and it effectively provided acceptable results. The results were good both with data augmentation and better with weighted loss. In general, the generalization power of Neural Networks together with the use of a weighted loss function was the key for outstanding models. In fact, a weighted loss function does not have the problem to create synthetic data that may

not represent the original class. Moreover, a deep neural network approach does not require any pre-processing so that it exempts us from performing a complex operation such as DWT or PCA, which is exhausting for an embedded device with low computational power and autonomy. The main reason for this advantage is due to the neural networks' ability to autonomous learn distinctive features through the various layers, from the most general to the most specific. For example, we would expect the initial layers of that CNN to capture a beat as a simple smooth curve, and as we consider deeper layers, they capture more peculiarities such as the QRS complex, ST segment and so on. In this way, an optimal choice of the neural network's depth with its related operations can build a classifier able to detect only the relevant features without incurring into overfitting.

Secondly, we have used the NODE and LSTM. The first one was a novel structure with comparable results to [24]. The second model born by the idea that the ECG signal could be considered as temporal time series with sequences made of single samples rather than multiple beats. The significance of this intuition was justified by the respectable outcome. Then, we added to the previous models an attention mechanism following the trend in Natural Language Processing. Here we wanted to capture the fact that with respect to a particular arrhythmia, the model should pay more attention to some portion of the beat than others and in general, there are samples with low information content that must be ignored and others to give more weight. For example, we would expect samples in the QRS complex to have a higher attention weight than samples in the beat's prologue. In the end, the best model we have implemented was LSTM with Luong's Attention.

Finally, we used a patient-centric approach. In particular, we wanted to implement a model personalized for the end user, since our final goal was to implement an arrhythmia detection module on a wearable device. The approach we used was Transfer Learning and the outcome was good but there is a large margin of improvement. Initially, we used a pre-trained neural network as a feature extractor for the single and then SVM. The idea here is that the layers of a neural network learn features from the most general to the most specific, and in this way, the one before the last layer can be a good generalization.

As a second experiment, we first tuned all the model on a single patient and then only the last layers. Generally, the second technique is more popular because as we said before the initial layers of a neural network capture general features and only the final layers are focused on the peculiarities of the single patient. This is why also for our case, the latter approach tends to have a better score. However, for the evaluation we need to take into consideration some classes in the test data are low populated and statistically insignificant.

Better results can be obtained with more irregular beats for each patient, but this is not a good practice to solve the issue because in practice a user will experience for a long time regular beats, and the module still needs to find arrhythmias. In general, the confusion matrices in this case, show worse results than the previous situations with all the dataset. This means the latter model suffers from some sort

of overfitting because it does not learn to generalize arrhythmias for completely new patients. However, the only solution to solve this issue is to acquire more data.

This work could be further developed by implementing the provided algorithms on embedded devices. The models we have presented can be applied in a medical context to support cardiologists or for personal use. In addition to this, with the advent of smart cars the proposed algorithms with small variations, can be used to monitor the state of the driver and detect dangerous situations such as tiredness or altered states.

Bibliography

[1] Testing and reporting performance results of cardiac rhythm and st segment measurement algorithms. *AAMI EC57*.

[2] Akrivopoulos, O., Amaxilatis, D., Antoniou, A., and Chatzigiannakis, I. Design and evaluation of a person-centric heart monitoring system over fog computing infrastructure. In *Proceedings of the First International Workshop on Human-centered Sensing, Networking, and Systems*, HumanSys'17, pp. 25–30. ACM, New York, NY, USA (2017). ISBN 978-1-4503-5480-6. Available from: http://doi.acm.org/10.1145/3144730.3144736, doi:10.1145/3144730.3144736.

[3] Akrivopoulos, O., Amaxilatis, D., Mavrommati, I., and Chatzigiannakis, I. Utilising fog computing for developing a person-centric heart monitoring system. *Intelligent Environments*, (2018), 9.

[4] Akrivopoulos, O., Amaxilatis, D., Mavrommati, I., and Chatzigiannakis, I. Utilising fog computing for developing a person-centric heart monitoring system. *JAISE*, **11** (2019), 237. Available from: https://doi.org/10.3233/AIS-190523, doi:10.3233/AIS-190523.

[5] Akrivopoulos, O., Chatzigiannakis, I., Tselios, C., and Antoniou, A. On the deployment of healthcare applications over fog computing infrastructure. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 288–293 (2017). doi:10.1109/COMPSAC.2017.178.

[6] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. (2016). Available from: http://arxiv.org/abs/1409.0473, arXiv:1409.0473v7.

[7] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg (2006). ISBN 0387310738.

[8] Bowyer, K. W., Chawla, N. V., Hall, L. O., and Kegelmeyer, W. P. SMOTE: synthetic minority over-sampling technique. *CoRR*, **abs/1106.1813** (2011). Available from: http://arxiv.org/abs/1106.1813, arXiv:1106.1813.

[9] Cecilia Vinzio Maggio, A., Bonomini, M., Laciar, E., and Arini, P. *Quantification of Ventricular Repolarization Dispersion Using Digital Processing of the Surface ECG* (2012). ISBN 978-953-307-923-3. doi:10.5772/23050.

[10] Chatzigiannakis, I. Computing in the fog: Recent technological advances and development techniques. In *Intelligent Environments 2018 - Workshop Proceedings of the 14th International Conference on Intelligent Environments, Rome, Italy, 25-28 June 2018*, pp. 14–17 (2018). Available from: https://doi.org/10.3233/978-1-61499-874-7-14, doi:10.3233/978-1-61499-874-7-14.

