Carolina Martins Barbosa Rodrigues Afonso

Changing Perspectives: Interlead Conversion in Electrocardiographic Signals

U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
June 2020

Carolina Martins Barbosa Rodrigues Afonso

Changing Perspectives: Interlead Conversion in Electrocardiographic Signals

Dissertation

Supervisor: Miguel Tavares Coimbra
Co-supervisor: João Tiago Ribeiro Pinto

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
June 2020

First of all, I would like to thank INESC TEC for allowing me to complete my master's thesis, in an area that has always captivated me. I want to thank João Ribeiro Pinto, for always guiding me in the best way, helping me to evolve, and always wanting to go further. To professor Miguel Coimbra, for showing me the area of computer vision and making me choose it as the basis for the development of my work. Second, I thank my parents who always supported me during these years of college. Finally to all my friends who were with me during this stage and to the Faculdade de Ciências that was my second home during these years.

Abstract

The electrocardiogram (ECG) is a physiological signal that measures the conduction of electrical currents through the heart, that allow for its contraction and relaxation to maintain adequate blood flow[3]. In the standard configuration, widely used in medical settings for heart monitoring and diagnosis, the ECG is acquired using electrodes in contact with the subject's wrists, ankles, and chest. When connecting two electrodes, we get a view of the heart, named Lead. In this dissertation, we studied how to convert those Leads into others, with the goal of helping with the diagnosis of some heart diseases by predicting the Leads that are missing from the recordings without having to submit the patient to various exams. For this task, we relied on signal processing and machine learning techniques with special focus on robust deep learning architectures, such as autoencoders, and implemented an algorithm that can be adaptable for the conversion into other Leads. This work paves the way towards easier and more comfortable ECG exams without losing potential for diagnosis - one Lead will be sufficient as the algorithm is able to accurately convert it into all other heart perspectives.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

After research on Lead conversion on electrocardiograms (ECG), we concluded that there was no great progress in the idea that we came to propose in this thesis. A normal electrocardiogram consists of a recording of 12 Leads, and what we are going to see is the implementation of an algorithm that allows us to convert these Leads into each other. “What is the purpose of this idea?”, “what is its contribution to medicine?”, “is it necessary to have the possibility of this conversion?”, “what is its use?”. So many questions that came to us but that also served as motivation for this implementation. In this section we will introduce our idea, explaining our motivation, objectives, contributions, and finally defining the structure of the thesis.

1.1 Motivation

The electrocardiogram (ECG) is a physiological signal that measures the conduction of electrical currents through the heart, that allows for its contraction and relaxation to maintain adequate blood flow[3]. In the standard configuration, widely used in medical settings for heart monitoring and diagnosis is acquired using electrodes in contact with the subject’s wrists, ankles, and chest.

Depending on the location of the electrodes, the electrocardiogram offers a different perspective of the heart. Hence, the usefulness of each Lead depends on the expected application. In biometrics, the most commonly used channel is Lead I (right arm - left arm) for improved usability and user comfort.

In healthcare, each Lead offers insight into specific conditions or irregularities. However, in several cases, the most favorable Lead may be unavailable, and professionals(or automatic systems) may be forced to use less adequate Leads.

Despite ECG in machine learning is usually used for arrhythmia detection, in this thesis, we will focus the use of ECGs as a biosignal for the development of methods for automatic conversion of Leads in electrocardiograph signals, which will help in the diagnose of some heart diseases, without having to submit the patient to various exams.

As a result of this thesis, it is expected to have an algorithm to convert, as faithfully as possible, ECG signals from certain Leads into target Leads, being this targets the Leads we need depending on the task. This will be achieved by resorting to signal processing and machine learning techniques, with a special focus on robust deep learning architectures.

Focusing on biometric applications, and the development of our work we are mainly going to have as the conversion target the Lead I, due to its acquisition being more accepted since the electrodes are positioned on the wrists[27], but the algorithm is expected to be adaptable for the conversion into other Leads.

1.2 Objectives

The big question we want to see answered in this thesis is, "is it possible?". All this uncertainty that comes together when we start research on a topic in an almost pioneering way, that is, when we do not find large studies on what we want to have some ideas that support our decisions, it always gives that additional motivation to continue and prove that we will achieve something. Thinking now in a more scientific way, what we want is to realize if the conversion of Leads is possible, so that, the information given by each one of them is not lost.

To forward our thesis so that we can get these answers, we organize our tasks as follows:

- Prior art study: bibliography review on ECG acquisition, Leads, datasets;
- Define the experimental protocol;
- Prepare data;
- Perform preliminary experiments;

- Design and implementation of the conversion model;
- Training of the implemented model;
- Evaluate performance and compare with the state-of-the-art;

1.3 Contribution

The contributing factor behind the implementation of this new conversion system is purely to be in par with changing technology and to enhance the existing deep learning techniques that are efficient. Resulting in higher quality and accuracy.

Our approach to this problem, consists in the creation of an algorithm to convert ECG signals from certain Leads, into a target one and the way we are planning to tackle this problem is by implementing an autoencoder that will have as an input the Lead to convert, code it an decode it into another Lead. If we can develop this idea and have good results, we also aim to extend this conversion so that instead of having only one target, we will have several.

The next image shows, the architecture we design for our algorithm:

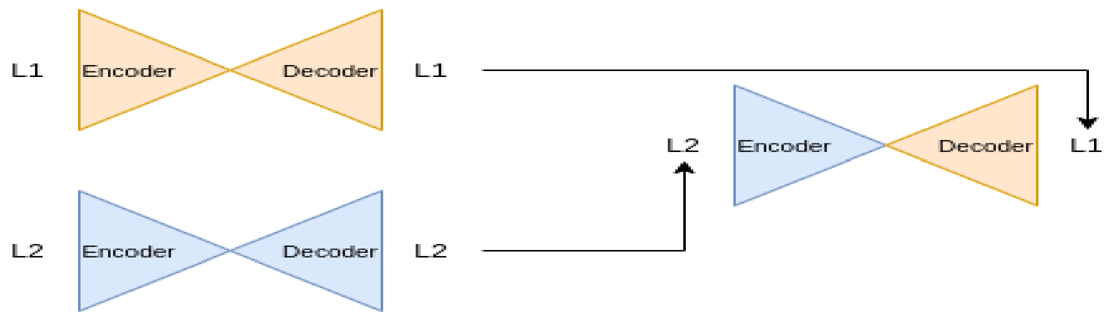

Figure 1.1: General architecture of our algorithm - diagram

As we can see in the diagram above that shows the idea of our algorithm. What we plan to do is to create autoencoders that take a Lead and rebuild it, and then combine the encoders of the different Leads with a decoder that will convert to the Lead I. What we want with this creation of an autoencoder per Lead is that the respective encoder will be able to create coding for each one of them so that when we go to the decoder it will be able to find the correlations and thus convert into another.

Understanding all different Leads, and the way an autoencoder works is a key factor for successful results. Therefore before we proceed to explain the objectives for this work, the first thing to clarify is the definition of the word “Lead” in the ECG context. A Lead refers to an imaginary line between two ECG electrodes.[29] The electrical activity of this Lead is measured and recorded as part of the ECG. The 12 Leads are made up of three bipolar and nine monopolar Leads. The three bipolar Leads are the electrical potentials between the right and left arm (Lead I), the right arm and left foot (Lead II), and between the left arm and left foot (Lead III). For the monopolar Leads, four different artificial reference points are constructed. Using these reference points, the potentials appearing on the left arm (VL), the right arm (VR), the left foot (VF), and on the six chest electrodes (V1-V6) are measured.[3]

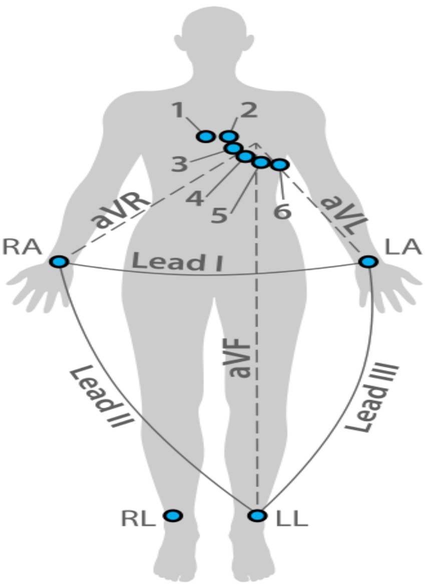

Figure 1.2: Standard 12-Lead configuration[27]

This will pave the way towards the obsolescence of laborious and uncomfortable multi-Lead ECG exams - one Lead will be sufficient as the algorithm would be able to accurately convert it into all other heart perspectives.

1.4 Thesis Structure

This thesis is organized as follows: we have eight chapters, **Background** (chapter 2), in this chapter we described every tool, concept and techniques we use during the development task, **Related work** (chapter 3), in here we describe all the related work we found regarding what we needed for the thesis, but mainly this chapter is used as an introduction for the basic concepts we need to be familiarized with before start the implementation, **Methodology** (chapter 4), in here we describe all the work done, **Experimental Methodology** (chapter 5), where we explain everything that is useful to give a better understanding of the methods, **Results** (chapter 6) where we show and explain all the results in a quantitative and qualitative approach and give some lights on improvements that we think it can be done, **Discussion** (chapter 7), here we discuss everything that can be improved and done as future work having as basis our model, and the last one, **Conclusion** (chapter 9), where we do a recap of the all thesis explaining the strongest and weakest points that we have at this point, with the hope this first approach to this theme will help and attract more research to this field.

Chapter 2

Background

As mention in the previous section, the goal of this thesis is the implementation of a Lead convert algorithm, that is adaptable to convert any kind of Lead. Before beginning this implementation it is important to understand all the techniques, methods, terms, and everything else related to this topic, and that's why this chapter exists. In this section, we are going to explain in detail all these things, to help understand our algorithm and its purpose.

In a theoretical approach, our algorithm is going to have four steps: data normalization, encoder, decoder, and evaluation. In the first step, we are going to make sure that all the data (electrocardiogram (ECG) signals) have the same properties, such as size, find the location of its peaks, and are all center in the same peak. When we get to the encoder part, here we are already in the autoencoder, and we are expecting that the signal is being learned, to guarantee that in the next phase (decoder) the signal is reconstructed as accurately as possible. For the last step, we are going to apply metrics to verify the accuracy of the reconstruction.

Before we start explaining what are autoencoders and how they work, we are going to start explaining some terms, that are the key for the understanding, not only our algorithm but also the motivation that led us to try and create it. When we started our research on this topic we realized quickly that not a lot of work has been done in this kind of conversion, and so that was one of the main reasons that make us interested in trying it.

Since the main focus of this thesis is signal processing, one of the terms that we must know is biological signals because we are working with ECGs, as well as the signal processing techniques themselves to be able to understand further on the transformations that were made to the ECG signal.

2.1 Biosignals

A biosignal is any signal in human beings that can be continually measured and monitored like electrocardiogram (ECG) from the heart, electromyogram (EMG) from the muscles, electroencephalogram (EEG) from the brain and electrooculogram (EOG) from the eyes, among others.[8]

The practical applications for biosignals are associated with the monitoring of physical activity. In medicine, they are used to test resistance to physical activity[8], how we will see it later on, when we talk about ECGs and how they are recorded.

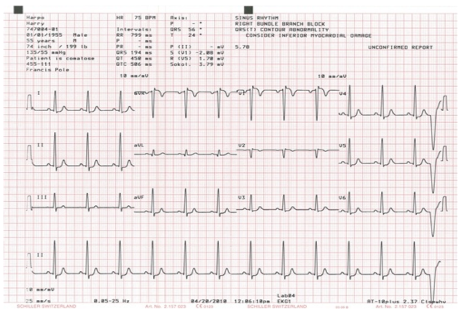

Figure 2.1: ECG record[11]

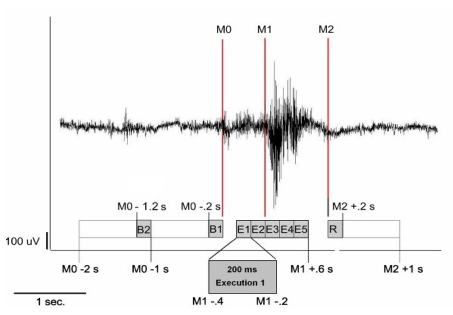

Figure 2.2: EMG record[30]

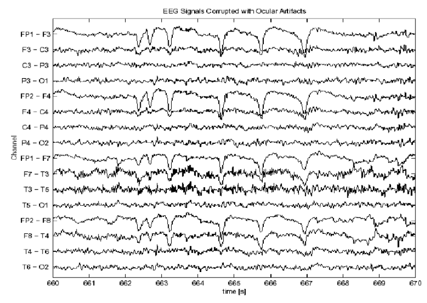

Figure 2.3: EEG record[21]

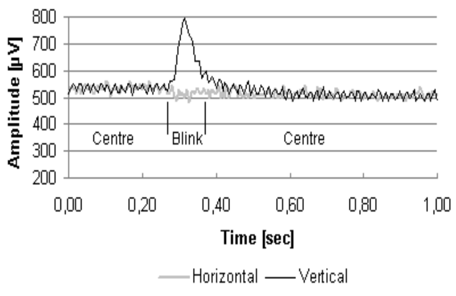

Figure 2.4: EOG record[31]

The biosignals can be continuous ($x(t)$), discrete ($x(n)$, where n = 0,1,2,3...), deterministic and random. Continuous signals can be defined over continuous values of time and space. Discrete signals define only a subset of regularly spaced points in time, continuous signals from the human body are converted to discrete signals by a process called sampling and they can be analyzed and interpreted by a computer.[10] Deterministic biosignals can be described by mathematical functions or rules, it illustrates the initial conditions. Deterministic processes can be divided into:

- **Transitional processes** - action system measured biosignals that are recorded only once.
- **Established processes** - the system will vary periodically in time or does not change at all.

A periodic signal is a good example of a deterministic biosignal, $x(t) = x(t + nT)$, repeats itself every T units in time, so T represents the period. The other kind of biosignal is random, and we can divide this type into random stochastic signals, that contain uncertainty in the parameters that describe them, so it cannot be precisely described by mathematical functions, it is often analyzed using statistical techniques with probability distributions or simple statistical measures such as the mean and standard deviation, and stationary random signals, where the statistics or frequency spectra of the signal remain constant over time.[10]

2.1.1 Electrocardiogram (ECG)

Once we define a biosignal and since we are working with one in this thesis, we should understand what an ECG is, not only as a biosignal but what is used for, and with that being said, the ECG is an exam used for diagnosing, because it evaluates the heart function. Despite all the continuous evolution of technology, the ECG still holds the main role in the research of heart diseases, it consists of an exam that detects the electrical activity of the heart. Each contraction of the heart muscle is commanded by small electrical impulses that are generated by it. This set of electrical impulses generates a signal - the ECG -, and which is formed by a set of P, Q, R, S and T waves that are repeated over time.[27] The ECG can also identify normal transmission patterns and the generation of the electrical impulses. It has great value in the evaluation of some types of heart abnormalities, including heart valves diseases, cardiomyopathy, pericarditis, and cardiac squeals of hypertension. Being a harmless exam and inexpensive, it is most useful for the study of the heart.

There are three types of ECG, resting, stress, and the Holter exam: The resting is done with the patient lying down and with his bare torso. The ideal situation is that the patient before the exam made no effort in the past 10 minutes, and has not smoked in the past 30 minutes. The patient should also avoid drinking cold water because it can change the signal. During the exam 10 electrodes are placed on the patient (6 on the chest, 2 on the wrist, and 2 placed on the ankles). The electrodes are wires that you attach to the patient to record the ECG. They allow Leads to be calculated. For regular ECG recordings, the variations in electrical potentials in 12 different directions (Leads) out of the ten electrodes are measured. After placing all the electrodes, they are connected to a machine that will perform the ECG. The exam only takes a couple of minutes, and during that time the machine records the electrical signals and draws them in a specific paper. The resting ECG is one of the most performed exams because it is cheap, fast, and easy to do. On the other hand, stress ECG requires another kind of equipment, like a treadmill or an exercise bike, which makes it more expensive. This exam allows the evaluation of the heart in stress conditions. If the resting ECG is normal and the history of the patient is suggestive of heart disease, the stress ECG can show alterations not before seen in the other. The Holter exam, in which it is recorded the heart activity during 24 hours, let us study the behavior of the heart during a day, comparing it with activities carried out and with the patient symptoms, that are recorded in a diary.[1]

The ECG tracing consists of 5 elements: wave P, interval PR, complex QRS, segment ST and wave T. The wave P corresponds to the depolarization of the auricle, the interval PR is the time since the beginning of the depolarization of the auricles and ventricles, the complex QRS corresponds to the depolarization of the ventricles, the segment ST is the time between the end of depolarization and the beginning of the repolarization of the ventricles and the wave T is the repolarization of the ventricles, that are ready to another contraction. Each heartbeat is made of a wave P, one complex QRS and one wave T.[20]

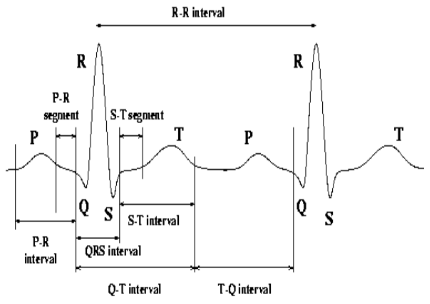

Figure 2.5: Representation of the fiducial landmarks in a heartbeat[20]

2.1.2 Signal processing

Signal processing is the analysis or modification of signals using fundamental theory, applications, and algorithms, to extract information from them in a way that makes them more suitable for specific applications. The processing can be done in an analogical or digital way. This process uses mathematics, statistics, computing, heuristics and linguistic representations, formalism and representation techniques, modeling, analysis, synthesis, discovery, recovery, detection, acquisition, extraction, learning, security, and forensics[23]. The sample for signal processing can include, sounds, images, temporal series, telecommunications signals, among others.

