MASTER'S THESIS

MASTER IN MICROELECTRONICS: DESIGN AND APPLICATIONS OF MICRO/NANOSCALE SYSTEMS

Cyber-Physical Systems for Remote Animal Monitoring

Automated birdsong recognition: A case study of *Falco Naumanni*

Author
Carmen Lozano Pons

Supervisors
Jorge Fernández Berni
Rocío del Río Fernández

Seville, 2020

Cyber-Physical Systems for Remote Animal Monitoring

Carmen Lozano Pons

Abstract

The aim of this project is to find a procedure for the acoustic recognition of a target species, the lesser kestrel (*Falco naumanni*). A bibliographical research in birdsong recognition has been made to find all possible tools out.

Among several techniques for feature extraction and data classification, Mel Frequency Cepstrum Coefficients and a shallow neural network have been chosen, respectively, to test their functionality.

For that task, recordings of different species of birds are required, so a database has been prepared based on local recordings found in Xeno-canto database.

Once the method has been evaluated, it is implemented in the microcontroller of a commercial low-cost device, AudioMoth, specifically designed for acoustic monitoring. This was performed by modifying and expanding the functionality of the project developed by the company.

Keywords: Acoustic, Lesser kestrel, Birdsong, Automated recognition, MFCC, Neural network, Commercial

Acknowledgements

To my supervisors, Jorge Fernández and Rocío del Río, and Javier Bustamante, for their guidance and encouragement during this stage, especially for giving me the opportunity to introduce me to nature conservation, one of my life values.

To my parents and friends, who listened to me every time I had a problem and clapped me when I solved it, even if they didn't understand anything I was saying.

To those who enjoy birding, recording birdsongs and sharing with everyone. Without them, this work could not be conducted. I am really grateful to José Carlos Sires, who is the author of the majority of the data I used. What a coincidence that I ended up in his selfless lessons in times of quarantine.

Thank you.

Contents

List of Figures

List of Tables

Acronyms

ADC Analog to Digital Converter

ARM Acorn RISC Machine

CMSIS Cortex Microcontroller Software Interface Standard

DC Direct Current

DCT Discrete Cosine Transform

DMA Direct Memory Access

DFT Discrete Fourier Transform

DSP Digital Signal Processing

EBI External Bus Interface

FFT Fast Fourier Transform

FPU Floating Point Unit

GMM Gaussian Mixture Model

HMM Hidden Markov Model

LED Light Emitting Diode

LPC Linear Predictive Coding

LPCC Linear Prediction Cepstrum Coefficients

FFT Fast Fourier Transform

FN False Negative

FP False Positive

MEMS Micro-Electro-Mechanical System

MFCC Mel Frequency Cepstral Coefficients

MP3 MPEG-1 Audio Layer III

NN Neural Network

NPV Negative Predictive Value

PCB Printed Circuit Board

RAM Random Access Memory

RISC Reduced Instruction Set Computer

SOM Self-Organising Map

SPI Serial Peripheral Interface

SRAM Static Random Access Memory

TN True Negative

TP True Positive

USB Universal Serial Bus

WAV Waveform Audio File

Chapter 1

Introduction

Fast environmental changes can impact species populations, producing rising, decline or even extinctions of species, having a dramatic impact on biodiversity.

For conservation programs, it is necessary to evaluate the size of populations and to keep tracking of them in the long term. Acoustic monitoring tools are very useful when monitoring breeding behaviour or population dynamics, allowing for continuous recording without constant direct-human monitoring. These tools also resolve monitoring in inaccessible ecosystems.

However, they can produce weeks or months of field recordings, which is time-consuming for an expert to verify how many and what species appear. Therefore automatic recognition can help in this process.

This project, which has been realized in collaboration with Doñana Biological Station[1], pursues automatic recognition of *Falco naumanni* (Fleischer, 1818) in their natural habitat employing a low-cost acoustic detector.

In this chapter, objectives are presented, followed by a brief description of the target species and the state of the art.

In subsequent chapters, the chosen procedure and its implementation in the acoustic monitoring tool will be detailed.

1.1 Objetives

The aim of this work is to develop a method for automatic recognition of *F. naumanni*, based on the sounds that this species emits. It would allow to discern target species from those ones which it coexists with, so subsequent ecological studies about size colony or population state could be accomplished.

This method will be evaluated and afterwards, implemented on a low-cost platform, AudioMoth, which is a device intended for acoustic detection.

The whole system may be applied to the monitoring of a local colony in the future, relating number of occurrences to number of breeding pairs and their reproductive success.

1.2 Lesser kestrel

As said before, our target species is the lesser kestrel (*F. naumanni*), a small falcon of the order Falconiformes.

[1] http://www.ebd.csic.es/inicio

1.2.1 Biology and distribution

These birds are gregarious, and they hunt in flocks chiefly flying insects [PMH01]. This kind of living, in colonies, has to be taken into account because they are likely to produce simultaneous vocalizations.

In Spain, breeding population is mainly distributed in the south-west (below parallel 42). Mediterranean populations come from reintroduction programs (except for a small population in Murcia). The species makes premigratory movements towards the north of the Peninsula before beginning migration to Africa. During this stay (late June to early October) they can congregate in large roosts of more than 1000 individuals. In Spain, some of them spend the winter in the Ebro valley, Castilla y León, Extremadura and Andalusia, although most stay in Africa. However, the exact wintering areas of the population are unknown [TS04].

In their colonies there are frequently jackdaws, parrots, sparrows, starlings, swift, doves...

1.2.2 Threats

The major cause of the decline of kestrels has been habitat degradation due to agricultural intensification and land use changes [IB10]. Main threats can be summarized in:

1. Breeding success reduction. Factors that lead to that are decrease access to prey, loss of suitable breeding sites or deliberate destruction of nest sites.
2. Adult mortality increase. Electrocution in electrical facilities and elimination of pre-migration roosting sites are the reasons for this increase.
3. Lack of effective conservation. That happens because protection and monitoring of migration areas are difficult to coordinate.

1.2.3 Sounds

Songbird vocal production apparatus is composed of the lungs (source of airpumping), the syrinx and the vocal tract (cavities formed by the trachea, mouth and beak). The songbird syrinx (Fig. 1.1) contains two sound sources, each consisting of a pair of labia. They are set into vibration by a passing airstream. This primary acoustic signal is then filtered as it passes through the vocal tract [RG10]. This process is outlined in Fig. 1.2.

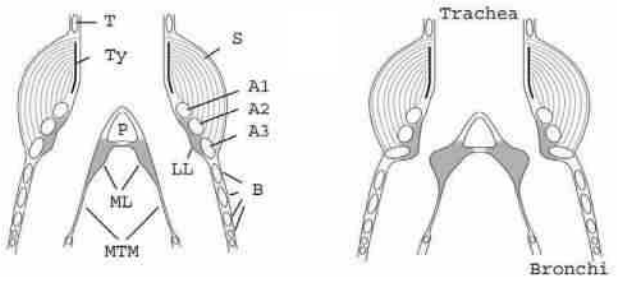


Figure 1.1: Schematic of a cross section of the syrinx. ML and LL indicate the labia [RG10].

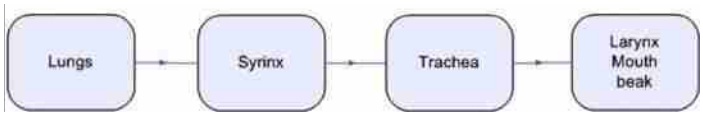


Figure 1.2: Sound production model [Pot+14].

Birds produce different sounds to communicate. These can be categorized in calls, songs and mechanical sounds (bill-clapping, drumming...). Calls are usually composed of short monosyllabic sounds

that may have many functions (begging, warning, alarm, territorial...). While all birds produce calls, only some birds make songs, which are longer and with a more complex structure [Pot+14].

A bird may generate the same song with short or long duration in different situations. Further, they adapt their sounds according to the environment or also generate incomplete calls in critical situations, especially during the breeding season when they are occupied with incubation and chick rearing [PMC18].

Specifically, the vocalizations of kestrels are heard manly in the breeding season, and consist of screams or chattering notes. Also, many species protest loudly when the nest is visited [CL11].

A lesser kestrel can produce different kinds of sounds: begging calls, alarm calls, plaintive and rising *whee* trills, interaction calls and flying calls. Last ones are the most representative and easy to hear, so they were selected for identification. It consists of a trisyllabic rasping *chae-chae-chae* (very different from common kestrel [RP20] [PMH01]). Sometimes, they extend this call, making a polysylabic sequence. Its spectrogram can be observed in Fig. 1.3. In that figure, three vocalizations are distinguished. They exhibit different levels of distortion because of the distance between the bird and the microphone.

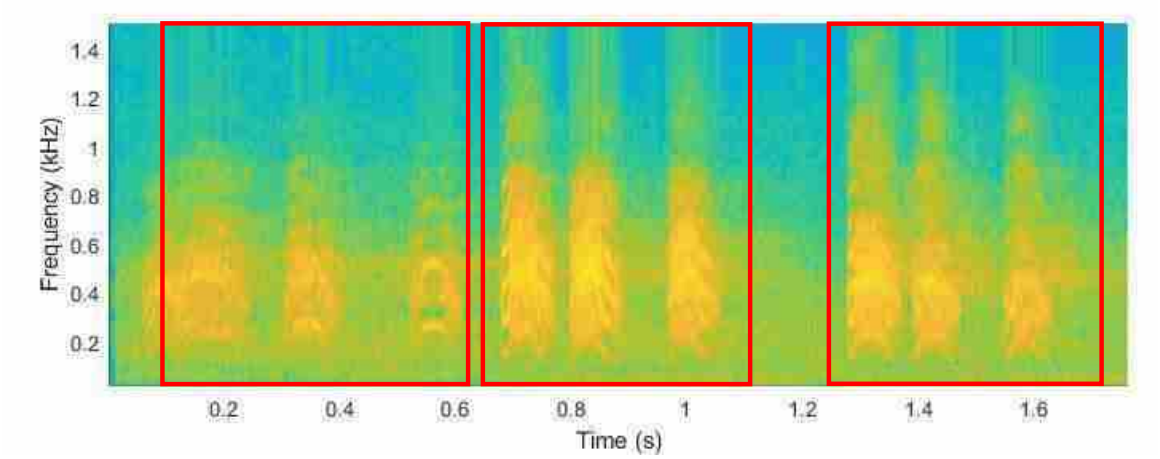


Figure 1.3: Flying calls of *F. naumanni* marked by boxes[2].

1.3 State of the art

In this section, the methods currently available related to the essential parts of a birdsong recogniser are summarised. These main parts are preprocessing (specifically noise reduction), call detection or segmentation, feature choice, and classification.

1.3.1 Noise reduction

Field recordings include any sounds that the recorder can detect, including birdsong of interest and many other biophonies, geophonies and anthrophonies. Removing the noise prior to birdsong detection and recognition is constrained by the temporal and frequency overlap of the noise and the birdsong.

The most common approach is to apply standard signal processing techniques and to perform noise profiling, followed by filtering. However, there are situations where high-pass, low-pass, or band-pass filters cannot be applied successfully (because they would remove frequency components of the birds' frequency range [PMC18]).

[2]It can be heard in https://drive.google.com/file/d/1FAtR--VGPLO6PtRSfeogR_aOEMHLsi28/view?usp=sharing

The Wiener filter is not useful for birdsong because it assumes that the signal and noise are stationary and that spectral information is available. Unfortunately, the solution for this, adaptative Wiener filter, suffers from signal distortion [PMC18].

A recent approach is the use of wavelets to remove noise from field recordings collected with automatic sound recorders. Wavelet denoising eliminates this quasi-stationary noise no matter whether it is wideband or narrowband, provided that it is approximately Gaussian [Pri+20].

An alternative to audio analysis is treating the spectrograms as images and applying image analysis methods to clean them, but this method does not permit the construction of a denoised sound file from the cleaned spectrogram [PMC18].

1.3.2 Segmentation

In order to recognise the birdsong in the recording, it is needed to isolate calls first. This is not simple because sometimes syllables are not followed by a silent interval and are not separated clearly. Merging the syllables that are very close to each other is common practice in syllable detection. Further, background noise challenges isolation.

