UNIVERSIDADE D
COIMBRA

João Francisco Louro Parente

DEVELOPMENT OF MACHINE LEARNING TOOLS FOR THE EXTRACTION OF BEHAVIORAL AND PHYSIOLOGICAL PARAMETERS OF BIRDS IN THEIR NATURAL ENVIRONMENT

Dissertação no âmbito do Mestrado Integrado em Engenharia Física orientada pelo Professor Doutor Filipe Manuel Almeida Veloso e pelo Professor Doutor Francisco Filipe Bento Neves e apresentada ao Departamento de Física da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2023

• U  C •

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

João Francisco Louro Parente

Development of machine learning tools for the extraction of behavioral and physiological parameters of birds in their natural environment

Thesis submitted to the
University of Coimbra for the degree of
Master in Engineering Physics

Supervisors:
Filipe Manuel Almeida Veloso
Francisco Filipe Bento Neves

Coimbra, 2023

Agradecimentos

Quero começar por agradecer à professora Ana Nunes, por me ter explicado matemática duma maneira tão intuitiva que me permitiu seguir o meu estudo na área das ciências. À professora Ana Cordeiro e ao professor Faustino, que explicavam física com tanto gosto e paixão que me fizeram gostar de física também. Ao professor João Sá, que me ensinou tudo o que sei de programação, se não fosse pelo esforço que fez para explicar todo o raciocínio de construção de código não teria escolhido esta tese.

Aos meus orientadores, Francisco Neves e Filipe Veloso, que me aturaram por três anos com dúvidas de iniciante, becos sem saída e texto mal escrito, muito obrigado pela paciência e tempo - foi esse esforço que deu qualidade a esta tese. Ao João Silva que, sem querer, me abriu a porta do mundo de Linux.

Aos meus amigos velhos, Gil, Lucy e Alex, que desde o secundário estiveram sempre ao meu lado em finos no Piano e divagações de 4 da manhã. Aos meus amigos de faculdade, Bea, Uly, Catarina, Carvalhão, Leonor, Carlos e Ruben, pelos risos em química geral, pelas conversas na varanda, pelos jantares a enrolar sushi, pelos almoços nas Azuis e por todas as outras aventuras. A todos os restantes que me ajudaram nestes seis anos. Às pessoas do BEST, que me mostraram o que é trabalhar em equipa, puseram-me fora da minha zona de conforto e ensinaram-me tanta coisa que vou levar comigo para o resto da minha vida. A Natália, por me aturar nas minhas divagações e ajudar-me sempre em tudo o que precisava.

A minha família, por me terem ajudado tanto nesta fase da minha vida, por terem apoiado todas as minhas decisões.

Obrigado a todos.

Resumo

Os ambientes costeiros são essenciais para muitas espécies, incluindo as gaivotas das espécies *Larus michahellis* e *Ichthyaetus audouinii*, que podem ser afetadas pela atividade humana. Para melhor compreender os efeitos da atividade humana nas gaivotas, os seus níveis de *stress* foram medidos através da análise do seu ritmo cardíaco. Para atingir este objetivo, foram construídos ovos falsos com microfones para registar o ritmo cardíaco das gaivotas.

Neste trabalho, foi desenvolvida uma abordagem de aprendizagem de máquina que combinou características extraídas dos Coeficientes Cepstrais de Frequência de Mel (CCFM) e Redes Neuronais Artificiais (RNA) para analisar o áudio e identificar os segmentos que tinham batimentos cardíacos. A combinação que apresentou os melhores resultados extraiu 15 médias e 15 desvios-padrão dos Coeficientes Cepstrais de Frequência de Mel extraídos de uma janela de áudio de 5 segundos e, em seguida, analisou-os com uma Rede Neuronal Artificial de seis camadas, em forma de losango. Identificou com êxito o batimento cardíaco no áudio. Após a identificação, esses segmentos foram analisados por uma combinação de Rede Neural Convolucional (RNC) com Rede Neural Recorrente (RNR) usando células de Memória de Curto Longo Prazo (MCLP) para localizar os batimentos cardíacos individuais. A combinação que obteve os melhores resultados utilizou duas camadas convolucionais de uma dimensão, uma camada de agrupamento máximo, três camadas de células de Memória de Curto Longo Prazo e uma camada densa. A identificação de batimentos cardíacos nas amostras mostrou resultados promissores, com as flutuações da saída da RNA a corresponderem à localização dos batimentos cardíacos.

Abstract

Coastal environments are essential for many species, including seagulls of the *Larus michahellis* and *Ichthyaetus audouinii* species, which can be impacted by human activity. To better understand the effects of human activity on seagulls, their stress levels were measured by analyzing their heart rate. To achieve this goal, dummy eggs with microphones were built to record the seagull heart beat.

In this work, a machine learning approach that combined features extracted from Mel-frequency Cepstral Coefficients (MFCC) and Artificial Neural Networks (ANN) was developed to analyze the audio and identify the segments that had heartbeats. The combination that showed the best results extracted 15 averages and 15 standard deviations from the Mel-frequency Cepstral Coefficients extracted from a 5-second audio window, then analyzed them with a six layer, diamond shape, feedforward Artificial Neural Network. It successfully identified the heart beat in the audio. After identification, these segments were analyzed by a combination of Convolutional Neural Network (CNN) with Recurrent Neural Network (RNN) using Long Short-Term Memory (LSTM) cells to locate the individual heartbeats. The combination that got the best results used two one dimension convolutional layers, a max pooling layer, three layers with Long Short-Term Memory cells and one dense layer. The identification of heartbeats within the samples showed promising results, with the fluctuations in the ANN output corresponding to the location of the heartbeats.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This work aims at finding solutions to some problems found by the Centro de Ciências do Mar e do Ambiente (MARE) [1] research group, in the Ecology and Conservation of Top Predators (ECOTOP) [2] project, in particular, the measurement of the birds' heart rate.

The MARE research group is a Portuguese marine sciences' investigation center dedicated to the study of marine environments and to develop ways to promote sustainable growth. The ECOTOP project, aims at quantifying the human impact on the coastal populations of seagulls of the genus *Larus*.

Within this project two features were used to determine the seagulls' health: the heart rate, that was measured in Beats Per Minute (BPM) and incubation temperature, that was measured in degrees Celsius.

To determine the heart rate, the MARE researchers were identifying each heartbeat manually and from them determining the heart rate. From the heart rate, the health of the bird can be inferred [3].

This method, although effective, was limited by the amount of data a researcher could listen at a given time and moved the focus of the researcher away from data analysis and in to data processing.

This work focuses on developing an automated method for heart beat analysis. It was developed at the Laboratory of Instrumentation and Experimental Particle Physics (LIP) [4] that, as the main institution for experimental particle physics in Portugal, shared its knowledge in sensors, electronics and software with the ECOTOP project through the Competence Center in Monitoring and Control (CCMC).

1.1 State of the art

For the development of this type of work, several techniques can be used. The work is usually divided in data extraction and data analysis.

1.1.1 Data extraction

For the extraction of behavioral characteristics of a bird several techniques can be used such as:

- Observation
- ECG implant
- Dummy egg with a sensor

In observation, the bird behavioral characteristics of the bird are inferred by direct observation. However, this method is more prone to errors [5] than the other two, where behavioral parameters are inferred from the extraction of physiological parameters.

ECG implants inside the bird's body allow to measure the heart rate more accurately than all other methods [6] and from it, factors such as stress and energy consumption can be deduced [5]. This method has the ethical problem of implanting a sensor inside an animal's body. This action can also invalidate the data because then human interaction and consequent stress can be associated with the act of implanting this device.

The use of a dummy egg with a sensor inside is a non-invasive method of extracting the bird heart beat. It only takes a few minutes to place in the nest [7] making it the method of choice for analyzing human influence on birds [8].

There are several sensors that can be used inside the dummy egg to obtain the bird's heart beat, such as infrared [8, 6] or dedicated microphones [7].

Infrared sensors were used in the beginning of this branch of research because of the limitations of the available microphones by that time. Some limitations of infrared sensors include the necessity of being in direct contact with the bird's body, which demands the absence of feathers between the sensor and the skin. Additionally, a transparent window in the egg is needed to be able to emit and receive the infrared light. Furthermore, the sensor can only record a signal when it is in contact with the body. [6]

Because of this, this technology only worked on certain species of penguins.

With the developments in the sound capturing technology, it became more feasible to place high sensitivity microphones in small dummy eggs [7]. Using this type of sensor simplifies data analysis, but has the disadvantage of being prone to noise. Despite this, it has become a popular way of extracting data, thanks to its low interference in the birds' habits.

The study developed by LIP and MARE used a dummy egg to record the sounds emitted by the birds. This dummy egg had a microphone, thermometer, data acquisition system, internal memory, and batteries as can be seen in Figure 1.1.

1.1.2 Data analysis

The main goal of the data analysis is to calculate the heart rate, from which other parameters can be inferred, (e.g. the energy consumption or stress levels). [3, 8].

If the data is detailed enough, each heartbeat can be divided in two peaks corresponding to S1 and S2 [1] from which it is possible to make more detailed analyses and detect cardiac anomalies.

There are several ways to analyze the dummy egg data. A possible way is direct analysis, which consists of listening to the sound file and/or viewing the correspondent wave forms and manually count heartbeats as a function of time. This way is time-consuming and expensive, especially when there are several hundred hours of audio to analyze. Because of these factors, there is an increasing need to automate this process.

The automation process can be divided into two phases, feature extraction and information extraction.

For the automation of this process, the software of choice tends to be Matlab [7, 3, 5]. A common process in automated approaches is a band pass filter for lower frequencies, which removes a part of the noise.

The approach developed by [3] involved passing the sound through a Fourier transform, taking a twelve-second moving average and matching frequencies to BPMs. After this, multiple statistical tests were done to verify the quality of the deductions and sort into classes of data. These included tests such as: Circular-linear correlations to analyze the periodicity of the signal, two tailed t-test to quantify the difference between classes, Tukey-Kramer honestly significant difference to test how reliable the data was among others.

In the work presented in [5] multiple methods were covered. All methods started with a frequency filter and passing it through a Hilbert transform to obtain the envelope of the sound wave. After this, the data was analyzed by four methods.

- Short time Fourier transform
- Sliding auto correlation
- Synchrosqueezed transform
- Peak detection of envelope signal

These systems have larger error margins than a manual analysis but allow processing large amounts of data.

For the analysis of the data collected in this work, artificial neural networks were

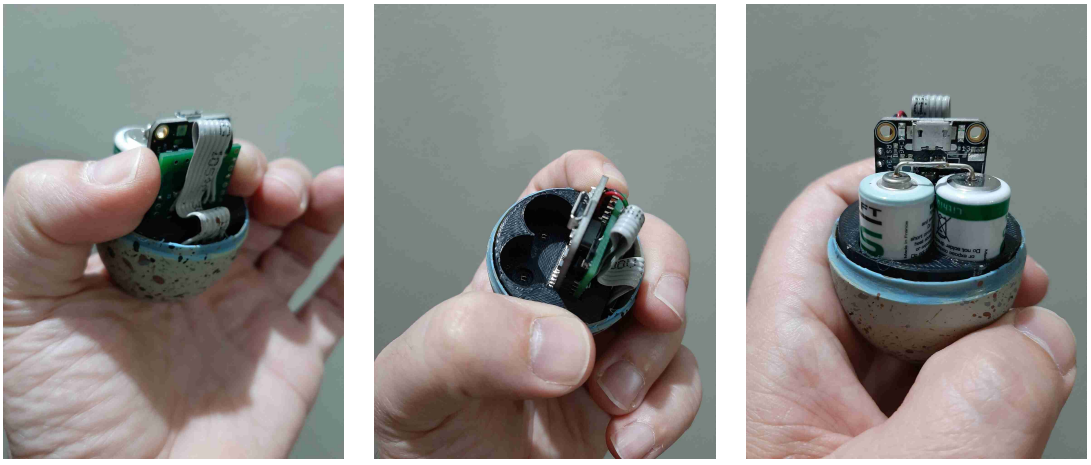
[1] S1 is the first heart sound, it occurs when the atrioventricular valves close during the ventricular systole phase. S2 in the second heart sound, it occurs when the aortic and pulmonic valves close during the ventricular diastole phase. [9]

used. This technique was not covered in the specific articles on bird heart beat analysis, however, it was already implemented in human heart beat analysis [10]. Due to their ability to adapt and learn, they have potential to analyze data more accurately than classical methods. Furthermore, artificial neural networks such as convolutional neural networks and recurrent neural networks have shown promising results in the area of signal analysis and heartbeat distinction [10, 11].

1.2 Previous work developed at LIP

To measure the heart rate and temperature of the seagulls, dummy eggs that resemble the ones belonging to seagulls, were developed. These eggs have a microphone to detect heart beat, and a thermometer to detect the incubating temperature of the seagull.

The outer layer of the eggs is a plastic shell that is covered with a rubber membrane. Inside they have a microphone, thermometer, microcontroller and batteries. The data is stored in a microSD card. A 16 Gb card can hold up to 1180 hours of data, however the batteries need to be replaced after 36 hours of use. The configuration of the dummy egg can be seen in Figure 1.1.



(a) Back view (b) Top view (c) Front view

Figure 1.1: Configuration of the dummy egg.

The heart beat was stored in the "Waveform Audio File Format" (WAV) audio file. The sample rate was initially set at 44 100 Hz but since the signal concentrates at the lower frequencies (as can be seen in figure 1.2), it was progressively lowered until it reached the value of 2 016 Hz. With this sample frequency, the maximum audio frequency the device could record is 1 008 Hz, as given by the Nyquist sampling theorem.

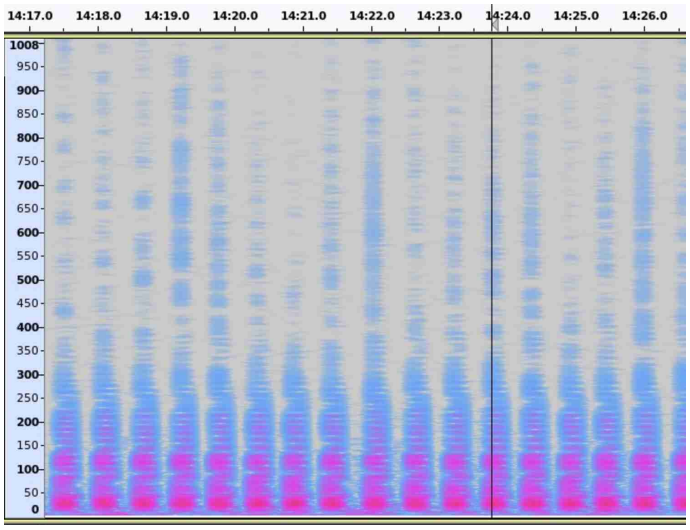


Figure 1.2: Heart beat spectrogram example.

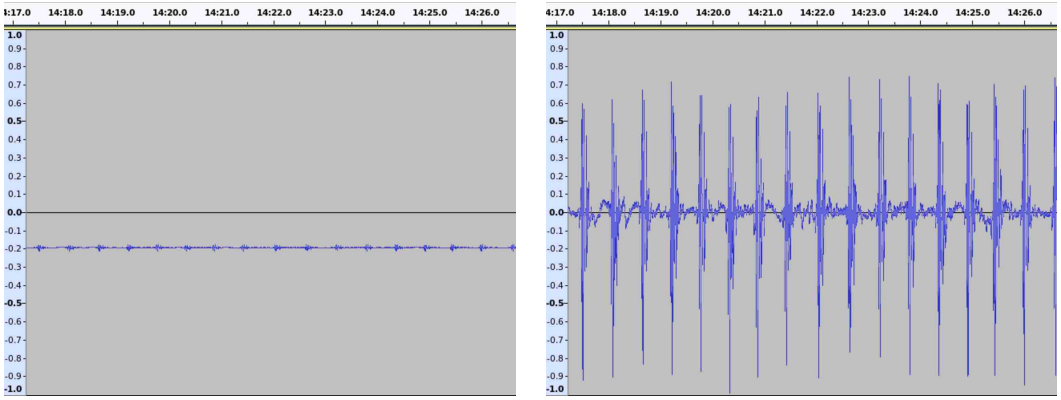
When starting this work, two methods were already available to analyze the heart beat stored in the audio files. They are described in subsection 1.2.1 *Method one* and subsection 1.2.2 *Method two*.

1.2.1 Method one

This method identified peaks in the audio that corresponded to heartbeats and from them calculated the heart rate. To achieve this, it followed these steps:

1. Window selection: A 15 s window is selected.

2. Sound amplification: The sound of a heart beat is faint, with a peak to peak amplitude of approximately 0.05 in a scale from -1 to 1, so it was amplified until it had an average amplitude of 0.5.



(a) Audio before amplification

(b) Audio after amplification

Figure 1.3: Audio amplification.

3. Low pass filter: Most of the energy is dissipated through low energy waves, as can be seen in the Figure 1.2, as such all frequencies above 4 Hz were removed.

4. Minimum, maximum finder: The peaks of the audio in a 500 samples window were found and corresponded to heartbeats.

5. Heart rate calculator: The mean value of the time differences between the maximums is determined and from it the heart rate.

1.2.2 Method two

1. Window selection as in subsection 1.2.1 *Method one*.

2. Sound amplification as in subsection 1.2.1 *Method one*.

3. Smoothing and filtering: A Hann function [12] was applied to a 3000 sample window to obtain the power/magnitude of the signal of interest.

4. Band pass: The resulting wave was passed through a band pass filter with a range from 50 HZ to, 5000 Hz to remove noise.

5. Heartbeats finder: A Difference of Gaussians [13] filter is applied to the smoothed audio. The minimum sigma of the Gaussians was 100 and the maximum sigma was 500. All features extracted that have a response above the threshold of 0.01 were considered a heartbeat.

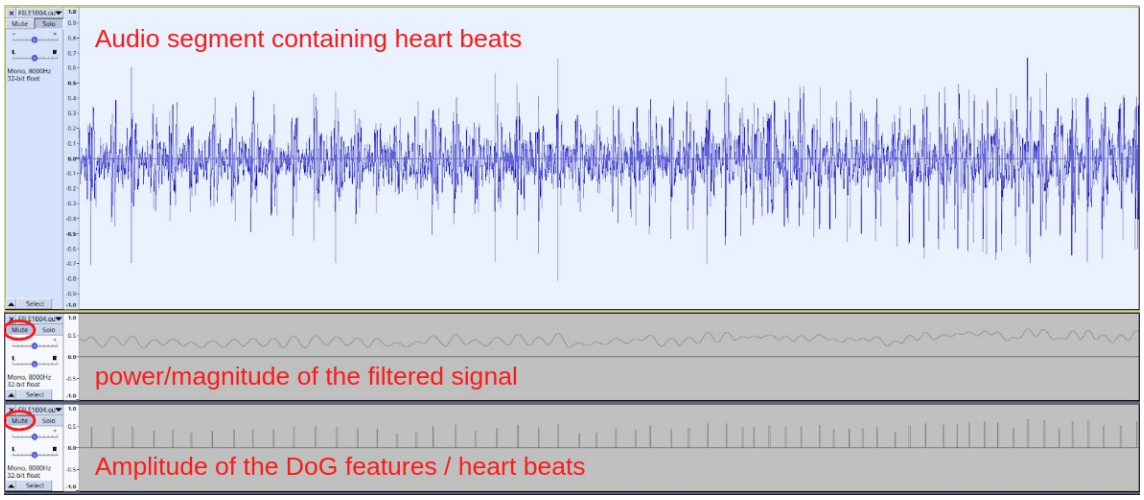


Figure 1.4: Heart beats detected with method 2.

1.3 Alternative methods

These methods had several issues, as they produced inconsistent results depending on the ambient noise and required manual parameter tuning for each individual dataset, which limited fully automatic analysis. Consequently, alternative methods were tested to address these issues.

Artificial Neural Networks (ANN) showed great potential to solve this kind of problem, given its ability to learn patterns that could not be detected by classical algorithms and to generalize them.

1.3.1 Objectives

The main objective of this research is to detect individual heart beats with an accuracy of 90%.

To reach this objective, the work passed through five stages:

1. Artificial signal generator.
2. Artificial noise generator.
3. ANN to distinguish signal (i.e., heart beat) from noise with an accuracy of at least 90%.
4. Artificial signal generator with identified heartbeats.
5. ANN to identify individual heartbeats with an accuracy of at least 90%.

The ANN that distinguishes signal from noise was necessary to improve the detection rates of the ANN responsible for identification of heartbeats.

Chapter 2

Background

The analysis of the audio files recorded using the dummy eggs was divided in two parts; feature extraction, for which the main tool was the Mel-Frequency Cepstral Coefficients (MFCC); and information extraction, which was done using Artificial Neural Networks (ANN).

For the identification of the heart beat in the data, a feedforward ANN was used. In this ANN, the inputs were features extracted from the MFCC.

For the identification of heartbeats in the audio signal, two types of ANNs, Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) were used together to extract this information directly from the raw data.

2.1 Mel-Frequency Cepstral Coefficients

The MFCC were chosen as the main way to extract information, as they were already used in similar problems [11].

They are a non-linear representation of the power spectrum of a time series, that when applied to a sound can better represent how a human would hear it [14].

They were extracted using the Librosa library [15] written in the Python programming language [16].

Computing the MFCC of an audio signal is a five-step process: framing the signal, windowing it, applying a Discrete Fourier Transform (DFT), applying a Mel filter bank, and finally applying a Discrete Cosine Transform (DCT) [17, 14]. Each of these steps is detailed below.

Framing the signal

The first step is to select a section of the audio, $S(t)$, with a short enough time span such as the spectral features of the sound can be considered stationary. This helps to stabilize the acoustic characteristics of the sound [17]. In this work, this

section of audio was called "MFCC window" and is referred by $S_m(t)$ where m is the index of the window and t the index of the audio sample in the window.

Windowing the signal

On each MFCC window, a Hann window filter is applied, this filter reduces the signal amplitude towards the edge of the window which enhances harmonics, smooths edges, and diminishes edge effects that occur when passing a sound to the frequency spectrum [17, 18]. The Hann filter is defined by the equation 2.1 [18].

$$H(n) = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi n}{N-1}\right) \quad , \quad 0 \leq n \leq N-1, \tag{2.1}$$

where n is the index of the sample of a time series and N is the total number of samples in the time series. This creates a function with the shape showed on figure 2.1.