The techniques of signal processing can show great use in the analysis and control of physic systems of interest to the most diverse researchers, not only electrical engineers but mechanical engineers. Nowadays signal processing is being brought to the attention of professionals in other fields of study like economy, biology, health, and they can integrate both health and computer technologies.

As we saw before, the fields to use signal processing are endless, so to get the same technique suitable for the different applications, we have different types of processing depending on the kind of signal we are dealing with, like the analogical signal, non-linear signal, discrete signal, and digital signal.

- **Analogical signal**: Any kind of processing used in analogical signals is done through analogical means. Analogical values are usually represented by tension, electric current, or electrical charge in electrical devices. Analogical signal processing is done in signals that have not been scanned yet, like radio, telephone and radar signals, television systems. This process involves electrical linear and non-linear circuits.

- **Non-linear signal**: Signal non-linear processing involves the analysis and processing of signals produced by non-linear systems and can be determined by time, frequency, or space-time domains[4].

- **Discrete signal**: Discrete signal is a temporal series that consists of a quantity sequence, a function on the discrete integer domain. Each value in the sequence is called a sample and is defined only in discrete points in time. Different from a continuous signal, a discrete one is not a function of a continuous argument. However, the function can be generated through a continuous signal of the sampling. We can obtain discrete signals in several ways, but normally they are classified into two groups[18]. The first one is the acquisition of values of an analogical signal in time, a process called signal sampling. The second one is the accumulation of value in time, for example, the number of people that enter a room per day. The concept of discrete signal processing, establish a mathematical base for digital signal processing without quantization.

- **Digital signal**: The digital processing signal (DSP) is the mathematical manipulation process of a signal, to modify it or improve it in some way. Is characterized through the discrete signal representation. It has several computational techniques that can be used directly in a computational system based on IBM-PC pattern, without needing to use specific hardware like FPGAs or micro-controllers. The Fourier transformations are one of the examples of this kind of processing. The goal of DSP is to measure, filter, and compress analogical signals. Usually, the first step for the conversion from analogical to digital is the sampling of the signal and then digitalize it using a converter analogical-digital, that transforms the analogical signal in a flow of digital discrete values. Often we need to have the conversion done oppositely, so a converter digital-analogical is also needed. The process of digital processing is more complex than the analogical signal processing, this application has a lot of computational power, which allows a lot of advantages, when comparing with the analogical signal processing in various applications, like detection and correction of errors in the transmission, as well as compression of data[6]. The DSP algorithms can be used in normal computers as well as computers that are specifically made for those types of processing. Nowadays, exists technologies used for signal processing like Field-programmable gate array (FPGA), digital signal controllers, among others.

2.2 Machine Learning

Machine learning (ML) is a sub-field of computer science engineers that evolved from the study of pattern recognition and the theory of machine learning in artificial intelligence. It explores the study and construction of algorithms that can learn from their mistakes and make predictions about data. Such algorithms operate by building a model from sample inputs to make predictions or decisions guided by the data rather than simply following inflexible and static programmed instructions. Whereas in artificial intelligence there are two types of reasoning, machine learning is only concerned with inductive.[16]

When ML appeared, its goal was to make computers modify and adapt their actions according to the tasks they would have to perform. In this way and practical terms the ML algorithms "learn" the task to be able to perform it without errors, this is, with high accuracy.[22]

Some parts of machine learning are closely linked to computational statistics. They focus on how to make predictions through the use of computers, with research focusing on the properties of statistical methods and their computational complexity. It has strong ties to mathematical optimization, which produces methods, theory, and application domains for this field. Machine learning is used in a variety of computational tasks that work on creating and programming explicit algorithms that are impractical.

Machine learning is sometimes confused with data mining, which is a sub-field that focuses more on exploratory data analysis and is known as unsupervised learning. In the field of data analysis, machine learning is a method used to plan complex models and algorithms that lend themselves to make predictions, in commercial use, this is known as predictive analysis. These analytical models allow researchers, data scientists, engineers, and analysts to produce reliable and repeatable decisions, results and discover hidden insights by learning the historical relationships and trends in the data.[12]

Some types of machine learning[22]:

- **Supervised learning** - In this kind of learning, the algorithm analyses all the training data, and throughout that learning process it generates a function that will be used to map new examples.

- **Unsupervised learning** - The algorithm looks for patterns that were not detected yet and try to match with other inputs, when finds a match between two inputs, a category is created.

- **Reinforcement learning** - This one differs from supervised learning, once it does not need the training data, instead it tries to create a balance between exploratory (analyzing unknown patterns) and exploitation work.

2.3 Deep Learning

Deep learning is a type of machine learning, that trains computers to perform the task as humans, which includes voice recognition and image identification. Instead of organizing data to be trained through predefined equations, deep learning sets up basic parameters about data and trains the computer to learn by itself through patterns of recognition in many processing layers, based on a collective of algorithms that try to model high-level abstractions of data, using a deep graph with many layers, composed by various linear and non-linear transformations.[15]

Deep learning is one of the bases of artificial intelligence (AI). The deep learning techniques have been improving the classification, recognition, detection, and describing capacities of computers, in one word, understanding. For example, deep learning is used in the classification of images, voice recognition, object detection, and content description. Systems like Siri are partially fed by deep learning.

An image can be represented in several ways, such as a vector of intensity values per pixel, in a more abstract way as a set of edges or regions with a particular shape. Some representations are better than others to simplify the learning task. One of the solutions that are achieved with deep learning, is the replacement of features made manually by efficient algorithms for supervised or semi-supervised feature learning and hierarchical feature extraction.[13]

Research in this area aims to make better representations and create models to learn these representations from large-scale unlabeled data. Some of the representations are inspired by advances in neuroscience and are based on the interpretation of information processing and communication patterns in a nervous system, such as neural coding that attempts to define a relationship between various stimuli and the associated neuronal responses in the brain.[26]

Various deep learning architectures, such as deep neural networks, deep convolutional neural networks, deep belief networks, and recurrent neural networks have been applied in areas such as computer vision, automatic speech recognition, natural language processing, audio recognition, and bioinformatics, where they have shown themselves capable of producing state of the art results in various tasks.

2.3.1 Autoencoders

As said in the previous chapter, an autoencoder is an unsupervised machine learning algorithm that takes an input (signal, in this thesis) and tries to reconstruct it using a fewer number of bits from the bottleneck also know as latent space. The compression in autoencoders is achieved by training the network and as it learns it tries as best as possible to represent the input at the bottleneck.

The main difference between a basic autoencoder and a neural network is the fact, that autoencoders have two symmetric parts, an encoder, and decoder, both have dimensions of compressed representations smaller than the dimensions of the input data. This allows us to learn the information from the compressed data to use in the reconstruction phase and not just the identity function that has the original input memorized. One of the things, we need to be careful when creating autoencoders, is when there are more nodes in the hidden layer, we are facing the problem that the autoencoder could be learning the so-called identity function, also known as null function, which means that the output is the same as the input, and it makes the autoencoder useless.[17]

Autoencoders are similar to dimensionality reduction techniques like Principal Component Analysis (PCA). They project the data from a higher dimension to a lower dimension using linear transformation and try to preserve the important features of the data while removing the others. The biggest difference between autoencoders and PCA is shown in the transformation part, PCA uses linear transformations and autoencoders use non-linear transformations.[17]

The most important properties an autoencoder should have are:

- Must be sensitive to inputs, to accurately reconstruct them.
- Need to be able to generalize well, even when are evaluating unseen data.

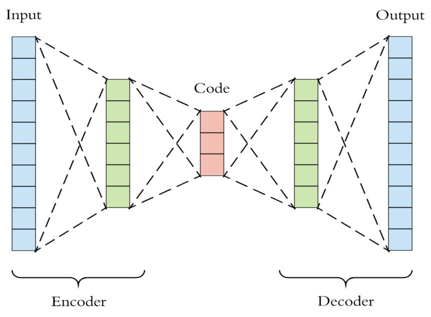

Figure 2.6: General architecture of an autoencoder[9]

As shown in the image above, the autoencoder can be divided into two parts: encoder and decoder.

- **Encoder**: Here the network compresses the input into a fewer number of bits. The space represented by these fewer number of bits is often called the latent-space or bottleneck. The bottleneck is also called the "maximum point of compression" since at this point the input is compressed to the maximum. These compressed bits that represent the original input are together called an encoding of the input. The encoder can be represented by a encoding function $h = f(x)$.[17]
- **Decoder**: Here the network tries to reconstruct the input using only the coding of the input. When the decoder can reconstruct the input exactly as it was given to the encoder, it is safe to say that the encoder can produce the best coding for the input with which the decoder can reconstruct well. The decoder can be represented by a decoded function $r = g(h)$.[17]

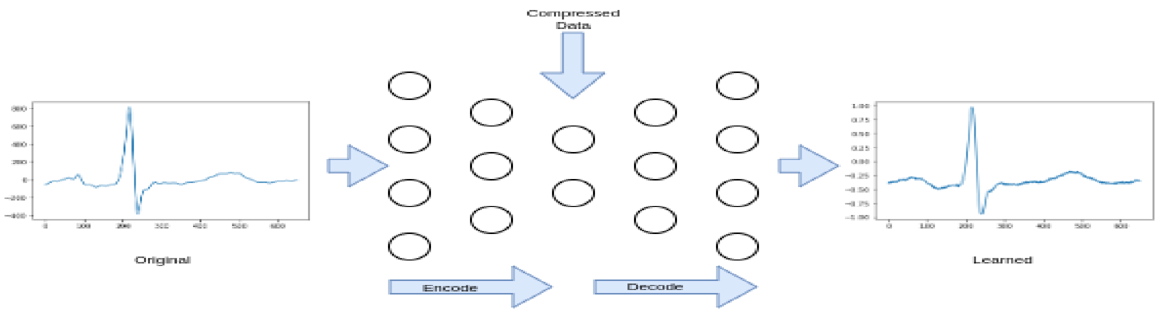

Figure 2.7: Representation of the different parts that make an autoencoder

The autoencoder attempts to learn an approximation to the identity function, to generate y similar to x. As we can see in the image above, the learn signal is a very close representation of the original input, with a little bit of excess noise.

To give a better explanation of the process of an autoencoder, let's see the example with the mnist database. The mnist database uses a large database of handwritten digits that is commonly used for training various image processing systems. If we assume that the x inputs are the pixel intensity values of a 10x10 image (100 pixels), so n = 100 and there are s = 50 hidden units in the L2 layer. Note that we also have $y \in R100$. Since there are only 50 hidden units, the network is forced to learn a "compressed" representation of the input, that is, given only the vector of activation of hidden units, it should try to "reconstruct" the input of 100 pixels x. If the input was completely random, the task of compression would be very difficult, but if there is a structure in the data, for example, if some of the input resources are correlated, this algorithm can discover them. This simple autoencoder ends up learning a low-dimensional representation very similar to PCA (principal component analysis).[17]

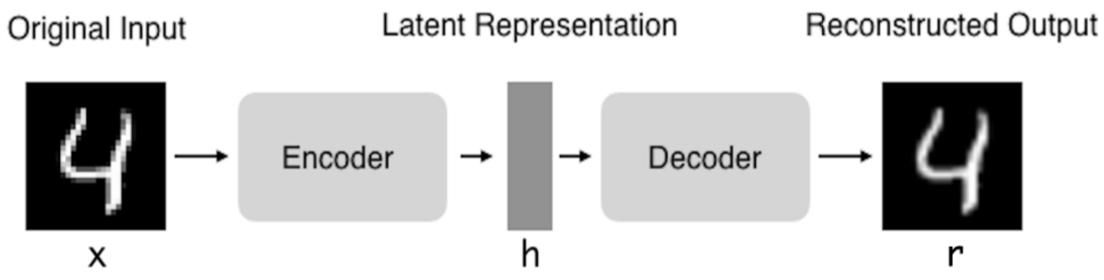

Figure 2.8: Example of the autoencoder with the mnist database[17]

If the purpose of the autoencoders was to copy the input to the output, they would be useless. We hope that by training them, the latent representation h will have useful properties. This can be achieved by adding restrictions on the copy's task. One way to get useful features is to restrict h to dimensions less than x, in this case, the autoencoder is called under complete. When training an incomplete representation, we force the autoencoder to learn the most important features of the training data. If we give a lot of capacity, it can learn to perform the copy task without extracting any useful information about the data distribution. This can also occur if the dimension of the latent representation is the same as the input and, in this case, the autoencoder is overcomplete, where the dimension of the latent representation is greater than the input. In such cases, even a linear encoder and a linear decoder can learn to copy input to output without learning anything useful about data distribution. Ideally, someone could successfully train any autoencoder architecture, choosing the size of the code and the capacity of the encoder and decoder based on the complexity of the distribution to be modeled .[17]

As we mention, autoencoders can be under complete or overcomplete. In the first one, the hidden layer h has a lower dimension than the input and in the second one, the hidden layer h has higher dimensions than the input. Beyond these two types of autoencoders, there are others, that we may be familiarized with to expand our knowledge in the matter.

- **Under complete autoencoder**: an autoencoder is under complete when it's code dimension is less than the input dimension. One of the advantages of being under complete is that when the learning task is being performed, it will force the autoencoder to extract the most important properties from the input data. This learning process can be described through the minimisation of the loss function: $L(x, f(g(x)))$, where L is a loss function penalising $g(f(x))$ for being dissimilar from x, such as the mean squared error. When the decoder is linear and L is the mean squared error, the under complete autoencoder learns to span the same subspace as PCA. [15]. This kind of autoencoder can be used in, dimensionality reduction, visualization, feature extraction, and how to learn binary codes.

- **Overcomplete autoencoder**: an autoencoder is overcomplete when h, is higher than x. Normally, this may cause it to trivially learn to copy input to output. The loss function has two parts where the first one is the loss function, it consists in the difference between the input and output data and the second part, is used as a regularization term which prevents the autoencoder from overfitting, $L(x, g(h)) + regulariser$.

- **Sparse autoencoder**: a sparse autoencoder differ from a normal autoencoder in the training criteria, it uses a sparsity penalty. In most cases, the loss function is obtained by penalizing the activation of hidden layers, so that, only a few nodes will be activated when a sample is fed to the network. This method, guarantee that the autoencoder is learning latent representations rather than, redundant information given by the input data. There are two different ways to construct sparsity penalty, L1 regularization, and KL-divergence.[24]

- **Denoising autoencoder**: denoising autoencoders are used for feature selection and extraction, the way they work, is by randomly turning some of the input values to zero. The number of nodes that have their values changed is dependent on the quantity of data that is fed to the network. When the loss function is being calculated, it is important to compare the output with the input and not with the altered one.

- **Contractive autoencoder**: contractive autoencoder is an unsupervised deep learning technique that helps the neural network encode unlabelled data. This kind of autoencoders makes the encoding less sensitive to small variations in its training dataset. This works by adding a regulariser, to the cost or objective function we are trying to minimize.

- **Variational autoencoder**: uses the variational approach for learning latent representation, which creates an additional loss component and a new training algorithm called Stochastic Gradient Variational Bayes.

- **Convolutional autoencoders**: Unlike a simple autoencoder, these approach the filtering task differently. Instead of choosing the filters, we let the model learn which ones are the best that minimize the error in the reconstruction. After these filters are learned, they can be applied to any input to extract features. They explore the convolutional operator, in a way that the model can learn the input in a smaller and simpler set of signals and which from it will reconstruct the input.