This step has been approached in different ways [PMC18]:

- Energy-based song detection coupled with thresholding, based on the assumption that the sections where the birds sing carry more energy than the other parts. This depends on the present noise and the proximity to the recorder. It tends to fail to detect faint songs, but also detects periods of noise that exceed the chosen threshold.
- Treating the spectrogram as an image and use median clipping, whereby points are identified as birdsong if they are greater than some pre-defined multiple of the median of the relevant column and row of the spectrogram. This procedure together with basic shape morphology methods can be used to improve the process.
- Applying a sinusoidal detection method (assuming that the number of species in a given recording is known).
- Methods that are based on the temporal patterns in the frequency bands of the target species and on the noise estimation from each band followed by spectral subtraction to avoid noise.

However, for long field recordings, which often contain a great deal of ambient noise in addition to the sounds of various animals, separating segmentation from recognition is not always effective. Firstly, each loud sound in the recording will be extracted, even those that are not biophonic, and secondly, quiet calls will be ignored since they do not rise sufficiently far above the noise level. Finally, the recorded soundscape often combines a small number of species of interest together with more common species, and segmenting all of the sounds individually is inefficient [Pri+20].

1.3.3 Feature extraction

To produce inputs for a classification algorithm, relevant features must be extracted from the call.

Possible features can be as simple as the sequence of amplitudes or the raw spectrogram values, but usually is better to utilize more descriptive features.

Features can be based on the amplitude plot (bandwidth, number of zero crossings...), on the energy of the signal within the window, or on the short-time Fourier transform data.

Some examples that have been used in the literature are Linear Predictive Coding (LPC) and its extension, Linear Prediction Cepstrum Coefficients (LPCC), that were initially used for encoding human speech, as well as Mel Frequency Cepstral Coefficients (MFCC), that use the Mel scale filter bank. Mostly, MFCC are used with their first and second order derivatives in order to capture dynamic features of the vocal tract [PMC18].

Another option is employing features learnt automatically from data, for example, creating learnt

bases using spherical k-means to obtain an automatic feature transformation [SP14].

Wavelet analysis is an alternative too, and it can avoid the trade-off in time-frequency resolution, present in Fourier analysis [Pri+20].

Despite the amount of techniques that have been proposed, there is no clear evidence for and against different representations for birdsong [PMC18].

1.3.4 Data classification

Once a feature representation has been chosen, feature vectors extracted from the sound are fed into a machine learning algorithm, which will cluster those that are similar, either in an unsupervised or supervised way.

Machine learning algorithms that have been used in the literature for birdsong recognition are briefly described below [PMC18]:

- A statistical learning method called Gaussian Mixture Model (GMM). It assumes that observations (such as recorded bird song) come from a weighted combination of inputs that can be described by Gaussian random variables.
- The most commonly-used form of machine learning for birdsong recognition is the Neural Network (NN). However, its ability to discriminate between species degrades as a larger variety of calls is introduced.
- Neural networks that perform unsupervised learning cluster similar calls together rather than using human labels to recognise calls from the same species. These ones are also commonly used, particularly the Self-Organising Map (SOM).
- Support Vector Machine. It is a machine learning approach that maps data into a higher dimensional space where it can be linearly separated.
- Decision trees. At each node of the tree, a split is performed using just one of the features from the input vector. A sequence of these splits enables a decision as to species. Random forests are used too. They consist of a collection of decision trees created with random partitions of the data. Their outputs are combined by majority voting.
- In order to recognise a sequence of syllables, the Hidden Markov Model (HMM) can be applied. This method creates a time-dependent probability distribution showing how likely certain syllables are to follow from others.
- Spectrogram cross-correlation takes a segment of the spectrogram and computes the cross-correlation with a set of template calls. It is proven to be successful when scanning a specific species with limited call variations.

The performance of most of these methods progressively degrades as more calls from more species are introduced [PMC18].

Chapter 2

Audio signal processing and inference

This chapter describes how the dataset was acquired and presents the song identification algorithm.
It is worth mentioning that segmentation has been avoided because of the reasons explained in Subsec. 1.3.2, whereas for feature extraction, MFCCs will be employed since they have been widely used in automatic bird recognition.
Finally, data classification will be performed by a neural network, because [PMC18] recommends this approach when there are few labelled data available, which is this case.
This is visually outlined in Fig. 2.1.

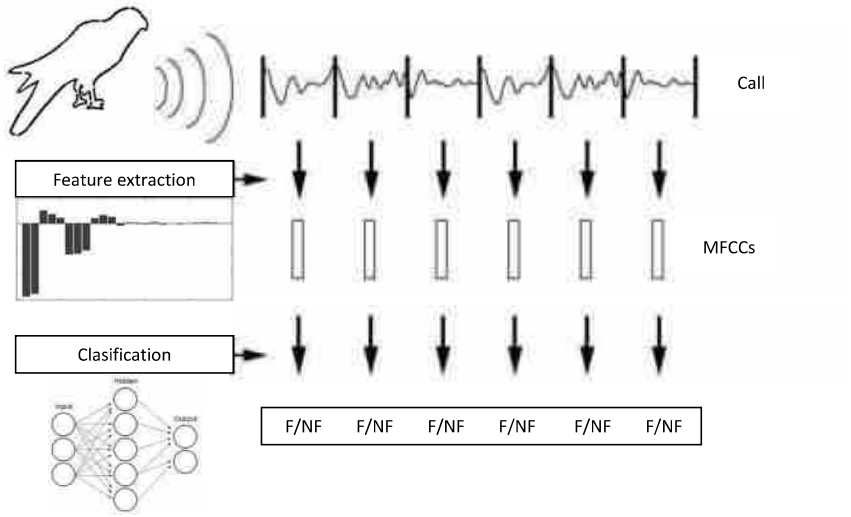


Figure 2.1: Procedure for lesser kestrel recognition. The signal is divided in segments, MFCCs are extracted from each one of these segments, and the neural network classifies them in one of two possible classes, Falco and Not-Falco (N/NF).

2.1 Data selection

2.1.1 Database

For a real-world application, it is important to work with data from real field. To accomplish that, several audio recordings have been downloaded from *www.xeno-canto.org*, a collaborative project dedicated to share bird sounds.

Location is associated with each song (see Fig. 2.2), so those ones recorded in western Andalusia were selected, due to the fact that there can be large intra-species song variability [PMC18].

Quality of audio recordings are also specified with letters 'A', 'B', 'C' and 'D'. In this work, audio recordings labelled with 'A' and 'B' were used. The theory seems to be that by using high quality data for training, the algorithms will deal well with noise in true recordings, although this is not tested, and does not seem particularly likely [PMC18].

All recordings have a sampling rate of 44100 Hz.

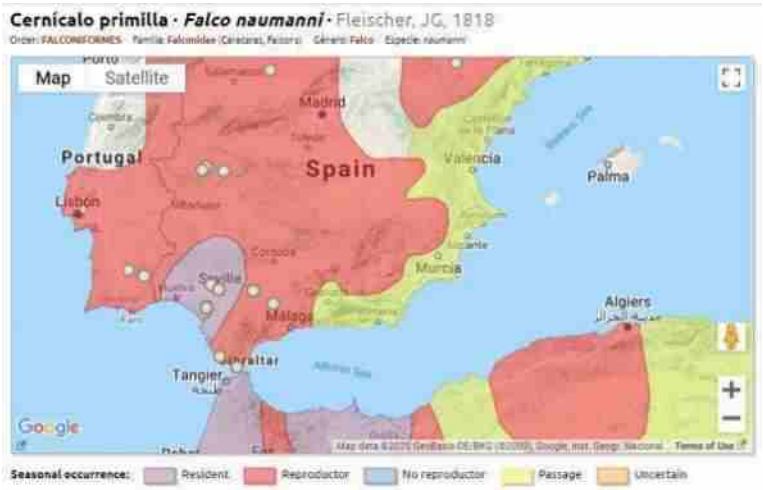


Figure 2.2: Map representation of the places where there are recordings.

Apart from lesser kestrel, other species which live in the same area were selected (Table 2.1). The total time of audio signal after data preparation of each species (Subsec. 2.1.2) is indicated, as well as the number of resulted inputs (groups of MFCC for training the neural network).

Species	Total time (s)	Inputs to NN
Falco naumanni	222.50	8747
Streptopelia decaocto	19.08	801
Sturnus unicolor	41.00	1720
Falco tinnunculus	12.13	473
Coloeus monedula	17.07	682
Apus Apus	76.67	3241
Emberiza calandra	85.87	3673
Upupa epops	45.67	1862
Columba livia	332.66	14146
Passer passer	37.40	1542
Background	477.42	20560

Table 2.1: List of species selected to perform the training of the system.

2.1.2 Data preparation

Syllable segmentation was done manually employing Audacity[1], a free and open-source digital audio editor, to make sure that quiet intersyllabic sounds are not employed in data training. Thus, reverberation

[1] https://www.audacityteam.org

has been carefully avoided, as it can be seen in Fig. 2.3.

Audacity can point regions with power spectrum below or above a chosen power value. To delimit the volume of a detectable sound, that value was fixed to -20 dB, so all lesser kestrel sounds for training are above that level.

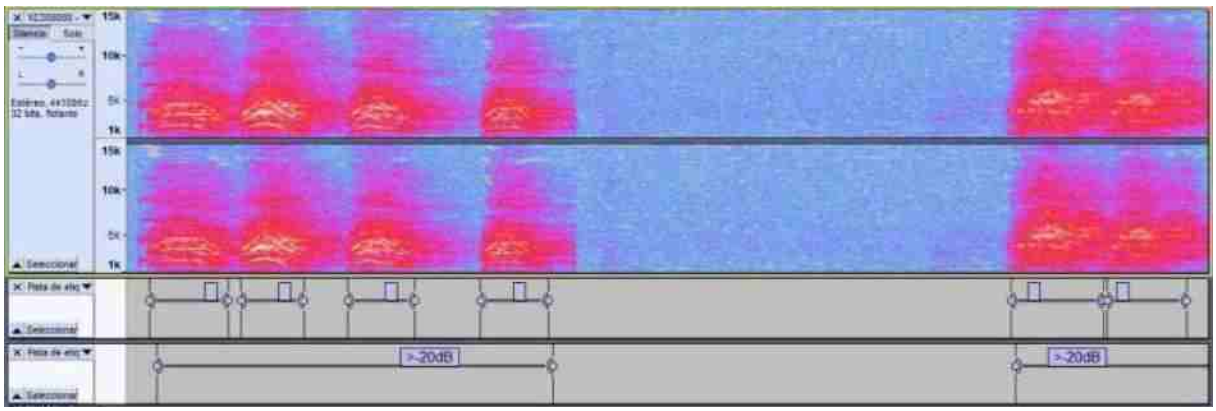


Figure 2.3: Syllable segmentation. Audacity generates labels indicating a sound pressure level above -20 dB (labels below). However, the manually extracted segments (labels above) avoid reverberation.

2.2 Feature extraction

As said before, feature extraction is based on MFCCs. In this section, its basis is presented, as well as some details about its implementation. This procedure is applied both to the training data and to the data to classify.

2.2.1 MFCCs

MFCCs are a representation of an acoustic signal as a result of a cosine transform of the real logarithm of the short-term energy spectrum expressed on a mel-frequency scale. Davis and Mermelstein were the first who used 'Mel Frequency Cepstral Coefficient' term [DM80]. They combined perceptual distributed triangular filters with cosine transform of logarithms of the energy filter outputs. They only propose a general description (without detailing number, shape or overlap between filters), so researchers adapt the algorithm to their experiments.

They have been widely used in speech recognition because they are based on human auditory perception (which may not be applicable for birds).

The calculation of these features includes the following steps [RV14]:

1. Apply the following pre-emphasis filter:

$$x_f(n) = x(n) - a \cdot s(n-1) \tag{2.1}$$

   where a is usually between 0.9 and 1. The signal is filtered with the aim to amplify high frequencies, which are affected by a -6 dB/decade attenuation caused by sound production in humans [d'A+05]. Not all authors consider this step.
2. Frame the signal into short frames with optional overlap. For stable acoustic characteristics, signal needs to be examined over a sufficiently short period of time. Therefore, signal analysis must always be carried out on short segments where the speech signal is assumed to be stationary. Short-term spectral measurements are typically carried out over 20 ms windows.
3. Multiply each frame with a Hamming window to keep the continuity of the boundaries in the frame,

in order to enhance the harmonics, smooth the edges, and reduce the edge effect while taking the Discrete Fourier Transform (DFT) on the signal. The Hamming window equation is given by:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \qquad 0 \leq n \leq N-1 \tag{2.2}$$

where N is the number of points of the window.