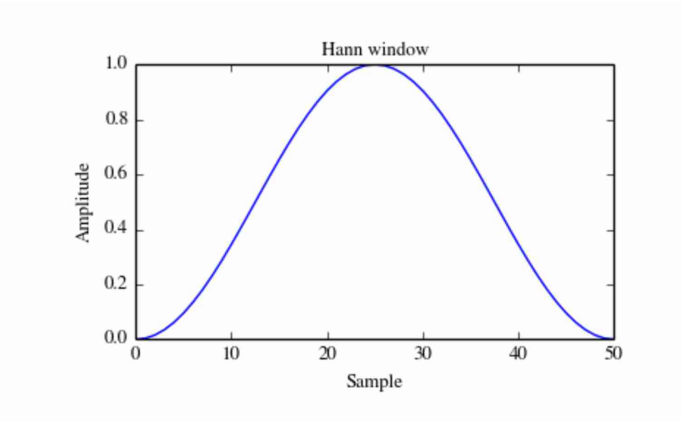


Figure 2.1: Hann window $H(n)$ [19].

This filter was multiplied by the signal, as is described in equation 2.2 [18].

$$S'_m(t) = H(t) \cdot S_m(t). \tag{2.2}$$

Discrete Fourier transform (DFT)

Next, the frequency spectrum of the sound is computed for each MFCC window. This is done using the Discrete Fourier Transform (DFT) that is defined in equation 2.3 [14]:

$$X_m(k) = \sum_{t=0}^{T-1} S'_m(t) e^{-\frac{2\pi i t k}{T}} \quad , \quad k = 1, 2, 3, \cdots, T-1, \tag{2.3}$$

where k is the index of the spectrum, T is the total number of samples in $S_m(t)$, $X_m(k)$ represents the DFT of $S'_m(t)$ at frequency, $f = k \cdot F_s/T$ [20]. F_s is the sample rate of the audio.

Mel filter bank

The Mel filter is defined by expression 2.4 [21].

$$q = 2595 \log\left(1 + \frac{f}{700}\right), \tag{2.4}$$

where f corresponds to a frequency and q the frequency modulated to the Mel filter. The inverse expression is shown in equation 2.5.

$$f(q) = 700(10^{\frac{q}{2595}} - 1). \tag{2.5}$$

These equations are necessary to build the Mel filter bank. This is a set of triangular band-pass filters that act on the output of the DFT [14]. The transfer function of the filter can be described by equation 2.6 [21]:

$$V_q(k) = \begin{cases} 0 & k < f(q-1) \\ \frac{k-f(q-1)}{f(q)-f(q-1)} & f(q-1) \leq k < f(q) \\ 1 & k = f(q) \\ \frac{f(q+1)-k}{f(q+1)-f(q)} & f(q) < k \leq f(q+1) \\ 0 & k > f(q+1) \end{cases} . \tag{2.6}$$

Besides this, a normalizing factor for each filter is also calculated using equation 2.7 [14].

$$A = \sum_{k=0}^{K} |V_q(k)|^2, \tag{2.7}$$

where K is the total number of indexes in the DCT.

After these equations are defined, the Mel filter bank is applied using equation 2.8 [14].

$$B_m(q) = \frac{1}{A} \sum_{k=0}^{K} |V_q(k) X_m(k)|^2 \quad , \quad q = 1, 2, 3, \cdots, Q, \tag{2.8}$$

where Q is the number of triangular filters that build the Mel spectrum.

Discrete cosine transform (DCT)

For the final step in the calculation of the MFCC the logarithm of equation 2.8 is passed through a DCT, as is described in equation 2.9.

$$\mathrm{MFCC}_m(p) = \frac{1}{Q}\sum_{q=1}^{Q}\log(B_m(q))\cos\left(\frac{\pi p(2q+1)}{Q}\right) \quad , \quad p = 1, 2, 3, \cdots, P, \quad (2.9)$$

where p is the index of the MFCC and P is the number of MFCCs. P must always be less than Q.

2.2 Artificial Neural Networks

Artificial Neural Networks (ANNs) are designed to mimic the working process of a brain by having nodes (artificial neurons) that can communicate with each other using connections (artificial synapses), this makes ANNs a good solution to solve nonlinear problems [22].

The nodes are grouped in layers. In a simple, feed-forward ANN, there are three types of layers (see figure 2.2): the input layer, where the data is inserted; hidden layers, that add complexity and flexibility to the model; and the output layer, that shows the final result.

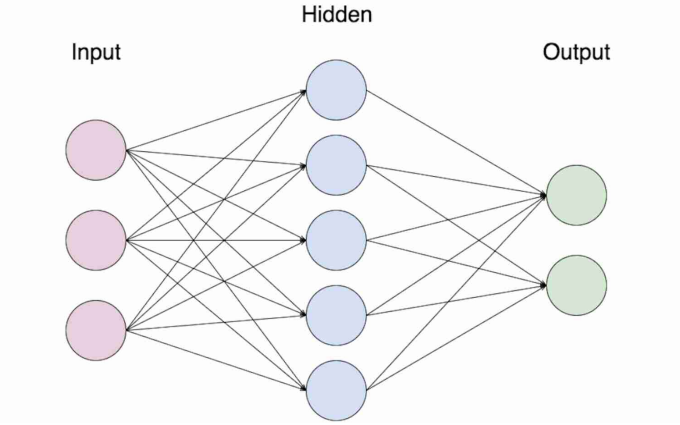


Figure 2.2: Architecture of a simple ANN [23].

Each node is defined by x_g^l, where l represents the index of the layer and g the index of the node [1]. Each x_g^l has a set of inputs x_g^{l-1}, that are the outputs from the previous layer nodes. Each node of the previous layer is multiplied by a weight $w_{d,g}^{j,l}$, where j represents the index of the previous layer and d the index of the corresponding node

[1] If g is not explicitly mentioned, it is referring to all nodes in that layer. If l is 1 then it is referring to the input nodes.

of the previous layer, [2] and summed together, as can be seen in equation 2.10. In the case of the input layer, its inputs are the data.

$$I(x_g^l) = \sum_{d=1}^{D} w_{d,g}^{j,l} \cdot x_d^j, \tag{2.10}$$

where D is the number of neurons in the previous layer.

A bias b_g^l is added, as can be seen in formula 2.11 to adjust the ANN for a possible dataset offset.

$$z(x_g^l) = I(x_g^l) + b_g^l. \tag{2.11}$$

This is then modulated by an activation function $U(z(x_g^l))$ [3] [24]. These steps generate the output of a single node.

This output is then transmitted to the nodes in the next layer by the connections until it reaches the final layer.

In this work, the output layer will present the label using one-hot encoding [25], a method in which the label is an array with the same size as the number of categories. Each category has a corresponding index, so the array is all zeros except for the index of the category it corresponds to, being one in this case. This reduces correlation between categories when compared to other methods of presenting labels, such as decimal encoding.

After an ANN is assembled, it is necessary to optimize its weights and biases, so it can output the right information for each input [26]. This is done during the learning phase. In this work the ANN learned by supervised learning, in this process, an input is given to the ANN, from which an output is generated, this output is then compared to the expected one and an error is calculated with an error function. This error is backpropagated through the ANN (explained in subsection 2.2.3 *ANN optimization*) changing the weights and biases of the nodes and connections with the goal of minimizing the overall error.

To build and train the ANN, the Keras [27] library with the TensorFlow [28] backend from the Python programming language [16] was used.

2.2.1 Activation functions

The activation functions modulates the output of each node, so the ANN can learn non-linear relations between the input and the output [29]. Usually the same activation function is applied to all nodes of a layer, and it can change from layer to layer depending on the goals for each one.

[2] If g and/or d are not explicitly mentioned, it is referring to all weights in that layer.

[3] This will be explained in subsection 2.2.1 *Activation functions*

In this work, the following activation functions were used:

Linear Activation Function

This activation function is defined in expression 2.12 and its graphical representation is shown in Figure 2.3.

$$linear(z) = z. \tag{2.12}$$

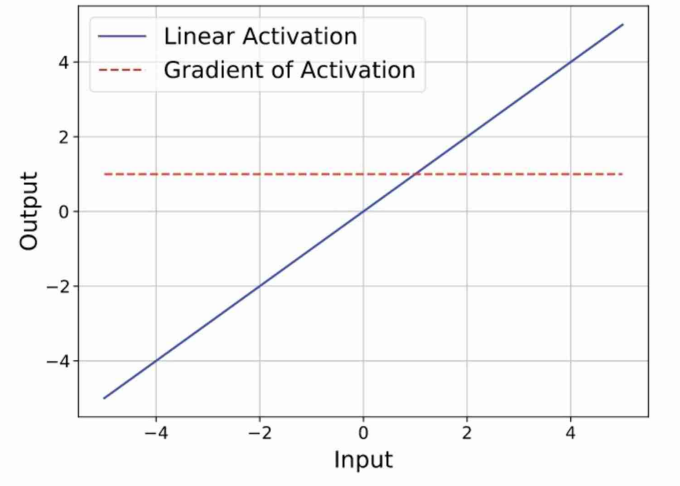


Figure 2.3: Linear activation and its gradient [29].

As it keeps the linearity of the ANN, it was used in the last layer of all tested networks. In these layers, the goal was to transfer information, not to learn from it.

Rectified Linear Unit (ReLU) Activation Function

This activation function is defined in expression 2.13 and its graphical representation is shown in figure 2.4.

$$ReLU(z) = \begin{cases} z & if \quad z \geq 0 \\ 0 & if \quad z < 0 \end{cases} \tag{2.13}$$

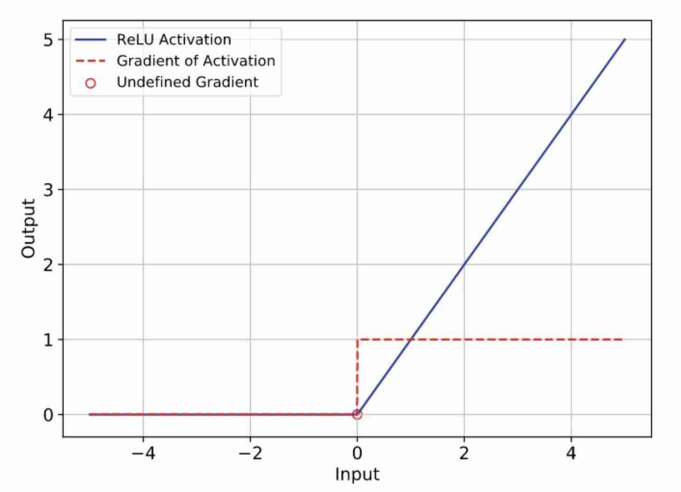


Figure 2.4: ReLU activation and its gradient [29].

It is the most used activation function because it promotes gradient stability, is computationally cheaper, and can create a sparse ANN [30], improving its generalization. A sparse ANN is possible by the creation of dead nodes. In some circumstances, this can be a disadvantage [29].

Dead nodes refer to nodes that output zero for all inputs. This occurs when a node encounters a situation where its weighted sum of inputs falls below zero for all training data [31]. These nodes cannot learn as its gradient is also zero (see figure 2.4) as such they decrease the ANN efficiency by consuming computational resources while not providing any useful contribution.

Sigmoid and Softmax Activation Functions

The Sigmoid activation function is defined in expression 2.14 and its graphical representation is shown in figure 2.5.

$$Sigmoid(z) = \frac{1}{1 + e^{-z}}. \tag{2.14}$$

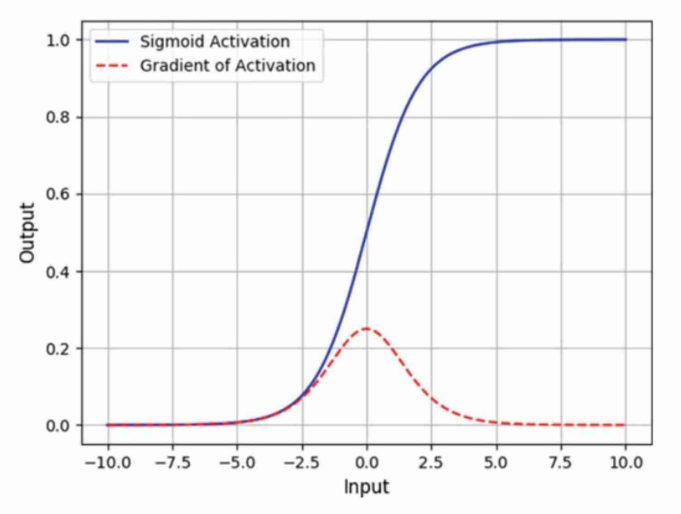


Figure 2.5: Sigmoid activation and its gradient [29].

This function was designed for binary classification, so it gives a result that is between 0 and 1 that can be interpreted as a probability [29]. The main problem of the sigmoid activation is that, as can be seen in figure 2.5, the greater the input absolute value closer to zero the gradient becomes. This learning gradient becomes smaller as it propagates backward through the ANN during backpropagation, this is known as the vanishing gradient problem, and it can stall the ANN learning process.

The softmax activation is an implementation of the sigmoid for a vector instead of a scalar [29]. It is defined in expression 2.15.

$$Softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad , \quad i = 1, \cdots, K \quad , \quad \mathbf{x} = (z_1, \cdots, z_K). \tag{2.15}$$

Tanh (Hyperbolic Tan) Activation

The tanh activation is defined in expression 2.16 and its graphical representation is shown in figure 2.6.

$$Tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}. \tag{2.16}$$

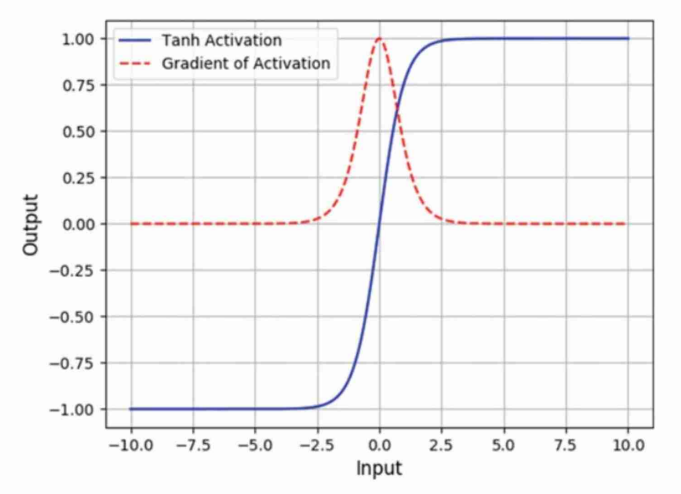


Figure 2.6: Tanh activation and its gradient [29].

Like the sigmoid it is symmetric, but converges faster as the maximum gradient activation of the function is one, making it four times greater than the one of the sigmoid [29].

2.2.2 Other types of layers

Besides the activation layers, there were also two other types of layers that were applied in this work. These layers are only used in the training phase of the ANN.

Batch Normalization

The batch normalization layer applies a transformation to the inputs of the layer by subtracting the mean and dividing the standard deviation of their values. Empirical tests prove that this normalization speeds up and stabilizes the learning process and allows for higher learning rates [32]. However, the application of this layer requires larger batches and is more computationally expensive.

Dropout

In this layer, each node has a probability of being deactivated, along with their incoming and outgoing connections, during the training phase of the batch [33]. This prevents them from co-adapting to each other, minimizing the problem of overfitting (explained in subsection 2.2.3 *ANN optimization*) and creates several sparse NN inside the main one, improving generalization.

2.2.3 ANN optimization

As it was described in, section 2.2 *Artificial Neural Networks* the value of each node (including output nodes) x_g^l is defined by the equation 2.17.

$$x_g^l = U\left(\left(\sum_{d=1}^{D} w_{d,g}^{j,l} \cdot x_d^j\right) + b_g^l\right). \tag{2.17}$$

For the purpose of ANN optimization, $w_{d,g}^{j,l}$ and b_g^l are the trainable parameters of the layers [34]. For the initialization of the ANN, these parameters are filled with small random values. Then the ANN gradually adjusts these parameters during its training phase to give more accurate results.

To adjust the trainable parameters, the first step is to choose an input x^1 and compute the ANN output $y^{predicted}$. The output is compared to the expected value $y^{expected}$ [34]. This comparison is quantified by the usage of the error function. In this work, the chosen function was the mean-squared error [35] that is defined by equation 2.18.

$$J = \frac{1}{Y}\sum_{g=1}^{Y}\left(y_g^{predicted} - y_g^{expected}\right)^2, \tag{2.18}$$

where J is the value of the error, Y is the number of output nodes.

$y^{predicted}$ is a function of the trainable parameters (see equation 2.17) as such the error function can be described as $J(\theta)$ where θ represent the full set of trainable parameters (weights and biases).

To train the ANN, the partial derivative of each parameter θ is calculated. This set of partial derivatives is the error function gradient and can be represented by $\nabla_\theta J(\theta)$. To compute the partial derivative the backpropagation algorithm is used, which utilizes the chain rule to do the calculations. This can be done because all operations used in an ANN are differentiable [34].

Then its parameters (θ) must be updated in the opposite direction of its gradient [36]. How much each parameter is optimized is given by the optimization function and by the learning rate η. In this work, two optimization functions were used: Stochastic Gradient Descent (SGD) (explained in section 2.2.3 *Stochastic Gradient Descent*) and Adam (explained in section 2.2.3 *Adam*).

Usually the optimization of the parameters of the error function is not done for each data point, but instead the losses for a small set of data points are calculated and before errors being back-propagated, this set is called a batch [37].

These optimizations batches are done across the data until the end of the dataset is reached. This set of batches that encompasses the dataset is called an epoch.

One of the problems that is encountered in the parameter optimization is overfitting [38], which is when the neural network does not generalize what it learned from the training data to unseen data.

To monitor the overfitting of the network, the data is divided in three sets, the validation dataset, the training dataset and the test dataset. The learning process is done only with the training dataset and after each epoch both datasets are feed to the network and the accuracies and losses measured. This creates four values, train accuracy, train loss, validation accuracy, validation loss. The difference between these two sets of measurements (difference between accuracies and between losses) is correlated to overfitting because it provides a way to inspect how the ANN preforms with seen data (training dataset) versus unseen data (validation dataset), if the performance values for the seen data continue to improve and are significant better than the ones for the unseen data, which may even degrade, then the ANN is memorizing the validation dataset instead of learning data features. The test dataset is only feed to the final ANN to prove its quality.

In this work there was a limited amount of data available, so to have a relevant enough dataset for the training phase the test dataset was only used in a few ANNs. In some ANN where the goal was only to have a proof of concept and the data was even more limited, the validation dataset was not used in favor of only using the test dataset.

Stochastic Gradient Descent

There are several variations of the SGD, in this work it was used the mini-batch SGD. In it, the parameters are updated for each batch of the training dataset. The first step in this method is to compute the average of the loss function gradient in the batch, as is defined in equation 2.19 [37].

$$\Delta\theta = \frac{1}{s}\sum_{i=0}^{s} \nabla_\theta J(\theta), \tag{2.19}$$

where s the size of the batch.

Then the parameters are updated as is defined in equation 2.20.

$$\theta_{new} = \theta - \eta \cdot \Delta\theta. \tag{2.20}$$

Performing updates on each batch reduces fluctuations in the optimization of the error function, which improves convergence. [36]

Adam

The Adaptive Moment Estimation (Adam) is an optimizer that computes adaptive learning rates for each parameter it updates. To do this, it keeps an exponentially decaying average of past gradients m_t and an exponentially decaying average of past squared gradients v_t

These decaying averages are calculated as defined in equations 2.21 and 2.22.

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla_\theta J(\theta). \tag{2.21}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla_\theta J(\theta))^2 . \tag{2.22}$$

The parameters m_{t-1} and v_{t-1} are initialized as zeros so in the first iterations there is a bias towards zero, to counteract this the values are bias-corrected using equations 2.23 and 2.24.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \tag{2.23}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}, \tag{2.24}$$

where β_1^t and β_2^t are β_1 and β_2 raised to the power of t, the iteration number.

These averages are used to update the parameters of the error equation according to equation 2.25.

$$\theta_{new} = \theta - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t. \tag{2.25}$$

In these equations β_1, β_2 and ϵ are values defined by the user that can be changed to adapt to different types of data. The authors of this optimizer propose 0.9 for β_1, 0.999 for β_2 and 10^{-8} for ϵ [39].

2.2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [40] are a type of ANN that applies a convolution between the data and a set of filters to identify patterns in the data, reducing the need for preprocessing. They have three types of layers: convolutional, pooling, and fully-connected layers. They provide superior performance with image and audio classification, but are also more computationally expensive.

Convolutional layer

This layer extracts features from the input data, for this it makes a convolution between the input and a feature detector (also known as kernel or filter) that is an array with the same dimensions (e.g., 1D for sound, 2D for gray scale images or 3D for color images) as the input but a smaller size. Then an element-wise nonlinear activation function (usually ReLU, sigmoid or tanh) is applied and the feature map created [41].

There can be multiple kernels and each one creates their own feature map [41].

Pooling layer

This layer reduces the dimensions of the data by applying an operation that reduces the number of inputs of the kernel to a single value. The two most used operations are [40]:

- Max pooling: The kernel selects the sample with the maximum value from the feature map.
- Average pooling: The kernel calculates the average from the feature map.

Due to time constrains, only the max pooling layer was used in this work.

These layers are usually in between convolutional layers. By stacking several convolutional and pooling layers, higher-level features can be extracted [41].

Although information is lost in this layer, it helps to reduce complexity, improving efficiency, and decreasing the risk of overfitting [40].

Fully-Connect layer

This layer, uses as input the features obtained by the previous layers and acts on them to extract information. For this, an ANN is used as it was described in section 2.2 *Artificial Neural Networks*.

2.2.5 Recurrent Neural Networks

RNNs are a type of ANN that introduce the notion of time to a model by taking into account values derived from the previous state of the network to make a classification of the current data point [42]. This type of analysis is mostly used for solving temporal problems such as language translation or speech recognition.

For a data point, x^1, two inputs are given to the RNN, the previous time steps [4] hidden values $h_{\tau-1}$ and x^1, and the network will give two outputs $y^{predicted}$ which represent the classification and h_τ that are the hidden values that encode the network's state.

[4]Time steps are represented by the letter τ.

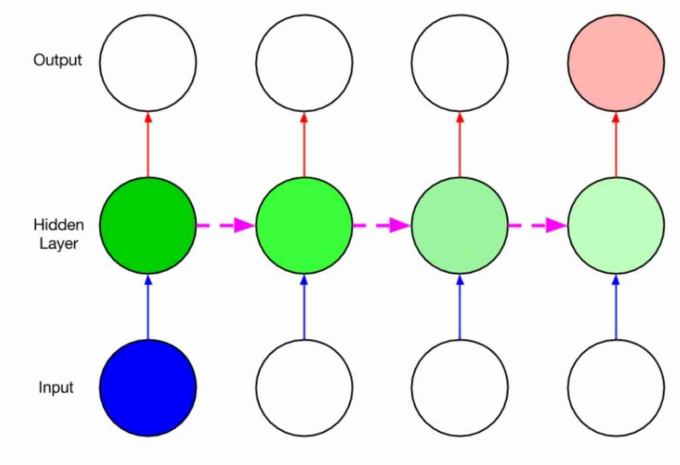


Figure 2.7: Architecture of an RNN with one input, one output and one recurrent hidden unit with its propagation through the time steps [42].