After getting acquainted with several types of autoencoders, we must see what their applications are to understand if it is sensible to use an autoencoder depending on the task we have to accomplish.

Currently, data denoising and dimensionality reduction for data visualization are considered two main interesting practical applications of autoencoders. With appropriate dimensionality and sparse constraints, autoencoders can learn more interesting data projections than PCA or other basic techniques. Autoencoders learn automatically from data examples, this means that it is easy to train specialized instances of the algorithm that will perform well on a specific type of input and that do not require any new resource engineering, only the appropriate training data.

Another application is as a preliminary task to recognize images with CNNs (Convolutional Neural Networks). Looking at the image above, we can observe that the output is the image of the number 4 much smoother, that happens because we removed noise from data and then use the output to train a CNN model. Autoencoders are trained to preserve as much information as possible when input is passed through the encoder and then through the decoder.

2.3.2 Convolution neural network (CNN)

After studying what autoencoders are, we must also understand what convolutional neural networks (CNN) are, to understand which of these tools will be most suitable to carry out our conversion.

A CNN is a class of artificial neural network of the feed-forward type, which has been successfully applied in the processing and analysis of digital images. A CNN uses a variation of multi-layer perceptrons designed to demand the least possible pre-processing. These networks are also known as shift invariant or space invariant artificial neural networks[33].

A CNN tends to demand a minimum level of pre-processing when compared to other image classification algorithms. This means that the network learns the filters that in a traditional algorithm would need to be implemented manually. This independence of prior knowledge and human effort in the development of its basic functionalities can be considered the greatest advantage of its application. This type of network is used mainly in image recognition and video processing, although it has already been successfully applied in experiments involving voice processing and natural language.

To sum up, we learn that the autoencoder receives an input and reproduces it as closely as possible. In theory this seems pretty simple, but normally an autoencoder have more than just an input and an output layer, without them its purpose would be useless, because it would just copy the input to the output. So between those layers the autoencoder has hidden layers, that will be becoming smaller, in terms of the number of nodes, until reach the smaller size possible, called bottleneck, in which, if the number of nodes is less than the number of pixels, the network has to compress the data (get rid of the meaningless features), happening the same way in reverse, if the number of nodes is greater than the number of pixels, the network has to decompress the data.

On the other hand, a CNN is design for data with spatial structure, like images, they are composed of many filters, that convolve across the data producing an activation at every position in the region. This activation creates a feature map, that consists in the representation of how much data is activate in that region according to what we are trying to identify or classify with the CNN. They are space invariant, which means, that the image distribution does not depend on the specific location of the object in the input image.

2.3.3 Keras and Tensorflow

Autoencoders use libraries such as keras and tensorflow, where they get the necessary tools to train the network. With that being said, keras is a python library applied to neural networks. It is capable of running on top of tensorflow and was created to quickly train networks. It is user-friendly, modular and extensible. Keras contains many implementations such as activation functions, layers and optimizers and has tools that make the creation of the codings of the different inputs (images or data in text) simpler.[7]

Like keras, tensorflow is a machine learning library that is mainly used to detect and decipher patterns. Tensorflow creates and trains machine learning models using high-level API like keras. It is easy to train and has a simple and flexible architecture.[2]

2.3.4 Optimizers and Loss Functions

One of the goals of machine learning is to have the smallest possible difference between the predicted output and the true output. This difference is called the loss function. To reduce this loss, we have to find an optimisation for the different weights. Combining all this, we guarantee that we can make a good forecast of our results. To achieve this goal, we run several iterations for the different weights, which will help to determine the minimum cost. Being this whole method known for the gradient descendent.

Gradient descent is an iterative optimisation algorithm that reduces the cost of the function, thus helping models to make more favourable predictions. The way this algorithm works is, since the gradient shows the direction of the increase, to find the minimum cost, we have to consult the gradient in the opposite direction, updating the values to minimize the loss. $\theta = \theta - \eta \bigtriangledown j(\theta; x, y)$, where $\theta \rightarrow$ weight, $\eta \rightarrow$ learning rate and $\bigtriangledown j(\theta; x, y) \rightarrow$ gradient of weight θ. Types of gradient:

- **Batch Gradient Descent**
- **Stochastic Gradient Descent**
- **Mini batch gradient descent**

As we have already noticed, an important part of decreasing the loss function, relies in the optimisation and for this it is important to understand the role of optimizers. Their main task is to update the weight of the parameters to minimize the loss function.

This function works as a "pilot" for the optimizer, that is, informs it so that, it can see if it is going in the right direction to reach the global minimum.

Types of optimizers[7]:

- **Adagrad (Adaptive gradient algorithm)**: An adaptive optimizer, since it has specific parameters, which are adaptable according to the frequency with which they are updated during training. The more updates you receive, the smaller the changes.

- **Adadelta**: Is a robust extension of the previous one. Adapts the learning rates, based on the gradients updates, instead of accumulating them. In this way it continues to learn regardless of the amount of updates that have been made.

- **RMSprop**: The key points of this optimizer are, to keep the gradients average and divide the gradient by the root of its average. It uses plain momentum, the central version maintains the average of the gradients and uses that average to estimate the variation.

- **Adam**: This optimizer has a stochastic gradient, which is based on the adaptive estimation of the first and second order moments. It radically reduces Adagrad learning rates. It can be seen as a combination of Adagrad and RMSprop, since it works well with sparse gradients like Adagrad, and it also works well with nonstagionary settings like RMSprop. Implements the exponential average of the gradients, so that in this way to scale the learning rates instead of making the average as in Adagrad. It keeps an exponentially decaying average of past gradients. It is computationally efficient and requires little memory. How it works:

  1. Update the exponential mean of the gradients (m_t), and the square of the gradients (v_t) which is the estimate of the first and second moment.
  2. Parameters $\beta 1, \beta 2 \in [0,1]$ control the decay rate of the averages.
  3. $m_t = \beta 1 m_{t-1} + (1 - \beta 1) g_t$, $v_t = \beta 2 v_{t-1} + (1 - \beta 2) g_t^2$

- **Nadam**: Like Adam is essentially RMSprop with momentum. It is used with gradients that have noise or that have high curvatures. The learning process is accelerated by the sum of the exponential decay of the averages of the current and previous gradients.

The loss function is an important component of neural networks. The loss is a prediction of the error that the network has, and the function that calculates this value is called the loss function. As already explained, what we know is that the loss is used to calculate the gradients and these are used to update the weights of the network. Very simply, this is what we need to train a network. As we are using Keras and TensorFlow and they offer several loss functions depending on our goals and before explaining the function we chose for our autoencoder, here is a brief explanation of the loss functions[7]:

- **Mean Squared Error (MSE)**: Is indicated for when it is a regression problem. This loss is the mean of squared differences between the target and the predicted values.
- **Binary Crossentropy (BCE)**: Is used for binary classification problems. When we use this function, we only need an output node that classifies the data into two classes. The output values vary between 0 and 1.
- **Categorical Crossentropy (CCE)**: Is used when we have a multi-class classification. To use this function, it is necessary to have the same number of output nodes and classes.

Once the key terms, methods and techniques for implementing the algorithm we intend to create are known, we begin the research on the work that has already been done in this area, to have a clearer idea of how we will approach the problem, focusing on this in the next chapter.

Chapter 3

Related Work

With the reading of the previous chapter, we are left with the perception of all the tools and techniques that we will use for our implementation, this chapter now serves to explain some ideas of some articles that we had as a basis for our development. With our analysis of them, we were able to get a clearer idea of how to tackle the problem and start to hope that it is possible to achieve results with this conversion so that they have the necessary quality to be used.

3.1 Conversion of ECG Leads

Even though the most known practical application of biometrics is recognition and identification, other applications have been growing related to the medical image, that works in helping the diagnose process. With that being said, in this thesis, we are using signal processing and machine learning to try and do the conversion of leads. About this topic, not a lot has been done, the only article that we found that was close to the type of conversion that we wanted to do was: the conversion of an ambulatory ECG to a standard 12 Lead ECG. Using the approach proposed by Zhu et al.[34], to start and try to create an idea of what is the best way to tackle this conversion, we decided it would be beneficial to explain all their process. Before we can explain the process we needed to be up to date with some definitions, like an ambulatory ECG, (is also called ambulatory electrocardiography, ambulatory EKG or a Holter monitor, is a battery-operated, portable device that measures and tape-records the heart's electrical activity continuously, usually for a period of 24 to 48 hours so that any irregular heart activity can be correlated with a person's activity. The device uses electrodes or small conducting patches placed on the chest and attached to a small recording monitor that is carried in a pocket or a small pouch worn around the neck), and standard 12-lead (is a representation of the heart's electrical activity recorded from electrodes on the body surface), to know their similarities and differences to understand how the conversion is going to take place.

The process followed in the article was[34]:

1. They collected different ECG recordings, using a 12-lead recorder. These readings originated from a resting state. To simulate ambulatory readings, unlike the previous ones, these were done with the volunteers seated. For both, the electrodes were in the same position.

2. As a basis for the creation of this algorithm, they followed the lead theory of electrocardiogram, which shows the relation between 12-lead ECG and the electric current generator is given by:

    - $V_{ambulatory} = M_{ambulatory} * p$ and $V_{std} = M_{std} * p$, p is the vector of the electric current dipole sources. $V_{ambulatory}$ and V_{std} are the vectors of the ambulatory ECG. $M_{ambulatory}$ and M_{std} are the lead matrices.

3. For evaluation and using the previous equations, they deduced that the 12-lead ECG can be calculated by the equation $V_{std.derived} = \beta_{avg*} V_{ambulatory}$ and made this assessment for the different waves (P, Q, QRS, ST, S, T)

Unlike the work of Zhu et al., in this dissertation we aim to achieve is not the conversion from one system to another (ambulatory ECG $\rightarrow$ standard 12-Lead ECG), but rather, converting the views we get when we do an ECG, in others. To allow the diagnosis to be made when we have fewer data. For this, we use autoencoders as a base tool for our algorithm.

An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. An autoencoder aims to learn a representation (coding) for a set of data, usually for dimensionality reduction, by training the network to ignore the background noise of the signal. Alongside the reduction side, it has a reconstructing side (decoder), where the autoencoder tries to generate as close as possible a representation from the input data, hence the name. For the basic model there are several variants, intending to force the learned representations to acquire useful properties, for example, autoencoders sparse and denoising. Autoencoders are used to solve problems related to face recognition.

In medicine, the definition of an ECG is a simple, non-invasive procedure. Electrodes are placed on the skin of the chest and connected in a specific order to a machine that, when turned on, measures electrical activity all over the heart.

The algorithm we are creating in this thesis, in the medicine field aims to change the perspective of the signal to show certain medical conditions. Heart disease refers to conditions that involve the heart, its vessels, muscles, valves or internal electric pathways responsible for muscular contraction.

As we already mention, there are three major types of ECG:

- **Resting ECG** - The patient is lying down for this type of ECG. No movement is allowed during the test, as electrical impulses generated by other muscles may interfere with those generated by your heart. This type of ECG usually takes 5 to 10 minutes.
- **Ambulatory ECG** - if the patient has an ambulatory or Holter ECG he wears a portable recording device for at least 24 hours. He is free to move around normally while the monitor is attached. This type of ECG is used for people whose symptoms are intermittent (stop-start) and may not show up on a resting ECG, and for people recovering from heart attack to ensure that their heart is function properly. The patient records his symptoms in a diary, and note when they occur so that his own experience can be compared with the ECG.
- **Cardiac stress test** - This test is used to record your ECG while you ride on an exercise bike or walk on a treadmill. This type of ECG takes about 15 to 30 minutes to complete.

3.2 Conversion of ECG Graph into Digital Format

As we have seen, the ECG consists of a set of waves that are repeated over time.[27] These are created by electrical impulses that are captured by the electrodes and recorded in a graph. The process of digitizing the ECGs is done, mainly to get rid of the background noise that the signal has.[32]

To understand the conversion method done by Virgin et al.[32], it describes one of the possible ways to do it following the pipeline:

1. **ECG graph** - Recording the ECG.

2. **Scanning** - The signals are scanned and saved as JPEG images.
3. **Crop Region of interesting** - Choose the region of the ECG, that has the greatest amount of information.
4. **Image binarization** - It transforms the image into a binary representation, reducing the number of colors to a binary level. This is an important process for the transformation of a biomedical map to a digital signal.
5. **Gradient based feature extraction** - Creating a vector that stores the magnitude and the gradient.
6. **Noise rejection** - Removes background noise from the image.
7. **Image thinning** - Eliminates signal redundancy.
8. **Edge detection** - Detects the edges of the signal in the image to separate the signal from the grid present on the ECG paper. The edges are found due to the difference in brightness between the pixels.
9. **Pixel to vector conversion** - The algorithm passes through the image and extracts a vector (1D). It starts at the beginning of the image until a point that represents the first stretch, after that for the second stretch, and so on until the last one that will coincide with the end of the image.
10. **Digitised data** - Creates a file with the exact values of the pixels present on the ECG.

3.3 Deepfakes

One of the applications of the autoencoders is the deepfakes, so in this section, we are going to see how does this process work.

The word deepfake was born from the junction of deep learning with fake. Deepfake is a technique based on the following, having as input a dataset of images (static or video format) takes a person present in those images and changes their appearance to someone else on an existing piece of media, or something simpler like changing the layout of the lips, using machine learning techniques, such as autoencoders and Generative adversarial networks (GANs)[25]. Deepfakes appeared for the first time when three researchers launched a program called "Video Rewrite Program" in 1997, this program edited videos of people speaking with a text other than the original, compellingly.

The process of creating a deepfake happens as follows: An image is fed to the encoder, and is compressed, having as result the image represented by a lower-dimensional structure, that normally is called a base vector or a latent face. Depending on the architecture of the network we are working with, the latent face might not look like a face. After, when this latent face passes through the decoder, the face is reconstructed. In general, an autoencoder have loss functions, which means that the reconstructed face will not have the same level of detail that was present in the original image.[35]

In the point of view of the programmer, we can say that they have full control over the network, when it comes to the number of layers, how many nodes they have and how they are connected, being this the most important part because it is where all the knowledge of the network is stored. Each edge has a weight, and find the set of weights that makes the autoencoder work right is a long process. When we talk about training a network, that means, we are optimizing the weights to reach a specific goal. In case of a traditional autoencoder, the performance of the network is shown by the accuracy given in the reconstruction of the original image from its latent representation.[35]

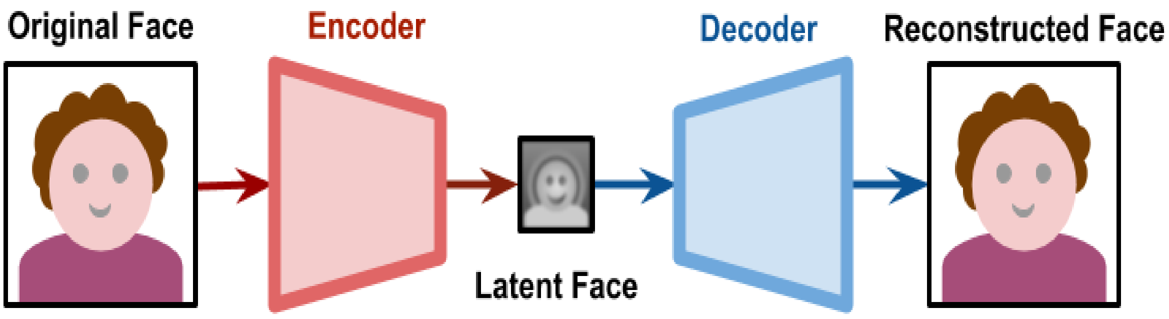

Figure 3.1: Representation of the phases of the autoencoder[35]

One thing that we should never forget, is the fact that, if we train two autoencoders separately, they are going to be incompatible with each other, because the latent face obtained in each one, will be based in specific features that each network " marked" as meaningful during the training process, but if two autoencoders are separately trained on different faces, their latent spaces will represent different features. The thing that makes the swapping process possible is to find a way, where we force the two latent faces to be coded with the same features. We can solve this just by sharing the same encoder for each network, and different decoders.[35]

All the latent faces are generated by the same encoder, which means that the encoder needs to be able to identify the features that are shared by both images. Once all the faces share a similar structure, it is not expected that the encoder learn the concept of the face itself.[35]

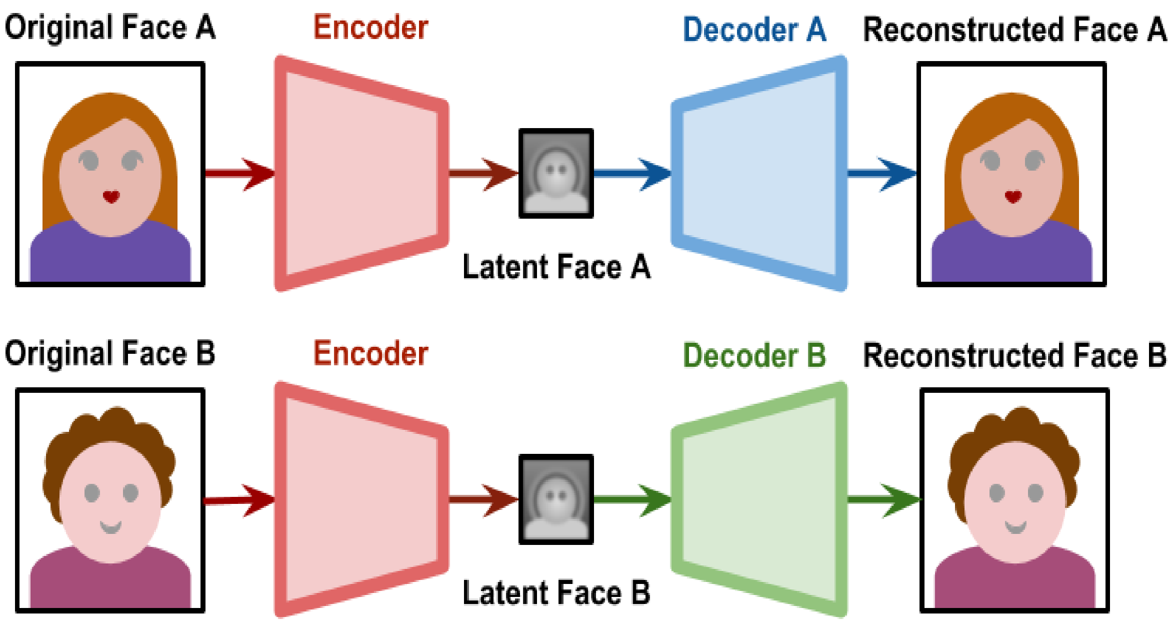

Figure 3.2: Representation of two different autoencoders that will focus on different features[35]

When all the training process is finished, we are going to start generating the deepfakes. if the network learns well what makes a face, the latent space will represent facial expressions and orientations.