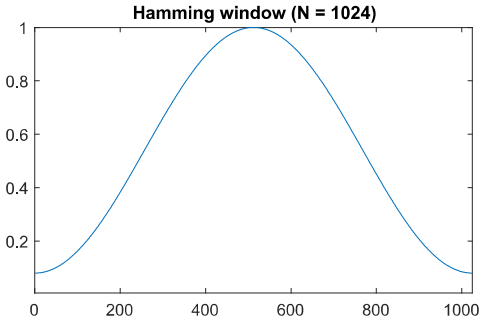


Figure 2.4: Hamming window of 1024 points.

4. Calculate the power spectrum of each frame.
5. Warp the spectrum along the frequency axis into the mel-frequency axis. The human auditory system does not perceive pitch linearly, so a transformation to a scale based on human perceived frequency is done:

$$m_f(f) = 1127\ln(1 + f/700) \tag{2.3}$$

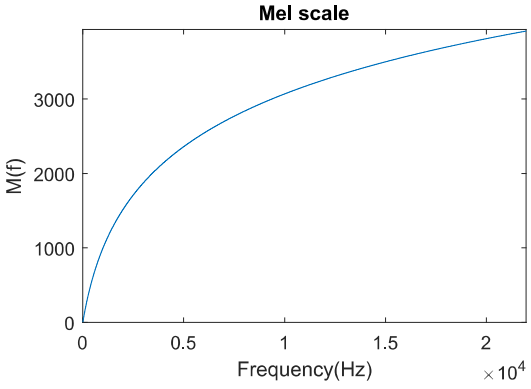


Figure 2.5: Mel scale.

where f denotes physical frequency in Hz and m_f denotes perceived frequency.

6. The resulted warped spectrum is convolved with band-pass filters, implemented in frequency

domain:

$$s(m) = \sum_{k=0}^{N-1} \left[|X(k)|^2 H_m(k) \right] \qquad 0 \leq m \leq M-1 \tag{2.4}$$

where M is the number of mel filters. $H_m(k)$ is the weight given to the k-th energy spectrum bin contributing to the m-th output band. The most commonly used filter shape is triangular. One possible definition of these bins can be given by:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \frac{2(k-f(m-1))}{f(m)-f(m-1)} & f(m-1) \leq k \leq f(m) \\ \frac{2(f(m-1)-k)}{f(m+1)-f(m)} & f(m) \leq k \leq f(m+1) \\ 0 & k > f(m+1) \end{cases} \tag{2.5}$$

with m ranging from 0 to $M-1$. This expression is represented by Fig. 2.6. The filter bank, normalized by its area, is shown in Fig. 2.7.

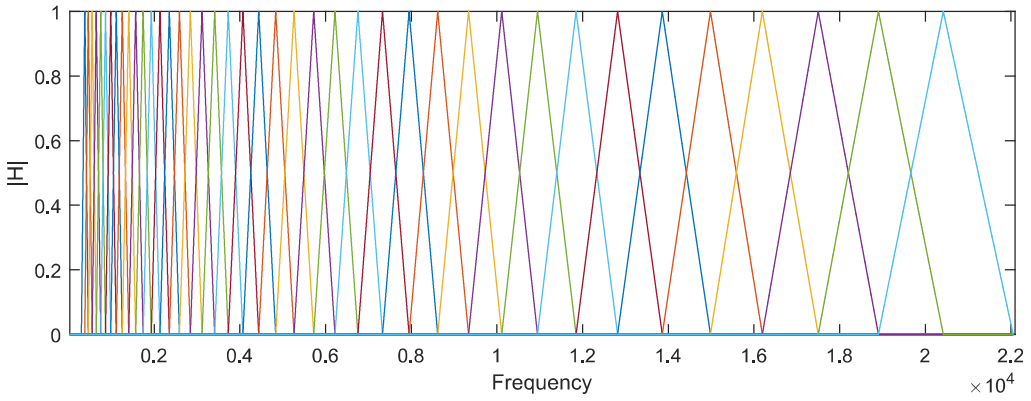


Figure 2.6: Mel filter banks without area normalization.

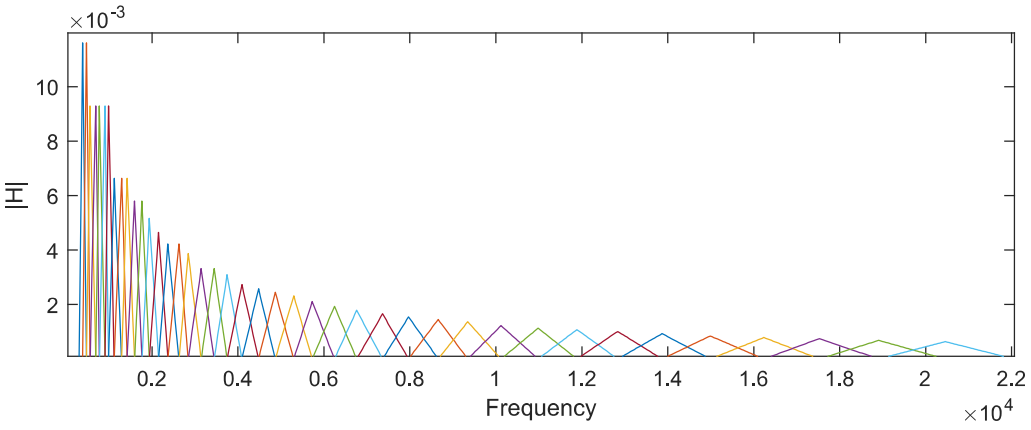


Figure 2.7: Mel filter banks with area normalization.

7. Take the logarithm of all filter bank energies. Then, take the Discrete Cosine Transform (DCT) of the log filter bank energies:

$$c(n) = \sum_{m=0}^{M-1} \log_{10}[s(m)] \cos\left[\frac{\pi n(m-0.5)}{M}\right]; \qquad n = 0,1,2,...,C-1 \tag{2.6}$$

where $c(n)$ are the cepstral coefficients and C is the number of MFFCs. The logarithm is done because to perceive doubled volume, it is necessary to increase the energy eight times. Additionally, the DCT decorrelates the energies, which tend to be correlated in adjacent bands since the vocal tract is smooth.
Traditional MFCC systems use only 13 cepstral coefficients because high coefficients represent fast changes and degrade performance. The zeroed coefficient is often excluded since it represents the average log-energy of the input signal (little speaker-specific information).

With the aim of capturing temporal dynamic information, the first derivatives of cepstral coefficients are also calculated, and they are the so-called delta coefficients. They represent MFCC changes. For a given MFCC c_i in a temporal sequence, the delta features are given by:

$$\delta c_m(n) = \frac{\sum_{i=1}^{T} i(c_{i+m}(n) - c_{i-m}(n))}{2\sum_{i=1}^{T} i^2} \tag{2.7}$$

where $c_m(n)$ denotes the n-th feature for the m-th time frame and T is the number of successive frames used for computation. A typical T value is 2.

2.2.2 Use of MFCCs

After explaining the general procedure of MFCC extraction, we next describe how it has been applied to the present work.

The incoming audio signal was resampled to 32 kHz and it was divided into frames of 32 ms (to adjust the window length to 1024 samples and make easier the subsequent programming in C). Then, in each frame, MFCCs were extracted following the scheme of Fig. 2.8, in order to reduce dimensionality.

Instead of calculating the power spectrum, the absolute value of the FFT was taken to reduce computational cost. The number of filters banks was 41, which [Gra+10] found appropriate to reduce the error rate.

The weights of each band-pass filter were normalized by its corresponding area (Fig. 2.7).

On the other hand, the number of coefficients were 12, so the zeroth MFCC was not included. Both the shape of Mel filter banks and the exclusion of the zeroth coefficient resulted in better performance when tested (see Sec. C).

Delta-MFCC were calculated too, employing the difference of five MFCCs ($T = 2$), as indicated in Fig. 2.9. On the other hand, when calculating deltas, it was necessary to repeat the first and last MFCC coefficients $T/2$ times at the beginning and at the end in order to not produce a discontinuity.

In conclusion, the feature vector included 12 MFCCs and their 12 corresponding deltas.

To see the code of the implemented script, please consult Appendix B. A specific code has been developed instead of using the MATLAB predefined function, for better control of parameters. With the same parameters, results from our code and MATLAB code have been compared, showing the same behaviour.

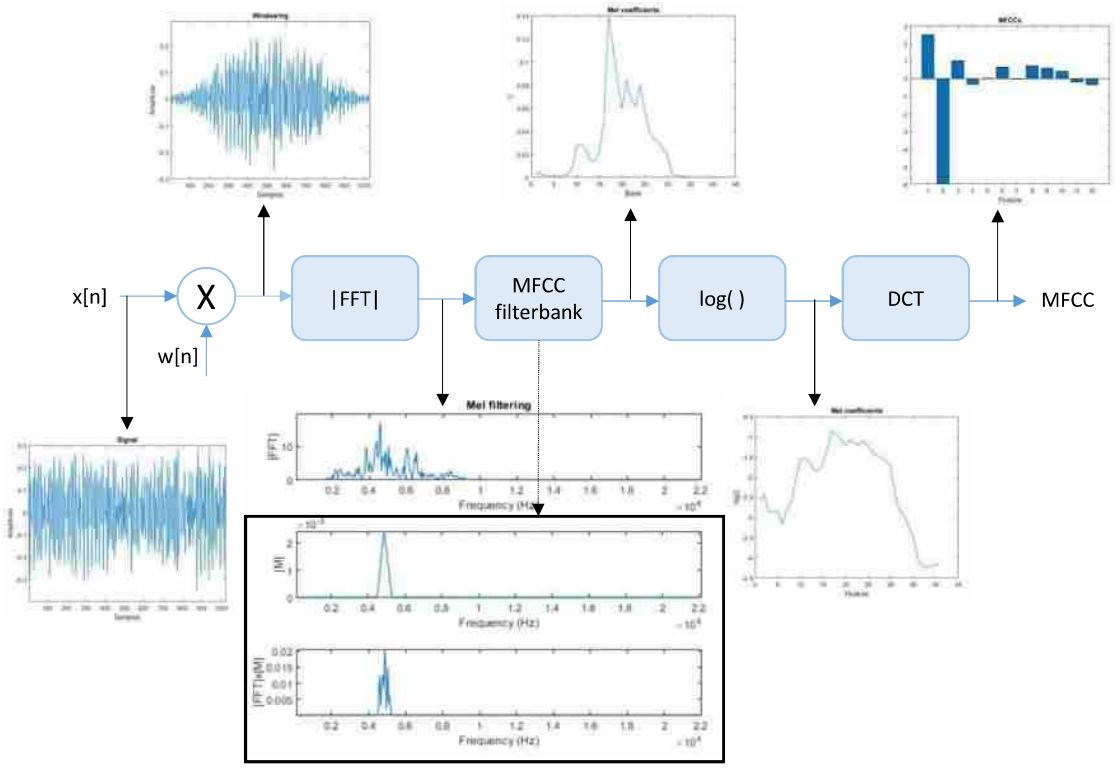


Figure 2.8: Block diagram of the Mel Frequency Cepstral Coefficients extraction procedure.

	c(0)	c(1)	...	c(C-1)
	$c_1(0)$	$c_1(1)$	...	$c_1(C-1)$
	$c_1(0)$	$c_1(1)$	...	$c_1(C-1)$
	$c_1(0)$	$c_1(1)$	...	$c_1(C-1)$
	$c_1(0)$	$c_1(1)$	...	$c_1(C-1)$
frame1	$c_1(0)$	$c_1(1)$	...	$c_1(C-1)$
frame2	$c_2(0)$	$c_2(1)$	...	$c_2(C-1)$
frame3	$c_3(0)$	$c_3(1)$	...	$c_3(C-1)$
frame4	$c_4(0)$	$c_4(1)$	...	$c_4(C-1)$
frame5	$c_5(0)$	$c_5(1)$	...	$c_5(C-1)$
...	...	...	...	...
frame$^\infty$	$c_\infty(0)$	$c_\infty(1)$	...	$c_\infty(C-1)$
	$c_\infty(0)$	$c_\infty(1)$	...	$c_\infty(C-1)$
	$c_\infty(0)$	$c_\infty(1)$	...	$c_\infty(C-1)$
	$c_\infty(0)$	$c_\infty(1)$	...	$c_\infty(C-1)$
	$c_\infty(0)$	$c_\infty(1)$	...	$c_\infty(C-1)$

	Δc(0)	Δc(1)	...	Δc(C-1)
frame1	$\Delta c_1(0)$	$\Delta c_1(1)$	...	$\Delta c_1(C-1)$
frame2	$\Delta c_2(0)$	$\Delta c_2(1)$	...	$\Delta c_2(C-1)$
frame3	$\Delta c_3(0)$	$\Delta c_3(1)$	...	$\Delta c_3(C-1)$
frame4	$\Delta c_4(0)$	$\Delta c_4(1)$	...	$\Delta c_4(C-1)$
frame5	$\Delta c_5(0)$	$\Delta c_5(1)$	...	$\Delta c_5(C-1)$
...	...	...	...	...
frame$^\infty$	$\Delta c_\infty(0)$	$\Delta c_\infty(1)$	...	$\Delta c_\infty(C-1)$

Figure 2.9: Example of calculation of one delta feature (yellow) from adjacent frames.