RNNs tend to run into two problems, exploding and vanishing gradients. These issues stem from the size of the gradient, which is the slope of the loss function. When this gradient is small it tends to become smaller, decreasing how much the weights are updated, thus preventing the algorithm from learning. When it is too big, the model becomes unstable. The weights become too large and will eventually cause a number overflow. One of the solutions is to truncated backpropagation through time (TBPTT), setting a maximum number of time steps from which the error is propagated [42].

Some RNNs do not have typical nodes, but instead, memory cells that regulate the flow of information in the network and through the iterations. In this work, only one type was used, the Long short-term memory cell.

Long short-term memory

The Long short-term memory (LSTM) cells were developed as a solution to the problem of vanishing gradients. That is, if the input that is influencing the current one is not in the recent past, the RNN may not be able to make the correct prediction. These memory cells store information through the network iterations accessing them when needed, mitigating the vanishing gradient problem. In this model the hidden layers do not have nodes but LSTM cells that have a self connected memory connection and three multiplicative gates: an input gate, an output gate, and a forget gate [43].

The gates receive inputs from the network, they are summed as is described in equation 2.10 and then modulated by an activation function, usually a sigmoid (see section 2.2.1 *Sigmoid and Softmax Activation Functions*). A gate with a value of 1 is an open gate, and a gate with a value of 0 is a closed gate. The value of the gate value is then multiplied by the input in the case of the input gate, by the output in the case of the output gate or by the recurrent unit in the case of the forget gate [43].

Each memory cell is made from six elements that can be seen on figure 2.8 and described on the list below.

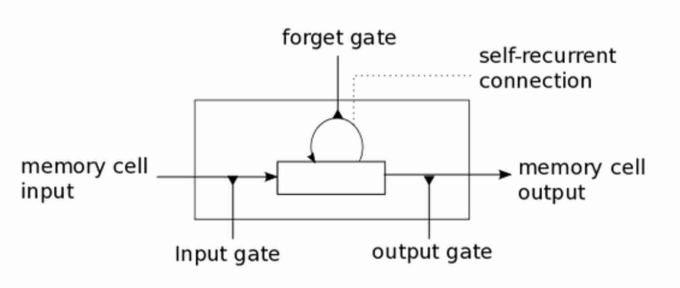


Figure 2.8: Representation of a LSTM memory cell [44].

- Memory cell input, that receives the data from the previous layer and previous hidden layer, and applies the transformation described in section 2.2 *Artificial Neural Networks*.
- Input gate, that is multiplied by the value of the input node.
- Self-recurrent connection, a node with linear activation that has as inputs the value from the input gate and a recurrent connection to itself. This state is usually passed through an activation function.
- Forget gate, that is used to erase, if necessary, the contents saved by the internal state.
- Output gate, that is multiplied by the value of the internal state.
- Memory cell input, that is the memory cell output.

2.2.6 Metrics

The quality of the developed ANNs was evaluated using the following metrics: Loss, Confusion Matrix, Accuracy, Precision and Recall.

- Loss: This measures the inconsistency between the predicted output of the ANN and the expected output. Its value is given by the error function (subsection 2.2.3 *ANN optimization*)
- Confusion Matrix: This is a table that provides a visual representation of an ANN's performance by presenting the predicted categories compared to the actual categories. In the case of a single-output ANN, the confusion matrix is represented by a 2×2 matrix, with the categories described in table 2.1.

Total population	Positive	Negative
Positive	True Positive (TP)	False Negative (FN)
Negative	False Positive (TN)	True Negative (TN)

Table 2.1: Single output confusion matrix.

- Accuracy: This is the proportion of correctly classified data out of the total number of instances in the dataset. It can be described by equation 2.26.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.26}$$

  It measures effectiveness of a model in making correct predictions.

- Precision: This metric is the proportion of true positive predictions out of all instances predicted as positive. It can be described by equation 2.27.

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{2.27}$$

  It measures the accuracy of positive predictions.

- Recall: This metric is the proportion of true positive predictions out of all positive instances. It can be described by equation 2.28.

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{2.28}$$

  Recall provides an indication of the ANNs ability to avoid false negative errors.

Chapter 3

Implementation

3.1 Urban Sound 8K

The first stage in this work was to gain the necessary knowledge to create and implement an ANN. So instead of directly using the data from the dummy eggs, it was used an already known and labeled dataset called Urban Sound 8K [45].

3.1.1 Dataset

The dataset has 8 732 sound samples, each lasting around 4 s. Each sound sample is labeled and belongs to one of ten categories, as is showed in table 3.1.

With this dataset, the goal was to build an algorithm that could distinguish the different types of sounds. To achieve this there were two parts, the feature extraction and the ANN. Both of them passed through several iterations until the most optimal configuration was found.

It was chosen to start with this dataset because all audio samples are already classified. In the dummy eggs the data is a raw stream of audio with no labels, by starting with a dataset that is already labeled it is possible to refine the classification algorithm in areas such as: type of features to extract, how to extract them, how much detail the features need, general shape of the ANN, types of layers, optimizers, among others.

This does not mean all the optimization work is done, both parts of this algorithm were later refined to adapt to the real data, but some optimization was done using this dataset.

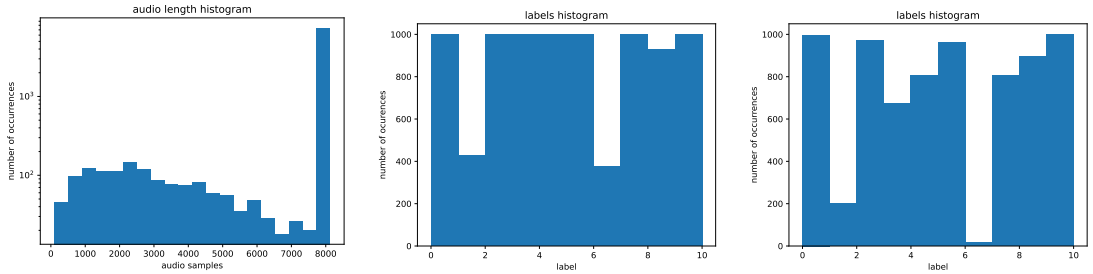
To achieve a greater similarity between the Urban Sound 8K dataset and the output data from the dummy eggs, the sampling rates were matched by down sampling the Urban Sound 8K dataset from a sample rate of 44 100 Hz to 2 016 Hz.

Not all of the sound files had the same length neither were equally distributed among the categories. These disparities do not critically affected the work, but it

was necessary to be aware of them to better understand the results.

From the 8 732 sound files in the dataset, only 7 327 had a usable audio length. The length was considered usable if it had at least, 8037 samples. The reasoning for this value is explained in subsection 3.1.2 *Feature Extraction*.

Figure 3.1(a) shows the distribution of the length of the audio files in samples. Figure 3.1(b) shows the discrepancies between the number of files per category, and figure 3.1(c) shows that they increase even further when only audio files with usable audio length are considered. A more detailed representation of the distribution of files per category can be found in table 3.1.



(a) Distribution of the length of the files in samples

(b) Distribution of files per category

(c) Distribution of usable files per category

Figure 3.1: Distribution of files.

Category	Description	Number of files	Number of usable files
Category 0	Air conditioner	1000	997
Category 1	Car horn	429	203
Category 2	Children playing	1000	969
Category 3	Dog barking	1000	675
Category 4	Drilling	1000	805
Category 5	Engine idling	1000	961
Category 6	Gun shot	374	16
Category 7	Jackhammer	1000	804
Category 8	Siren	929	897
Category 9	Street music	1000	1000

Table 3.1: Number of files per category.

The variations between categories explain why some of them trained better than others. An ANN trained with larger datasets could more easily distinguish between features and statistical variations. As such, ANNs trained with larger datasets can give better results, that is why the elements in category nine with 1000 elements can train better that the ones on category six that has only 16.

This difference in category size did not put restrains on the goals of this part of

the work, as there were still seven categories with more than 800 samples, which was sufficient to develop and test the ANN.

3.1.2 Feature Extraction

Several feature extraction methods, effectively consisting of variations in the MFCC processing, were tested. To compare them, these features were used as inputs to the ANN developed in section 3.1.3 *ANN model 2* only changing the input layer, so it matches the number of features extracted. This way, the ANN was the classifier that measured the quality of the feature extraction method, outputting higher accuracies and lowers losses for methods that extract more relevant data.

In this work, both the ANN and the features' extractor were developed at the same time. In this dissertation, it was chosen to present the information in separate sections to simplify the comparison of the implemented methods.

The data was split in an 80/20 fraction, where 80% of the data were used for training and 20% for validation. The ANN had the shape presented in table 3.2.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	x	yes
2	Dense	ReLU	250	yes
3	Dense	Softmax	250	yes
4	Output	Linear	10	no

Table 3.2: Architecture of ANN used for measuring the quality of the features, where x will be the number of features.

The optimizer was the SGD (see section 2.2.3 *Stochastic Gradient Descent*) with a 0.1 learning rate. It was set to ran for 8000 epochs in batches of 32 elements.

All feature extraction methods tested were operations done on the MFCC block. These were calculated in the following order (see flowchart 3.2): first a 188 sample window was selected, from here a user selected number of MFCCs were extracted, then a stride of 47 samples was applied, this cycle repeated until the window went past the end of the file.

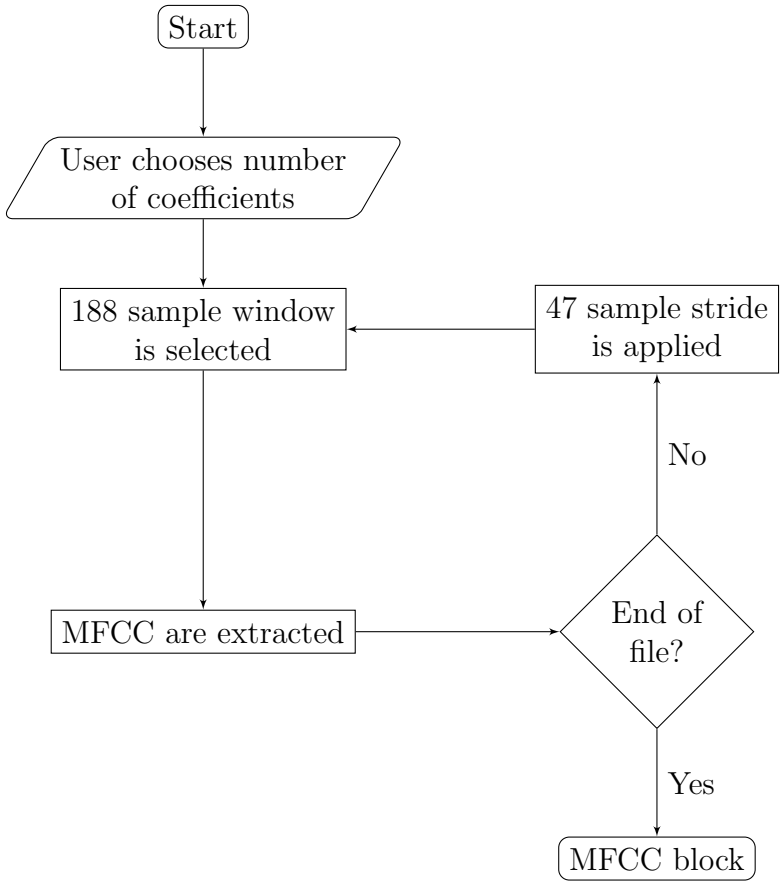


Figure 3.2: MFCC extraction from audio.

One consequence of this approach was that in the last three MFCC, less than 188 samples are extracted from the file, so the window was zero padded. To avoid this, those windows were removed.

In this work, the term MFCC referred to a single column of coefficients extracted from one window, and MFCC block refers to a set of MFCCs that can be represented in a matrix in which the rows corresponded to the coefficients and the columns to the windows across the time steps.

The Librosa default values for the window and the stride were used. With these values, the window had a length of 93 ms and the stride had a length one fourth the window size. At a sample rate of 2016 Hz it corresponded to a window of 188 samples and a stride of 47 samples.

Before the audio went through this process, the number of samples was reduced, to equal a multiple of the length of the stride. This was a way to enforce that all files have the same number of windows and all the windows have the same number of samples.

A 4 s audio at a sampling rate of 2016 Hz has 8064 samples. The highest multiple of 47 that is less than 8064 is 8037, so all files were reduced to this value.

Extracting MFCCs from these audio files with 8037 samples each, with windows of 188 samples at a stride of 47 samples, gave 171 MFCCs. As the last 3 MFCCs were removed, the final number of MFCCs was 168.

A summary of the ANN models trained using different MFCCs configurations can be found in table 3.3.

Model	Number of MFCCs	Averages	Standard deviations	Size of window	Number of features	Validation loss	Validation accuracy
Model 1	40	yes	no	188	40	0.0163	88.74%
Model 2	30	yes	yes	188	60	0.0143	90.86%
Model 3	30	no	no	188	5040	0.0658	53.00%
Model 4	30	yes	no	376	40	0.0151	89.96%

Table 3.3: Summary of results given by the ANN.

Feature extraction model 1

For this extraction method, it was chosen to extract 40 coefficients. From the resulting MFCC block, the average across the time steps was calculated, giving a total of 40 feature per file. The extraction process is illustrated in the flowchart 3.3.

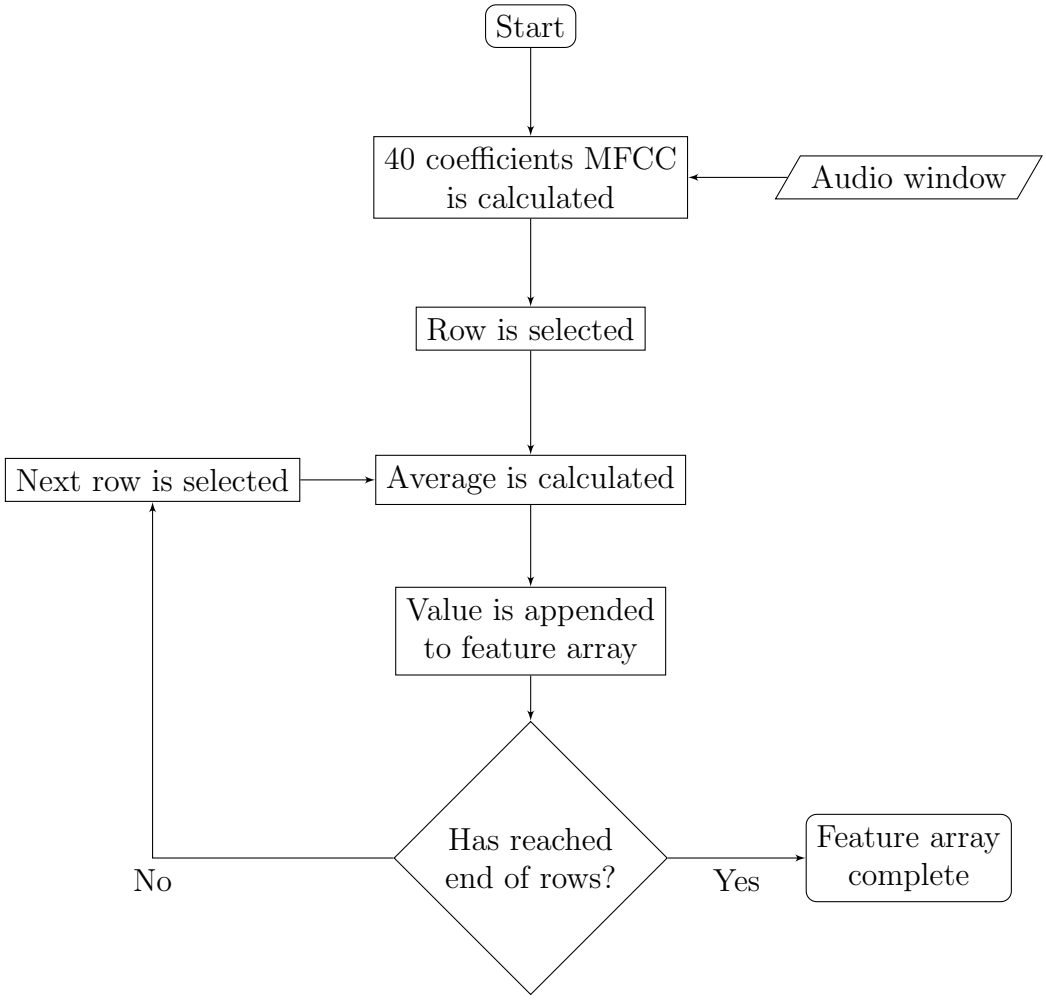
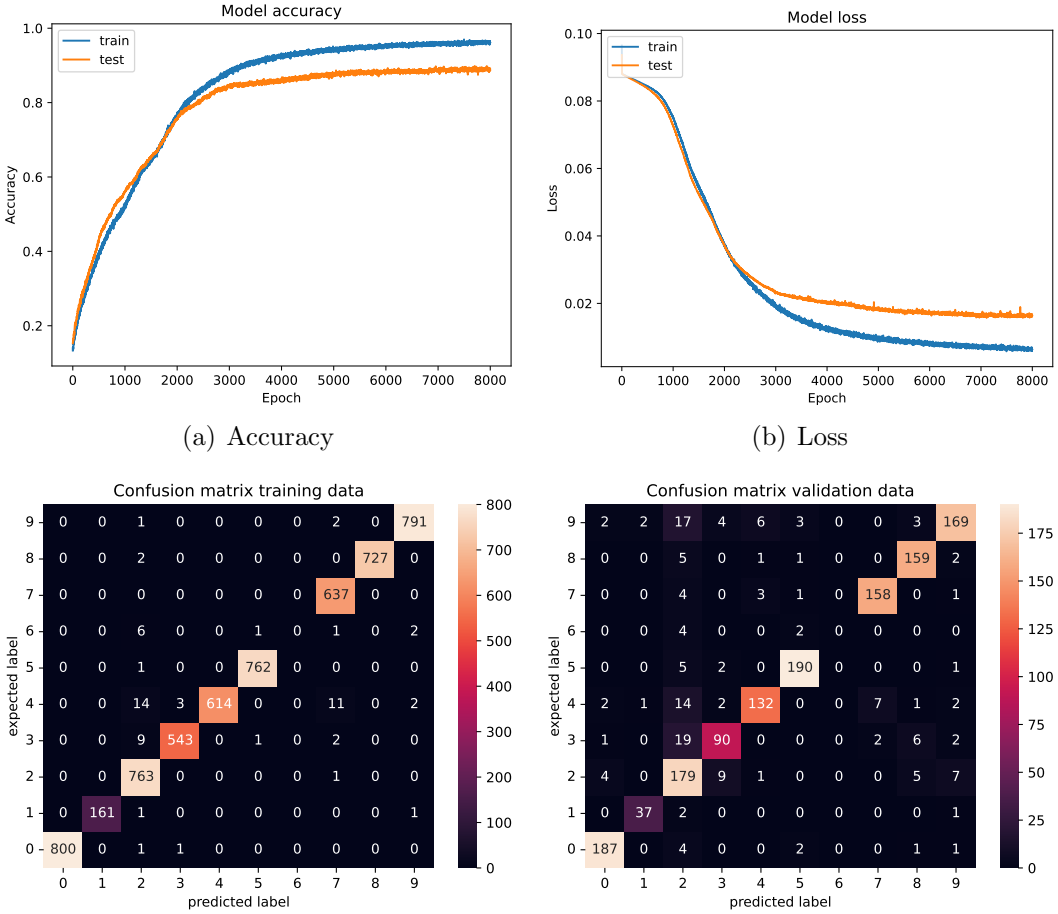


Figure 3.3: Block diagram for the feature extraction model 1.

The ANN trained with this data gave the results displayed on figure 3.4:



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.4: ANN model 2 evolution with the feature extraction model 1.

After training the loss function of the ANN had a value of 0.0064, the accuracy was 96.18%, the validation loss function had a value of 0.0163 and the validation accuracy was 88.74%.

The confusion matrix reveals that the ANN incorrectly classified categories three, four, and nine as category two, indicating a potential spectral similarity between these categories. Additionally, no instances were correctly identified as category six, which can be attributed to the limited number of entries in that category.

The ANN trained with this extraction method had a validation accuracy of 88.74% that is lower than the goal of 90% making it not suitable for this work (see subsection 1.3.1 *Objectives*). The validation and train accuracies have a difference of 7.44 percentage points.

Feature extraction model 2

This configuration used as inputs the averages and standard deviations across a MFCC block of 30 coefficients. This gave each sound sample 60 features. The extraction process can be seen in the flowchart 3.5.