To sum up, to have a good deepfake, we need to make sure that the two subjects that are fed as input, share as many similarities as possible, and in this way, we can be sure that the encoder that is shared between them can generalize the features. The image below represents the way a deepfake is formed:

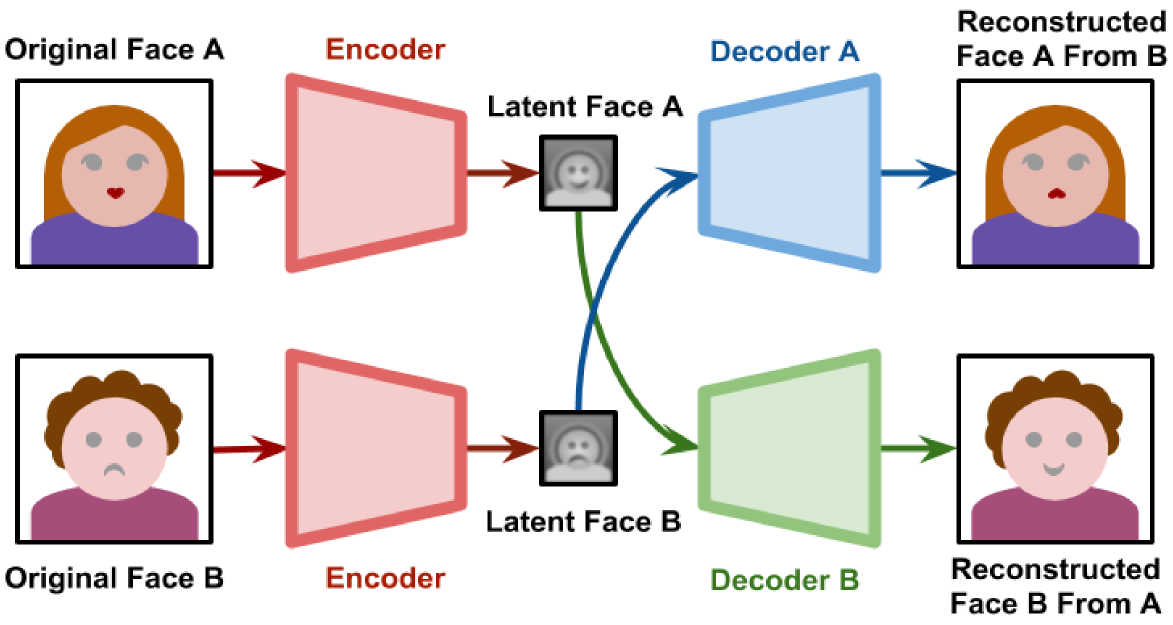

Figure 3.3: Autoencoder deepfake creation scheme[35]

When we look at this process of creating deepfakes, we can see how it relates to what we are developing.

First, we can immediately see that they use autoencoders and what differs from our work is that they receive images instead of signals. In the first phase, they have an autoencoder that receives an image and tries to reconstruct it, following this idea, that's how we started the development of our algorithm.

Second, they advance to the conversion part and for that, they create two autoencoders so that each one of them "learns" the most important features and change the decoders so that the image is converted. Again, and correspondingly this is how we did our conversions and in this way for the development of our conversion algorithm, we use some of the ideas present in these examples, which we will be described in more detail in the next chapter.

Chapter 4

Methodology

This chapter is about the methodology of the thesis. In more detail, we are explaining the research approach, how we did the data collection, selection of samples, type of data, and the limitations of the project.

As already mentioned, this thesis aims at the creation of an algorithm to convert, electrocardiogram (ECG) signals from certain Leads into one target Lead. To achieve this goal, we purpose an algorithm to convert Leads with the help of deep learning techniques, such us, autoencoders.

4.1 Materials

The dataset used was PTB[5] and was obtained through physionet[14] (the research resource for complex physiologic signals) is a collection of ECG and has the following specifications:

- 16 input channels, (14 for ECGs, 1 for respiration, 1 for line voltage).
- Input voltage: ±16 mV, compensated offset voltage up to ±300 mV.
- Input resistance: 100Ω(DC).
- Resolution: 16 bit with 0.5 $\mu V/LSB$(2000 A/D units per mV).
- Bandwith: 0-1 kHz (synchronous sampling of all channels).
- Noise voltage: max. 10 μV(pp), respectively $3\mu V$(RMS) with input short circuit.

- Online recording of skin resistance.
- Noise level recording during signal collection.

The database contains 549 records from 290 subjects (aged 17 to 87, mean 57.2; 209 men, mean age 55.5, and 81 women, mean age 61.6; ages were not recorded for 1 female and 14 male subjects). Each subject is represented by one to five records. There are no subjects numbered 124, 132, 134, or 161. Each record includes 15 simultaneously measured signals: the conventional 12 Leads (i, ii, iii, avr, avl, avf, v1, v2, v3, v4, v5, v6) together with the 3 Frank Lead ECGs (vx, vy, vz). Each signal digitized at 1000 samples per second, with a 16-bit resolution over a range of $\pm 16.384mV$. On special request to the contributors of the database, recordings may be available at sampling rates up to 10 kHz.[5]

Within the header (.hea) file of most of these ECGs. The record is a detailed clinical summary, including age, gender, diagnosis, and where applicable, data on medical history, medication and interventions, coronary artery pathology, ventriculography, echocardiography, and hemodynamics. The clinical summary is not available for 22 subjects.[5]

As we mentioned before, the PTB database was retrieved from the physionet, that it is a repository that stores medical data. It was created to catalog biomedical research and offers access to large collections of data for free.[14]

4.2 Pre-processing

We chose this database because at the time it fills all the requisites we needed to start testing the autoencoder, the signals did not need a lot of modifications to be trained by the autoencoder.

Since we are working with fully-connected autoencoders and with convolutional autoencoders, we got two different ways of pre-processing the same data.

4.3 Fully connected autoencoder

Starting with the fully-connected methodology, from the signal in the database, we only changed the way the signals were shown, that is, we got rid of background noise, we also centered all signals in R to get them all normalized to be used later in the process. The approach we used for this task was applying a low-pass filter (a low-pass filter is used to remove the higher frequencies in a signal of data) which receives four arguments data (signals), cutoff (40), fs (1000) and order (5), combined with one high-pass filter (a high-pass filter is used to remove the higher frequencies in a signal of data), which receives four arguments data (signals), cutoff (1), fs (1000) and order (5), to get the entire signal clear, so it is easier for the algorithm to identify the R-peaks that will be used to center the signal. After being cleared, we used the Pan Tompkins algorithm to detect the R-peaks and stored all of them in an array (this algorithm detects fiducial points and is one of the most used. The way it processes is: it starts by reducing noise using filters and uses a derivative to extract information from the QRS complex. With the help of a quadratic function and a window that moves along the signal, it highlights the high frequencies and with an adaptive threshold it can detect the R peaks[27]), and this we used the implementation that was present in the package py-ecg-detectors[28], so when it is time to cut the signal we can easily find the place where to center the new window, that is going to be 0.25s earlier and 0.4s after the R-peak.

Once all signals, were centered, filtered and with the same size, we created a pickle file, to store all of them separating the data in two arrays, train and test, so when we moved to the autoencoder we only need to load the file instead of preform the task above explained every time.

4.4 Convolutional autoencoder

For the pre-processing of the convolutional autoencoder, it was much simpler, from the process described before the only thing we did was cutting the signal, but even that was easy just because for this one with did not need to find the R-peaks of the signal, we just cut the signals second by second. The way with did this was by creating a function (cut signal) that received 3 parameters the signal, and the measures we wanted for the new window. After cutting all the signals we just needed to reshape it in order to add the channel, so in this way, the autoencoder could train it.

Since we are organizing the data so that it can be trained by the autoencoder, it is necessary to divide it into two groups, training and testing, for that, we took the 549 readings and divide them so that we have 349 in the training group and 200 in the test group. After doing this division, we apply the transformations described above, both for one and for the other.

4.5 Autoencoder

We moved on to create the autoencoder, that will perform the conversion. With the development of our search and implementation, we created four autoencoders. We started with the fully-connected than transform it so that it would give multiple outputs, this idea was done, so it would bring some improvement to what we already had. After having both of them working, and because we were always looking for ways to improve our results, it arose the idea of transform both of them into convolutional ones. In this section, we will be describing the way we did it.

4.5.1 Fully-connected

As was mention before, the first autoencoder we implemented, was the fully-connected one, which consisted of 9 Dense layers on the encoder part, 1 for the bottleneck, and 4 for the encode and decode part.

This autoencoder started to be just a “prediction machine”. This being, its main task was just to try and predict the Lead that was fed to the autoencoder as input. In this way, we had a better understanding of what the autoencoder was learning for each Lead. So in this part of training what we did was just changing the parameters, such as, number of the epochs, learning rates, optimizers, among others to seek the best combination.

When we felt we found a good combination we started to transform this “prediction machine” into a conversion one. For this to happen, in the previous step, when we were training the prediction, we end up with an autoencoder for each Lead we trained, so what we did was change the decoders, the same way we learn when we read about the deepfakes in the previous section, and start to give one Lead and try to convert into other.

The data fed to the autoencoder was a signal that was treated according to the pre-processing described above.

When the results were reflecting what we predict it should happen we move on to transform in a multi-output autoencoder.

4.5.2 Fully connected multi-output

We decided to improve our autoencoder so that instead of passing one Lead and converting to another, we pass one Lead and try to convert into several, this way we are with a more efficient, faster process and mainly one autoencoder will be sufficient to get all the views of the different Leads.

For the creation of this autoencoder, we took what we had already implemented and the change we made was, instead of just having a decoder, we now have one per Lead. To make what we are saying more visible we can observe our new autoencoder in the following diagram:

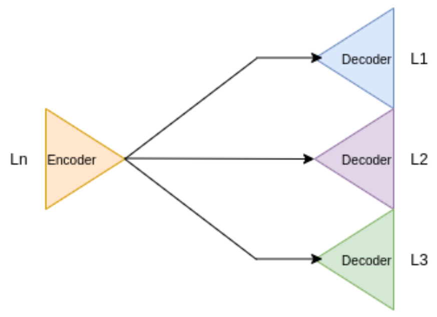

Figure 4.1: Autoencoder multi-output architecture

As you can see from the diagram, the process that will happen now will be, first, a Lead is fed to the autoencoder and that Lead will be converted into the others, in this case in Lead I, II, and III. To achieve this goal, what we did was to create an encoder that has the function of learning the main features of the Leads that we want to convert in, and then all the information that is gathered in the bottleneck instead of being given to just one decoder, it will be given to 3 and the different decoders will try to reconstruct the signal in each own Lead.

For that, and before we performed the training, it was necessary to reorganize the data, so that the conversion is carried out successfully. So what we did was, take the data and create a training vector (xtrain) that, in this case, has the 3 types of Leads, so in this way, the encoder can learn not only about one Lead but about as many Leads as the number of decoders and we also created a vector for each Lead (lead(n)) that has the same peak in the same position but of another Lead. To clarify what has been said, in practice what we have is the following: the first heartbeat of the xtrain is the one that has the peak R in Lead I in the 331's sample. What will correspond to the 331 centered heartbeat in Lead II, and the heartbeat centered on Lead III and so on. These two will be the first heartbeats of Lead II and Lead III, respectively, and the first heartbeat of xtrain will also be in the first position of Lead I (since it is from Lead I).

The architecture we defined for the multi-output autoencoder was: an input layer, 4 Dense layers on the encoder part, 1 for the bottleneck, and for each decoder we have 4 Dense layers and 1 output layer.

4.5.3 Convolutional autoencoder

After finishing the multi-output autoencoder, with changes in the parameters and the number of layers and seeing that the results were already what we expected, we decided not to let our goals stop there and so explore the creation of a convolutional autoencoder, basing this implementation on a UNet, with the change that instead of using 2D convolutional layers, we opted for 1D.

Before proceeding with the visualization of the results we obtained, let's understand what a UNet is, it is a neural network that has a "U" architecture and as it happens in autoencoders it has two symmetrical parts. The left part, "contracting path", where the convolutional process takes place and a right part, "expansive path", made up of transposed convolutional layers.

The diagram below shows the network's operation in a more intuitive way:

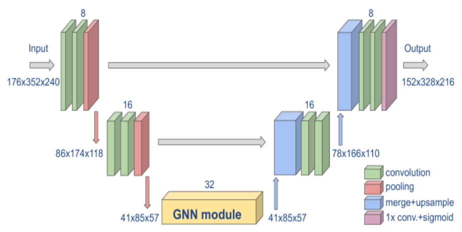

Figure 4.2: Representation of the UNet general architecture[19]

The image above represents the convolution process, where the green layers are the Conv1D and the red one is the max-pooling layer where the sample is downsized, our autoencoder is made by three of this combination of layers. After this process is done and the sample is in its smaller size we start the expansive path, where the sample is going to expand until the same size as the input.

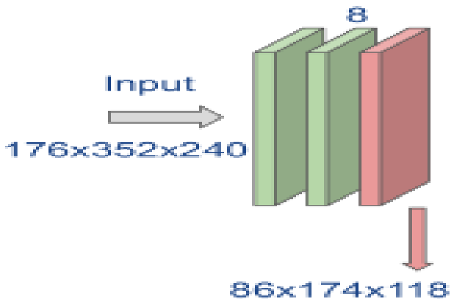

Figure 4.3: Representation of the contracting path in the Unet[19]

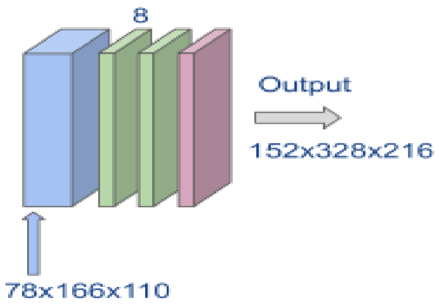

Figure 4.4: Representation of the expansive path in the Unet[19]

The image above represents the deconvolution process, where the blue layer is the up-sampling and the green layers are the Conv1DTranspose, our autoencoder is made by three of this combination of layers. After this process is done and the sample is in its original size the output is formed and the prediction is done.