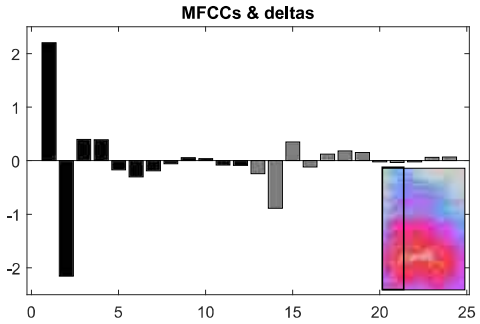


Figure 2.10: An example of the MFCC values of a lesser kestrel call window. The MFCCs are in black and the deltas in gray. Inside, an inset with the vocalization with the processed window marked.

2.3 Classification

This section is organized as the previous one. The foundation of the selected machine learning method, a shallow neural network, is presented, followed by its specific implementation.

2.3.1 Neural networks: theory

Architecture

Artificial neural networks are computing systems inspired by biological neural networks that constitute animal brains. They are applied to a wide variety of problems, for example, pattern classification.

They are composed of interconnected artificial neurons (Fig. 2.11). The neuron output is calculated as:

$$a = f(\mathbf{W}\mathbf{p} + b) \tag{2.8}$$

Inputs $\mathbf{p}$ are multiplied by the scalar weight matrix, $\mathbf{W}$, and the bias b is added. The output goes into a transfer or activation function f, which produces the output a. Transfer functions can be linear or non-linear functions. Typical transfer functions are presented in Fig. 2.12.

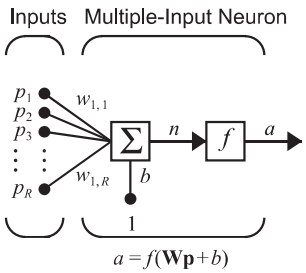


Figure 2.11: Neuron with R inputs [Dem+14].

Name	Input/Output Relation	Icon
Hard Limit	$a = 0 \quad n < 0$ $a = 1 \quad n \geq 0$	
Symmetrical Hard Limit	$a = -1 \quad n < 0$ $a = +1 \quad n \geq 0$	
Linear	$a = n$	
Saturating Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n \leq 1$ $a = 1 \quad n > 1$	
Symmetric Saturating Linear	$a = -1 \quad n < -1$ $a = n \quad -1 \leq n \leq 1$ $a = 1 \quad n > 1$	
Log-Sigmoid	$a = \frac{1}{1 + e^{-n}}$	
Hyperbolic Tangent Sigmoid	$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$	
Positive Linear	$a = 0 \quad n < 0$ $a = n \quad 0 \leq n$	
Softmax	$a = \frac{e^{n}}{\sum_i e^{n}}$	

Figure 2.12: Table with several transfer functions [Dem+14].

Furthermore, neurons often associate in layers to operate in parallel. Each input is connected to each of the neurons.

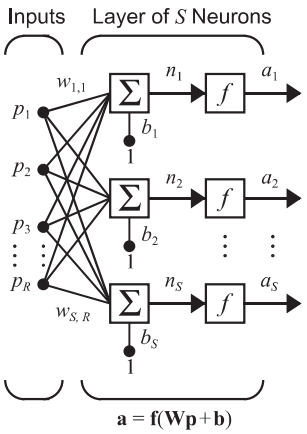


Figure 2.13: Layer of S neurons [Dem+14].

In this case, **W** is a matrix in which the row index indicates the destination neuron associated with

that weight, while the column index indicates the source of the input for that weight:

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix} \tag{2.9}$$

To build a network, several layers L are interconnected, each with their own $\mathbf{W}^L$ matrix and bias vector $\mathbf{b}^L$.

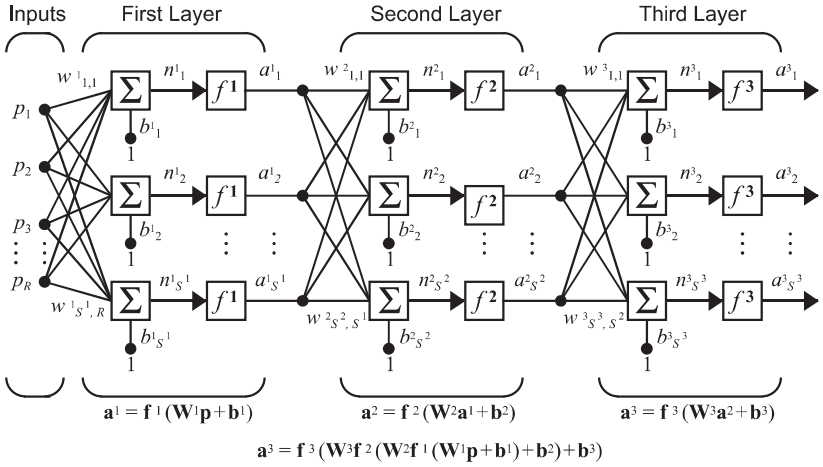


Figure 2.14: Three-layer network [Dem+14].

Training

Training the network means to adjust weights and bias vectors in order to reach the desired output from an input vector. This adjustment is mediated by a learning rule. There are many types of learning rules. In general, in supervised learning, where example input-output pairs are available, the learning rule can be expressed by:

$$\mathbf{W}_{new} = \mathbf{W}_{old} + \alpha \mathbf{e}\mathbf{p}^T \tag{2.10}$$

$$\mathbf{b}_{new} = \mathbf{b}_{old} + \alpha \mathbf{e} \tag{2.11}$$

where $\mathbf{e}$ is the error between target outputs $\mathbf{t}$ and the outputs given by the network. Error performance can be computed in several ways. One of them is the mean square error, which is employed in the Backpropagation algorithm, an approximate steepest descent algorithm. The algorithm should adjust the network parameters in order to minimize the mean square error $F(\mathbf{x})$:

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})] \tag{2.12}$$

The mean square error can be approximated by:

$$\widehat{F} = [\mathbf{t}(k) - \mathbf{a}(k)]^T [\mathbf{t}(k) - \mathbf{a}(k)] = \mathbf{e}^T(k)\mathbf{e(k)} \tag{2.13}$$

where the expectation of the squared error has been replaced by the squared error at iteration k [Dem+14].

The steepest descent algorithm for the approximated mean square error is:

$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha \frac{\partial \widehat{F}}{\partial w_{i,j}^m} \tag{2.14}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha \frac{\partial \widehat{F}}{\partial b_i^m} \tag{2.15}$$

To find the derivatives, the chain rule is employed:

$$\frac{\partial \widehat{F}}{\partial w_{i,j}^m} = \frac{\partial \widehat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^m} \tag{2.16}$$

$$\frac{\partial \widehat{F}}{\partial b_i^m} = \frac{\partial \widehat{F}}{\partial n_i^m} \times \frac{\partial \partial n_i^m}{\partial b_i^m} \tag{2.17}$$

where $n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$. The first derivative can be defined as sensitivity $s_i^m \equiv \frac{\partial \widehat{F}}{\partial n_i^m}$ and:

$$\mathbf{s}^m = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \widehat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1} \tag{2.18}$$

where:

$$\dot{\mathbf{F}}(\mathbf{n}^m) = \begin{bmatrix} \frac{\partial f^m(n_1^m)}{\partial n_1^m} & 0 & \cdots & 0 \\ 0 & \frac{\partial f^m(n_2^m)}{\partial n_2^m} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial f^m(n_S^m)}{\partial n_S^m} \end{bmatrix} \tag{2.19}$$

Finally, the updates of the weights and the biases presented in Eqs. 2.14 and 2.15 can be expressed as:

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m(\mathbf{a}^{m-1})^T \tag{2.20}$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m \tag{2.21}$$

The Backpropagation algorithm derives its name from the sensitivities being propagated backwards through the network from the last layer to the first layer, as can be observed in the last equations.

It is important to remark that this is not the only training algorithm. There are modifications of it, each one with their advantages and disadvantages [Dem+14].

2.3.2 Network design

The neural network design has been made in MATLAB based on its own auto-generated code. That part of the code is explained in Appendix B.

Structure

The problem is outlined as a binary classification problem. The input layer consists of 24 neurons with a hyperbolic tangent activation function, whose inputs are the 12 MFCCs with their corresponding delta coefficients. A vector of length 24 altogether. Training was accomplished also with 26 coefficients (with 13 MFCCs and their deltas), including the zeroth coefficient. However, the former gave better results (Appendix C). There is only a hidden layer, composed of 2 neurons with hyperbolic tangent activation functions too. The addition of more neurons in the hidden layer did not improve the accuracy.

At the end of the network, there are two outputs activated by a softmax function, which represent the two classes: *F. naumanni* (F) and not - *F. naumanni* (NF). These outputs will be consistent estimates of the probabilities belonging to the classes.

The aforementioned activation functions were presented in Fig. 2.12.

To compare the two types of Mel filter bank shapes (see Figs. 2.6 and 2.7), the training has been repeated with and without area normalization, showing the former a better performance (Appendix C).

Finally, the structure of the network can be visualized in Fig. 2.15.

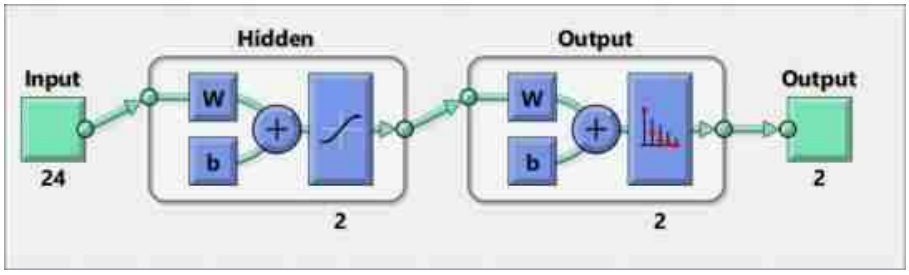


Figure 2.15: Neural network employed.

Training

The training data were divided into three sets: training, validation and testing, in a proportion 65-15-20. The validation data are used to avoid overfitting.

No preprocessing was applied. With the same inputs, normalization and no normalization of these inputs showed that the latter exhibits better performance (see Appendix C).

In relation to training algorithms, it is very difficult to know which one will be the fastest for a given problem. It depends on many factors (the complexity of the problem, the number of data points in the training set, the number of weights and biases in the network, the error goal, the use of the network...). In a previous work [DBD], the authors compared various training algorithms and they came to the conclusion that Resilient Backpropagation and Scaled Conjugate Gradient Backpropagation were the best for pattern recognition.

Multilayer networks typically use sigmoid transfer functions in the hidden layers. These function are characterized by the fact that their slopes must approach zero as the input becomes large. This causes a problem because it can cause small changes in the weights and biases, even though they are far from their optimal values. Resilient Backpropagation eliminates these effects because only the sign of the derivative can determine the direction of the weight update. The size of the weight change is determined

by a separate update value, whose magnitude increases if the weight continues to change in the same direction for several iterations. The Resilient Backpropagation training algorithm is the fastest on pattern recognition problems. Its performance also degrades as the error goal is reduced [DBD].