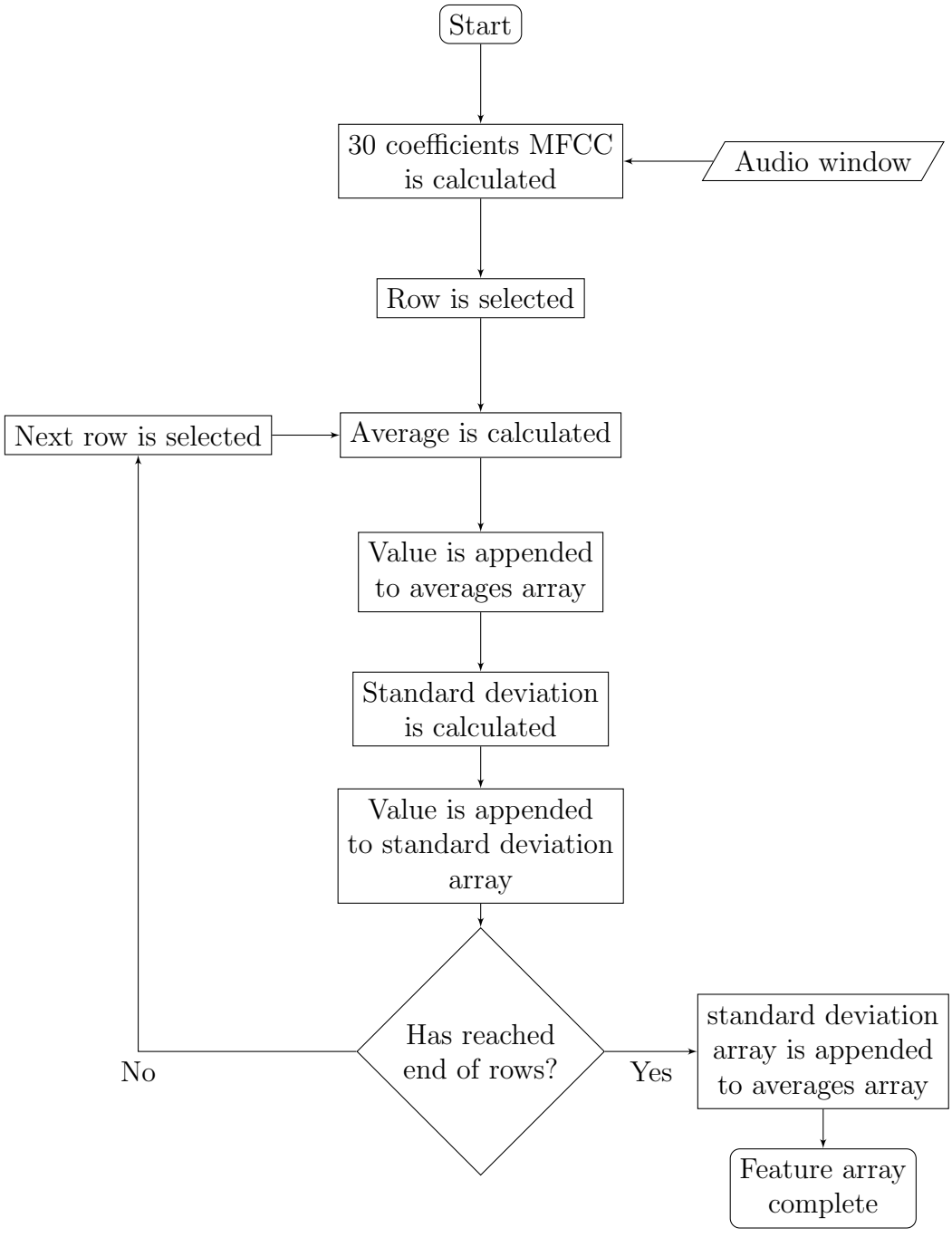
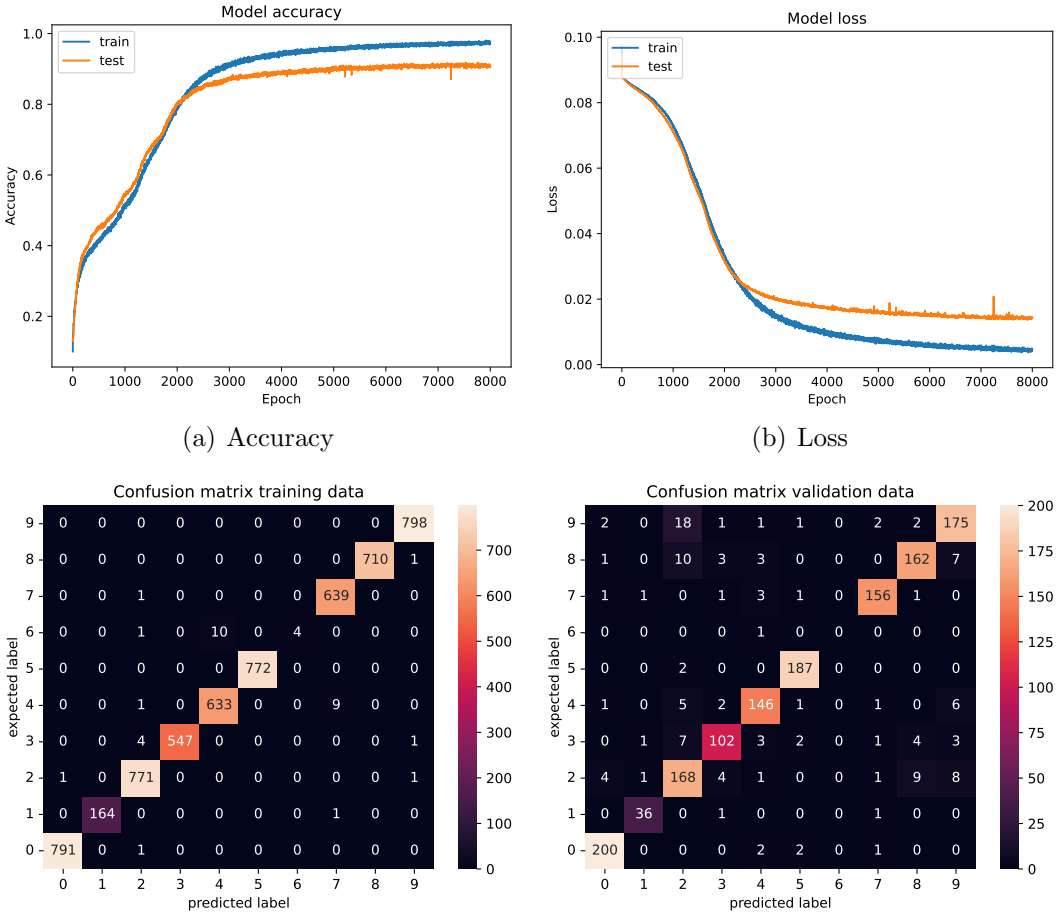


Figure 3.5: Block diagram for the feature extraction model 2.

This configuration gave the results that can be seen on figure 3.6.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.6: ANN model 2 evolution with feature extraction process 2.

After training the loss function of the ANN had a value of 0.0048, the accuracy was 97.17%, the validation loss function had a value of 0.0143 and the validation accuracy was 90.86%.

In this ANN, the confusion matrix reveals the categories eight and nine were mistakenly identified as category two, also indicating a spectral similarity.

This set of features had good results. Both the train accuracy and validation accuracy had values over 90%. The accuracies had a 6.31 percentage points difference.

Feature extraction model 3

Another test was done using all the entire MFCC block with 30 coefficients as the input. This gave all files 5040 features. As such, more information was given to the ANN. The extraction process can be seen in the flowchart 3.7.

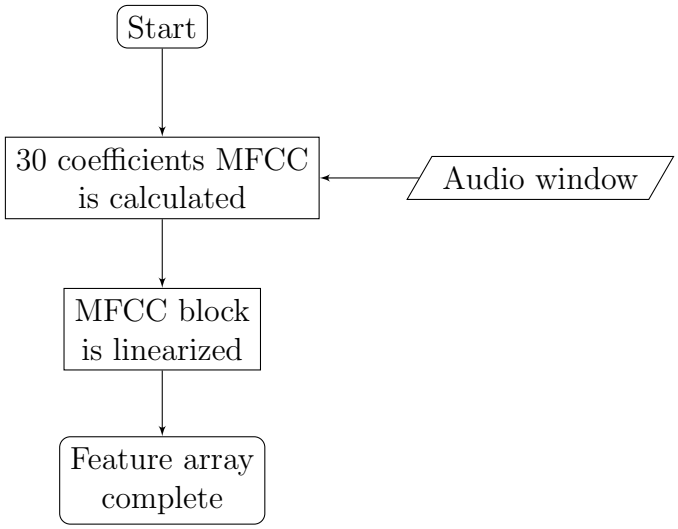
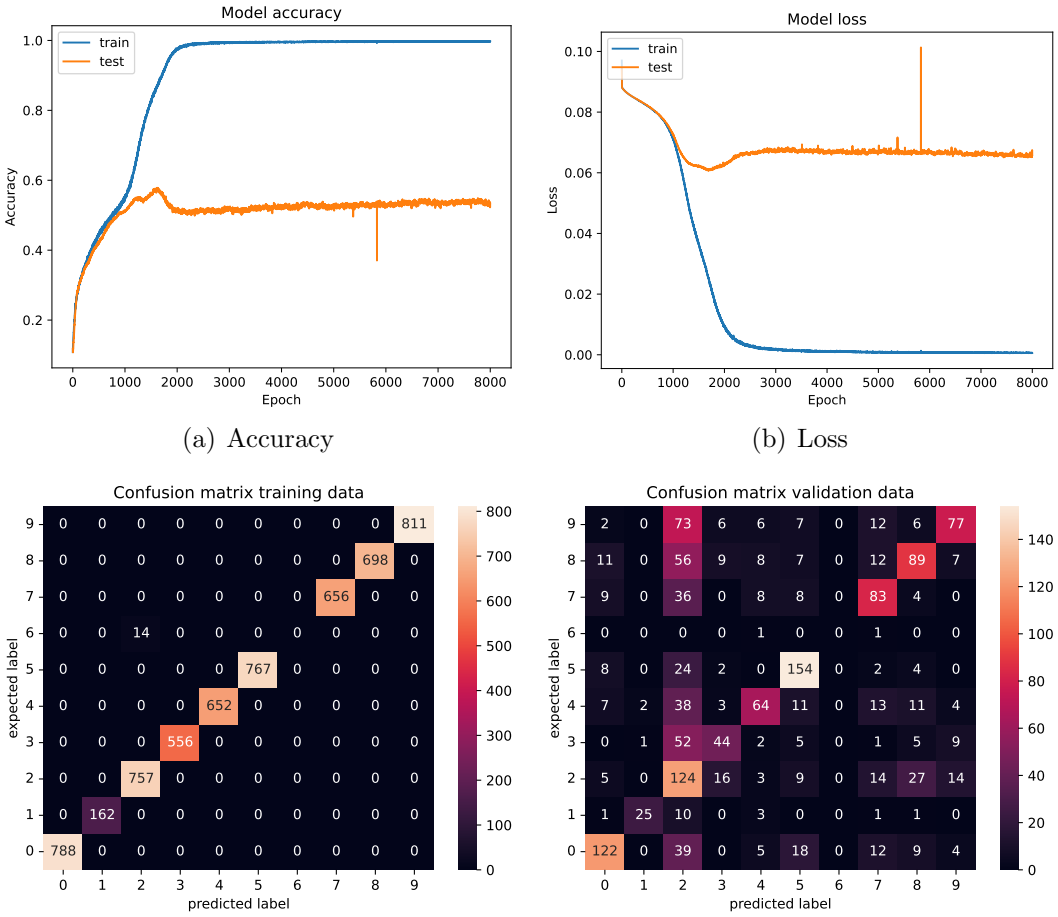


Figure 3.7: Block diagram for the feature extraction model 3.

This configuration gave the results that can be seen on figure 3.8.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.8: ANN model 2 evolution with feature extraction process 3.

After training the loss function of the ANN had a value of 0.0006, the accuracy was 99.71%, the validation loss function had a value of 0.0658 and the validation accuracy was 53.00%.

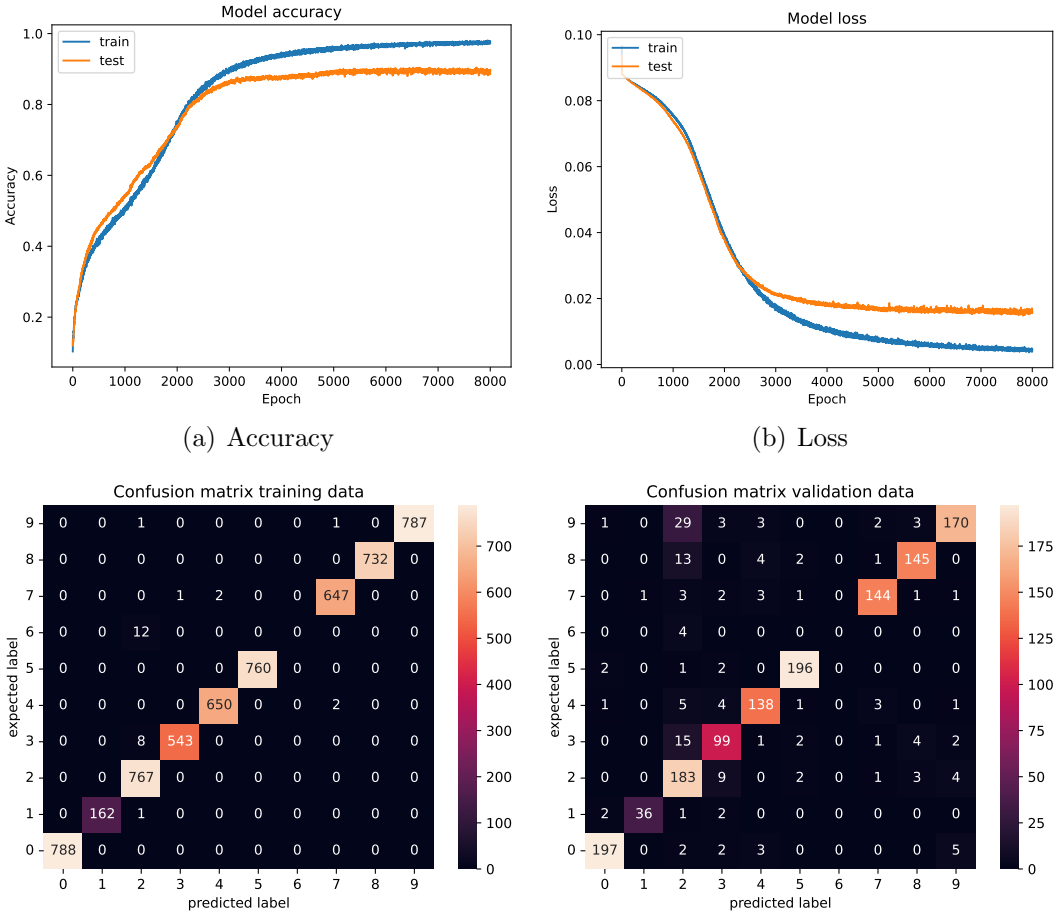
The learning curves indicate that the optimal performance of the ANN was achieved around the 1 500 th epoch, and the network did not show further improvement beyond the 2 000 th epoch. Moreover, the confusion matrices show that the ANN overfitted to the data. While the training dataset had only misidentified instances in category six (with the lowest number of instances), the validation dataset had numerous misclassifications.

The validation accuracy of this ANN was 37 percentage points below the set goal, and the train accuracy was almost 100%. This means the ANN overfitted and memorized the data instead of learning patterns from it, this made it not a suitable solution for this problem.

Feature extraction model 4

In the next test, the size of the window was doubled from 188 samples to 376 samples, as was the stride from 47 to 94 samples. One of the consequences of this approach is that the size of the MFCC matrix reduced in half from 168 to 84.

The ANN was trained using the method from section 3.1.2 *Feature extraction model 2* and gave the results that can be seen on figure 3.9.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.9: ANN model 2 evolution with feature extraction process 4.

After training the loss function of the ANN had a value of 0.0051, the accuracy was 96.78%, the validation loss function had a value of 0.0151 and the validation accuracy was 89.96%.

The confusion matrices highlight that this ANN experiences similar issues as the previous ones. It consistently misclassifies categories three, six, eight, and nine as category two.

These results were less accurate than the ones got with the 188 sample window. The lack of improvement is a sign that the extra information given by the increase in the audio samples is not relevant to the features that distinguish the categories.

From these tests it was concluded that the best features to extract were the 30 averages and 30 standard deviations with the 188 sample window as it was the only set of features that gave accuracies over 90% in both the train and the validation dataset.

3.1.3 Artificial Neural Network

In this section, several configurations of ANNs were tested using the Urban Sound 8K dataset. The input were the features extracted according to method described in section 3.1.2 *Feature extraction model 2*. The output was a ten element array displaying the categories in one-hot encoding.

In all tests the data was split in an 80/20 fraction, where 80% of the data was used for train and 20% for validation, no test dataset was used for the reasons given in the subsection 2.2.3 *ANN optimization*. All trains ran for 8000 epochs.

ANN model 1

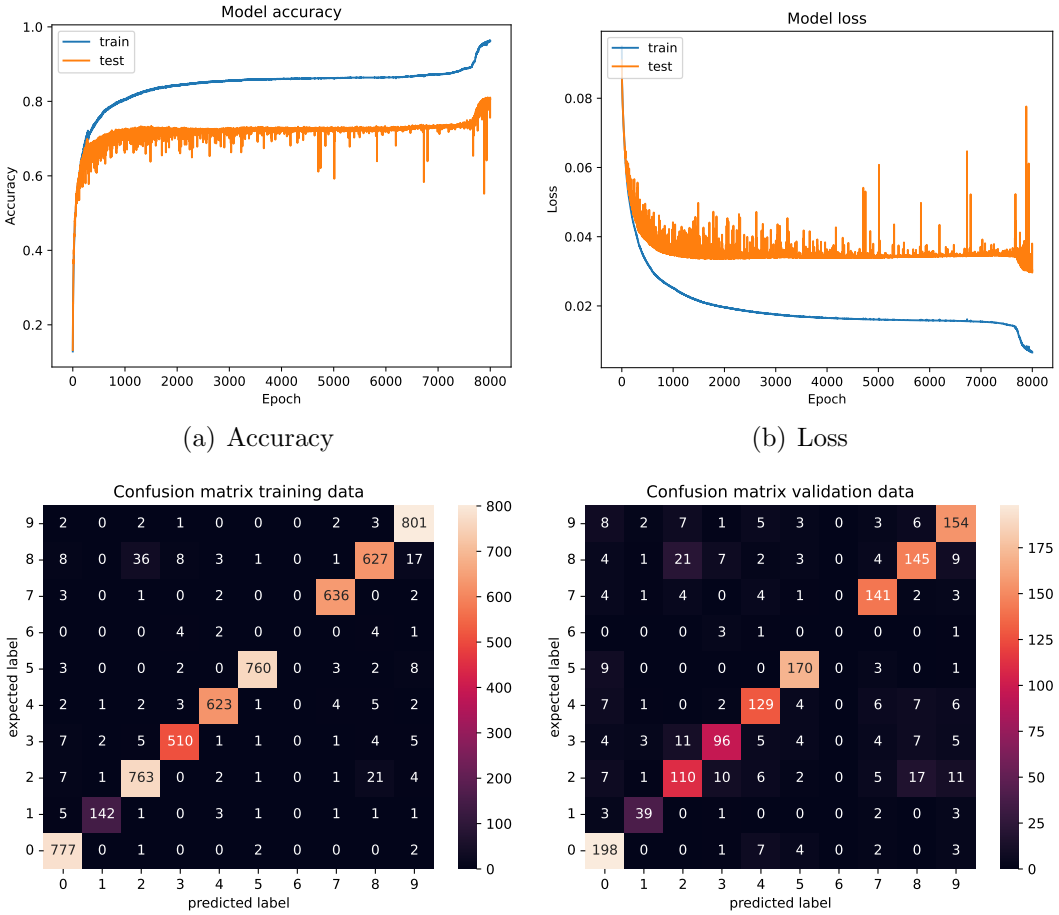
The first ANN that was made had the configuration presented in table 3.4.

Layer index	Layer type	Activation function	Number of nodes
1	Input	ReLU	60
2	Dense	ReLU	250
3	Dense	Softmax	250
4	Output	Linear	10

Table 3.4: Architecture of ANN model 1.

The results were obtained using the SGD optimizer with a 0.1 learning rate.

Training with this network gave the results that can be seen on figure 3.10.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.10: ANN model 1 evolution.

After training the loss function of the ANN had a value of 0.0066, the accuracy was 96.14%, the validation loss function had a value of 0.0303 and the validation accuracy was 80.63%.

The confusion matrices consistently reveal the misidentification, primarily focused on category two.

In this model, the learning curves of the accuracy and loss had several fluctuations this was a suboptimal learning process, as the ANN had large setbacks that reduce its quality.

During the final epochs of the learning phase, the ANN exhibited a sudden improvement in its performance. This occurrence suggests that, by chance, the ANN might have found another local minimum in the error function. Ideally, the training should have continued for more epochs to allow the model to stabilize at its lowest point. However, this was not done because it did not shorten the gap between the train accuracy and the validation accuracy as such it was deemed non-critical for the purposes of the work.

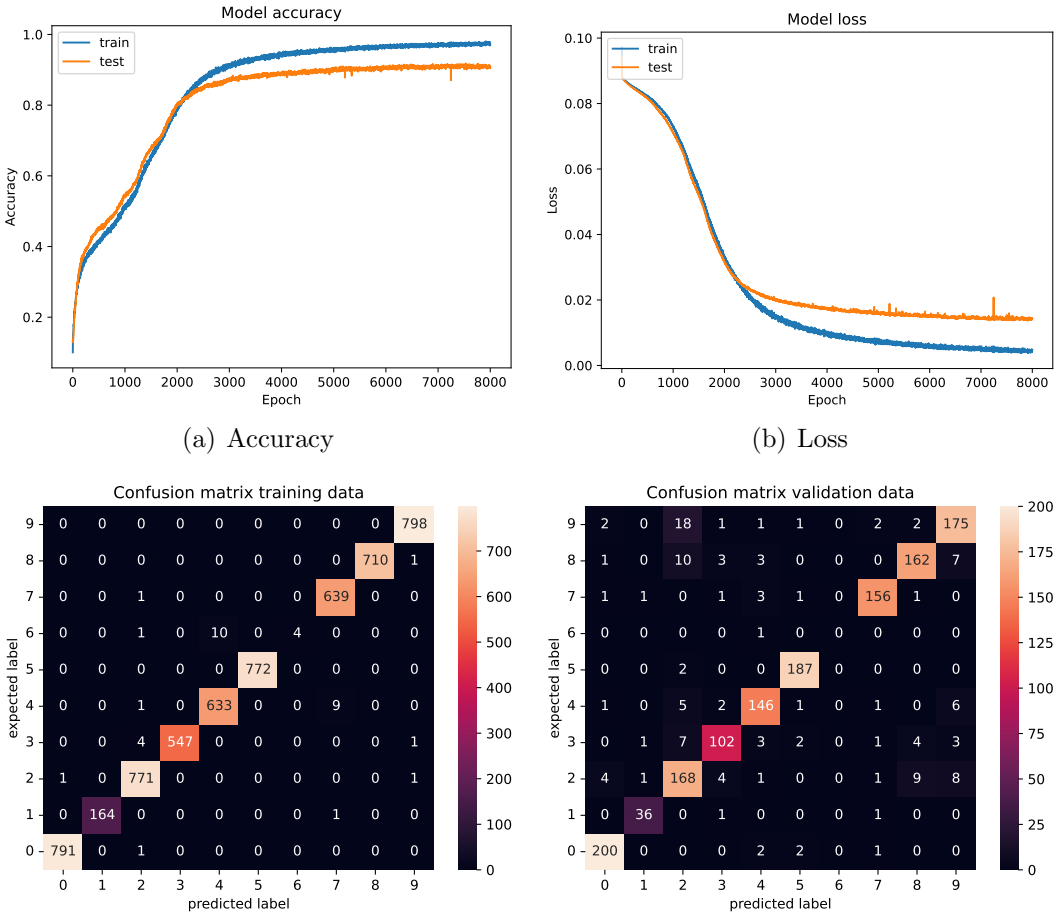
ANN model 2

To solve the problem of the ANN described in section 3.1.3 *ANN model 1*, a batch normalization layer was added after each layer. This gave the configuration described in table 3.5.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	x	yes
2	Dense	ReLU	250	yes
3	Dense	Softmax	250	yes
4	Output	Linear	10	no

Table 3.5: Architecture of ANN model 2.