The architecture we defined for the convolutional autoencoder was: an input layer, 3 times the combination of 2 Conv1D layers and 1 max-pooling, on the encoder part, 2 Conv1D for the bottleneck, and for the decoder we have 3 times the combination of 1 upsampling layer and 2 Conv1DTranspose layers 1 output layer.

4.5.4 Convolutional autoencoder multi-output

As in the fully-connected autoencoders, we also implemented a multi-output one for the convolutional autoencoder. The way we did it was very similar to what happened in the fully-connected, we just add a decoder to each Lead.

In this way, the architecture we defined for the convolutional multi-output autoencoder was: an input layer, 3 times the combination of 2 Conv1D layers and 1 max-pooling, on the encoder part, 2 Conv1D for the bottleneck, and for the decoder, we have 3 times the combination of 1 upsampling layer and 2 Conv1DTranspose layers 1 output layer, and we have a combination of the decoder part for each Lead, having a better understanding when looking at figure 5.5 in the next chapter.

To sum up, this chapter serves to show all types of autoencoders that we implement, the way we did the pre-processing of the data, which database is used so that we can replicate the experience and obtain the same results, to have a better understanding of what happens in practice. Using the next chapter to show and explain the different architectures used for the autoencoders.

Chapter 5

Experimental Methodology

As we mentioned at the end of the last chapter, here we are going to explain in detail how we create our models and how we trained them. As already seen when reading the previous chapters, the goal of this thesis is to create an algorithm, that can perform the conversion of different Leads. To achieve this goal we chose to implement an autoencoder.

5.1 General approach of the autoencoders

Since in this thesis, we were dealing with electrocardiogram (ECG) signals, we downloaded the PTB database, that exists as an open-source in the physionet website. Once the data that we are going to feed to the autoencoder, needs to be normalized, we start by going through each signal and applying a low and high pass filter, to get rid of the background noise, then with the Pan_Tompkins[28] algorithm we found every R-peak in the signal and stored in an array. Once we had all the peaks, we moved on to cutting the signal center in R, to guarantee that all of them had the same size we begun by padding all the signal with zeros, then since we had the same window for each signal we center the R-peak and cut them. To end the data processing, we normalized the signal between[-1,1], and the result was:

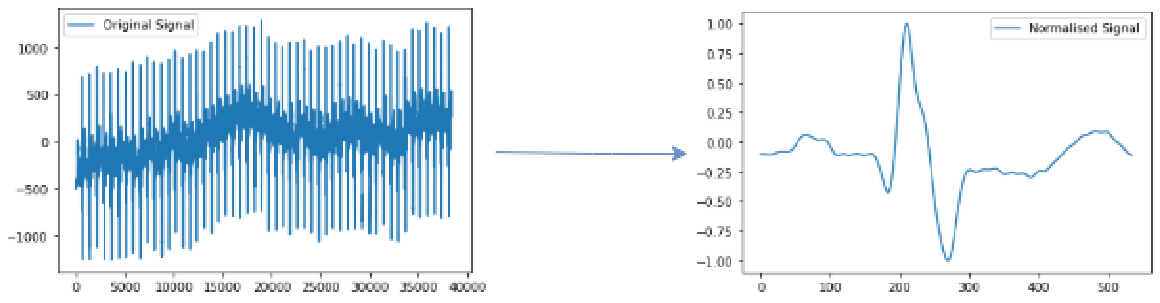

Figure 5.1: Normalisation of the signal

To end the data processing we stored all the signals in a dictionary where the key is the number of the Lead and in this way it is much easier and more intuitive to access the Lead we want in any sample.

With the progress in development, we implemented more than one autoencoder, each one performing the same task, but always with improvements, varying in the type of layers, that is, we started with a fully-connected autoencoder until a convolutional one.

With that being said, we started with the implementation of the autoencoder that has as the main task converting one Lead to another, evolving from this one to one that was able to convert a single Lead into several (for this example we only worked with the first three Leads, which was sufficient to understand that autoencoder's behavior). After having trained and tested the fully-connected autoencoder we advance for the convolutional ones where we implemented the same thing as before, one autoencoder that converts one Lead into another and one that converted it to several.

Below are represented the different diagrams, that show how we arrange all the layers from the input to the output.

Autoencoder 1-1 Lead:

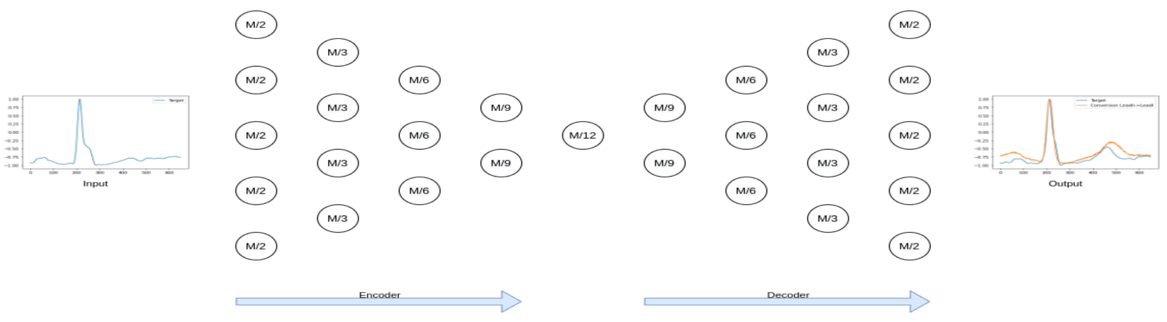

Figure 5.2: Conversion autoencoder architecture

- x $\rightarrow$ Input layer
- b,c,d,e $\rightarrow$ Hidden layers, encoder
- f $\rightarrow$ Bottleneck
- g,h,i,j $\rightarrow$ Hidden layers, decoder
- y $\rightarrow$ Output layer

This first autoencoder, as its name implies, has the function of converting one Lead into another.

In this diagram we can see that it is a fully-connected autoencoder and that it has symmetric encoder and decoder, this symmetry is because the output must be the same size as the input and as in the encoder part this size is being reduced, we have to have a decoder that has the same layers for it to be completely reconstructed. When we look at the layers we see that the input was reduced three times from layer to layer until the bottleneck less in the first one, which was only one and symmetrically in the decoder it grew by three units.

It was trained with the Adam optimizer, a batch_size of 32, 300 epochs, and a validation_split of 0.2. Having these results:

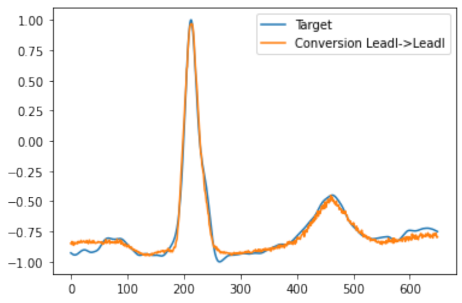

Figure 5.3: Conversion Lead I $\rightarrow$ Lead I

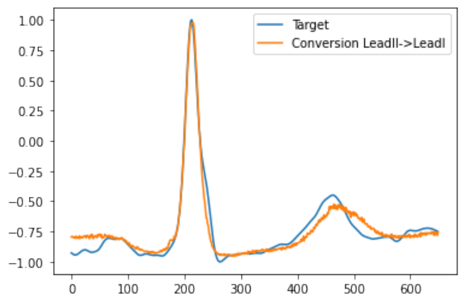

Figure 5.4: Conversion Lead II $\rightarrow$ Lead I

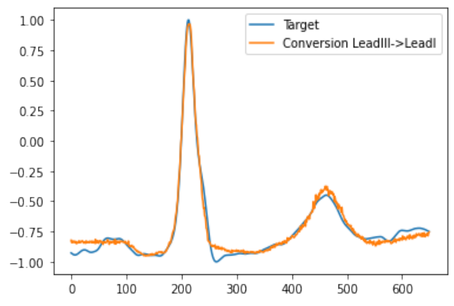

Figure 5.5: Conversion Lead III → Lead I

Autoencoder 1-N Leads:

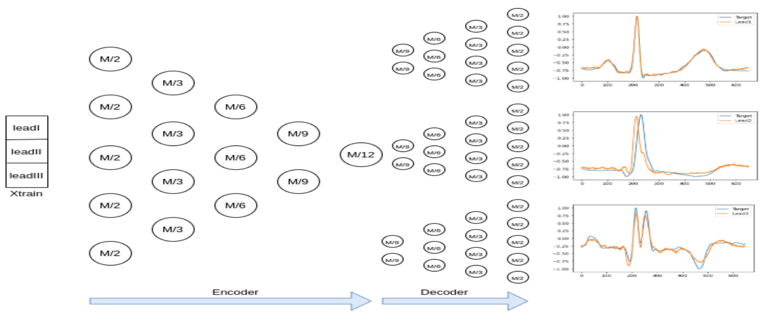

Figure 5.6: Conversion autoencoder multi-output architecture

- x → Input layer
- b,c,d,e → Hidden layers, encoder
- f → Bottleneck
- g1,h1,i1,j1 → Hidden layers, decoder1
- g2,h2,i2,j2 → Hidden layers, decoder2
- g3,h3,i3,j3 → Hidden layers, decoder3
- y1 → Output layer1
- y2 → Output layer2
- y3 → Output layer3

In the same way, as in the previous autoencoder, this is also a fully-connected autoencoder that also has symmetry between the encoder and the decoder. The big difference from this to the previous one is that instead of having one decoder, there are several, in this case, three because we are only converting to the first three Leads, but the idea of this autoencoder is to have as many decoders as the Leads we need to convert.

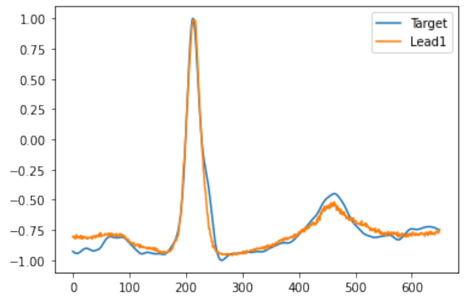

Figure 5.7: Conversion xtrain → Lead I

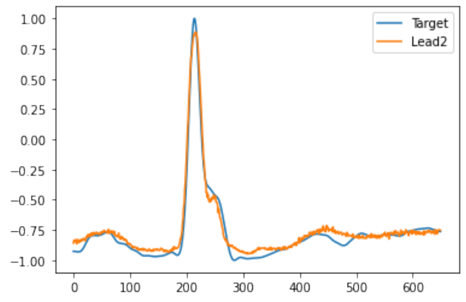

Figure 5.8: Conversion xtrain → Lead II

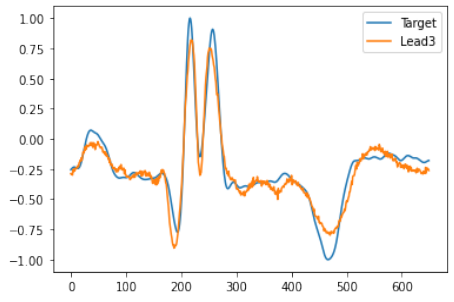

Figure 5.9: Conversion xtrain → Lead III

Convolutional autoencoder 1-1 lead:

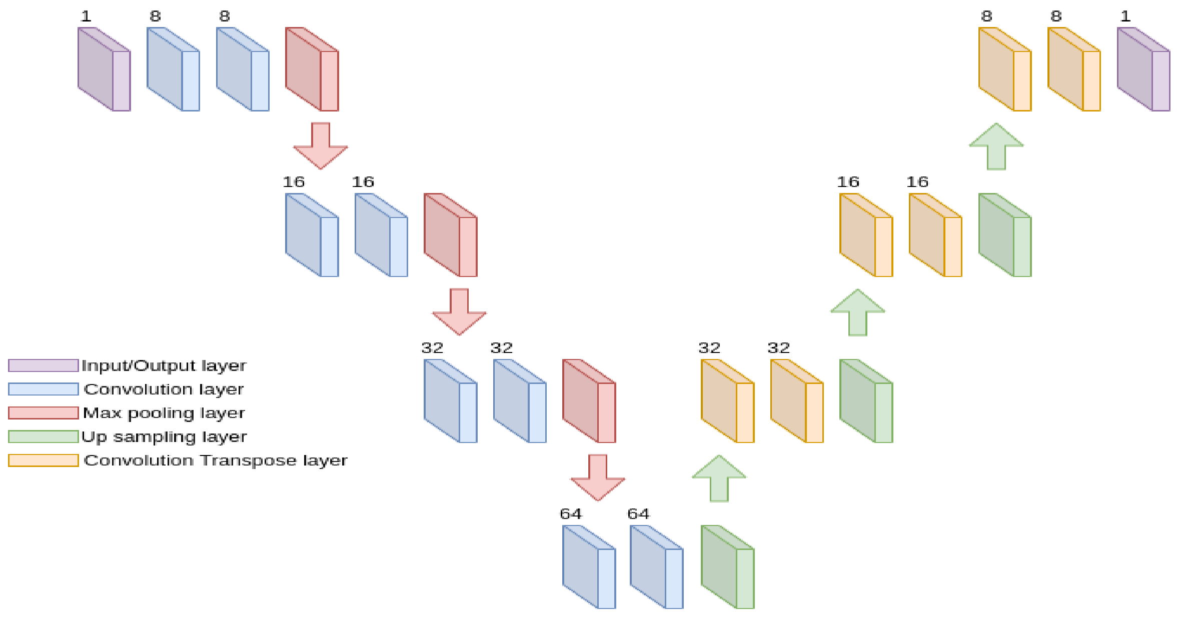

Figure 5.10: Convolution autoencoder architecture

For the convolutional autoencoder, the architecture that we chose was a combination of convolution layers with max-pooling for the contraction part of the network, and a combination of convolution transposed layers with upsampling for the reconstructive side of the network as we can have a clear perception of this network in the following diagram:

- x → Input layer
- a,c,d,e → Convolutional layers
- g,h,i → Convolutional transposed layers
- b → Max pooling layer
- f → Up sampling layer
- y → Output layer

Here we are already dealing with another type of autoencoder, the convolutional one. The main difference of this autoencoder to the other two is the fact that the layers are now convolutional, which means that here the signal will vary in size in the max-pooling layers (in the encoder part) and in the upsampling (in the decoder part), this way we will be able to extract more information about the different waves, to achieve a more complete representation of the converted signals. One of the great advantages that this autoencoder brought, was how much easier it made the process of pre-processing the data since it was not necessary to detect peaks and center the signals, but it was only necessary to cut the signals according to the time we found enough.

It was trained with the Adam optimizer, a batch_size of 32, 500 epochs, and a validation_data = (lead, target) and shuffle=False. Having these results:

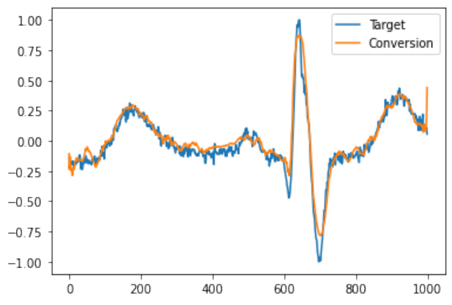

Figure 5.11: Conversion Lead I → Lead I

Convolutional autoencoder 1-N lead:

Following the idea of fully-connected autoencoders, we also created a convolutional autoencoder that converts one Lead into several, and as in the previous example, we also only created three different decoders, but it is also adaptable and we can add a decoder for each Lead that we want to convert.