Scaled Conjugate Gradient Backpropagation seems to perform well over a wide variety of problems, specifically for networks with a large number of weights. It is almost as fast as Resilient Backpropagation on pattern recognition problems. Its performance does not degrade as quickly as Resilient Backpropagation performance does when the error is reduced. This algorithm calculates the derivatives as backpropagation does, but also computes the maximum feasible step α in that direction, so it converges with less iterations [Roo+19].

Since learning time is not a restriction and the configured network has only one hidden layer, Scaled Conjugate Gradient Backpropagation was chosen as training algorithm.

To calculate the network performance, the cross-entropy loss function was employed:

$$CE = -t_F \log(y_F) - (1 - t_F) \log(1 - y_F) \tag{2.22}$$

where t_F and y_F are the groundtruth and the score for class F, respectively. It returns a result that heavily penalizes outputs that are extremely inaccurate, with very little penalty for fairly correct classifications[2].

The training occurs according to default parameters of the Scaled Conjugate Gradient Backpropagation implementation. Training will stop when:

- The maximum number of 'epochs' to train (1000) is reached.
- The performance is minimized to the 'goal' (0).
- The performance gradient falls below 'minimum performance gradient' (10^{-6}).
- The validation performance has consecutively increased after 'maximum validation failures' (6) to avoid overfitting training data.

Validation

The training was accomplished until epoch 115, when the validation performance started to decreased while training performance continued increasing.

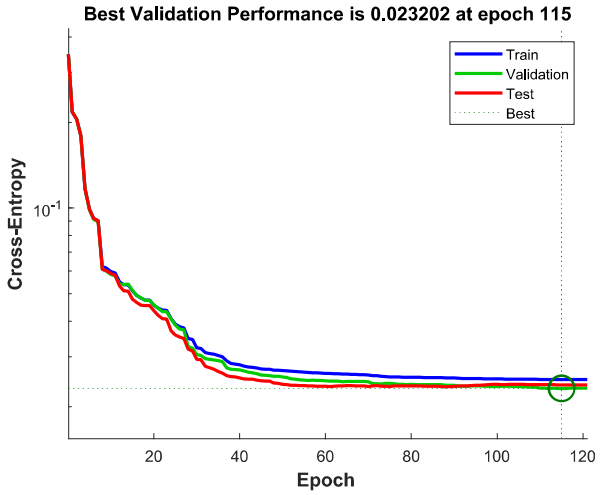


Figure 2.16: Training Learning Curve monitoring the network performance while training.

[2]When the groundtruth is 1, the first term is activated, whereas when it is 0, the activated term is the second one. Depending on the accuracy of the output of the network, CE will be almost zero or not.

The results in terms of the network's classification are depicted in Fig. 2.18, which shows the confusion matrix. In the confusion matrix, recall, precision, specificity, negative predictive value and accuracy can be observed (visually defined in Fig. 2.17). *F. naumanni* (class F or 1) was classified with a mean accuracy of 98.3%.

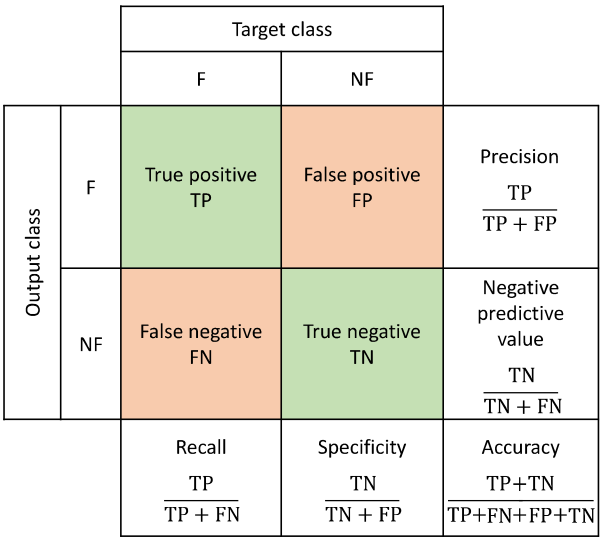


Figure 2.17: Confusion matrix with the meanings of the boxes.

In Fig. 2.19, the receiver operating characteristic (ROC) curve is included, that illustrates the ability of the binary classifier as its discrimination threshold is varied. The choice of which threshold to use will be discussed in the next subsection.
The Area Under the Curve (AUC) is a tool to evaluate the performance of a classifier. The larger the AUC, the better the model can distinguish between classes. For a perfect classifier, AUC is 1, while for a classifier that randomly allocates observations to classes, AUC is 0.5. In this case, the value given by MATLAB is 0.9995, close to 1.

2.3.3 Call score

At this point, from 32 ms of recording, a classification result between F and NF is obtained. However, only one positive would not be enough to quantify calls. It should be followed by more positives because a call lasts more.

In Figs. 2.20 and 2.21, some results are shown. It can be observed a continuous highly probability at the output of the neural network when a call of *F. naumanni* occurs.

According to this, a temporal study should be carried out. A typical trisyllabic call lasts about half a second (see Fig. 1.3). Thus, when a high probability of F class is detected, the probability of F is summed across adjacent frames. That value is compared with a threshold to distinguish an entire call.

In short, probabilities of 16 windows are added and compared with a threshold th. This process has been tested for different values of that threshold:

- $th = 16 \cdot 0.6$
- $th = 16 \cdot 0.7$
- $th = 16 \cdot 0.75$
- $th = 16 \cdot 0.8$
- $th = 16 \cdot 0.9$

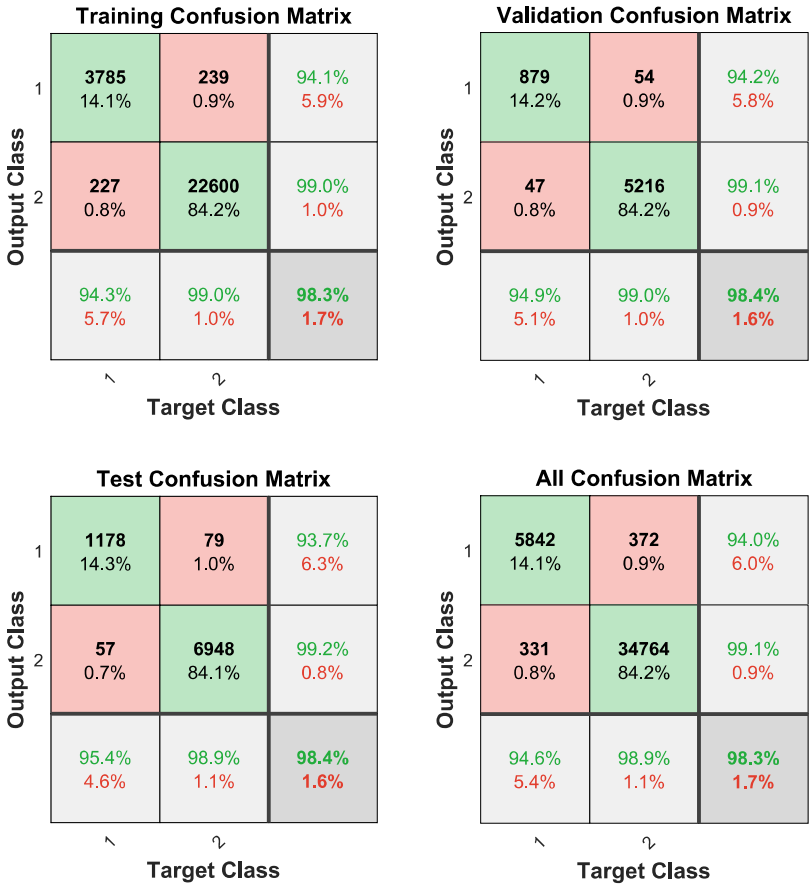


Figure 2.18: Confusion matrices for each dataset: training, validation, and test.

If the addition exceeds the threshold, a call would have been identified.

Some recordings, different from those chosen for training, have been selected to follow this approach. These recordings belong to all species used in training. Their duration can be examined in Table 2.2, as well as their Xeno-canto reference in Appendix A. This validation step is shown in Figs. 2.22 and 2.23.

Once the count has been done, the results, that is, false and true positives and negatives, are shown in Table 2.3.

With these data, precision, negative predictive value (NPV), recall, specificity and accuracy can be calculated and presented in Table 2.4.

According to Table 2.4, specificity and NPV are pretty good. So, in order to choose a threshold, it is necessary to take attention to recall and precision. When the threshold is risen, precision does too, making positives more reliable. It is important to note that many FNs are originated in vocalizations below -20 dB (a limit that was established when selecting datasets, see Sec. 2.1.2). On the contrary, recall decreases, so missing positives increase. Nevertheless, the majority of FPs come from three specific recordings (XC308077, XC282760 and XC299528, see Appendix A). They coincide with sounds of *F. naumanni* not included in training or with recordings from other birds that emit different vocalizations apart from those employed in training. So, the situation could improve if more types of birdsounds (of the same species) and more species are included in the training.

A balanced situation can be 69.25% in precision and 65.75% in recall, so the selected threshold would be $16 \cdot 0.4$. In any case, most of FPs come from *F. naumanni* and FNs are below -20 dB, as said before.

Species	Total duration	Background	Bird vocalizations
Falco naumanni	21:46	15:48	05:58
Streptopelia decaocto	00:48	00:14	00:34
Sturnus unicolor	01:13	00:43	00:30
Falco tinnunculus	00:57	00:16	00:41
Coloeus monedula	05:24	03:48	01:36
Apus Apus	00:33	00:08	00:25
Emberiza calandra	02:39	01:38	01:01
Upupa epops	01:46	01:25	00:21
Columba livia	00:20	00:10	00:10
Passer Domesticus	00:19	00:01	00:18

Table 2.2: Duration (mm:ss) of the recordings used in validation, breaking down bird vocalizations and background.

	TP	FN	FP	TN
$th = 16 \cdot 0.3$	359	76	215	26551
$th = 16 \cdot 0.4$	286	149	127	26639
$th = 16 \cdot 0.5$	215	220	84	26682
$th = 16 \cdot 0.6$	146	289	44	26722
$th = 16 \cdot 0.7$	99	336	20	26746

Table 2.3: True positives, false negatives, false positives and true negatives of the validation of syllable detection.

	Precision	NPV	Recall	Specificity	Accuracy
$th = 16 \cdot 0.3$	0.62543	0.99715	0.82528	0.99197	0.98930
$th = 16 \cdot 0.4$	0.69249	0.99444	0.65747	0.99526	0.98985
$th = 16 \cdot 0.5$	0.71906	0.99182	0.49425	0.99686	0.98882
$th = 16 \cdot 0.6$	0.76842	0.98930	0.33563	0.99836	0.98776
$th = 16 \cdot 0.7$	0.83193	0.98759	0.22759	0.99925	0.98691

Table 2.4: Precision, negative predictive value, recall, specificity and accuracy of the validation of syllable detection.

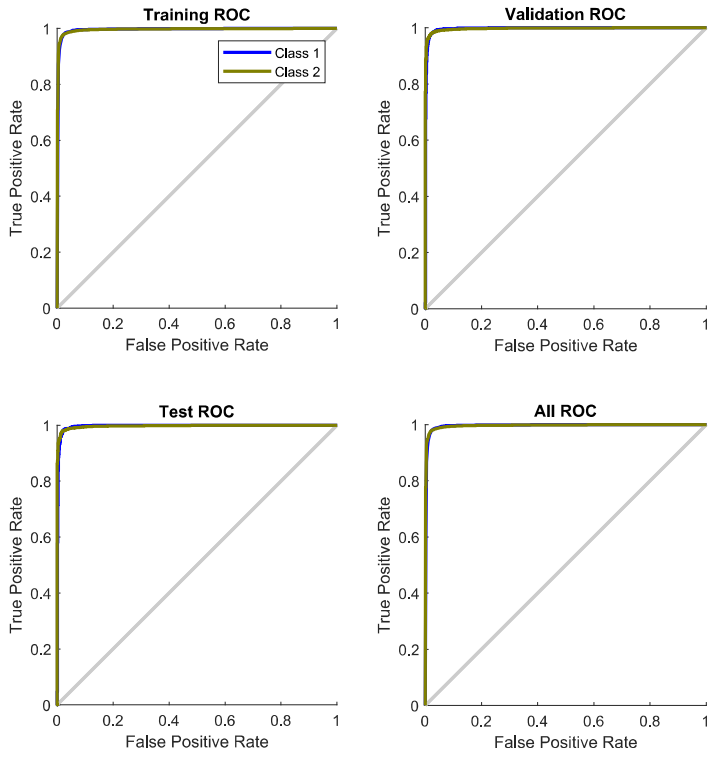


Figure 2.19: ROC curves for test, validation and training.