This model took more epochs to learn, but the learning process was smoother and improved its accuracy, as can be seen in the figure 3.11.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.11: ANN model 2 evolution.

After training, the loss function of the ANN had a value of 0.0048, the accuracy was 97.17%, the validation loss function had a value of 0.0143 and the validation accuracy was 90.86%.

The batch normalization slowed down the learning process but improved its quality as it can be seen by the lack of fluctuations and the decrease in the difference between the train and validation curves, as such it was used in all the other ANN.

3.2 Seagull audio, separating noise from signal

For this part of the work, the goal was to have a software tool to analyze the dummy egg sound file and distinguish the parts that have heart beat from the ones that do not.

These audio files had a sample rate of 2 016 Hz and each file had 20 min of length. The audio in these files was manually divided and assigned in two categories: signal and noise. The signal corresponded to the heart beat of the bird, and the noise to everything else.

The noise was further divided in two categories: ambient noise, e.g., the wind, waves and distant shrieks, and "shot" noise, that corresponded to peaking on the egg, moving the egg inside the nest and nearby shrieks.

Another way to express this is that ambient noise was caused by the natural environment, and "shot" noise was caused by the nesting seagull. As a consequence, the signal muffled the ambient noise, so when there was a signal there was no ambient noise. "Shot" noise, because of its nature, always happened when there was signal and muffled it. One of the consequences of this is that the shot noise was less likely to be removed than the ambient noise. This is not just because of its overlap with the signal, but also because it happened less frequently.

This muffling of the ambient noise by the signal and of the signal by the shot noise happened because of two reasons: the proximity of the source of the sound and the intensity of the sound emitted by the source. The source of the signal (seagull) was much closer than the source of the ambient noise (e.g. waves, wind on leaves). Besides, when the seagull sits on the egg its body blocks most of the ambient noise, only leaving the signal. In the case of the shot noise the source of the sound was at the same distance as the signal source, but a seagull shriek was much louder than a heart beat, so it muffled the signal.

An ANN was used to distinguish the heart beat from noise (both types). To train this ANN, a labeled dataset was needed. To create the dataset, a fake signal generator and a fake noise generator were made.

3.2.1 Fake signal generator

Creating a completely artificial beat was considered, but after a few tests, it was concluded that the heartbeats were too complex to be simulated accurately.

As such, the generation of a file needed a set of real heartbeats that were selected from the dummy eggs' audio files. These heartbeats were selected in windows, 500 samples in length, encompassing the heartbeat. This length was chosen because, from a visual inspection, it was long enough to encompass a heartbeat and short enough to create a wide range of BPM. The audio files from which the heartbeats were selected were chosen randomly, and three to five random heartbeats were extracted from each one.

These heartbeats were extracted as audio files. A total of 777 heartbeats were extracted.

After the selection of the individual real heartbeats, the BPM and Signal to Noise Ratio (SNR) values were randomly chosen for 30 seconds of the file, as having a non-constant BPM and noise allowed training of more robust ANNs. A random real heartbeat was chosen from the ones previously saved, it was normalized so is had a zero average and its maximum absolute value was one. It was also zero padded until it had the desired heart rate period, and then it was appended to an array. This process repeated until the array was equivalent to a 30 s long audio file. White noise was then added according to the SNR defined in the beginning. From this file, a window five seconds in length was chosen. From this window, features were extracted as it is described in section 3.1.2 *Feature extraction model 2*. To the 60 extracted features, a label identifying it as a signal was assigned. Then a 2.5 s stride was applied to the window, and this cycle repeated until it reached the end of the 30 s long audio file. This fake signal generation and feature extraction is repeated until the file duration reached the desired total time. A block diagram of this extraction can be found on figure 3.12.

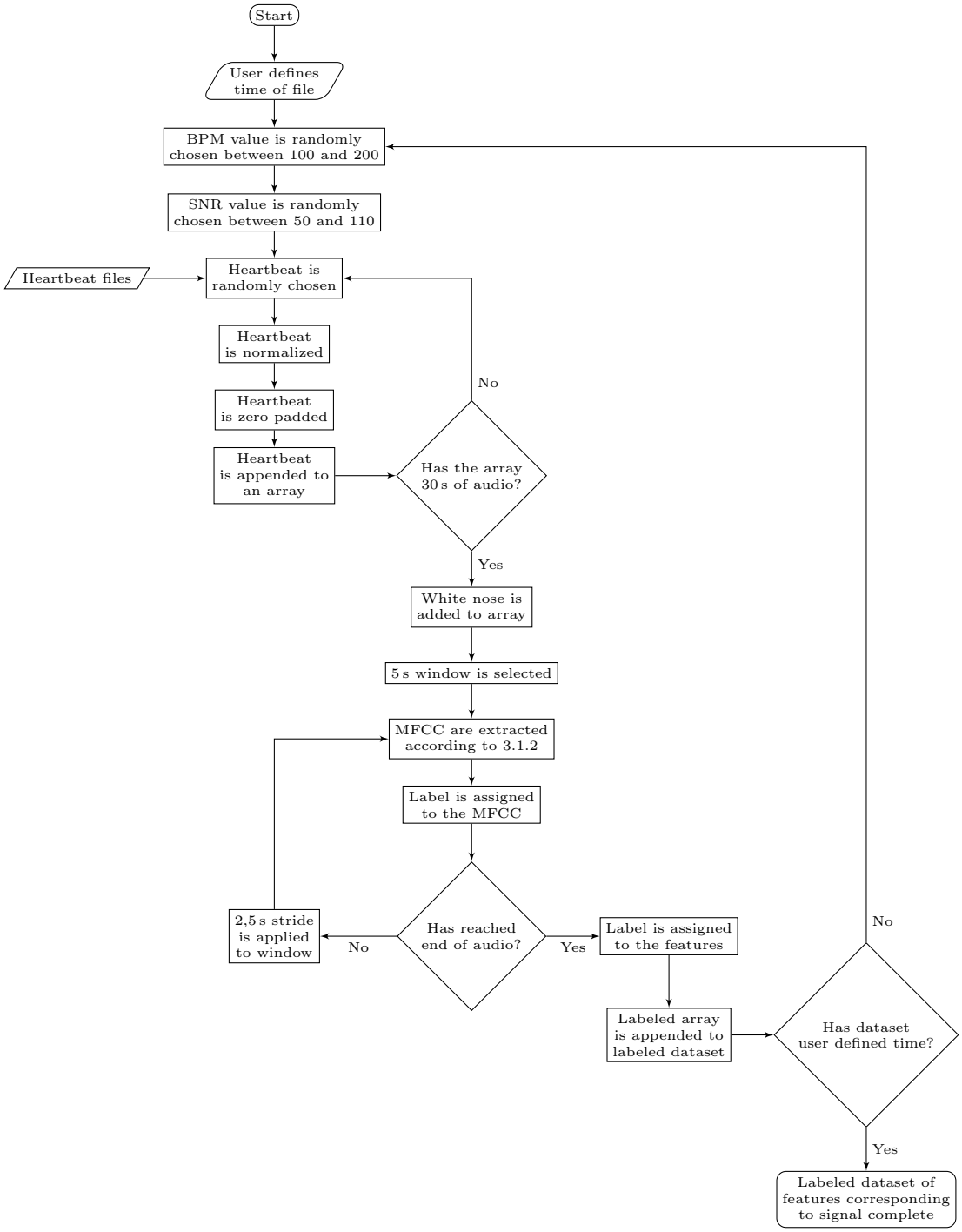


Figure 3.12: Block diagram of the fake signal generator.

3.2.2 Fake noise generator

To train an ANN, a noise file, without a heart beat, was also needed. This file was created using the Urban Sound 8K dataset. This dataset was chosen because it offered a large sound variety. It was hopped that it encompassed the two types of noise mentioned in section 3.2 *Seagull audio, separating noise from signal*, making the network more resilient.

To create it, a file was randomly chosen from the dataset, the features were extracted using the method described in section 3.1.2 *Feature extraction model 2* and a label identifying it as noise was added. This was repeated until there were the same number of data points for noise as there was for signal.

3.2.3 ANN train

Several ANN configurations were trained using the previously generated data are described in this section.

ANN model 3

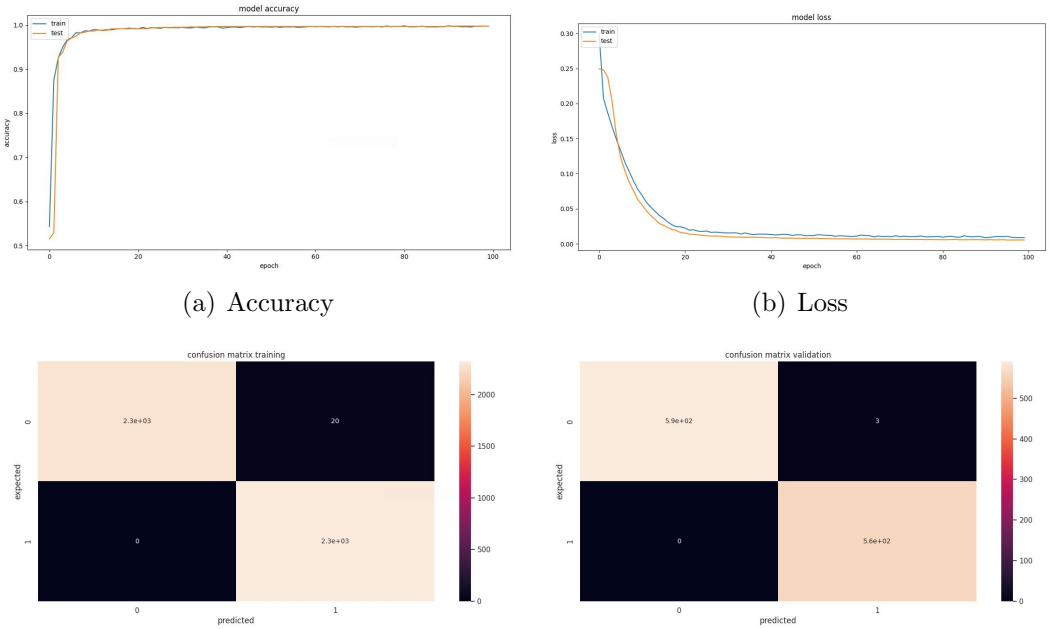
The first model had the shape presented in table 3.6.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	Softmax	60	yes
2	Dense	ReLU	20	yes
3	Dense	Softmax	20	yes
4	Output	Linear	1	no

Table 3.6: Architecture of ANN model 3.

The output was a number between zero and one, if it was one then it means the ANN detected a signal, if it was zero then the ANN detected noise (binary classification).

The dataset had 2880 points of data and 2880 points of noise. 80% of the data was used for training and 20% for validation, it did not have a test dataset as there was little data. The batch size was 32, it used the SGD optimizer with a 0.001 learning rate and the loss was mean squared error. It gave the results that can be seen on figure 3.13.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.13: ANN model 3 evolution.

After training, the loss function of the ANN had a value of 0.0057, the accuracy was 99.80%, the validation loss function had a value of 0.0037 and the validation accuracy was 99.74%. These results had an accuracy well above the one required.

The confusion matrices indicate the presence of some false positives, but their occurrence is negligible.

After this results, an improvement was made in the fake heart beat generation, instead of adding a layer of white noise to the fake heart beat, sounds from the Urban Sounds dataset were added. This was made to make the noise and the signal more similar, so the ANN would be more robust.

ANN model 4

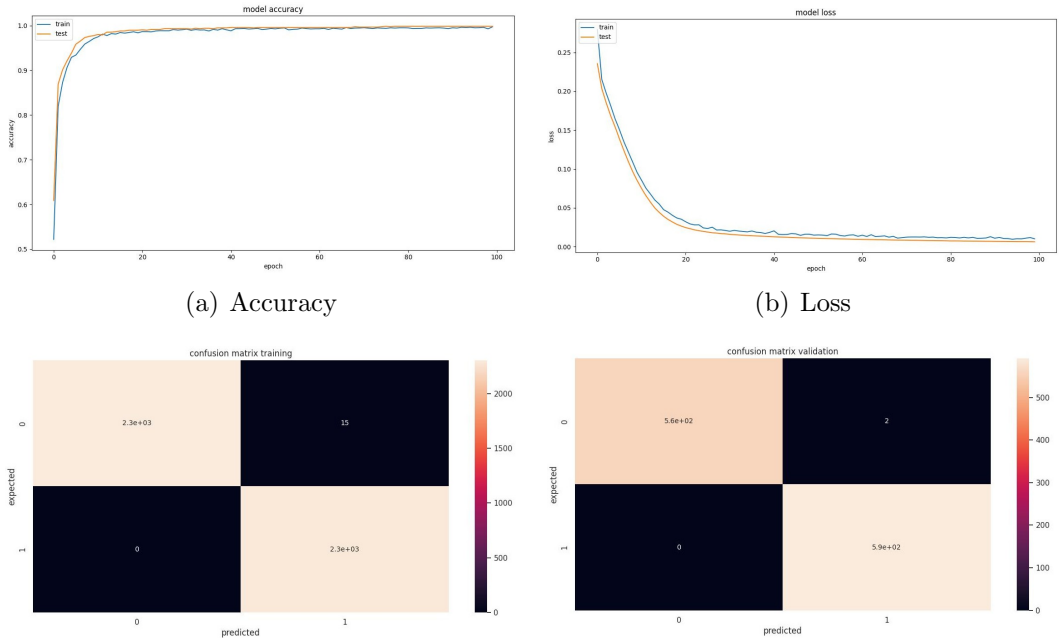
The ANN trained with this data had the shape presented in table 3.7.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	60	yes
2	Dense	ReLU	20	yes
3	Dense	Softmax	20	yes
4	Output	Linear	1	no

Table 3.7: Architecture of ANN model 4.

The train had 14515200 points of data and 14515200 points of noise. 80% of the

data was used for training and 20% for validation, the batch size was 32, it used the SGD optimizer with a 0.001 learning rate and the loss was mean squared error. After 100 epochs, it gave the results that can be seen on figure 3.14. As this was a simple viability test, no test dataset was used.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.14: ANN model 4 evolution.

After training, the loss function of the ANN had a value of 0.0103, the accuracy was 99.64%, the validation loss function had a value of 0.0126 and the validation accuracy was 99.58%. These results had an accuracy well above the one required.

Similar to the previous model, the confusion matrices reveal the presence of a few false positives, but their occurrence is negligible. The absence of false negatives and the presence of false positives indicate that the network has higher recall than precision. This is not ideal, as the subsequent part of the algorithm aims to identify heartbeats where any false positives could potentially compromise some results.

After these very encouraging results, the trained ANN was tested on real data. It was chosen a file that had only heart beat, so the expected output would be 1 across the whole file as the sound is analyzed in 5 s windows. From a visual inspection of the dummy egg audio file, the average heart rate of a seagull is approximately, 320 ± 18 BPM [46] so in one window there are in average 27 heart beats. However, the output of the ANN was a fluctuating line at around 0.5 as can be seen on figure 3.15.

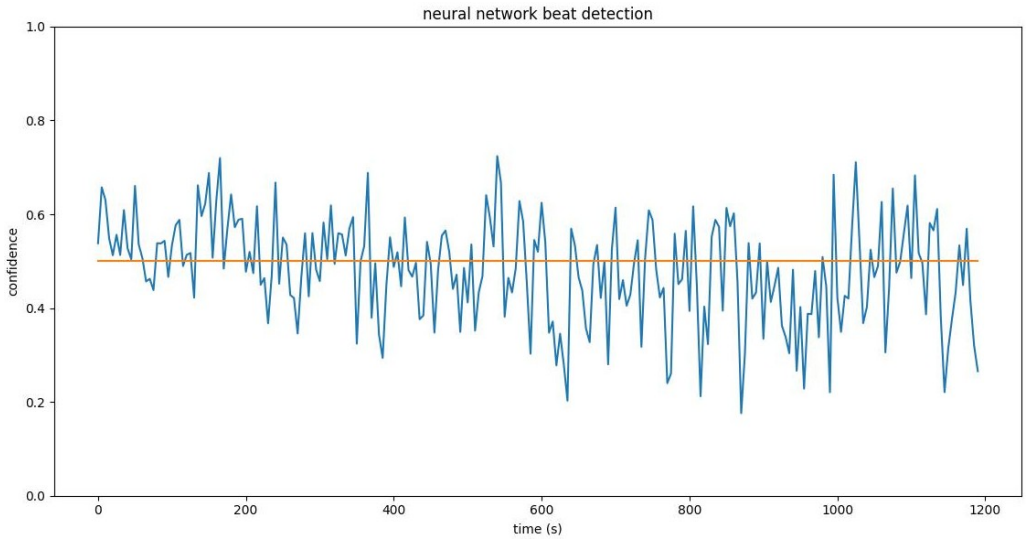
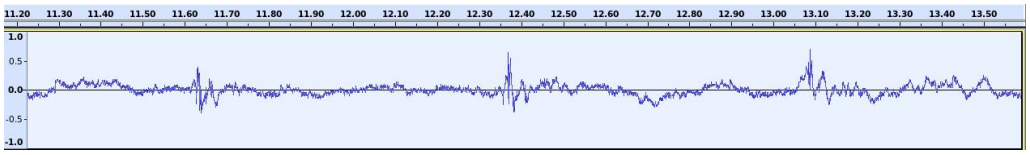


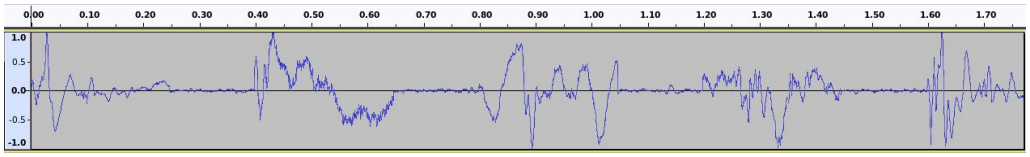
Figure 3.15: ANN model 4 signal identification.

This result is an indicator that the signal and/or noise generator did not accurately represent the real data.

In figure 3.16 the real heart beat are compared to the generated ones. In it, it can be seen that the real data has: greater fluctuation of the baseline noise, a better integration of the heartbeats with the noise, more similar heartbeats and non normalized heartbeats.



(a) Real heartbeats



(b) Fake heartbeats

Figure 3.16: Comparison between real and generated data.

With these differences taken into account, the signal and noise generators were redone.

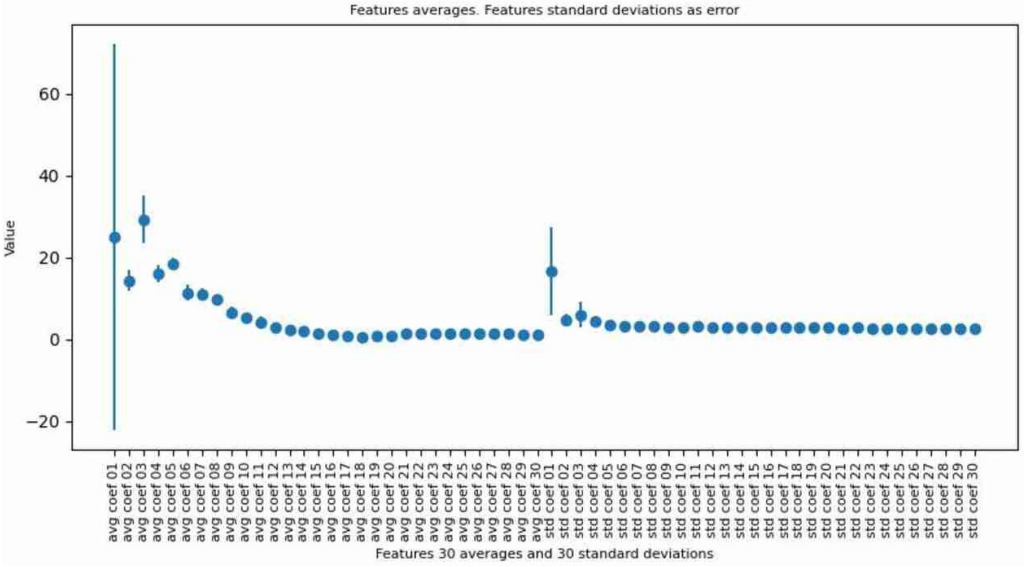
3.2.4 Revised fake signal and noise generator

To generate the labeled dataset, the sound files were manually inspected and blocks of the file that have either only beats or only noise were selected and saved separately as their own audio file.

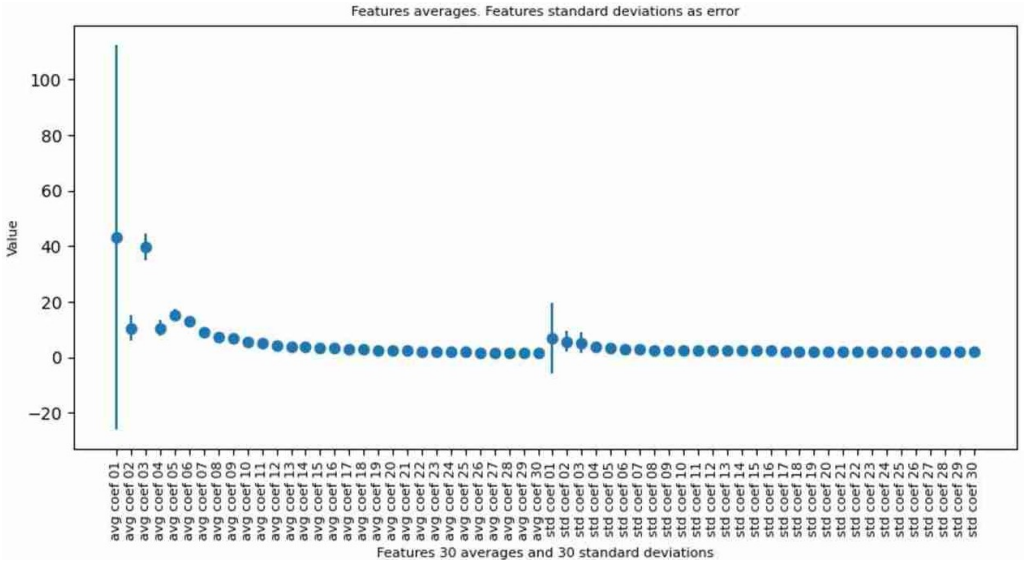
For this work, a total of 7 785 s of heart beat and 9 622 s seconds of noise were selected.