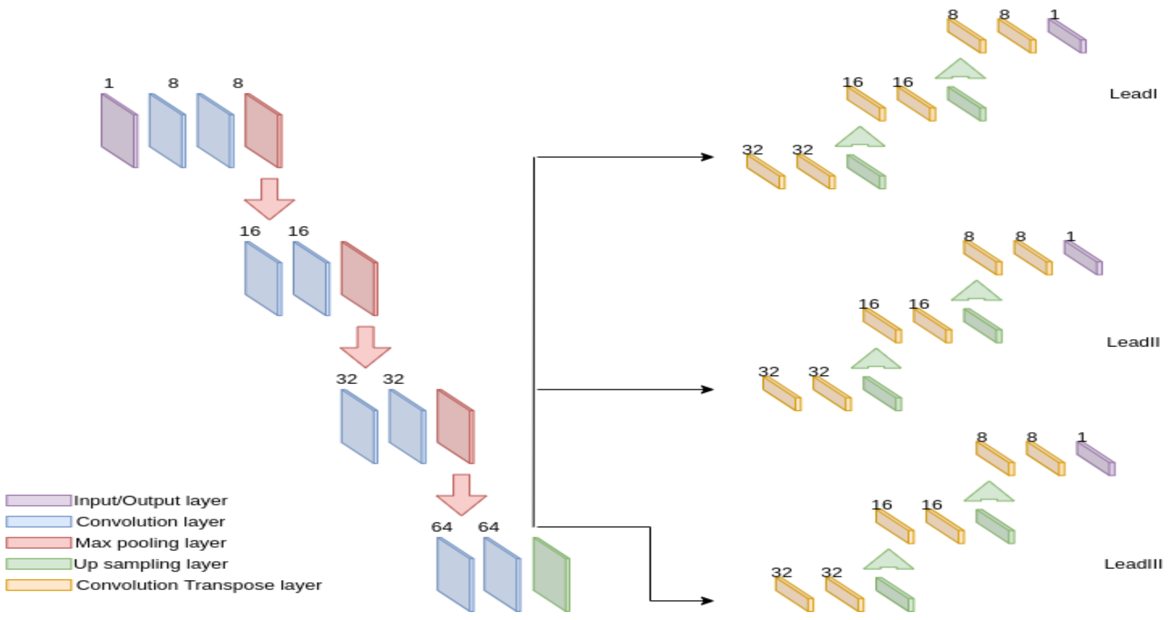

Figure 5.12: Convolution multi-output autoencoder architecture

- x → Input layer
- a,b,d,e,g,h,j,k → Convolution layers
- c,f,i → Max pooling layers
- m1,o1,q1,r1,t1,u1 → Convolutional transposed layers
- m2,o2,q2,r2,t2,u2 → Convolutional transposed layers
- m3,o3,q3,r3,t3,u3 → Convolutional transposed layers
- l1,p1,s1 → Up sampling layers
- l2,p2,s2 → Up sampling layers
- l3,p3,s3 → Up sampling layers
- y1 → Output layer1
- y2 → Output layer2
- y3 → Output layer3

After seeing all the architectures of the different autoencoders and having an idea of how they work, that is, how each layer interacts with the others, we will proceed to the next chapter, where we will analyze all the results obtained with these autoencoders, to have a notion which are the best and worst features of each one, as well as to understand which will be the most suitable to solve our conversion problem.

It was trained with the Adam optimizer, a batch_size of 32, 500 epochs, and a validation_data = (lead, target) and shuffle=False. Having these results:

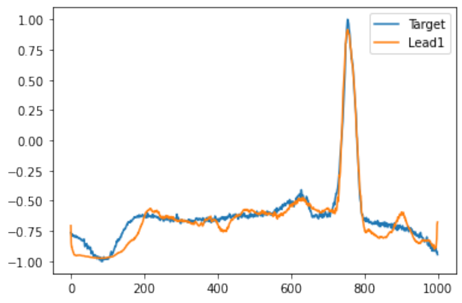

Figure 5.13: Conversion convolutional xtrain $\rightarrow$ Lead I

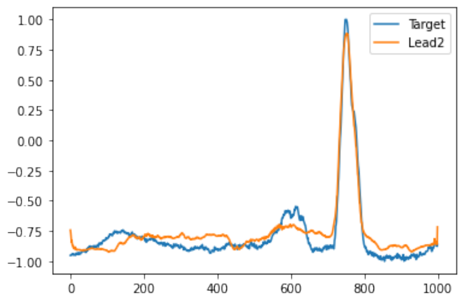

Figure 5.14: Conversion convolutional xtrain $\rightarrow$ Lead II

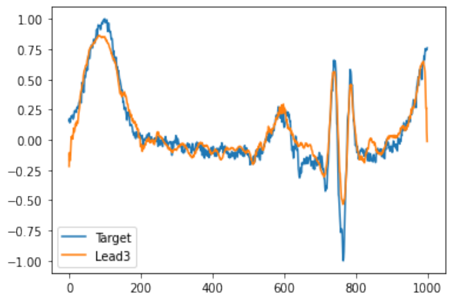

Figure 5.15: Conversion convolutional xtrain $\rightarrow$ Lead III

5.2 Evaluation metrics

In the next chapter, we are going to show and analyze all the results we got from the different autoencoders, in this way, one important value that we should analyze when using autoencoders is its loss value. This value shows us how different the output is from the input, and in this way we can see if the output is reliable or if it is so far off that it should not be trusted.

Since we are using Keras, we have access to functions that give us the loss value and different metrics where we can use to test in our data. With that being said, in our work, we explored three different metrics: mean square error (MSE) that calculates the average of the difference between the points of the original signal and the converted signal, the structural similarity (SSIM), is a metric that calculates the degradation of the signal, which is caused by processing. Thus giving us the perception of the difference between the two signals and the peak signal-to-noise ratio (PSNR) gives us the ratio between the highest peak and the noise that affects its representation.

These loss functions are calculated according to the following equations:

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{n=0}^{N}(y - \hat{y}_i)^2 \tag{5.1}$$

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.2}$$

$$PSNR = 20\log_{10}(\frac{MAX_f}{\sqrt{MSE}}) \tag{5.3}$$

5.3 Our approach to the analysis of the results

To have a more clear idea of how we decide to approach the analysis of our results we need to explain what did we change and how that influenced our work.

Our main approach was the analysis of the importance of the number of layers presented in each autoencoder, so what we did was, for each autoencoder we created 3 more, one with 7 layers, one with 9 and one with 11 for the fully connected, and for the convolutions, since the training process was too long, we decided to just run the test set changing the data.

Then for each autoencoder, we evaluate each metric above explained and compared all the results, relating them with each other, to understand which one is the best.

After explaining all the methods, and techniques used, not only how we design our autoencoders to get the results, but also the metrics and methods we use to test and analyze them, we move on to the next chapter, where we are going to show the results.

Chapter 6

Results

As the name of this chapter indicates, here we will be showing and analyzing the results obtained. In this way, we decided to divide this chapter into two main sections, the qualitative and the quantitative results, where the first one was divided into five subsections, one for each objective outlined during the development of the implementation of the algorithm.

To understand our analysis it is important to clarify the meaning of prediction and conversion in this context. So every time we use the word prediction we are referring to a conversion done from one Lead to the same Lead (e.g. Lead I $\rightarrow$ Lead I), and conversion when we are referring to conversion from one Lead to another (e.g. Lead I $\rightarrow$ Lead II)

6.1 Qualitative Results

For these set results that we are going to show throughout this section, we only consider the first three Leads, since it was the ones we used in all the points above exposed to try to standardize the results and achieve a more intuitive way to relate them. For this experiment, we used the test set of our data, so in this way, we are training the autoencoder with data that it has never seen and thus we can see if it is working correctly.

6.1.1 Prediction of the same Lead

As its name indicates, in this first objective we only made predictions. These predictions allowed us to have a general idea of how the autoencoder would work and also, as a starting point to understand what kind of influence each parameter would have on the final result. The following images show these predictions:

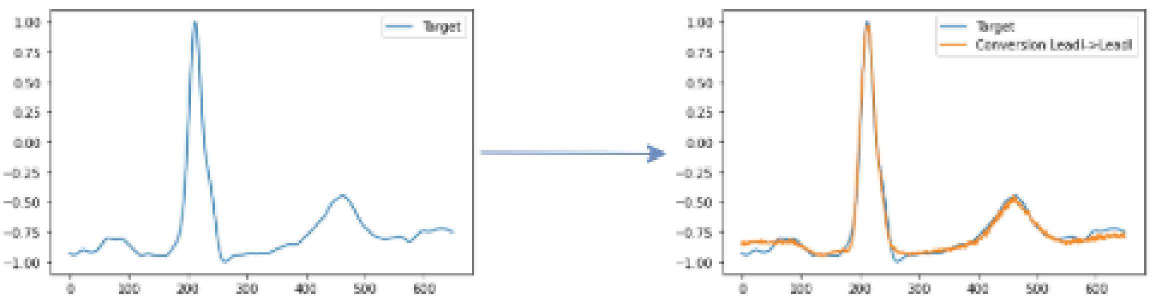

Figure 6.1: Prediction Lead I → Lead I

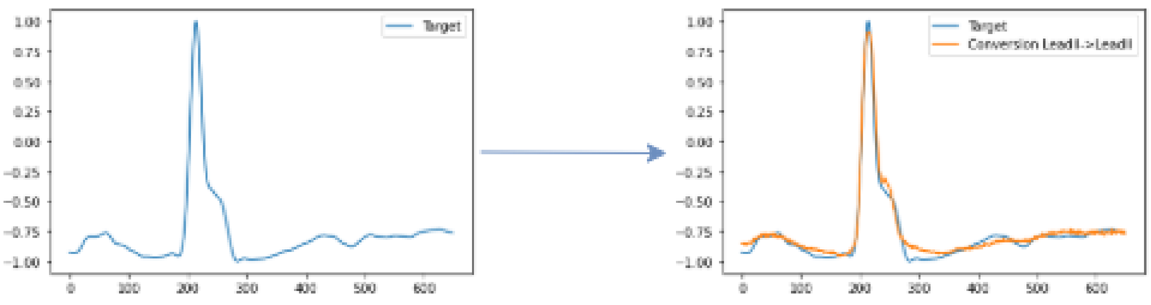

Figure 6.2: Prediction Lead II → Lead II

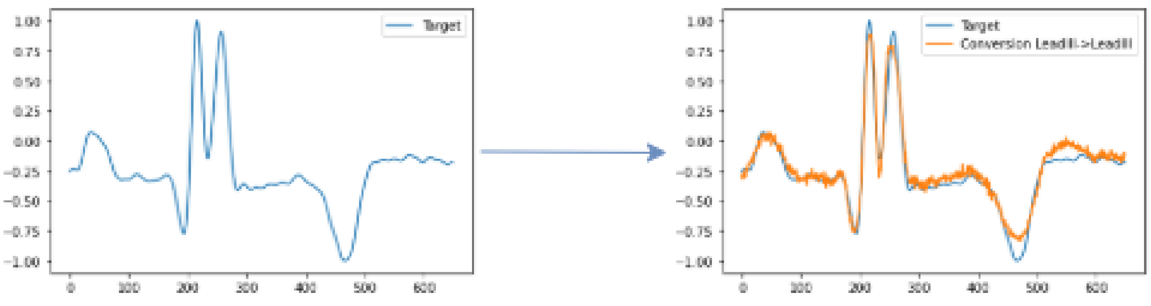

Figure 6.3: Prediction Lead III → Lead III

Image 6.1, 6.2, and 6.3 represent the prediction of Lead I, Lead II, and Lead III, respectively. For every image, the graph on the left (in blue), shows the original lead, and the one on the right is the result of our prediction (in orange). When we look at all the prediction graphs, we can see that the orange line overlaps, almost entirely the blue line, which shows us that the prediction is accurate, but even so we should analyze the parts of the signal that are badly reconstructed, and with that being said, what we are seeing in each image is a P wave, a QRS complex and T wave, which together represent a heartbeat. Analyzing each one, we can see that in all of them the P wave is always well predicted, concerning the QRS complex we can see that we always have a failure in the conversion of the wave S, being the inflection done with a small value of amplitude than the original signal. As for the wave T we can see that in Lead I, it is reconstructed correctly, in Lead II we lost some information as we can see, between 400 and 500, we have some areas where the orange line does not overlap with the blue one. The same happens in Lead III where the inflection of the predicted signal happens at an amplitude of -0.75 instead of -1.00 as in the original signal.

After analyzing the results, we can say that regardless of the Lead we are trying to predict when we feed to the autoencoder the lead we are trying to predict, that is a very accurate and efficient way, it can make a good prediction even with some loss, which can tell us, that the autoencoder is learning the key features of the different Leads, and can reconstruct the signal as it should be.

6.1.2 Conversion to Lead I

In this section, we will analyze the results we obtained when we made our firsts conversions. For this we use the same autoencoder as in the previous example, the only change being that, now we have a Lead and a target, which is Lead I for all conversions we made in this step.

As already said, the Lead that we will consider being the target is Lead I and it has this representation:

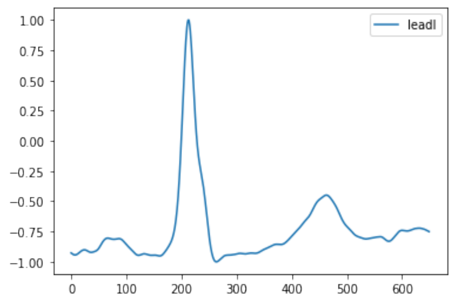

Figure 6.4: Target Lead representation - Lead I

The results of the conversions we obtained were as follows:

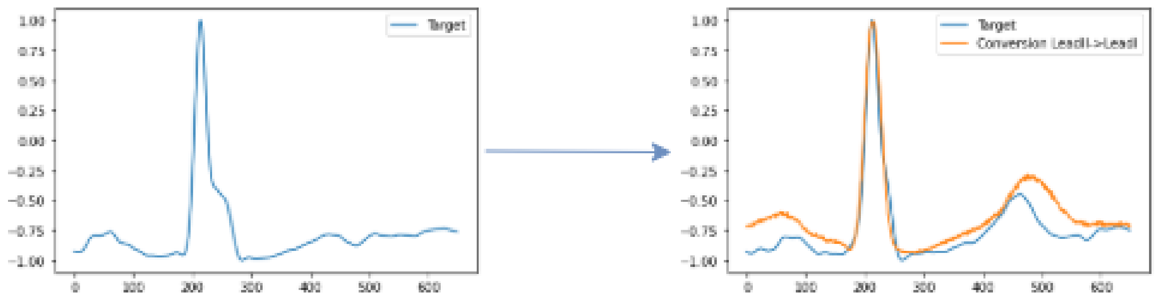

Figure 6.5: Conversion Lead II → Lead I

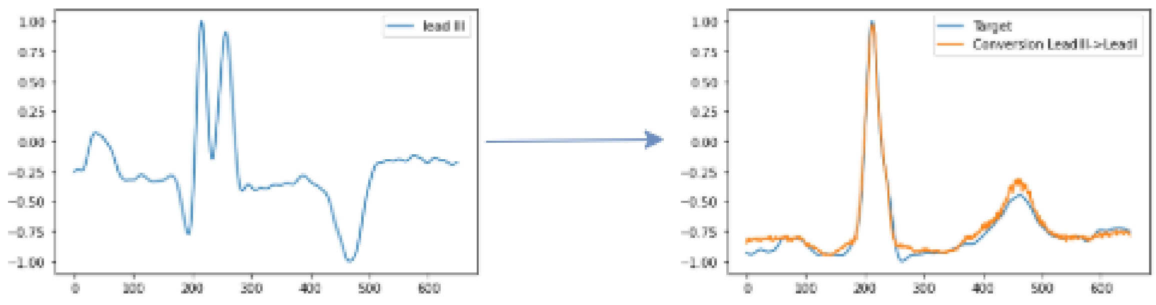

Figure 6.6: Conversion Lead III → Lead I

Image 6.5, 6.6 represents the conversion of Lead II and Lead III to Lead I, respectively. In the same way, as in the previous example, our results consist of two graphs, which in this case, as it is a conversion, the first graph, the one on the left (in blue), represents the Lead that will be fed to the autoencoder in order to be converted, and the second graph, the one on the right (in orange), represents the signal conversion. When we look at both conversions, we see that they were successful, since the orange line is a very close representation of the blue line.

When we look at the two images, but only at the predicted signal (orange signal), we can identify the Lead I, shown in figure 6.4, and in this way, we can conclude that the autoencoder is doing a good reconstruction of the different Leads. But when we compare the result (orange signal) with the original signal (blue signal), we can see that there are flaws, especially when we try to convert the Lead II, we can see that both the P and T waves were lost since the overlap between the two lines is null in case of wave P and close to null in wave T. In the QRS complex, there is a loss of both Q and S waves, but not as noticeable as the P and T waves, since here we have an overlap. When we observe the conversion of Lead III, we see that it was much better done than the previous one, with much less loss throughout the entire signal, with the most significant loss being in the S wave where the inflection was once again made in a smaller amplitude than the original signal, this different levels of accuracy could tell us that the graphic representation of the Lead III is closer to Lead I than Lead II and because of that, the conversion is done in a better way.

6.1.3 Conversion multi-output

For this new set of results, we will use an autoencoder, that instead of giving us only one result, gives us as many as the number of decoders we add, in this case, three, since we are using the first three Leads as we mentioned, at the beginning this chapter.

With that being said, we are using the multi-output autoencoder described in the previous section, where we have a common encoder and a decoder for each Lead, as represented in the image 5.3 of the previous chapter.