[11] Chatzigiannakis, I., Valchinov, E. S., Antoniou, A., Kalogeras, A. P., Alexakos, C. E., and Konstantinopoulos, P. Advanced observation and telemetry heart system utilizing wearable ECG device and a cloud platform. In *2015 IEEE Symposium on Computers and Communication, ISCC 2015, Larnaca, Cyprus, July 6-9, 2015*, pp. 25–30 (2015). Available from: https://doi.org/10.1109/ISCC.2015.7405449, doi:10.1109/ISCC.2015.7405449.

[12] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *CoRR*, **abs/1806.07366** (2018). Available from: http://arxiv.org/abs/1806.07366, arXiv:1806.07366.

[13] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, **abs/1406.1078** (2014). Available from: http://arxiv.org/abs/1406.1078, arXiv:1406.1078.

[14] colah's blog. Understanding lstm networks. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[15] Elhaj, F. A., Salim, N., Harris, A. R., Swee, T. T., and Ahmed, T. Arrhythmia recognition and classification using combined linear and nonlinear features of ecg signals. *Computer Methods and Programs in Biomedicine*, **127** (2016), 52 . Available from: http://www.sciencedirect.com/science/article/pii/S0169260715301097, doi:https://doi.org/10.1016/j.cmpb.2015.12.024.

[16] Elhaj, F. A., Salim, N., Harris, A. R., Swee, T. T., and Ahmed, T. Arrhythmia recognition and classification using combined linear and nonlinear features of ecg signals. *Computer Methods and Programs in Biomedicine*, **127** (2016), 52 . Available from: http://www.sciencedirect.com/science/article/pii/S0169260715301097, doi:https://doi.org/10.1016/j.cmpb.2015.12.024.

[17] Emanet, N. Ecg beat classification by using discrete wavelet transform and random forest algorithm. In *2009 Fifth International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control*, pp. 1–4 (2009). doi:10.1109/ICSCCW.2009.5379457.

[18] Emanet, N. Ecg beat classification by using discrete wavelet transform and random forest algorithm. In *2009 Fifth International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control*, pp. 1–4 (2009). doi:10.1109/ICSCCW.2009.5379457.

[19] for Visual Recognition, C. C. N. N. Convolutional neural networks (cnns / convnets). Available from: http://cs231n.github.io/convolutional-networks/.

[20] Gers, F. A. and Schmidhuber, J. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, vol. 3, pp. 189–194 vol.3 (2000). doi:10.1109/IJCNN.2000.861302.

[21] He, H., Bai, Y., Garcia, E. A., and Li, S. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In *IJCNN*, pp. 1322–1328. IEEE (2008). ISBN 978-1-4244-1821-3. Available from: http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2008.html#HeBGL08.

[22] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, **abs/1512.03385** (2015). Available from: http://arxiv.org/abs/1512.03385, arXiv:1512.03385.

[23] Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Comput.*, **9** (1997), 1735. Available from: http://dx.doi.org/10.1162/neco.1997.9.8.1735, doi:10.1162/neco.1997.9.8.1735.

[24] Kachuee, M., Fazeli, S., and Sarrafzadeh, M. ECG heartbeat classification: A deep transferable representation. *CoRR*, **abs/1805.00794** (2018). Available from: http://arxiv.org/abs/1805.00794, arXiv:1805.00794.

[25] Kuo, C. J. Understanding convolutional neural networks with A mathematical model. *CoRR*, **abs/1609.04112** (2016). Available from: http://arxiv.org/abs/1609.04112, arXiv:1609.04112.

[26] Luong, T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (2015). Available from: http://dx.doi.org/10.18653/v1/D15-1166, doi:10.18653/v1/d15-1166.

[27] MathWorks. Continuous wavelet transform and scale-based analysis. Available from: https://it.mathworks.com/help/wavelet/gs/continuous-wavelet-transform-and-scale-based-analysis.html.

[28] Moody GB, M. R. The impact of the mit-bih arrhythmia database. *IEEE Eng in Med and Biol*, **10** (2001), 45 .

[29] Organization, W. H. Cardiovascular diseases (cvds). Available from: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[30] Pascanu, R., Mikolov, T., and Bengio, Y. Understanding the exploding gradient problem. *CoRR*, **abs/1211.5063** (2012). Available from: http://arxiv.org/abs/1211.5063, arXiv:1211.5063.

[31] R. Anderson, J. *Cognitive Psychology and Its Implications* (2005).

[32] Scire, A., Tropeano, F., Anagnostopoulos, A., and Chatzigiannakis, I. Fog-computing-based heartbeat detection and arrhythmia classification using machine learning. *Algorithms*, **12** (2019). Available from: http://www.mdpi.com/1999-4893/12/2/32.

[33] Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition (2014).

[34] stackoverflow. What is the number of filter in cnn? Available from: https://stackoverflow.com/questions/36243536/what-is-the-number-of-filter-in-cnn.

[35] Taspinar, A. A guide for using the wavelet transform in machine learning. Available from: http://ataspinar.com/2018/12/21/a-guide-for-using-the-wavelet-transform-in-machine-learning/.

[36] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30* (edited by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett), pp. 5998–6008. Curran Associates, Inc. (2017). Available from: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[37] Walraven, G. *Basic Arrhythmias*. Pearson (2019).

[38] WILDML, D. L., Artificial Intelligence and NLP, D. B. Recurrent neural networks tutorial. Available from: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/.