Then a random, 10 000 sample window (4,96 s) was selected from one of these files. From it, features were extracted as described in section 3.1.2 *Feature extraction model 2* and saved in a CSV file with a label that identifies them as noise or signal. As such, the ANN was trained with complete real data segments instead of a generated segment using individual beats. This is thought to improve its realism.

Another enhancement of the signal generator was the reduction of the number of extracted features. Rather than calculating 30 coefficients for the MFCC block, which would result in 60 features (30 averages and 30 standard deviations), it was decided to reduce the number of coefficients to 15. From them, the averages and standard deviations were extracted, as done in section 3.1.2 *Feature extraction model 2*, resulting in a total of 30 features. This decision was done based on the distribution plots shown in figure 3.17. These plots revealed that the variance for the last MFCC coefficients averages and last MFCC coefficients standard deviations was negligible, and removing them could significantly reduce the training time of the ANN.



(a) Distribution plot for heart beat



(b) Distribution plot for noise

Figure 3.17: Extraction method of section 3.1.2 *Feature extraction model 2* averages and standard deviations.

3.2.5 ANN train

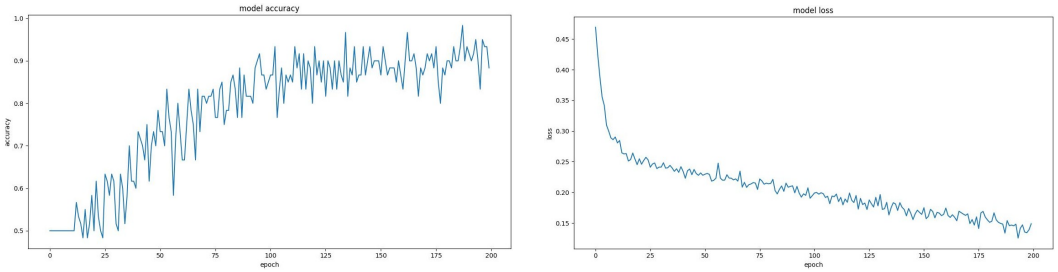
ANN model 5

This ANN model had the shape presented in table 3.8.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	30	yes
2	Dense	ReLU	20	yes
3	Dense	Softmax	20	yes
4	Output	Linear	1	no

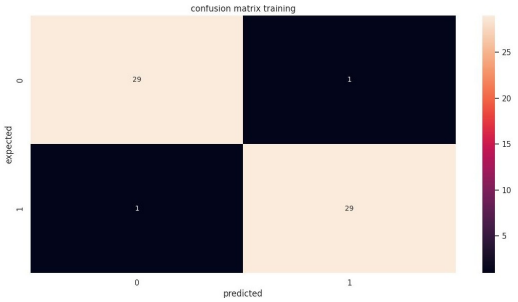
Table 3.8: Architecture of ANN model 5.

This ANN was trained with 521 data points [1] of noise and 521 data points of heart beat. The loss was calculated using the mean squared error, the optimizer was SGD with a 0.001 learning rate, the batch size was 10, and it ran for 200 epochs. It gave the results that can be seen on figure 3.18.



(a) Accuracy

(b) Loss



(c) Confusion matrix

Figure 3.18: ANN model 5 evolution.

In the final iteration, the loss function had a value of 0.1353 and the accuracy was 0.95%.

The confusion matrix of this ANN reveals that the number of false positives is equal to the number of false negatives. This indicates an improvement compared to previous results.

For this model there was a small amount of data available and for the first iterations

[1] data point refers to the the features extracted from the 10 000 sample window as explained in subsection 3.2.4 *Revised fake signal and noise generator*

the goal was only a rough optimization. As such, there was no validation dataset; all generated data was used to train the ANN.

A 20-minute file that, by chance, was half noise half signal was used as a test dataset. A spectrogram of this file can be found on figure 3.19.

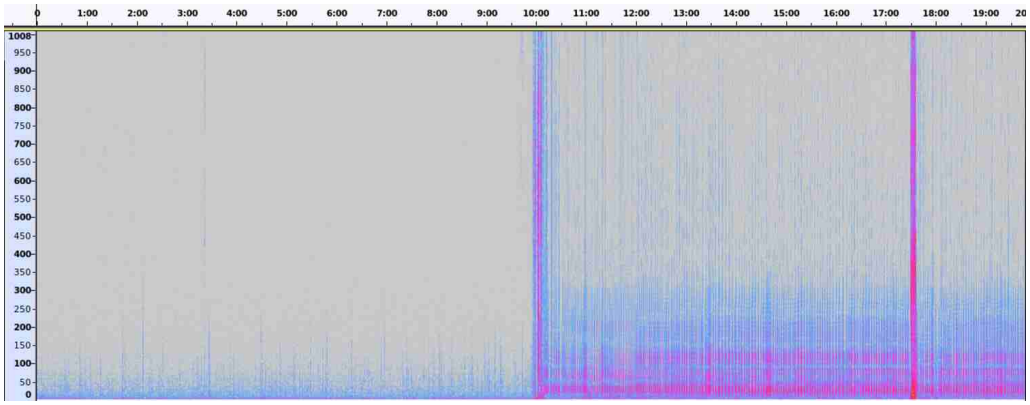


Figure 3.19: Spectrogram of the sound file that is half heart beat, half noise and was used as test dataset for the ANNs.

.

The detection results on this file can be seen on figure 3.20.

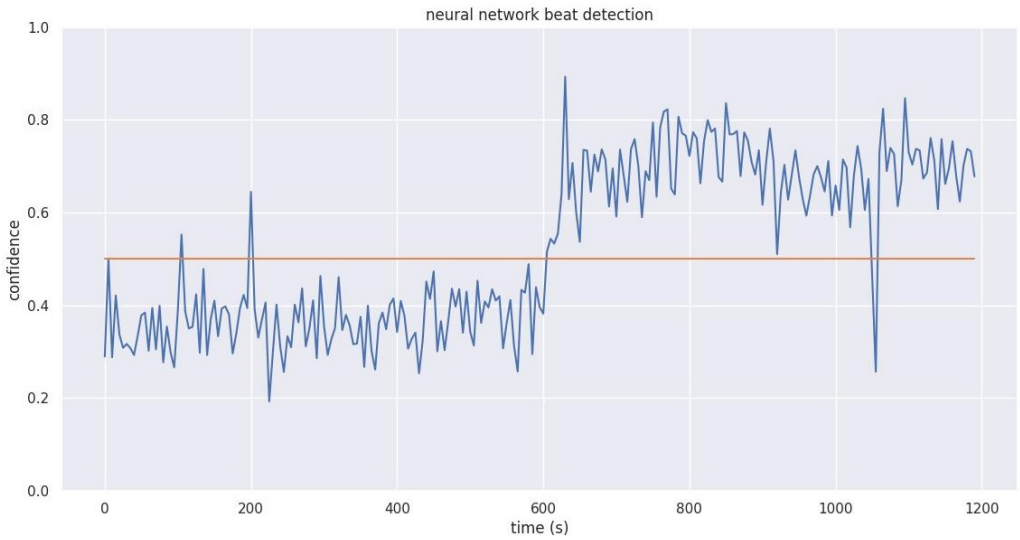


Figure 3.20: ANN model 5 signal identification.

.

The blue line is the output of the ANN and the orange line is the threshold of detection (0.5): anything above this line was considered as signal and anything below noise.

In general, the signal was well divided, with a clear demarcation where the noise ended and the signal began. However, the detection was closer to 0.5 than to the

extremes and there were a few outliers. The next iterations tried to solve this.

ANN model 6

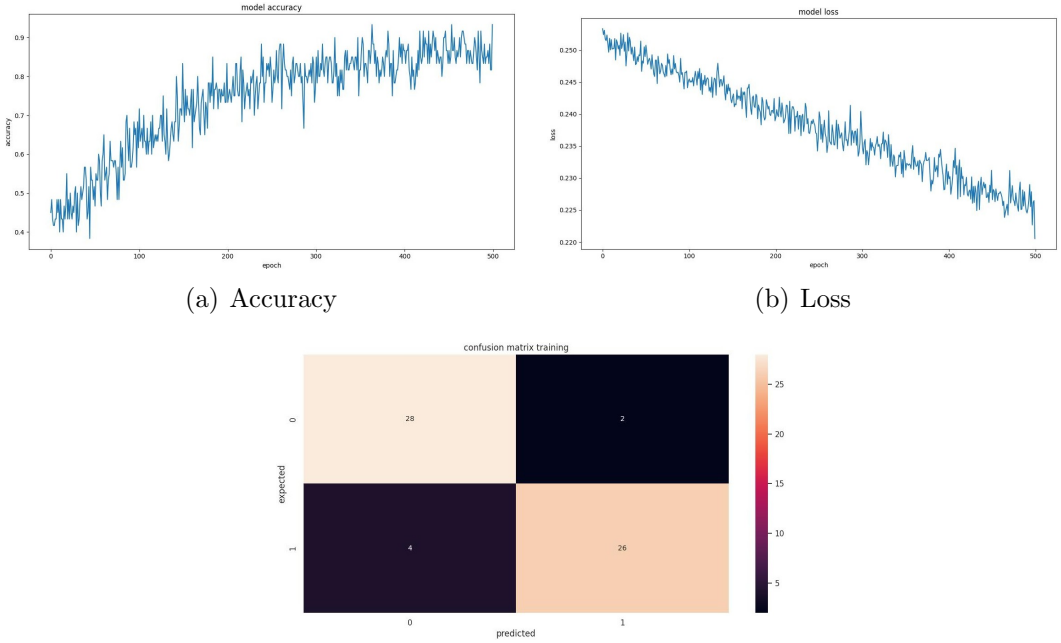
The next ANN had the shape presented in table 3.9.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	30	yes
2	Dense	ReLU	20	yes
3	Dense	Sigmoid	20	yes
4	Output	Linear	1	no

Table 3.9: Architecture of ANN model 6.

.

The loss was calculated using the mean squared error, the optimizer was SGD with a 0.001 learning rate, the batch size was 10, and it ran for 500 epochs. It gave the results that can be seen on figure 3.21.



(a) Accuracy

(b) Loss

(c) Confusion matrix

Figure 3.21: ANN model 6 evolution.

.

The detection results are on figure 3.22.

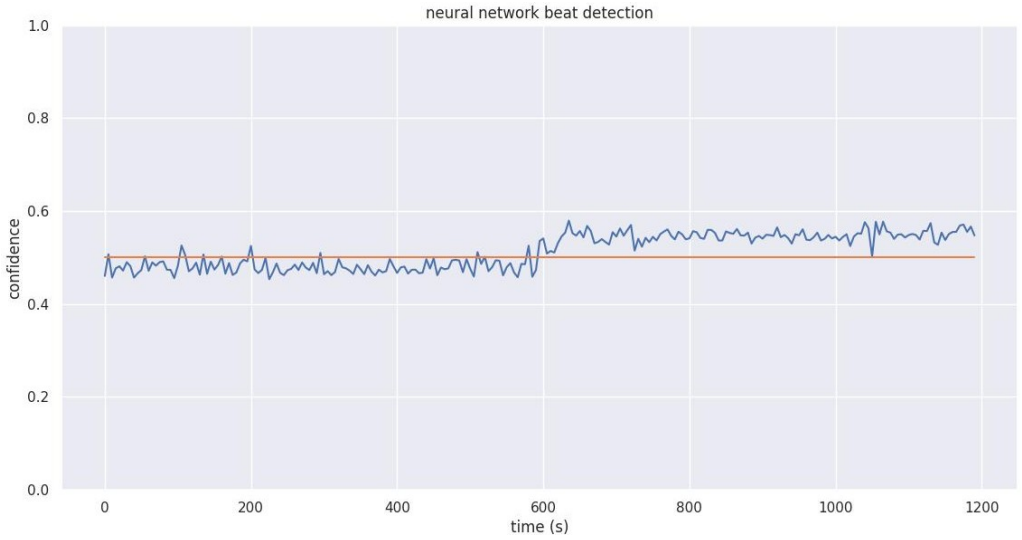


Figure 3.22: ANN model 6 signal identification.

In the final iteration, the loss function had a value of 0.2205 and the accuracy was 93.33%.

This change did not improve the results. There was still a division, but the outliers increased, and ANN output was even closer to the 0.5 mark than before.

ANN model 7

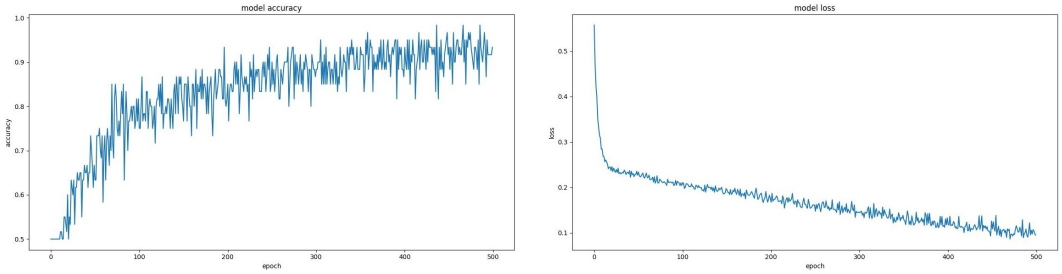
Following this, an ANN with the shape presented in table 3.10 was made.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	30	yes
2	Dense	ReLU	20	yes
3	Dense	Tanh	20	yes
4	Output	Linear	1	no

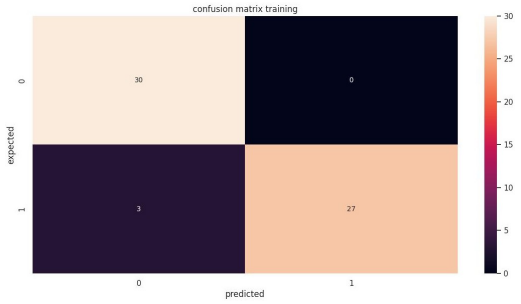
Table 3.10: Architecture of ANN model 7.

.

The loss was calculated using the mean squared error, the optimizer was SGD with a 0.001 learning rate, the batch size was 10, and it ran for 500 epochs. It gave the results that can be seen on figure 3.23.



(a) Accuracy (b) Loss



(c) Confusion matrix

Figure 3.23: ANN model 7 evolution.

In this ANN, there are no false positives, only false negatives, resulting in a precision of one. This precision value is preferred for this particular section of the work, as explained in detail in section 3.2.3 *ANN model 4*.

The detection results of the test file are on figure 3.24.

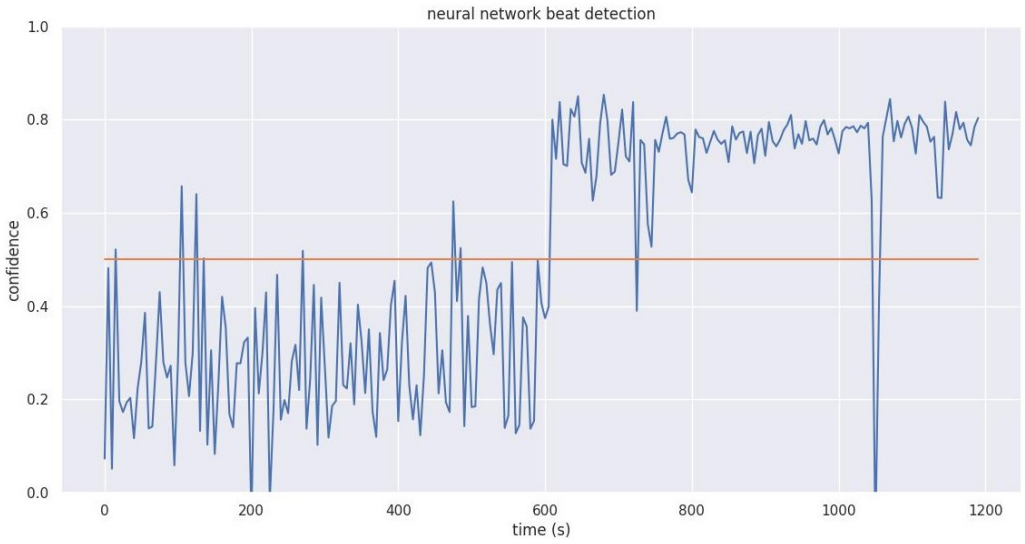


Figure 3.24: ANN model 7 signal identification.

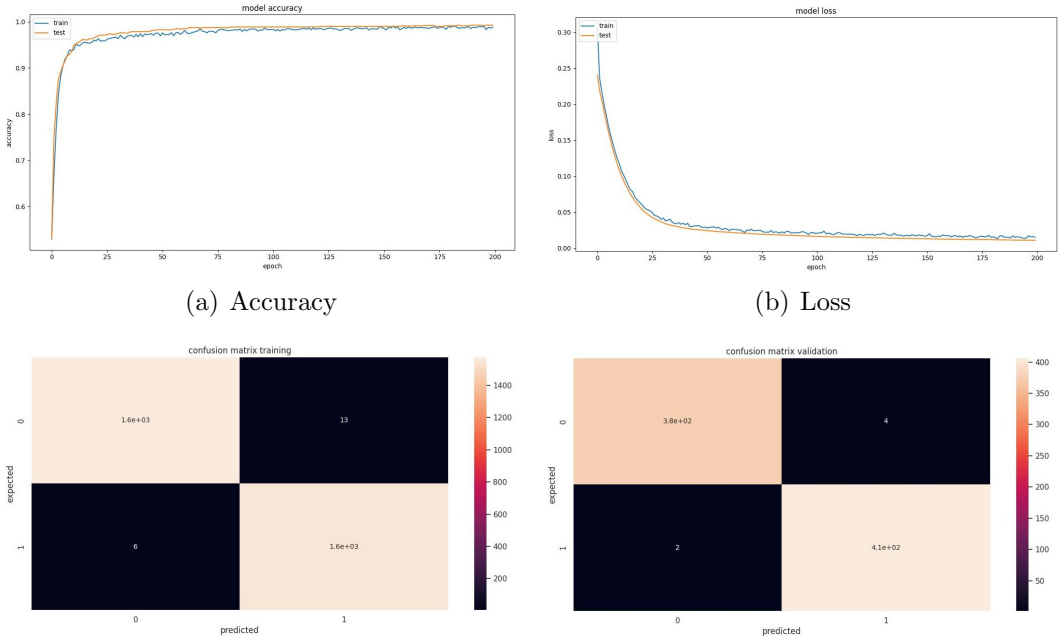
In the final iteration, the loss function had a value of 0.0949 and the accuracy was 93.33%.

When using the Tanh activation function, the results became more distributed across the available range, but the number of outliers increased compared to the ANN discussed in section 3.2.5 *ANN model 5*, rising from 3 to 9. In this work, it is preferable to prioritize a weaker detection capability with higher accuracy over a stronger but less accurate one.

ANN model 8

This ANN had the same configuration and presets as the one in section 3.2.5 *ANN model 5*, but it was trained with 1968 data points of noise and 1968 data points of signal for 500 epochs. As there was more data, the dataset was split in an 80/20 fraction, where 80% of the data was be used for training and 20% for validation.

It gave the results that can be seen on figure 3.25.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.25: ANN model 8 evolution.

After training the loss function of the ANN had a value of 0.0155, the accuracy was 98.79%, the validation loss function had a value of 0.0112 and the validation accuracy was 99.24%.

The detection results of the test file are on figure 3.26.

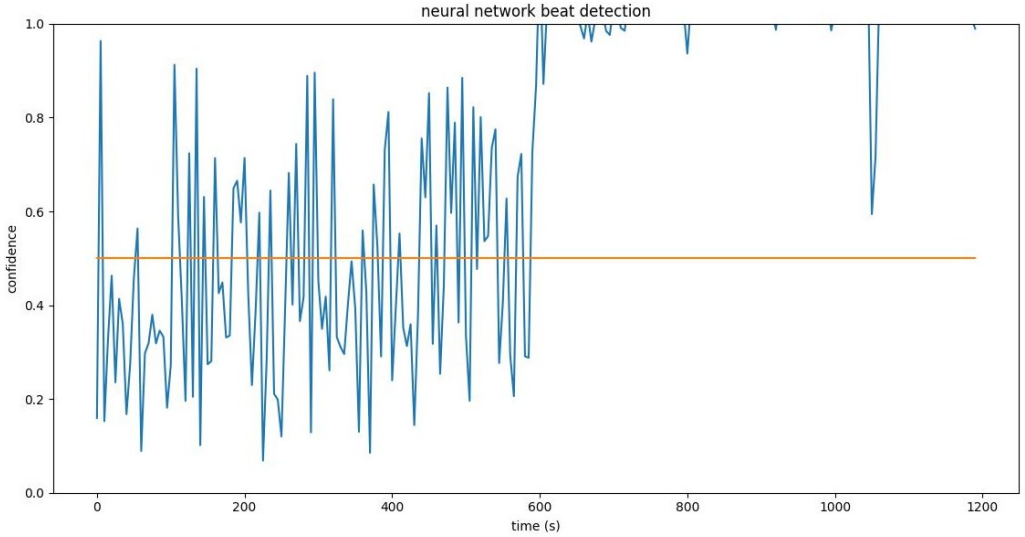


Figure 3.26: ANN model 8 signal identification.

Although the ANN had good results when using the train data when it analyzed the test file, the output was not usable. However, it did detect a difference between signal and noise.