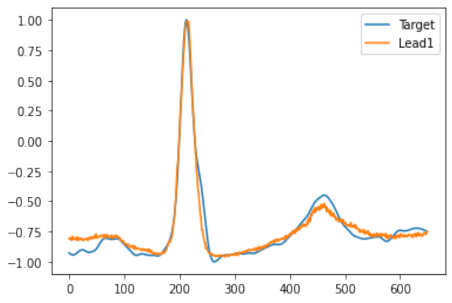

Figure 6.7: Conversion xtrain → Lead I

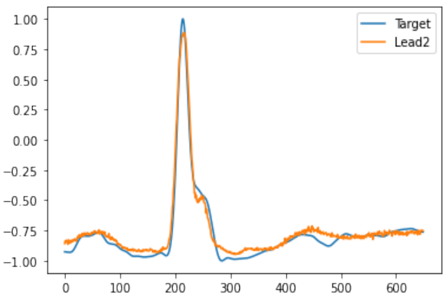

Figure 6.8: Conversion xtrain → Lead II

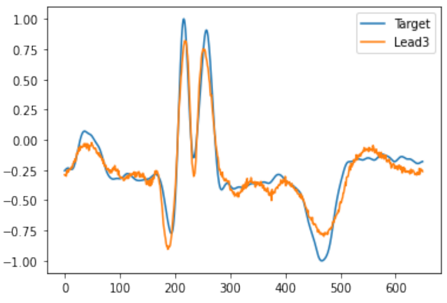

Figure 6.9: Conversion xtrain → Lead III

To obtain these results, it is important to mention that, contrary to what happened in the previous examples, in which we fed an xtrain array to the autoencoder, that consisted only of a specific Lead, which was the one we wanted to be learned by the encoder, here our xtrain array consists of the first three Leads, as we can also see in figure 5.3 of the previous chapter, and in this way our encoder will be able to learn the most important features of the Leads, to be able to reconstruct them.

As we can see from the figure of the results obtained, we can say that for Lead I the encoder is learning and being able to predict the Lead in its entirety, except for some parts where some information is lost. In the image of Lead II we see that the Lead was reconstructed but with a slight shift, but we can see that there in the area of the Q wave the autoencoder managed to reconstruct the inflection. In the Lead III image, we have a lot more losses than the other two, but regardless of that, we can say the prediction is done just by analyzing the shape of the signal.

6.1.4 Conversion/Prediction of leads using a convolutional autoencoder

As it was mention in the methodology chapter, for this autoencoder we had a different approach when it comes to pre-processing the data. To get the shape of the signal for training we just went throughout all the recordings we have, and cut the signals in little parts, in this case, with one second of length. Then we just reshape it, so we could add the channel that will represent the dimension of the signal. But as well as the previous examples we also divided the data into two groups: test and train to perform the training of the autoencoder with the set created for that task and then use the test set to verify if it learned the features, so that can convert unseen data.

The results obtained were the following:

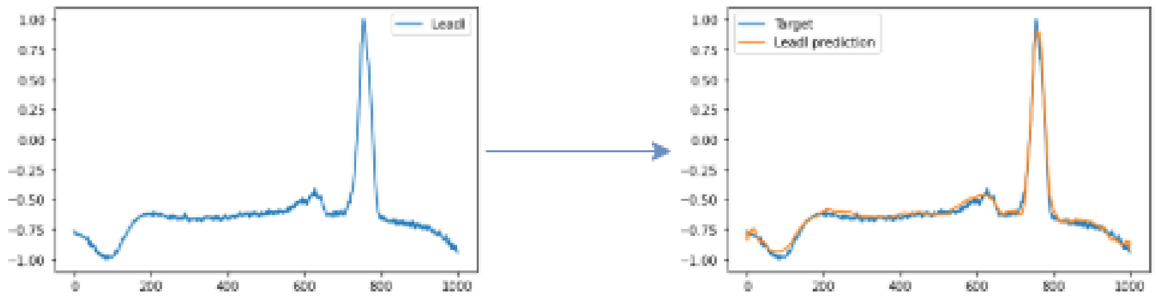

Figure 6.10: Prediction Lead I → Lead I

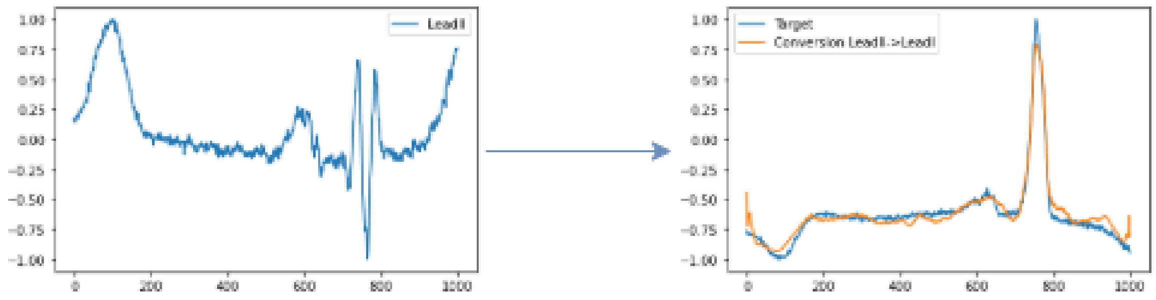

Figure 6.11: Conversion Lead II → Lead I

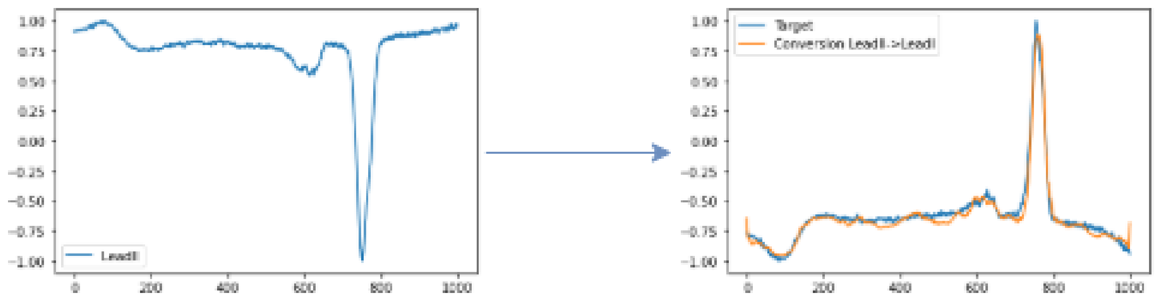

Figure 6.12: Conversion Lead III → Lead I

As in the previous examples, we are also using a test set, and as we can see the conversion results are pretty accurate. The way we trained this new autoencoder, was by combining the training we did for the predictions and the conversions in the previous autoencoder, being the following, we started by trying to predict the Leads that were given to the autoencoder as input, and that is what's represented in the figure 6.10, where we predicted the Lead I.

As we can observe in the image we have a total overlap of both signals, and, the contrary of what happened in the fully-connected autoencoder here we did not lose the information of wave S, which is a good indicator of improvement, when we compare with the previous results, although here we lost some information about the R-peak.

About the images 6.11 and 6.12, we are dealing with conversions instead of predictions, and because of that, we see that the converted signal (in orange) has a lot more variations than what we see in figure 6.10. When we compare these two conversions with the ones made by the fully-connected autoencoder we can see that has a lot of improvement, in the way that this autoencoder was able to capture all the information needed to reconstruct the signal, which indicates that the exploration of this autoencoder could lead us to better results.

6.1.5 Conversion convolutional autoencoder multi-output

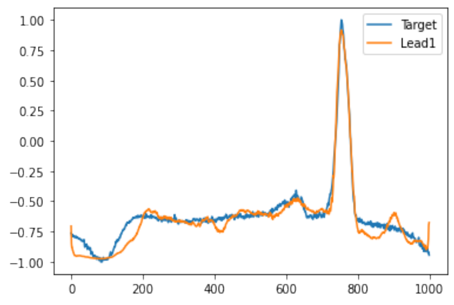

Figure 6.13: Prediction LeadI

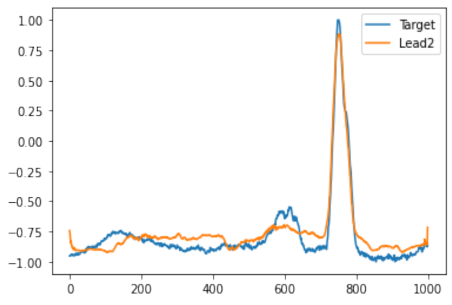

Figure 6.14: Conversion LeadII

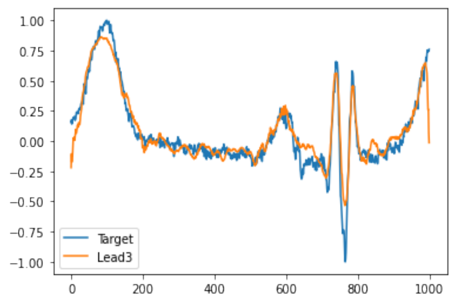

Figure 6.15: Conversion LeadIII

In the same way, as it happened in the fully-connected multi-output autoencoder, here we also generate an xtrain array that was also made up of the first three Leads, so that, the encoder could learn the main features, to gather the fiducial points necessary for it to be carried out in the conversion.

As we can see from the images, the autoencoder is managing to convert the Leads, but contrary to what happened in the convolutional that was not multi-output, this one lost more information about some waves, which may be happening due to the smaller number of data for each Lead, and because of that, the encoder does not have enough information to create a good coding. Even so, the results are satisfactory, which leads us to conclude that the convolutional autoencoder with some training and improvement of some parameters, such as increasing or reducing the number of layers, may be ideal for conversions.

6.2 Quantitative results

One of the important factors to evaluate when working with autoencoders, is the loss value, as it is this value that shows us the amount of information that we are losing during the conversion.

To have a more in-depth analysis of the loss values, we chose to make comparisons of those same values, by comparing autoencoders with different number of layers and also conversions from different Leads to Lead I and then use the same method to analyse the conversion from a Lead to several, and thus have a more concrete idea of our conversions and mainly understand what factors contribute directly to the conversion of these Leads, whether to obtain better or worse results. So we start by showing three tables:

xtrain	**leadI → leadI**			**leadII → leadI**			**leadIII → leadI**		
Loss	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**
1	0.009	0.943	68.509	0.062	0.829	60.241	0.046	0.830	61.544
2	0.009	0.947	68.465	0.059	0.838	60.450	0.046	0.847	61.521
3	0.010	0.937	67.940	0.062	0.824	60.185	0.055	0.814	60.710
4	0.009	0.947	68.630	0.058	0.840	60.464	0.041	0.863	62.055
5	0.009	0.943	68.486	0.060	0.819	60.315	0.046	0.844	61.522
6	0.008	0.950	68.981	0.070	0.781	59.702	0.054	0.781	60.844
7	0.011	0.928	67.582	0.065	0.822	60.019	0.047	0.856	61.401
8	0.008	0.949	68.913	0.057	0.846	60.562	0.041	0.862	61.954
9	0.009	0.949	68.721	0.066	0.812	59.905	0.042	0.855	61.875
10	0.010	0.937	68.187	0.064	0.829	60.090	0.042	0.854	61.878
AVG	**0.009**	**0.943**	**68.441**	**0.062**	**0.824**	**60.193**	**0.046**	**0.841**	**61.530**
Std dev	**0.003**	**0.022**	**1.399**	**0.013**	**0.065**	**0.762**	**0.014**	**0.082**	**1.345**

Table 6.1: Analysis of the different metrics in a 11 layer fully-connected autoencoder

xtrain	**leadI → leadI**			**leadII → leadI**			**leadIII → leadI**		
Loss	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**
1	0.007	0.959	69.878	0.063	0.798	60.160	0.043	0.862	61.847
2	0.008	0.950	69.164	0.060	0.824	60.322	0.046	0.851	61.541
3	0.007	0.955	69.455	0.059	0.825	60.412	0.043	0.858	61.803
4	0.008	0.951	69.152	0.065	0.835	59.995	0.046	0.841	61.506
5	0.008	0.952	69.307	0.062	0.825	60.242	0.044	0.848	61.716
6	0.008	0.954	69.170	0.054	0.847	60.812	0.044	0.854	61.743
7	0.008	0.955	68.981	0.059	0.824	60.450	0.046	0.847	61.498
8	0.007	0.958	69.697	0.058	0.840	60.523	0.043	0.853	61.792
9	0.007	0.958	69.605	0.055	0.844	60.707	0.046	0.846	61.480
10	0.008	0.955	69.325	0.061	0.835	60.289	0.052	0.797	60.986
AVG	**0.008**	**0.955**	**69.273**	**0.060**	**0.830**	**60.391**	**0.045**	**0.846**	**61.591**
Std dev	**0.001**	**0.009**	**0.897**	**0.011**	**0.049**	**0.817**	**0.009**	**0.065**	**0.861**

Table 6.2: Analysis of the different metrics in a 9 layer fully-connected autoencoder

xtrain	**leadI → leadI**			**leadII → leadI**			**leadIII → leadI**		
Loss	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**	**MSE**	**SSIM**	**PSNR**
1	0.006	0.960	70.162	0.064	0.831	60.083	0.045	0.844	61.552
2	0.006	0.959	70.057	0.058	0.840	60.482	0.054	0.800	60.832
3	0.007	0.961	69.877	0.062	0.813	60.237	0.053	0.831	60.925
4	0.007	0.958	69.834	0.052	0.838	61.002	0.055	0.813	60.689
5	0.006	0.960	70.148	0.058	0.836	60.491	0.050	0.824	61.129
6	0.006	0.963	70.125	0.051	0.855	61.027	0.040	0.863	62.119
7	0.005	0.969	70.849	0.058	0.839	60.513	0.048	0.841	61.335
8	0.006	0.967	70.690	0.063	0.828	60.153	0.051	0.826	61.067
9	0.006	0.964	70.499	0.053	0.844	60.857	0.046	0.857	61.492
10	0.006	0.963	70.236	0.056	0.832	60.683	0.045	0.839	61.642
AVG	**0.006**	**0.962**	**70.248**	**0.058**	**0.836**	**60.553**	**0.049**	**0.834**	**61.278**
Std dev	**0.002**	**0.011**	**1.015**	**0.013**	**0.042**	**0.944**	**0.015**	**0.063**	**1.43**

Table 6.3: Analysis of the different metrics in a 7 layer fully-connected autoencoder

The tables above represent the average loss values in conversions from Lead I, Lead II, and Lead III to Lead I in 10 training sessions, where we reduced the number of layers by two.

To analyze the loss tables, in addition to dividing only into different loss functions, we will also divide the number of layers in the autoencoders and divide into two groups, predictions, and conversion.

Before we look at the numbers, what makes sense is that the best autoencoder, at least in most cases, is the one with the greatest number of layers, since the more layers we have on the encoder part, the signal will be compressed more slowly, which will lead to greater clarity for the encoder, which are the focal points, which define the signal. "Is this what happens?", let's check now with an analysis of the tables.

- Predictions:
  - 11 layers: When we look at the results of the predictions in the 11-layer autoencoder, the largest variation of values we have, regardless of the loss function is not significant, being the highest with MSE of 0.008-0.011 (0.003 variation), with SSIM of 0.928-0.950 (0.022 variation) and with PSNR of 67,582-68,981 (1,399 variation) for Lead I.
  - 9 layers: When we look at the results of the predictions in the 9-layer autoencoder, the largest variation of values we have, regardless of the loss

  function is not significant, being the highest with MSE of 0.007-0.008 (0.001 variation), with SSIM of 0.950-0.959 (0.009 variation) and with PSNR of 68.981-69.878 (0.897 variation) for Lead I.

  – 7 layers: When we look at the results of the predictions in the 7-layer autoencoder, the largest variation of values we have, regardless of the loss function is not significant, being the highest with MSE of 0.005-0.007 (0.002 variation), with SSIM of 0.958-0.969 (0.011 variation) and with PSNR of 69.834-70.849 (1.015 variation) for Lead I.

  Analyzing the loss variations in the predictions of the three autoencoders we can conclude that the 9-layer autoencoder is the one that shows smaller values of variations, which shows that this is the one that behaves better with making predictions of the leads that are given to the autoencoder.