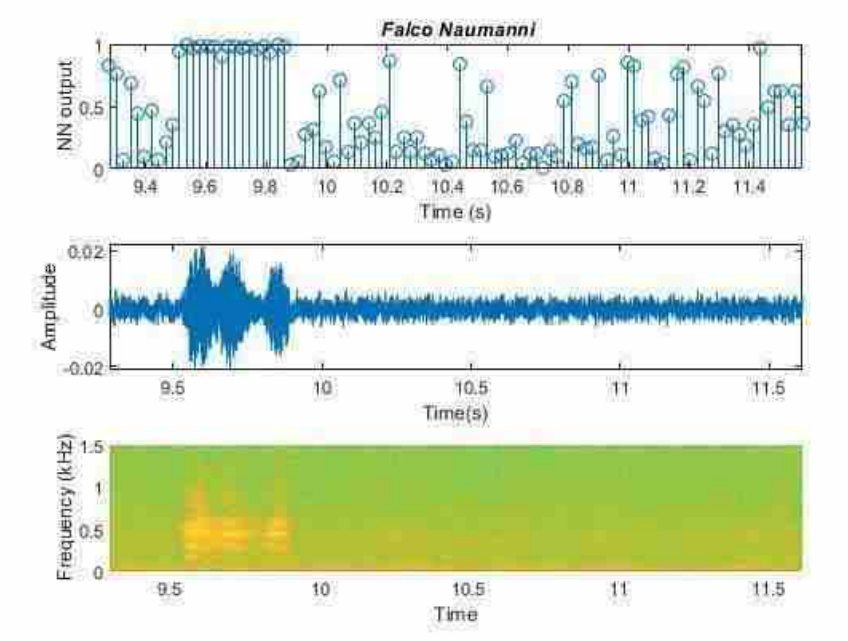


Figure 2.20: Output of the neural network, amplitude and spectrogram of a piece of recording of *F. naumanni*.

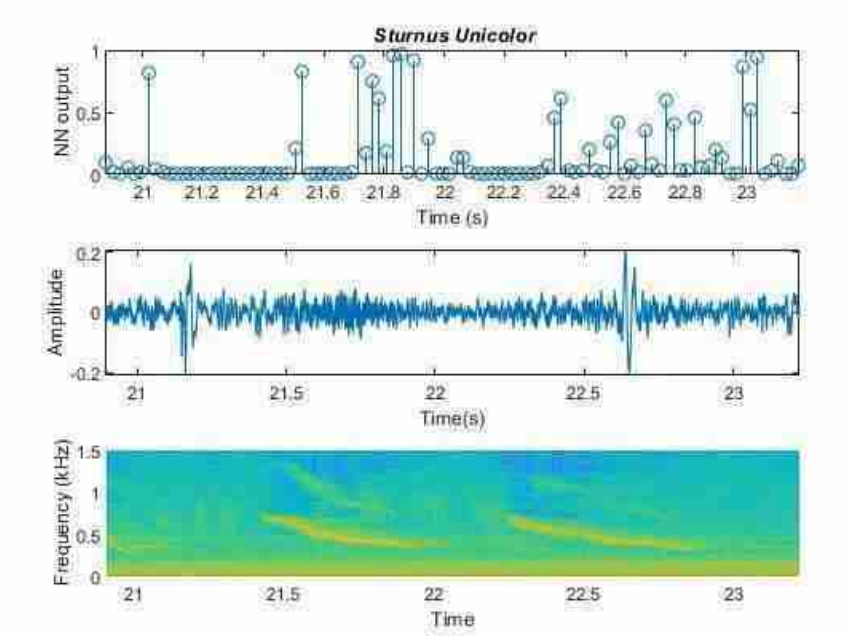


Figure 2.21: Output of the neural network, amplitude and spectrogram of a piece of recording of *Sturnus unicolor*.

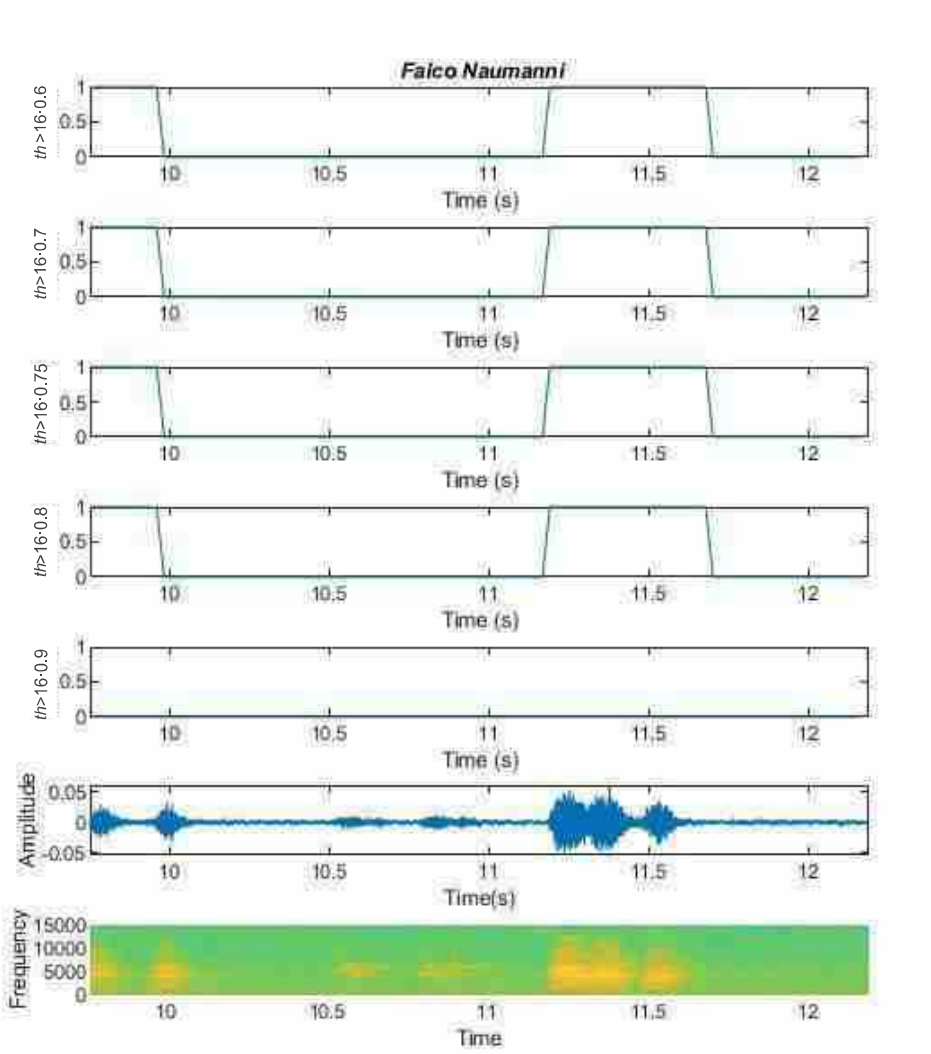


Figure 2.22: Search for *F. naumanni* calls. The penultimate and last plots are the amplitude and spectrogram of a *F. naumanni* recording. The first five plots point out the calls that would be detected using each proposed threshold. These five plots are the representation of vectors whose value is one in groups of 16 points if their added probabilities of being FN class exceed the thresholds. These detections were manually written down as TP, TN, FP and FN to perform the validation.

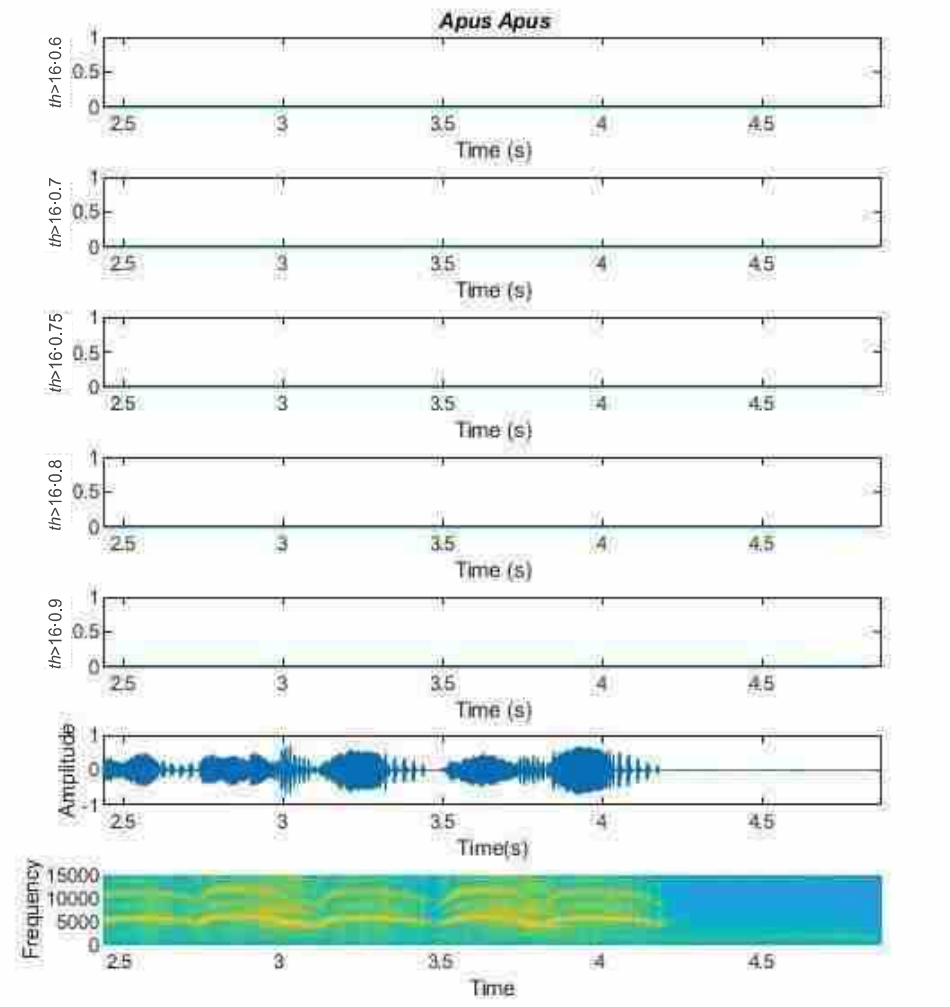


Figure 2.23: Search for *F. naumanni* calls in a recording of *Apus apus*. The penultimate and last plots are the amplitude and spectrogram of a *Apus apus* recording. No call is detected in this shot.

Chapter 3

System embedding

As previously mentioned, the designed processing was implemented on a low-cost acoustic detector, called AudioMoth. This facilitates acoustic detection and deployment in remote locations. At the same time, its low-power operation allows for long-term monitoring. In this chapter, AudioMoth is described and the implementation is presented.

3.1 Hardware description: AudioMoth

This credit-card sized device mainly consists of a micro-controller, a micro-electro-mechanical system (MEMS) microphone, and a printed circuit board (PCB). The latter includes a sidemounted switch, an USB port, red and green LEDs, and a microSD card port. Components are placed between the board and the battery holder[1] on the top layer of the two-layer PCB (see Fig. 3.1).