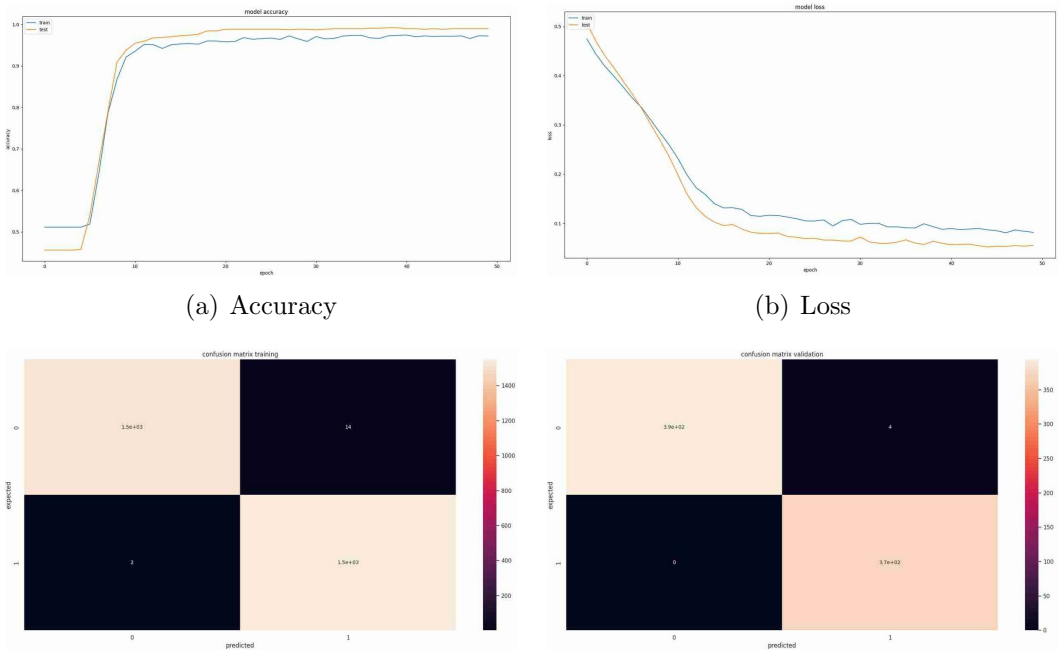
ANN model 9

In this model, the number of nodes was increased. It had the shape presented in table 3.11.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	30	yes
2	Dense	ReLU	100	yes
3	Dense	Softmax	50	yes
4	Output	Linear	1	no

Table 3.11: Architecture of ANN model 9.

The loss was calculated using the mean squared error, the optimizer was SGD with a 0.001 learning rate, the batch size was ten, and it ran for ten epochs. The results can be seen on figure 3.27.



(a) Accuracy

(b) Loss

(c) Confusion matrix for training data

(d) Confusion matrix for validation data

Figure 3.27: ANN model 9 evolution.

After training the loss function of the ANN had a value of 0.0817, the accuracy was 97.20%, the validation loss function had a value of 0.0552 and the validation accuracy was 98.96%.

The detection results of the test file are on figure 3.28.

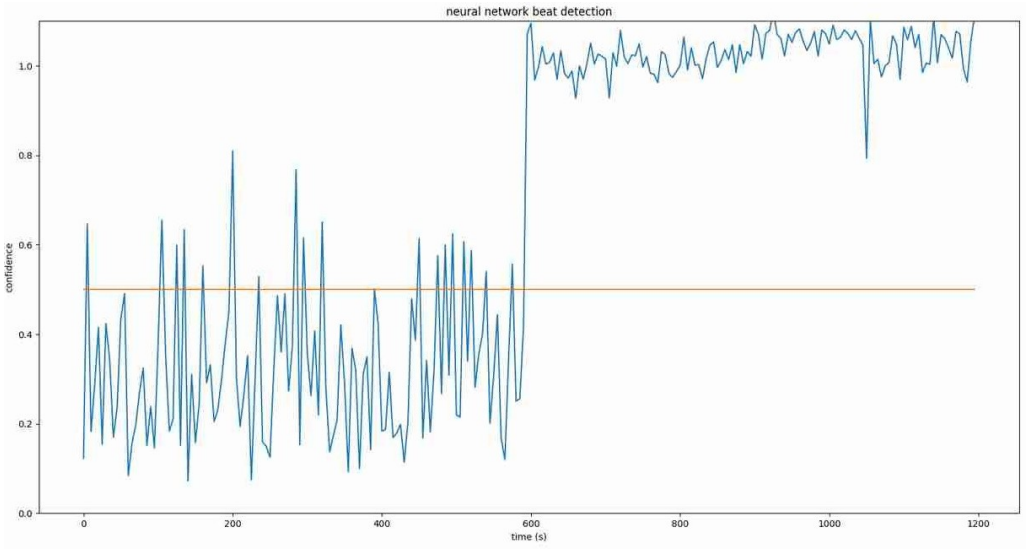


Figure 3.28: ANN model 9 signal identification.

To reduce the training time, the number of epochs in this model was reduced

from 500 to 50. This choice was based on a visual inspection of previous accuracy evolution graphs, which indicated that the majority of optimization occurs early in the training process.

With this configuration, the ANN could clearly make a division between signal and noise with greater separation than the one presented in section 3.2.5 *ANN model 5*, although there were more outliers.

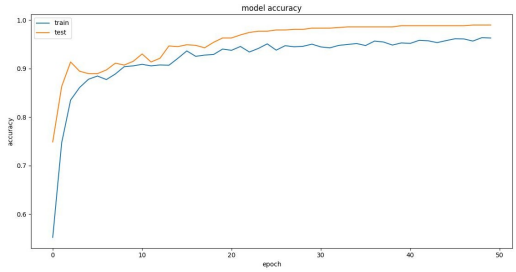
ANN model 10

In this model, the number of neurons was further increased, as can be seen in table 3.12.

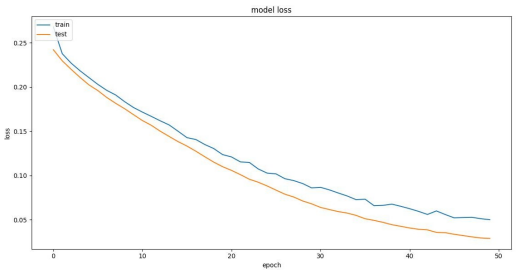
Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	30	yes
2	Dense	ReLU	100	yes
3	Dense	Softmax	100	yes
4	Output	Linear	1	no

Table 3.12: Architecture of ANN model 10.

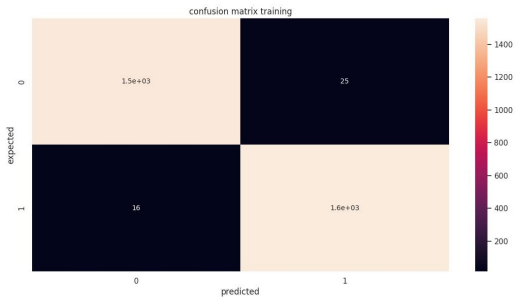
The loss was calculated using the mean squared error, the optimizer was SGD with a 0.001 learning rate, the batch size was 10, and it ran for 50 epochs. The results can be seen on figure 3.29.



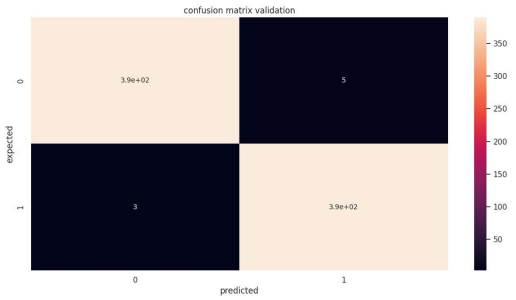
(a) Accuracy



(b) Loss



(c) Confusion matrix for training data



(d) Confusion matrix for validation data

Figure 3.29: ANN model 10 evolution.

The detection results of the test file are on figure 3.30.

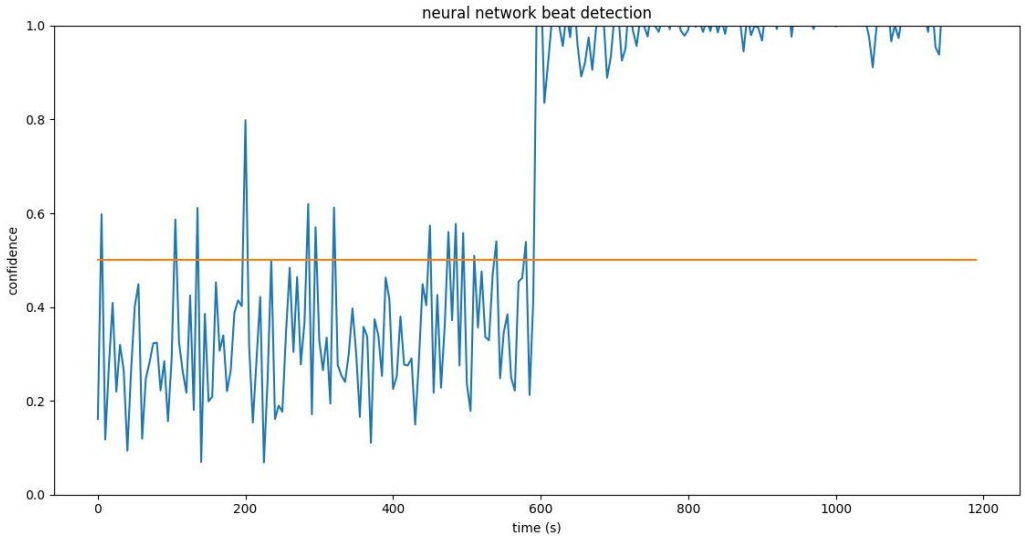


Figure 3.30: ANN model 10 signal identification.

These results are similar to the ones got by the ANN developed in section 3.2.5 *ANN model 8*.

ANN model 11

In this model, the final activation layer was changed as can be seen in table 3.13.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	30	yes
2	Dense	ReLU	100	yes
3	Dense	Sigmoid	100	yes
4	Output	Linear	1	no

Table 3.13: Architecture of ANN model 11.

The loss was calculated using the mean squared error, the optimizer was SGD with a 0.001 learning rate, the batch size was 10, and it ran for 50 epochs.

As the graphs of the evolution of the ANN did not have any information that was relevant for the optimization of the ANN, as such they will not be presented.

The detection results of the test file are on figure 3.31.

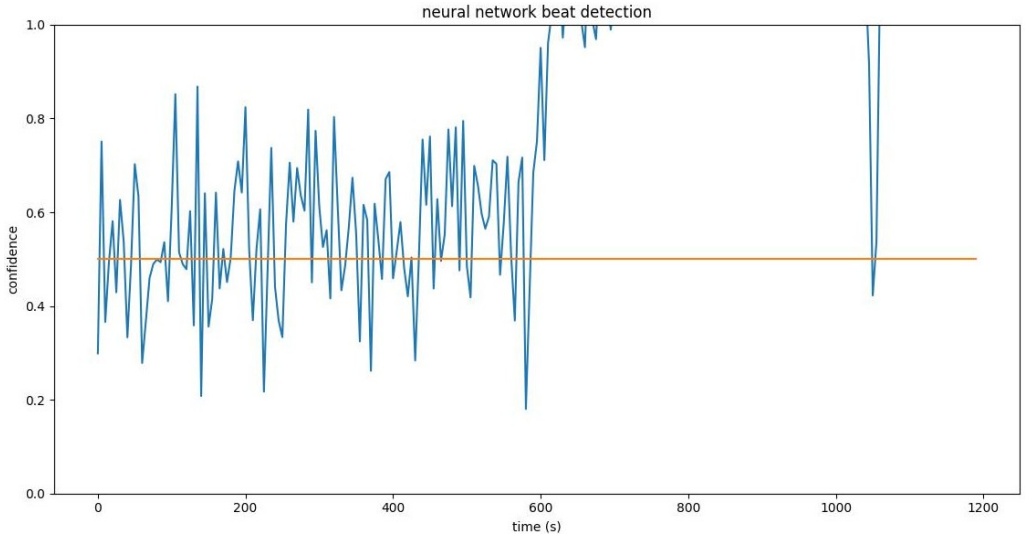


Figure 3.31: ANN model 11 signal identification.

The division still shares the same problems, as the one seen on section 3.2.5 *ANN model 8*.

It is worth noting that the last noise detection was not an outlier, in this point there was a shot noise that interrupted the signal. The detection of shot noise was a positive point for the ANN.

ANN model 12

As the ANN was not learning as expected, four things were changed: the number of nodes per layer of the ANN were changed to funnel like shape; a second node was added to give a confidence of the data point being noise; the range of the expected output was changed so 1 corresponded to signal, -1 corresponded to noise and 0 was the halfway point; the range of the y-axis on the graph of the confidence level of the signal/noise was increased, so it can represent values that exceed the range $[-1, 1]$.

This ANN architecture can be seen on table 3.14.

Layer index	Layer type	Activation function	Number of nodes	Batch normalization
1	Input	ReLU	30	yes
2	Dense	ReLU	256	yes
3	Dense	ReLU	64	yes
4	Dense	Sigmoid	16	yes
5	Output	Linear	2	no

Table 3.14: Architecture of ANN model 12.

The loss was calculated using the mean squared error, the optimizer was Adam, the batch size was 10, and it ran for 50 epochs.

The detection results of the test file are on figure 3.32.

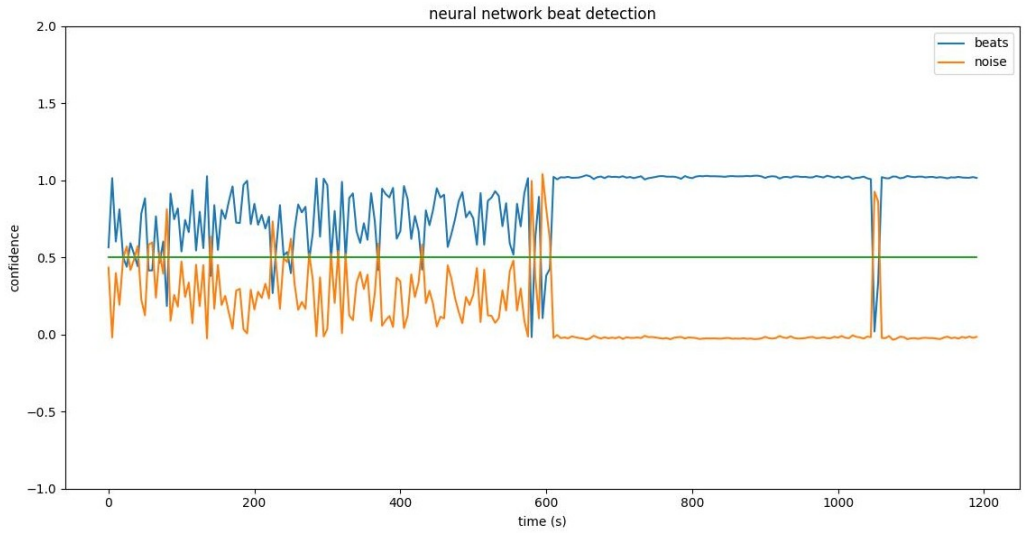


Figure 3.32: ANN model 12 signal identification.

This graph shows that the ANN could not detect noise, as the first part fluctuates too much to be considered usable. In the sections of the sound that contain noise, this ANN exhibits a significant number of false positives, incorrectly classifying noise as signal.

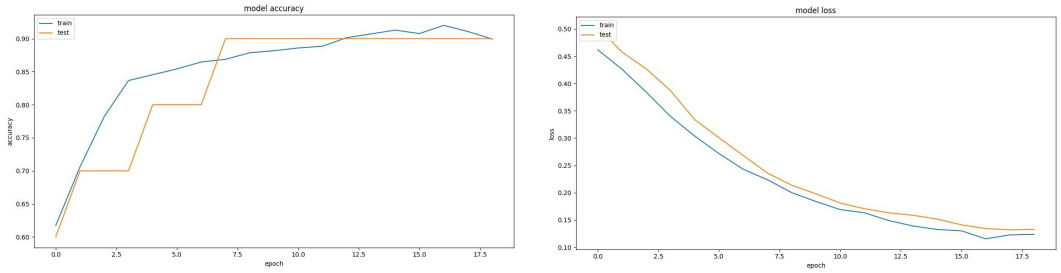
ANN model 13

In this model, a dropout was added to all layers except the last one. The shape of the ANN is presented in table 3.15.

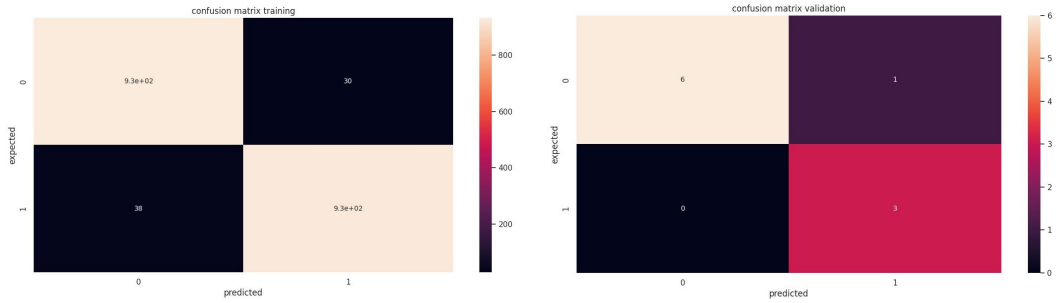
Layer index	Layer type	Activation function	Number of nodes	Batch normalization	Dropout value
1	Input	ReLU	30	yes	0.2
2	Dense	ReLU	16	yes	0.2
3	Dense	ReLU	64	yes	0.2
4	Dense	ReLU	64	yes	0.2
5	Dense	Softmax	16	yes	0.2
6	Output	Sigmoid	2	no	$N \backslash A$

Table 3.15: Architecture of ANN model 13.

The loss was calculated using the mean squared error, the optimizer was SGD with a 0.001 learning rate, the batch size was 10, and it ran for 19 epochs. The results can be seen on figure 3.33.



(a) Accuracy (b) Loss



(c) Confusion matrix for training data (d) Confusion matrix for validation data

Figure 3.33: ANN model 13 evolution.

The last iteration had a loss of 0.1236, an accuracy of 89.94%, a validation loss of 0.1328 and a validation accuracy of 90.00%.

This model only ran for 19 epochs because it had an early stooping callback that defined if the validation accuracy did not improve in five consecutive epochs, the train would stop.

The confusion matrix of the validation dataset demonstrates that the ANN has higher precision than recall.

The detection results of the test file are on figure 3.34.

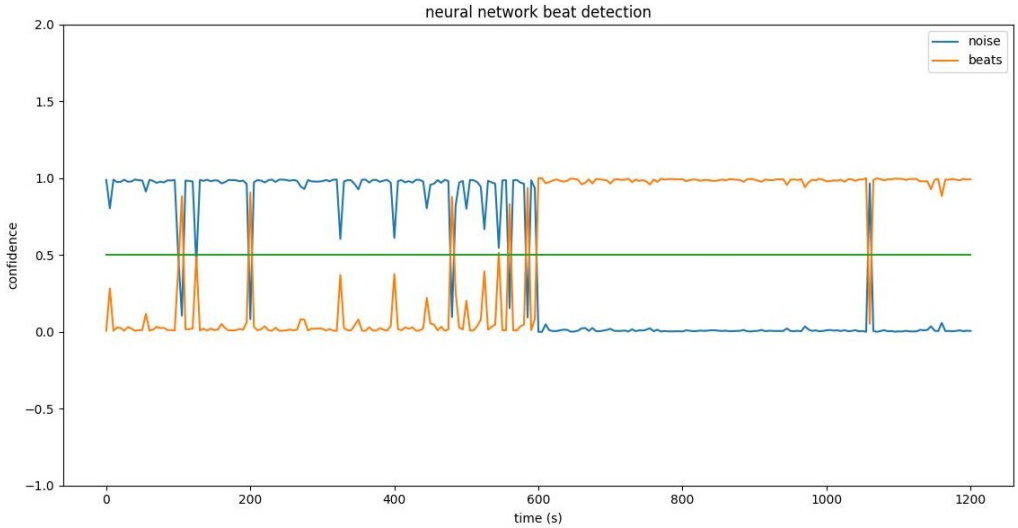


Figure 3.34: ANN model 13 signal identification.

The file analyzed with this ANN had a clear division of noise and signal that could be used by other programs.

Throughout these attempts, several key concepts were learned. Firstly, in the last layer of the network, using the sigmoid activation function instead of a linear activation function yielded best results. Secondly, employing a diamond shape for the ANN layers instead of a funnel like shape proved to be more effective. Thirdly, the inclusion of batch normalization layers helped stabilize the learning process and improve overall results. Lastly, incorporating dropout layers enhanced the resilience of the ANN and contributed to higher accuracy.

3.3 Seagull audio, identifying individual heartbeats

After the section of the audio with heart beat were identified, the next step was to identify the individual heartbeats. To identify them, RNNs were used.

The input sequence of the RNN was the sound of the fake egg, and the output sequence was a file the same size as the input one. In this file, each sample was a number between zero and one that corresponds to the probability of that sample being a heartbeat.

Fake data generator with labeled heartbeats

To train an RNN to identify heartbeats, a labeled dataset was needed. For this, it was used a modified version of the fake signal generator created in section 3.2.1.

This signal generator was discarded in the section 3.2 *Seagull audio, separating noise from signal*, but it was reused for this, as it was the simplest way to test if the RNN could detect a beat or not.

To generate the fake audio file, the user first specified the desired length of the file. Next, a random value between 100 and 200 was chosen as the BPM for the audio file. Then, a random heartbeat was selected, normalized to a zero average and a maximum absolute value of one, and padded with zeros to match the period of the heartbeat at the chosen BPM. This process was repeated until a seven-second audio clip was generated, which was then repeated to create the desired length specified by the user.

The label array had the same length as the data array and all indexes were zero except the 100 indexes after each heartbeat, which were one.

A block diagram of the generation of the fake heart beat and label can be found on figure 3.35.

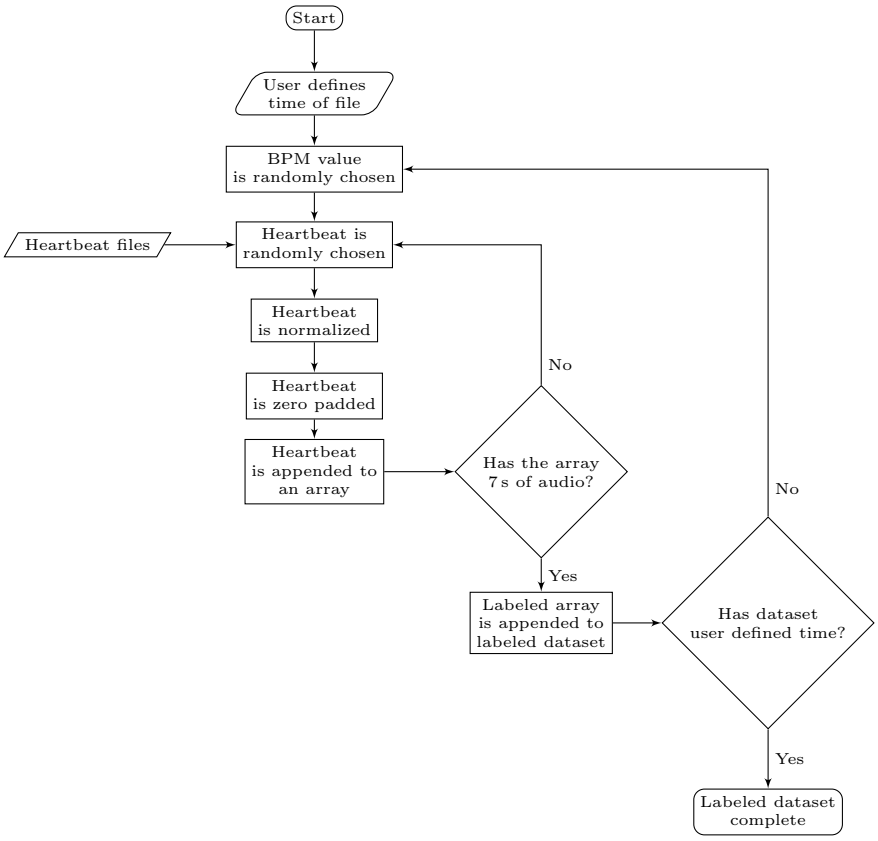


Figure 3.35: Fake heart beat and label creation block diagram.