- Conversions:
  - 11 layers: When we look at the results of the conversions in the 11-layer autoencoder we need to divide into conversions Lead II→Lead I and Lead III→Lead I, the largest variation of values we have, regardless of the loss function is not significant, being the highest with MSE of 0.057-0.070 (0.013 variation), with SSIM of 0.781-0.846 (0.065 variation) and with PSNR of 59.702-60.464 (0.762 variation) for Lead II→Lead I and the highest with MSE of 0.041-0.055 (0.014 variation), with SSIM of 0.781-0.863 (0.082 variation) and with PSNR of 60.710-62.055 (1.345 variation) for Lead III→Lead I.
  - 9 layers: When we look at the results of the conversions in the 9-layer autoencoder we need to divide into conversions Lead II→Lead I and Lead III→Lead I, the largest variation of values we have, regardless of the loss function is not significant, being the highest with MSE of 0.054-0.065 (0.011 variation), with SSIM of 0.798-0.847 (0.049 variation) and with PSNR of 59.995-60.812 (0.817 variation) for Lead II→Lead I and the highest with MSE of 0.043-0.052 (0.009 variation), with SSIM of 0.797-0.862 (0.065 variation) and with PSNR of 60.986-61.847 (0.861 variation) for Lead III→Lead I.
  - 7 layers: When we look at the results of the conversions in the 7-layer autoencoder we need to divide into conversions Lead II→Lead I and Lead III→Lead I, the largest variation of values we have, regardless of the loss function is not significant, being the highest with MSE of 0.051-0.064 (0.013

variation), with SSIM of 0.813-0.855 (0.042 variation) and with PSNR of 60.083-61.027 (0.944 variation) for Lead II→Lead I and the highest with MSE of 0.040-0.055 (0.015 variation), with SSIM of 0.800-0.863 (0.063 variation) and with PSNR of 60.689-62.119 (1.43 variation) for Lead III→Lead I.

Analyzing the loss variations in the conversions of the three autoencoders we can conclude that the 9-layer autoencoder is the one that shows smaller values of variations, which shows that this is the one that behaves better with making conversions of the leads that are given to the autoencoder.

Since in both examples, forecasts, and conversions, the autoencoder that shows the best results is the one with 9 layers, being the "average" in terms of layers that we created, we can conclude that this will be our best choice to try to improve and try to achieve the autoencoder that is the best at performing these tasks regardless of Leads. This is a great result because it is a good approximation of what we predicted that the result would be, thus being that, the best one is not the smallest, which is good. After all, when we have such a small number of layers, probably the input is just being copied to the output without having the learning process, from the autoencoder. With the same thought, it is also great that the biggest one is not better either, because when we get a lot of layers in the encoder part, we can be facing overfitting problems.

Now lets see what happens in the autoencoder multi-output:

xtrain	**xtrain → all:9layers**			**xtrain → all:11layers**			**xtrain → all:13layers**		
Loss (MSE)	**LeadI**	**LeadII**	**LeadIII**	**LeadI**	**LeadII**	**LeadIII**	**LeadI**	**LeadII**	**LeadIII**
1	0.600	0.609	0.404	0.584	0.600	0.418	0.583	0.603	0.402
2	0.584	0.605	0.426	0.592	0.618	0.414	0.531	0.612	0.437
3	0.560	0.549	0.413	0.595	0.599	0.410	0.575	0.605	0.408
4	0.565	0.620	0.412	0.592	0.595	0.419	0.567	0.606	0.416
5	0.585	0.593	0.407	0.596	0.600	0.408	0.580	0.615	0.406
6	0.552	0.610	0.402	0.576	0.589	0.418	0.583	0.605	0.395
7	0.588	0.611	0.416	0.588	0.587	0.409	0.612	0.618	0.412
8	0.582	0.599	0.415	0.583	0.617	0.420	0.582	0.608	0.394
9	0.605	0.602	0.399	0.581	0.615	0.409	0.595	0.604	0.412
10	0.580	0.590	0.421	0.610	0.622	0.375	0.597	0.620	0.405
AVG	**0.580**	**0.599**	**0.412**	**0.590**	**0.604**	**0.410**	**0.580**	**0.610**	**0.409**
Std dev	**0.053**	**0.071**	**0.027**	**0.034**	**0.035**	**0.045**	**0.081**	**0.017**	**0.043**

Table 6.4: Analysis of the different metrics in fully-connected multi-output autoencoder

Like the first example, we are obtaining these values, by training the autoencoder 10 times and reducing the number of layers. The values are also being calculated using the mean squared error loss function.

In this table, we are showing the loss values in the three Leads, for each change in the autoencoder. To evaluate these values, we calculate their average, obtaining the following results: 0.530, 0.535, and 0.533, for 9, 11, and 13 layer autoencoders, respectively.

When we look at these values, we see that there is not a big difference, so that it leads us to choose one. Thus, we decided to evaluate the variation in loss, having found the following values: 0.049, 0.038, and 0.047 for autoencoders of 9, 11, and 13 layers, respectively.

To find a good autoencoder, we look for the one with the smallest loss rate, as well as the smallest variation, since if these values are low, it shows that we can capture the maximum amount of information when we do the conversion. As through the analysis of the values in the table and the averages we calculate, we see that the values do not vary so that we can choose one. We can conclude that the performance of a multi-output autoencoder is not very influenced by the number of layers it has.

After training all these previous autoencoders, we had an idea, as it was mention before, to explore the convolution autoencoders. Since we are dealing with different

xtrain	**LeadI → LeadI**	**LeadII → LeadI**	**LeadIII → LeadI**
1	0.224	0.177	0.193
2	0.243	0.187	0.190
3	0.238	0.176	0.183
4	0.209	0.203	0.238
5	0.269	0.220	0.200
6	0.233	0.178	0.240
7	0.255	0.198	0.255
8	0.240	0.171	0.171
9	0.244	0.204	0.271
10	0.237	0.173	0.226
AVG	**0.239**	**0.189**	**0.217**
Std dev	**0.06**	**0.049**	**0.1**

Table 6.5: Analysis of the different metrics in a convolutional autoencoder

types of layers, we opt for change the training method, and instead of reducing the number of layers, we just train the testing set throughout 10 trains, like in the previous example and try to compare these results with the previous, to get an idea, if this autoencoder could be the way to go, to improve our work, and the values of loss we got are represented in the table below:

As it happens in the previous training this table, also has three columns, where the first one represents the prediction of Lead I, and the other two represent the conversion, and where all the loss values were calculated with MSE.

Analyzing now the columns that represent the conversions, we can see that comparing it with the others the values of loss are greater, this could be happening due to the lack of pre-processing, that we have of the signals in this example. Even so, when it comes to converting Leads this autoencoder seems promising, because, and as we saw in the results chapter, this autoencoder captures all waves in the heartbeat signal even without having to locate the R-peak before training, and that is something that did not happen in the fully-connected autoencoder.

To analyze these values, we chose to compare these results with those obtained in the previous ones, and we can see that when compared with the autoencoders, regardless of the number of layers, we can see that the convolutional one always presents higher loss rates, but we can forget that it also converts a much greater amount of information, and in this way, it makes sense that the loss is greater, because it also has more data that is being analyzed.

Even with worse loss results, we can see from the representations that the conversions made by the convolutional autoencoders are more complete but also present a greater amount of noise. This noise can also be the result of not filtering the signals in the processing part, being that in the fully-connected autoencoders when they are given to the autoencoder, they have already undergone a large part of the processing. This could be interesting also as future work, that is, seeing how much the pre-processing of the data is directly connected with the performance of the autoencoder.

To sum up, in this chapter we did a qualitative and quantitative analysis of the data, where in the first analysis, we showed and analyzed the graphic representations we obtained in our training, completing this analysis with a small discussion of the different loss values that were acquired through the calculations of different metrics, but which corroborated the graphical results, thus falling into the quantitative analysis of our data.

Chapter 7

Discussion

In this chapter, we will analyze our entire thesis, completely and concisely, to have a concrete idea of what were the high points and the low points of all our work, also showing ideas that can be put into practice as future work.

Going back to the beginning, this lead conversion idea was motivated by the lack of some leads in ECG readings that are online, which led us to think that at the medical level, it may be an impediment in the diagnosis of some diseases, and this way we proposed this conversion algorithm.

To achieve this algorithm, we resorted to the implementation of an autoencoder, since it was the one that managed to match all the characteristics we were looking for to have the best performance. We ended up creating four different ones, each one evolving from the previous one.

We started exploring fully-connected autoencoders, and in this way, we created two, one that received a Lead and converted to another and one that received a vector with some Lead samples and converted them in order to have a multi-output.

After being trained, and already with good results, we thought about how we could do to improve and that's when we came up with the idea of turning these two autoencoders into convolutional ones

The big improvement we achieved with convolutional was the fact that here two layers called max-pooling and upsampling are introduced, which transform the size of the signal and in this way get that autoencoder to extract a greater amount of information in the encoding part. Which in general resulted in better results.

The way we chose to analyze the results, which was by changing the number of layers in the different autoencoders, may not have been the best since it is a factor that does not influence our results in a very significant way. But even so, seeing the results obtained, we can conclude that we managed to achieve our goal.

Thus, taking into account our development and results, we believe that some points should be explored, both at the medical level and the biometric level.

As mentioned at the beginning of this discussion, one of the reasons that led us to make this creation was the lack of some Leads in ECG readings, so an idea would be to show our results to a cardiologist so that he could analyze them and give us feedback if these conversions are sufficient and reliable enough to be able to help in the diagnosis. At the biometric level, see if these conversions preserve the information so that it is used as a biometric feature in classification and identification processes. Finally, we can also try to improve our model using GANs or variational autoencoders for example.

In summary, for our purposes, the results were quite satisfactory. Knowing that there is always room to improve what, due to lack of time, we are unable to fully exploit the potential of convolutional autoencoders, we can always take our model and analyze its results differently, in order to find parameters that cause changes more notable than just removing and adding layers to see if we come to the same conclusions.

Chapter 8

Conclusion

This last chapter serves as the closure of our thesis, and in this way, we will use it to summarize the main points of our thesis.

Starting from the beginning, the objective of our thesis was the conversion of Leads from electrocardiograms (ECG). This idea came to us to try that with the creation of this algorithm we could somehow help in the field of medicine, in terms of disease diagnosis. We think it would be interesting to address this issue because, through the recordings that exist, it appears that not all Leads are present.

In this way, we use autoencoders, which consist of unsupervised neural networks that learn the features of a given input and proceed to its reconstruction. In the development of this thesis, we created four autoencoders: fully-connected, fully-connected multi-output, convolutional autoencoders, and convolutional multi-output autoencoders, respectively, each one being an improvement from the previous one.

We started by exploring the fully-connected ones, which are made up of dense layers and which work as follows: It has an encoder and decoder that are symmetric, in which the encoder compresses the input until the bottleneck, where an array is formed with the most important features that define the input, which is called coding. This coding will be used in the decoder part, as a basis for reconstructing the input. In the case of convolutional autoencoders, we also have symmetric encoder and decoder, but the difference is that in these the layers are convolutional, max-pooling, and transposed convolutional, upsampling, respectively. The biggest difference is that these layers change the size of the input at each iteration in the network, to discard information that is not relevant in defining the input.

From a purely programmatic view, the results obtained are quite good, but an idea to validate them, it would be, as already said, to show them to a cardiologist for him to analyze them and confirm the reliability of the data in medical terms.

To sum up, seeing everything we have managed to develop since the beginning of the research, we could not be more satisfied, since we achieved all the objectives we set ourselves in an initial phase. The result is a robust and adaptable algorithm that we hope will be a good tool in the process of diagnosing heart disease in a multi-lead ECG that was the ultimate goal from the beginning.

References

[1] Electrocardiogram(ecg). https://www.cuf.pt/saude-a-z/eletrocardiograma-ecg.

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Lena Biel, Ola Pettersson, Lennart Philipson, and Peter Wide. Ecg analysis: a new approach in human identification. *IEEE Transactions on Instrumentation and Measurement*, 50(3):808–812, 2001.

[4] Stephen A. Billings. *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. 2013.

[5] R Bousseljot, D Kreiseler, and A Schnabel. Nutzung der ekg-signaldatenbank cardiodat der ptb über das internet. *Biomedizinische Technik/Biomedical Engineering*, 40(s1):317–318, 1995.

[6] James D Broesch, Dag Stranneby, and 1959 Walker, William. *Digital signal processing : instant access*. Amsterdam ; Boston : Newnes/Elsevier, 2009. Includes bibliographical references (page 149) and index.

[7] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[8] Carlo J DeLuca, Per Bergman, Gianluca DeLuca, and L Donald Gilmore. Biosignal monitoring system and method, May 29 2001. US Patent 6,238,338.

[9] Arden Dertat. Applied deep learning-part 3: Autoencoders. *by Towardssdata-science. Oct*, 2017.

[10] Monty A Escabí. Biosignal processing. In *Introduction to Biomedical Engineering*, pages 549–625. Elsevier, 2005.

[11] Rune Fensli, Einar Gunnarson, and Torstein Gundersen. A wearable ecg-recording system for continuous arrhythmia monitoring in a wireless tele-home-care situation. In *18th IEEE Symposium on Computer-Based Medical Systems (CBMS'05)*, pages 407–412. IEEE, 2005.

[12] Jerome H Friedman. Data mining and statistics: What's the connection? *Computing Science and Statistics*, 29(1):3–9, 1998.

[13] Patrick O Glauner. Deep convolutional neural networks for smile recognition. *arXiv preprint arXiv:1508.06535*, 2015.

[14] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. Ch. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000 (June 13). Circulation Electronic Pages: http://circ.ahajournals.org/content/101/23/e215.full PMID:1085218; doi: 10.1161/01.CIR.101.23.e215.

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

[16] KWilliam L. Hosch. Machine learning. *Encyclopædia Britannica, inc.*, 2019.

[17] Nathan Hubens. Deep inside: Autoencoders. https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f.

[18] Dimitris G. Manolakis John G. Proakis. *Digital Signal Processing principles, algorithms, and applications*. PRENTICE-HALL INTERNATIONAL, INC. https://engineering.purdue.edu/ ee538/DSP$_{T}ext_{3}rdEdition.pdf$.

[19] Antonio Garcia-Uceda Juarez, Raghavendra Selvan, Zaigham Saghir, and Marleen de Bruijne. A joint 3d unet-graph neural network-based method for airway segmentation from chest cts. In *International Workshop on Machine Learning in Medical Imaging*, pages 583–591. Springer, 2019.

[20] S Karpagachelvi, M Arthanari, and M Sivakumar. Ecg feature extraction techniques-a survey approach. *arXiv preprint arXiv:1005.0957*, 2010.

[21] V. Krishnaveni, S. Jayaraman, S. Aravind, V. Hariharasudhan, and Ramadoss Kalidoss. Automatic identification and removal of ocular artifacts from eeg using wavelet transform. *Meas. Sci. Rev.*, 6, 11 2005.

[22] Stephen Marsland. *Machine learning: an algorithmic perspective.* CRC press, 2015.

[23] Jose Moura. What is signal processing? [president's message]. *Signal Processing Magazine, IEEE*, 26:6 – 6, 12 2009.

[24] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[25] Thanh Thi Nguyen, Cuong M Nguyen, Dung Tien Nguyen, Duc Thanh Nguyen, and Saeid Nahavandi. Deep learning for deepfakes creation and detection. *arXiv preprint arXiv:1909.11573*, 2019.

[26] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[27] João Ribeiro Pinto, Jaime S Cardoso, and André Lourenço. Evolution, current challenges, and future possibilities in ecg biometrics. *IEEE Access*, 6:34746–34776, 2018.

[28] Luis Howell Bernd Porr. A collection of 7 ecg heartbeat detection algorithms implemented in python., 06 2007. http://researchdata.gla.ac.uk/716/.

[29] David Selvitelli and Mark Tauer. Ecg lead wire organizer and dispenser, May 15 2012. US Patent 8,180,425.

[30] Inga Teismann, Olaf Steinsträter, Tobias Warnecke, Sonja Suntrup-Krueger, E. Ringelstein, Christo Pantev, and Rainer Dziewas. Tactile thermal oral stimulation increases the cortical representation of swallowing. *BMC neuroscience*, 10:71, 07 2009.

[31] Ali Usakli, Serkan Gurkan, Fabio Aloise, Giovanni Vecchiato, and Fabio Babiloni. On the use of electrooculogram for efficient human computer interfaces. 05 2009.

[32] G Angelo Virgin and V Vijaya Baskar. Conversion of ecg graph into digital. *International Journal of Pure and Applied Mathematics*, 118(17):469–484, 2018.

[33] Wei Zhang. Parallel distributed processing model with local space-invariant interconnections and its optical architecture. 1990.

[34] Xin Zhu, K Yoshida, W Yamanobe, Y Yamamoto, Wenxi Chen, and Daming Wei. Conversion of the ambulatory ecg to the standard 12-lead ecg: a preliminary study. In *IEEE EMBS Asian-Pacific Conference on Biomedical Engineering, 2003.*, pages 48–49. IEEE, 2003.

[35] Alan Zucconi. Understanding the technology behind deepfakes. https://www.alanzucconi.com/2018/03/14/understanding-the-technology-behind-deepfakes/.