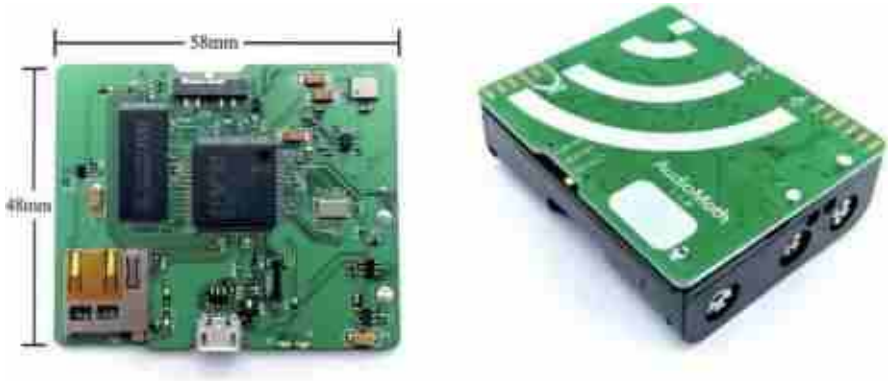


Figure 3.1: View of the PCB components and the mounted device [Hil+19].

AudioMoth is built around an ARM Cortex M4 micro-controller with a floating-point unit (FPU). The micro-controller links to an external static random access memory (SRAM) chip using an external bus interface (EBI). Direct memory access (DMA) allows the device to sleep in low energy modes while data are sampled and routed between the analog interface and the external SRAM chip. The latter consists of 2Mbit SRAM organized as 256K words by 8 bits.

Sound passes through a drill hole and it is captured by the MEMS microphone. It can be configured to record at many sample rates, up to 384 kHz. The audio circuitry begins at the microphone input, which is routed to the micro-controller analog peripherals. The microphone signal is amplified by op-amp circuitry,

[1] It is designed for 3 AA batteries.

whose gain can be controlled by software. Then, it is converted to digital samples by a 12-bit ADC and sent to the external SRAM using DMA, as commented before.

Data can be saved to the microSD card via the serial peripheral interface (SPI) bus. Only secure digital high-capacity (SDHC) microSD cards[2], formatted to FAT32, are compatible with AudioMoth.

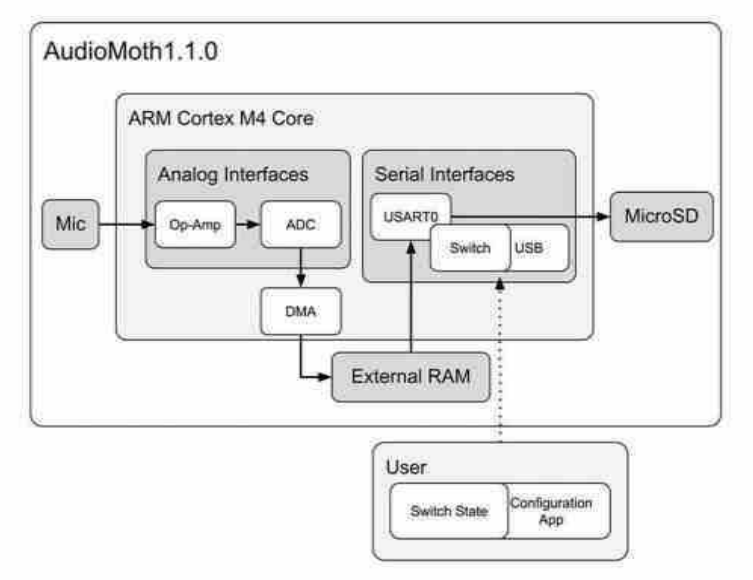


Figure 3.2: Hardware overview and typical operation flow [Hil+19].

The device can be powered from any 3.6 V – 20 V DC supply. The power supply connects to a regulator that converts the DC supply to a stable 3.3 V, or alternately the device can be supplied by 5 V USB power. The power circuitry also has digital noise isolation that uses a series resistor and ferrite bead to isolate the analog audio circuitry from the digital transmission lines.

3.2 Firmware description

The processing developed in Chapter 2 is needed to be implemented in AudioMoth. With this aim, the C project example created by Open Acoustic Devices has been modified to include appropriate functionality.

3.2.1 Basic firmware details

The basic firmware can be download from `https://github.com/OpenAcousticDevices/AudioMoth-Firmware-Basic`. The modification of only a part of the code allows for changing the configuration from Configuration App, to set recording periods, sample rate, gain, sleep duration or configure date (Fig. 3.3).

In this code, the SRAM chip is divided into an eight-element circular buffer. This buffer structure allows acoustic data to be filled in contiguous elements. When a whole element has been filled with a sufficient number of acoustic samples, the micro-controller wakes up to store those samples on the microSD card. Simultaneously, the next element is being filled, thus it allows a continuous stream of audio to be recorded to storage.

Once initialized, AudioMoth is waiting until required conditions (high power battery and time of next recording) are met (Fig. 3.4). In that case, the function that controls the recording, *makeRecording*(), is activated. Peripherals are initialized and recording parameters are calculated (Fig. 3.5). A callback

[2]For sample rates up to 48 kHz, a microSD card of at least speed class of 10 must be used. In addition, microSD cards must have a memory up to 32 GB.

Figure 3.3: Configuration example in AudioMoth Configuration App. Two recording periods are selected, as well as sample rate, gain, recording and sleep duration. At the end, the app informs about the number of files produced, their size and daily energy consumption.

function is defined to be invoked when the DMA transfer finishes. The callback function triggers an interruption and reactivates the DMA (Fig. 3.6). During that interruption, the eight-element buffer is being filtered and written (Figs. 3.7 and 3.8). Then, a Waveform audio file (WAV) is opened, and if an element of the buffer is filled, it will be saved to the microSD card (Fig. 3.5).

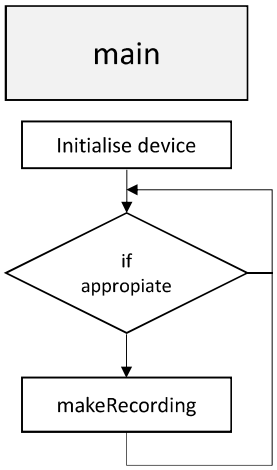


Figure 3.4: Flowchart of *main*().

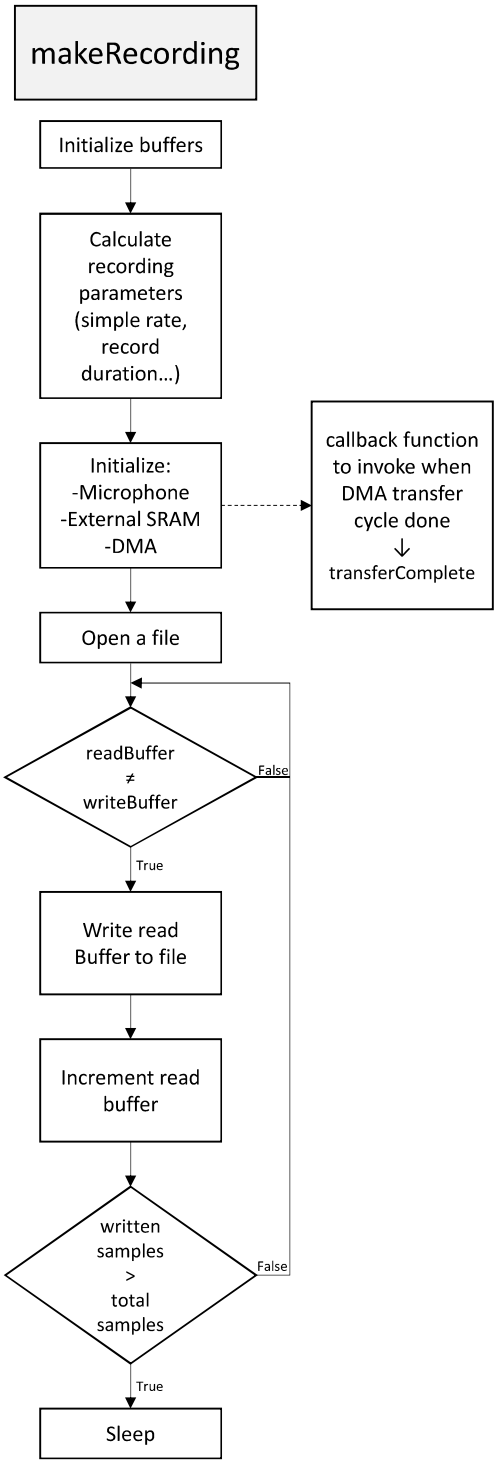


Figure 3.5: Flowchart of *makeRecording*(). This function initialises the recording and saves the content of the full elements of the circular buffer to the microSD card.

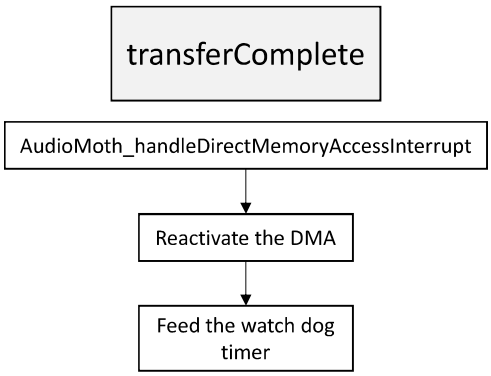


Figure 3.6: Flowchart of $transferComplete()$.

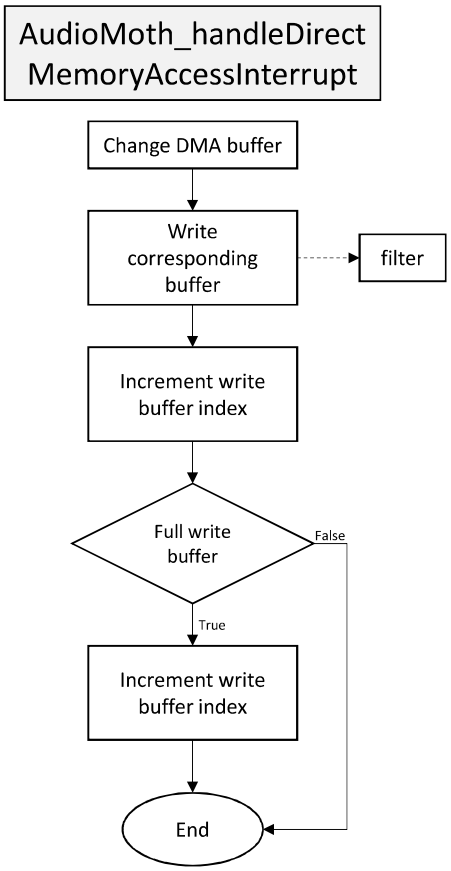


Figure 3.7: Flowchart of *AudioMoth_handleDirectMemoryAccessInterrupt*(). The content of DMA buffer is passed to the circular buffer. Actually, this is done by a function inside this function, $filter()$.

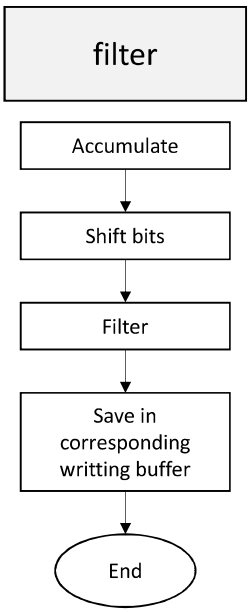


Figure 3.8: Flowchart of $filter()$. The signal is accumulated to avoid high frequency noise and filtered by a DC blocker implementation[3].

3.2.2 Firmware modification

In order to add the birdsong identification system, the function *makeRecording*() has been edited.

The eight element buffer has been transformed into a 128 element buffer to fill the SRAM chip of elements of 1024 samples (Fig. 3.9). Thus, the MFCC algorithm can be applied to every buffer. Nevertheless, when writing to the microSD, the number of samples to save in memory has been maintained, that is, the microcontroller waits until 16 elements (a subbuffer, so to say) have been analysed by the neural network to write them in memory. In this way, time spent on writing in memory is reduced. In addition, this subbuffer is useful to perform the procedure explained in Subsec. 2.3.3. With a specific sampling frequency (32 kHz), the subbuffer will contain half second, the time necessary to accumulate the probabilities, and it will mark the time at which a call happens.