RNN model 1

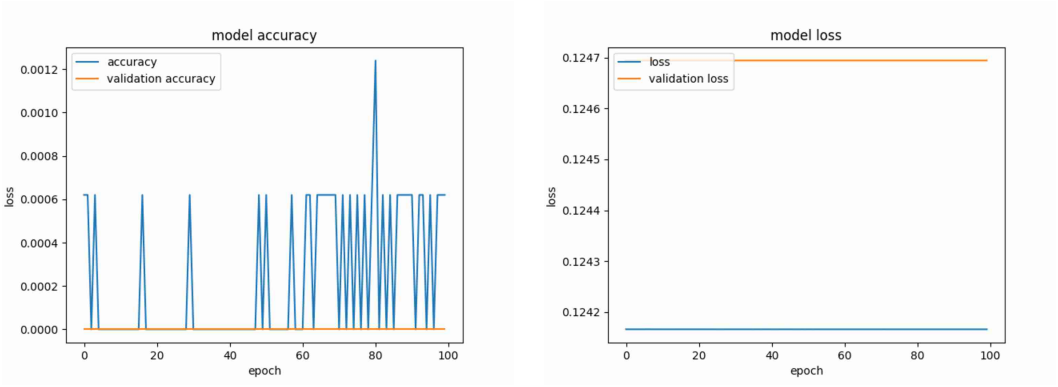
The first RNN had the architecture described in table 3.16.

Layer index	Layer type	Activation function	Nodes	Dropout	Units	Recurrent dropout	Return sequences
1	Dense	Linear	5000	na	na	na	na
2	LSTM	na	na	0.1	100	0.1	true
3	LSTM	na	na	0.1	100	0.1	true
4	LSTM	na	na	0.1	100	0.1	false
5	Dropout	na	na	0.1	na	na	na
6	Output	Softmax	5000	na	na	na	na

Table 3.16: Architecture of RNN model 1.

It was trained with 5000 s of data, of which 80% were used for training and 20% for validation. It ran for 100 epochs and had a batch size of 64. It used the Adam optimizer and measured its loss using the mean squared error.

This gave the results seen on figure 3.36.

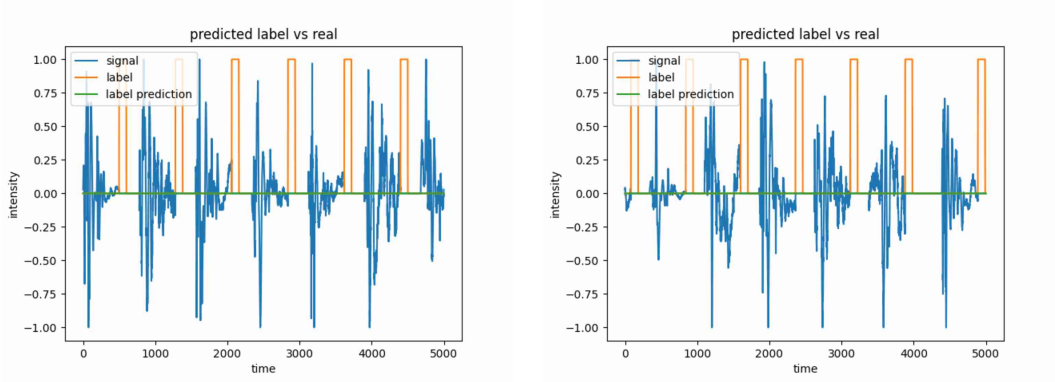


(a) Accuracy (b) Loss

Figure 3.36: RNN model 1 evolution.

After training the loss function of the RNN had a value of 0.1242, the accuracy was 00.06%, the validation loss function had a value of 0.1247 and the validation accuracy was 00.00%.

The results, obtained when using this RNN in the train dataset and validation dataset, are presented in figure 3.37.



(a) Prediction for the train dataset (b) Prediction for the validation dataset

Figure 3.37: RNN model 1 heartbeat identification.

As can be seen in the figure 3.36 the loss did not improve during the train that means that the value of the weights and biases were not altered during the training phase. As such, the RNN did not learn, making it unsuitable for this work. This could suggest that the ANN may require more than just LSTM cells to learn.

RNN model 2

To improve detection, convolutional layers were added to this model. Additionally, the dataset was increased, and the model was trained for more epochs than the previous version.

The decision to use CNNs was influenced by previous works that employed them [11, 10]. CNNs are useful for their ability to detect patterns through the use of convolutional layers, regardless of their location within the signal window. This is particularly beneficial in the context of heartbeat detection, as a heartbeat can occur at any position within the signal. Additionally, CNNs can effectively reduce dimensionality through the pooling layers. This dimensionality reduction can facilitate the subsequent RNN in accurately identifying the heartbeats within the data.

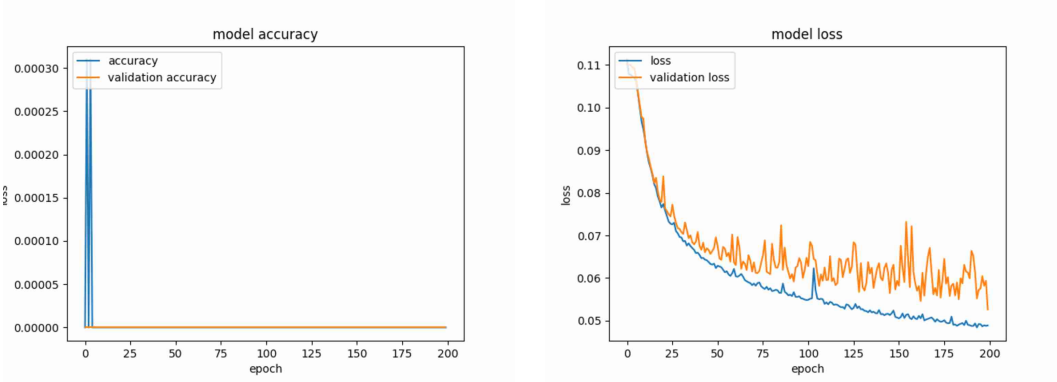
This model had the architecture described in table 3.17.

It was trained with 10 000 s of data, of which 80% were used for training and 20% for validation. It ran for 200 epochs and had a batch size of 64. It used the Adam optimizer and measured its loss using the mean squared error.

Its evolution can be seen on figure 3.38.

Layer index	Layer type	Activation function	Nodes	Dropout	Filters	Kernel size	Input shape	Pool size	Strides	Units	Recurrent dropout	Return sequences
1	1D Convolutional	ReLU	na	na	64	10	[5000,1]	na	na	na	na	na
2	1D Convolutional	ReLU	na	na	128	10	[5000,1]	na	na	na	na	na
3	Max pooling	na	na	na	na	na	na	10	2	na	na	na
4	Dropout	na	na	0.1	na	na	na	na	na	na	na	na
5	LSTM	na	na	0.1	na	na	na	na	na	64	0.1	true
6	LSTM	na	na	0.1	na	na	na	na	na	64	0.1	true
7	LSTM	na	na	0.1	na	na	na	na	na	64	0.1	false
8	Flatten	na	na	na	na	na	na	na	na	na	na	na
9	Dense	ReLU	64	na	na	na	na	na	na	na	na	na
10	Output	Linear	5000	na	na	na	na	na	na	na	na	na

Table 3.17: Architecture of RNN model 2.

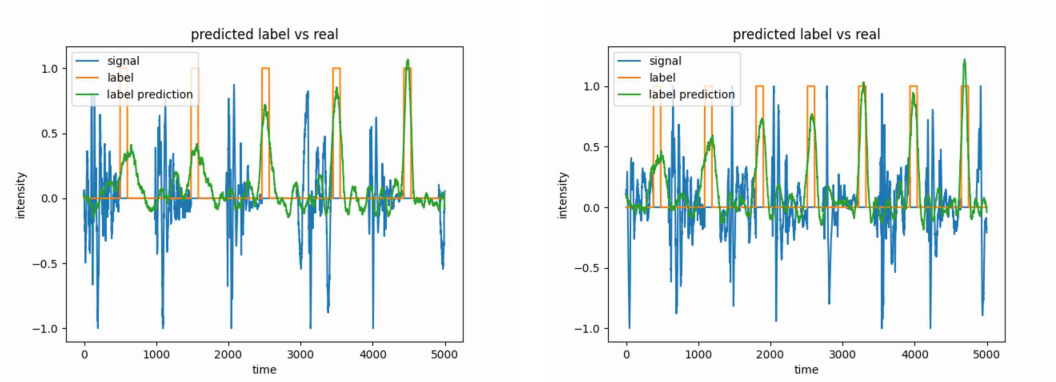


(a) Accuracy (b) Loss

Figure 3.38: RNN model 2 evolution.

After training the loss function of the ANN had a value of 0.0481, the accuracy was 00.00%, the validation loss function had a value of 0.0522 and the validation accuracy was 00.00%. The values for the accuracy can be explained by examining the way the Keras library calculates the accuracy[47].

The results obtained when using this RNN in the train dataset and validation dataset are presented in figure 3.39.



(a) Prediction for the train dataset (b) Prediction for the validation dataset

Figure 3.39: RNN model 2 heartbeat identification.

Although the accuracy was zero, the heartbeats were correctly identified, especially the last ones, this improvement in detection shows that the RNN is taking into account previous heartbeats to identify current ones. This RNN showed promising results and hinting that this method can work.

The discrepancy between the accuracy, loss, and actual identification of heartbeats can be explained by examining how accuracy is calculated in the Keras library [47] and comparing it to the criteria used for heartbeat identification. The Keras library defines accuracy based on an exact match between the predicted and expected values.

Since the output of the ANN may not precisely replicate the label square wave signal, the accuracy reported by the library is zero percent. However, it is important to note that the oscillations of the ANN's output do correspond to the variations of the label. Therefore, for the purpose of heartbeat identification, the matching of the label and oscillations was considered as successful identification, despite not achieving a perfect match according to the accuracy metric. Similarly, the lower-than-expected losses in this ANN can be attributed to the same reason.

Chapter 4

Conclusion

This work aimed to address the heart rate measurement challenges faced by the ECOTOP project. A proposed solution was developed to automate the identification of individual heartbeats in audio files.

To achieve this goal, two ANNs were employed, one for signal identification and another for heartbeat identification. Both ANNs were trained using supervised learning, and data generators were created for both of them.

The signal and noise data generator gave the best results by creating fake data directly from real data. This involved selecting segments containing only noise or heartbeats and creating a dataset to train the ANN. Features were then extracted from these data windows, and this research found that the best-performing features were 15 averages and 15 standard deviations over time from the MFCC block of each window.

The data was then fed into a diamond-shaped, feedforward ANN, achieving an accuracy of 89.94% and a validation accuracy of 90.00%. The ANN demonstrated a good separation between noise detection and signal detection, as evidenced by its analysis of the test file. The utilization of batch normalization, ReLU activation functions, and dropout significantly improved the ANN's classification capabilities.

Although the ANN did not technically achieve the initial goal of 90% accuracy, it fell short by only 0.06 percentage points in the train dataset, which was considered negligible.

The generator of heartbeat data was done by manually selecting heartbeats and saving them to a database, from them a sound file with a label identifying the heartbeats was made. A layer of white noise was also added to improve ANN resilience.

This data was feed to an ANN model that combined a CNN with an RNN using LSTM cells. In spite of the way the accuracy is calculated in the Keras library, which return an accuracy of 0% both for the validation and train datasets, the pulses can

be actually be correctly identified if using a method like the one suggested below. In this ANN the use of convolutional and pooling layers greatly improved its ability to detect the heartbeats.

4.1 Improvements

In this work, there are still several optimizations that can be made.

For example, all the articles that were initial read used MFCC as the way to extract features, and it was by that reason that they were used. However, a simple spectral analysis was never tried and if the heartbeats can be easily identified by visual inspection of the sound spectrogram, then an ANN could identify them as well. The heartbeats are also visible in the MFCCs, but it was never tested which one would be better.

The number of MFCCs could also be further optimized by reducing the number of variables until, for example, 90% of the original variance is reached [48]. This simple process could reduce the training time of the ANN.

The fake heart beat generator could be improved by adding noise to the files, as this can improve ANN generalization [49]. Another improvement would be simply adding more real data that creates the generated one.

The heart beat detection ANN output fluctuates throughout the file as it was seen in section 3.2.5 *ANN model 13* this hinder its capability of detecting signal. This can be improved by having, for example, a moving average that negates the effects of outliers, introducing a sense of time to the detection by correlating the data. Another option would be only considering signal values that were followed by a certain number of signal detections (e.g., only considering a detection signal if the previous three detections were also signal).

Another possible optimization for the heart beat detection ANN, is to use a RoC curve to find the optimal value for the threshold that defines a data point as signal or noise. For example, in section 3.2.5 *ANN model 10* if the threshold was at 0.7 instead of 0.5, the program would have identified all data points correctly except for one outlier.

The audio generator that produces the labeled heartbeats could be enhanced by applying some small improvements such as: Not doing sound amplification of the original heartbeat, instead only moving their average to zero and doing the amplification on the final sound. Applying a Hamming window to the heartbeat, so it can better integrate with the zero-pad. Using noise from the real audio to generate a noise layer that can be added to the generated heart beat.

The ANN that detects individual heartbeats is the section of this work that has more room to improve. Several aspects optimized, including the number of neurons, number of layers, types of activation layers, values for dropouts can all be optimized. The use of RNN cells like GRU, BLSTM, and BiGRU can be tested with BLSTM

likely to yield better results based on previous research [10].

The output can also have some post-processing, for example passing it through a function as in equation 4.1.

$$F(x) = \begin{cases} 1 & if \quad x \geq 0.5 \\ 0 & if \quad x < 0.5 \end{cases} . \tag{4.1}$$

This way, if the output is one for a certain number of samples (later to be defined) the program gives the information that a heartbeat just occurred.

Another possible improvement is to operate the RNN in continuous mode. In this mode, instead of providing the RNN with a fixed window of audio for analysis, the samples are continuously fed into it. This approach can enhance detection performance as the RNN can better identify later heartbeats, as was seen in section 3.3 *RNN model 2*.

Bibliography

[1] "MARE homepage." `https://www.mare-centre.pt/`. Accessed on 2022-08-02.

[2] "Ecotop homepage." `https://www.facebook.com/ecotop.mareuc/`. Accessed on 2022-08-02.

[3] U. Ellenberg, T. Mattern, and P. J. Seddon, "Heart rate responses provide an objective evaluation of human disturbance stimuli in breeding birds," *Conservation Physiology*, vol. 1, no. 1, 2013.

[4] "LIP homepage." `https://www.lip.pt/`. Accessed on 2022-08-02.

[5] C. Q. Howard, J. C. Hodgson, and L. P. Koh, "Heart rate measurement of nesting birds using a microphone in a plastic egg," in *Proceedings of ACOUSTICS*, vol. 7, 2018. ISBN: 9781510877382.

[6] A. J. Nimon, R. C. Schroter, and B. Stonehouse, "Heart rate of disturbed penguins," *Nature*, vol. 374, pp. 415–415, 1995.

[7] J. M. Arnold, R. Ordonez, D. A. Copeland, R. Nathan, J. M. Scornavacchi, D. J. Tyerman, and S. A. Oswald, "Simple and inexpensive devices to measure heart rates of incubating birds," *Journal of Field Ornithology*, vol. 82, no. 3, pp. 288–296, 2011.

[8] A. J. Nimon, R. C. Schroter, and R. K. Oxenham, "Artificial eggs: measuring heart rate and effects of disturbance in nesting penguins," *Physiology & Behavior*, vol. 60, no. 3, pp. 1019–1022, 1996.

[9] T.-E. Chen, S.-I. Yang, L.-T. Ho, K.-H. Tsai, Y.-H. Chen, Y.-F. Chang, Y.-H. Lai, S.-S. Wang, Y. Tsao, and C.-C. Wu, "S1 and s2 heart sound recognition using deep neural networks," *IEEE Transactions on Biomedical Engineering*, vol. 64, no. 2, pp. 372–380, 2016.

[10] S. Latif, M. Usman, R. Rana, and J. Qadir, "Phonocardiographic sensing using deep learning for abnormal heartbeat detection," *IEEE Sensors Journal*, vol. 18, no. 22, pp. 9393–9400, 2018.

[11] P. Narváez, S. Gutierrez, and W. S. Percybrooks, "Automatic segmentation and classification of heart sounds using modified empirical wavelet transform and power features," *Applied Sciences*, vol. 10, no. 14, p. 4791, 2020.

[12] J. O. Smith III, *Spectral Audio Signal Processing*. W3K Publishing, 2011. ISBN: 978-0-9745607-3-1.

[13] F. Akram, M. A. Garcia, and D. Puig, "Active contours driven by difference of Gaussians," *Scientific reports*, vol. 7, no. 1, p. 14984, 2017.

[14] L. Rabiner and R. Schafer, *Theory and applications of digital speech processing*. Prentice Hall Press, 2010. ISBN:978-0-13-603428-5.

[15] "Librosa homepage." `https://librosa.org/`. Accessed on 2022-08-04.

[16] "Python homepage." `https://www.python.org/`. Accessed on 2022-08-04.

[17] Z. K. Abdul and A. K. Al-Talabani, "Mel frequency cepstral coefficient and its applications: A review," *IEEE Access*, vol. 10, pp. 122136–122158, 2022. https://www.doi.org/10.1109/ACCESS.2022.3223444.

[18] S. W. Smith *et al.*, *The scientist and engineer's guide to digital signal processing*. California Technical Pub. San Diego, 1997. ISBN: 0-9660176-7-6.

[19] T. S. community, "scipy.signal.hann." `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.signal.hann.html`, 2014. Online; accessed on 2023-02-24.

[20] K. R. Rao, D. N. Kim, and J. J. Hwang, *Fast Fourier transform: algorithms and applications*. Springer, 2010. https://www.doi.org/10.1007/978-1-4020-6629-0.

[21] S. Karpagavalli and E. Chandra, "A review on automatic speech recognition architecture and approaches," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 9, pp. 393–404, 04 2016. https://www.doi.org/10.14257/ijsip.2016.9.4.34.

[22] E. Grossi and M. Buscema, "Introduction to artificial neural networks," *European journal of gastroenterology & hepatology*, vol. 19, pp. 1046–54, 01 2008. https://www.doi.org/10.1097/MEG.0b013e3282f198a0.

[23] J. Dacombe, "An introduction to artificial neural networks (with example)." `https://medium.com/@jamesdacombe/an-introduction-to-artificial-neural-networks-with-example-ad459bb6941b`, 2017-10-23. Online; accessed on 2023-02-28.

[24] M. Zakaria, A. Mabrouka, and S. Sarhan, "Artificial neural network: a brief overview," *neural networks*, vol. 1, p. 2, 2014.

[25] D. Harris and S. Harris, *Digital Design and Computer Architecture*. Morgan Kaufmann, 2007. ISBN: 9780080547060.

[26] P. Sharma, N. Malik, N. Akhtar, and H. Rohilla, "feedforward neural network: A review," *International Journal of Advanced Research in Engineering and Applied Sciences (IJAREAS)*, vol. 2, no. 10, pp. 25–34, 2013.

[27] "Keras homepage." `https://keras.io/`. Accessed on 2023-04-18.

[28] "Tensorflow homepage." `https://www.tensorflow.org/`. Accessed on 2023-04-18.

[29] W. Pedrycz and S.-M. Chen, *Deep learning: algorithms and applications*. Springer, 2020. https://www.doi.org/10.1007/978-3-030-31760-7.

[30] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through l_0 regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[31] S. C. Douglas and J. Yu, "Why relu units sometimes die: analysis of single-unit error backpropagation in neural networks," in *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, pp. 864–868, IEEE, 2018.

[32] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, "PLMR", 2015.

[33] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[34] F. Chollet, *Deep learning with Python*. Manning Publications Co., 2017. ISBN: 9781617294433.

[35] H. Pishro-Nik, *Introduction to Probability, Statistics, and Random Processes*. Kappa Research, LLC, 2014. ISBN: 9780990637202.

[36] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[37] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," *arXiv preprint arXiv:1804.07612*, 2018.

[38] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, vol. 1168, p. 022022, IOP Publishing, 2019.

[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[40] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, *et al.*, "Recent advances in convolutional neural networks," *Pattern recognition*, vol. 77, pp. 354–377, 2018.

[41] S. Sakib, N. Ahmed, A. J. Kabir, and H. Ahmed, "An overview of convolutional neural network: its architecture and applications," *Preprints*, 2019. https://www.doi.org/10.20944/preprints201811.0546.v4.

[42] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[43] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer Berlin Heidelberg, 2012. https://www.doi.org/10.1007/978-3-642-24797-2.

[44] Y. Chen, K. Zhong, J. Zhang, Q. Sun, and X. Zhao, "LSTM networks for mobile human activity recognition," in *2016 International conference on artificial intelligence: technologies and applications*, pp. 50–53, Atlantis Press, 2016.

[45] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *22nd ACM International Conference on Multimedia (ACM-MM'14)*, (Orlando, FL, USA), pp. 1041–1044, Nov. 2014.

[46] P. Johansson, L. Hermansson, and M. Henning, "Cardiovascular effects of clonidine in an avian species, larus argentatus," *Naunyn-Schmiedeberg's Archives of Pharmacology*, vol. 318, pp. 62–65, 1981.

[47] F. Chollet, "accuracy_metrics.py." `https://github.com/keras-team/keras/blob/f9336cc5114b4a9429a242deb264b707379646b7/keras/metrics/accuracy_metrics.py#L354`, 2023. Online; accessed on 2023-06-08.

[48] R. Zebari, A. Abdulazeez, D. Zeebaree, D. Zebari, and J. Saeed, "A comprehensive review of dimensionality reduction techniques for feature selection and feature extraction," *J. Appl. Sci. Technol. Trends*, vol. 1, no. 2, pp. 56–70, 2020.

[49] A. J. C. SHARKEY, "On combining artificial neural nets," *Connection science*, vol. 8, no. 3-4, pp. 299–314, 1996.