Acoustic data are stored in $int32\_t$ format. In order to process these data with the design system, it is necessary to normalize it and convert it to $float32\_t$. To carry the MFCC method out, a list of constants has been defined:

- Coefficients of the Hamming window.
- Constants of the DCT: $\frac{2}{sqrtN}$, $\frac{\pi}{2N}$.
- Filter banks in a vector and a matrix. The vector only contains the non-zero values of all the filters following each other. The 41x2 matrix includes in every row the index of the vector corresponding to the beginning of the specific filter and how many positions takes up (Fig. 3.10).

To calculate deltas, a small buffer of 5 elements is dynamically allocated in memory to accumulate MFCCs, so past and futures ones are available (Fig. 3.11). Since future information is required, the element of the buffer that is being written to the circular buffer is not the one that is being checked by the network.

To take advantage of the hardware resources, the FFT of the MFCC extraction procedure and the matrix operation of the neural network have been developed with the aid of Cortex Microcontroller Software

[3]Recursive filter specified by $y(n) = x(n) - x(n-1) + R \cdot y(n-1)$.

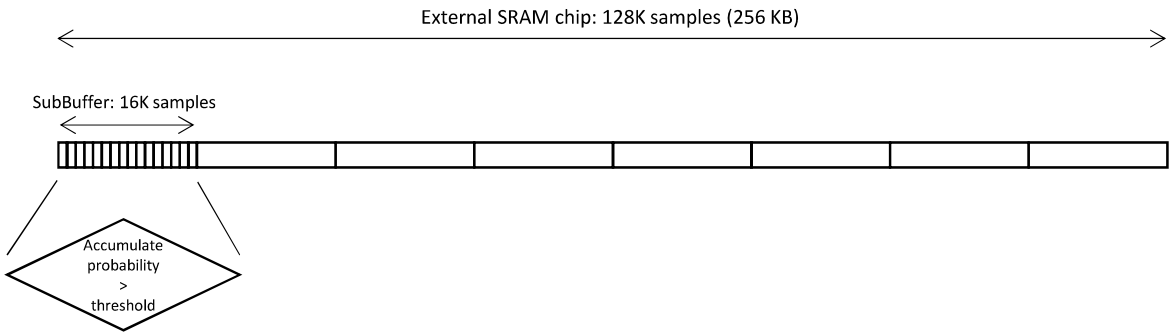
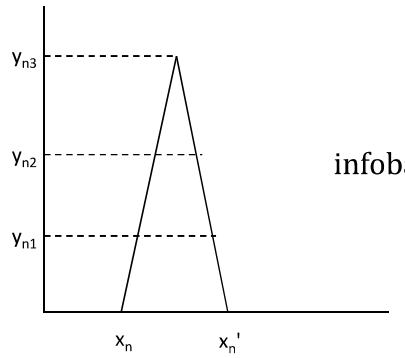


Figure 3.9: New division of the circular buffer. The content of the subbufer is accumulated to compare with the threshold *th*. In case of surpassing it, a call is reported.

$$\text{hbanks} = \{y_{1\,1}, y_{1\,2}, y_{1\,3}, y_{1\,2}, y_{1\,1}, \cdots, y_{41\,1}, \cdots, y_{41\,k}, \cdots, y_{41\,1}\}$$



$$\text{infobanks} = \begin{Bmatrix} x_1 & x_1' - x_1 \\ \vdots & \vdots \\ x_{41} & x_{41}' - x_{41} \end{Bmatrix}$$

Figure 3.10: Organization of the vector and matrix that describe the filter banks. The vector *hbanks* contains the non-zero values of the filters. Each row of the matrix *infobanks* indicates at what position the non-zero values start and for how long.

Interface Standard (CMSIS) library, which uses FPU and digital signal processing (DSP) resources. DCT has not been programmed this way because the required length is a power of two, and padding the signal could slightly change the results of the network.

The probability given by the output of the neural network is summed, and when the whole subbuffer has been analysed, the result is checked. If it exceeds the threshold, a text file is opened and the time and date are written on it after accessing to the backup real time clock (BURTC). Furthermore, the green LED is activated until the verification of next subbuffer.

Therefore, what we find in the microSD card is the configured recording and a text file with annotations of what time the calls of lesser kestrel were detected.

3.2.3 Further modifications

Some of the parameters that have been used in the system were edited after programming AudioMoth to implement the system in a simpler way. They are itemized here:

- Dividing the signal in steps of 1024 samples. Previously, the signal was divided into fragments of 20 ms duration. This was changed to match with binary system and be able to use floating point

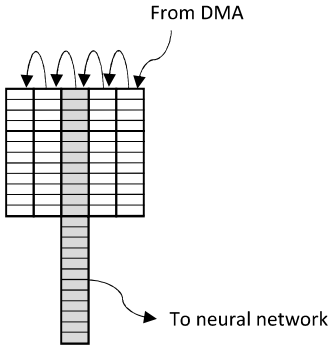


Figure 3.11: Buffer of 5 elements intended to contain the five groups of MFCCs (up) and their corresponding deltas (down).

operations.

- Setting the highest frequency in filter banks at 15 kHz. Datasets were MPEG-1 Audio Layer III (MP3) recordings (whereas AudioMoth generates WAV files), so the compression of these files precludes results from being exactly the same. In this way, higher frequency information does not disrupt the output of the network.
- Setting sampling frequency to 32 kHz. In this way, the signal accumulated in a subbuffer is equivalent to a call length, making easier to compare the threshold and then save the fragment to memory. In order to carry it out, it was necessary to resample the dataset to 32 kHz.

These changes forced the network to be retrained and validated.

3.2.4 Recording with AudioMoth

After flashing AudioMoth[4], several recordings were played on the computer, in order to mimic birdsongs. AudioMoth was placed next to the computer, as Fig. 3.12 shows[5]. It was checked that the green LED was activated when a *F. naumanni* emitted a call, as opposed to background noise or other birds singing.

Furthermore, the output of the neural network in AudioMoth was stored in the microSD card (apart from the recording) to be compared with the output of the neural network in MATLAB when the AudioMoth recording was selected as input of the system. That comparison[6] is shown in Fig. 3.13 and 3.14. This is a verification step to check that the processing done by AudioMoth is the same as MATLAB processing. In a recording of 21 seconds, the root mean square error (RMSE) between the two neural network outputs was 0.0924.

As commented in Subsec. 3.2.2, AudioMoth writes the date and time of occurrences to a text file. This can be visualised in Fig. 3.15.

Finally, an aspect that should be taken in account for a future deployment is energy consumption. The durability of the battery is principally determined by the recording periods. A continuous monitoring would demand approximately 300 mAh daily, according with AudioMoth Configuration App (Fig. 3.16). Nevertheless, shorter periods would reduce the required charge (see Fig. 3.3). This would be adjusted based on the activity of the colonies over the day.

[4]Configuration parameters are stored in FLASH initially and copied to the back-up RAM when the AudioMoth detects that it has experienced a power-up reset. So, when editing the default settings, it is needed to switch off and on.

[5]A video with the demonstration can be visualized in `https://drive.google.com/file/d/1VnWrPhLRWbhcV6yBqf85AcC2S3UAd32s/view?usp=sharing`

[6]Take into account that AudioMoth data are a rough approximation since the data had to be taken as *int* format because not all capabilities from the *printf()*-style functions were available and decimals could not be written to the text file.

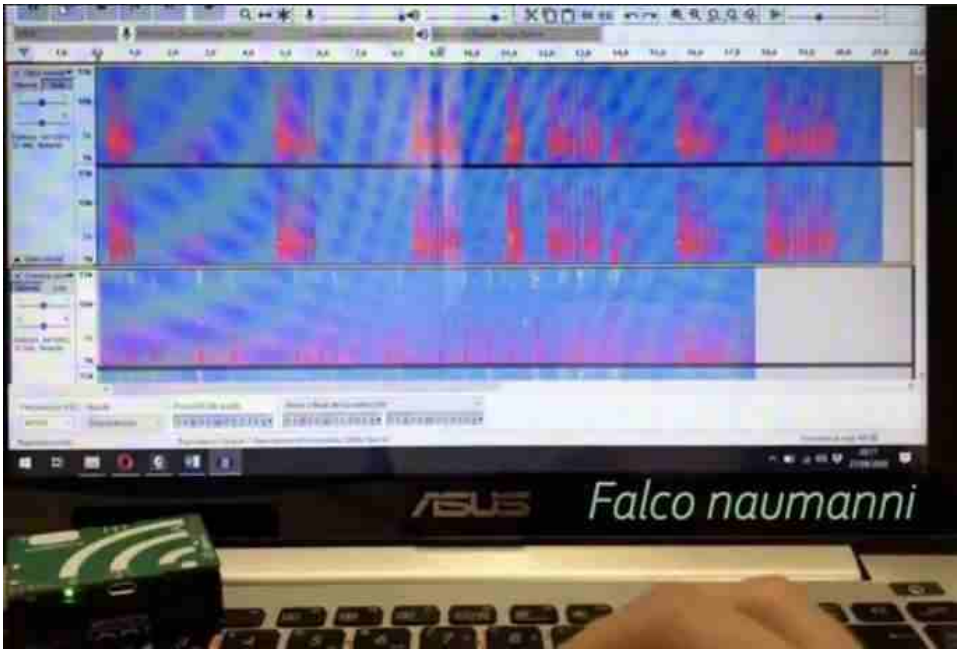


Figure 3.12: System operating test. AudioMoth is recording and it has detected a vocalization of *F. naumanni*. Consequently, the green LED (bottom left corner) has been activated.

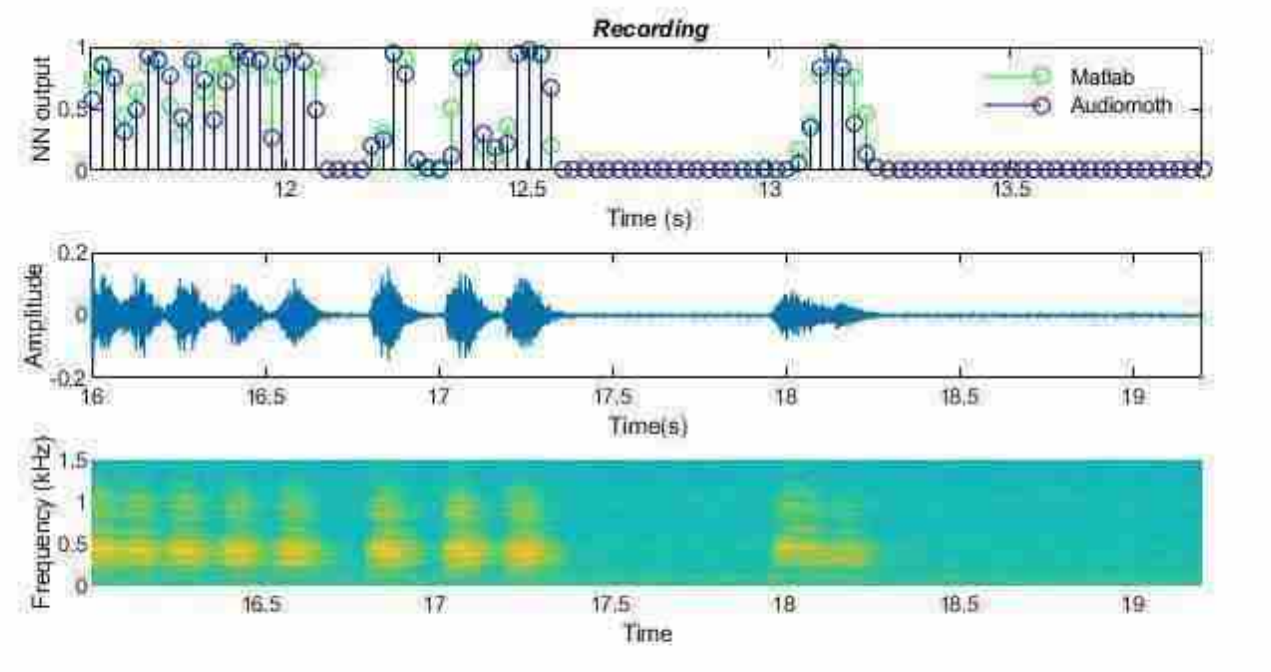


Figure 3.13: Output of neural networks when a *F. naumanni* is singing. In the first plot, the output of the neural network from AudioMoth processing (blue) is presented together with the output of the neural network from MATLAB whose system input is the recording from AudioMoth (green). Second and third plots are amplitude and spectrogram of the recording.

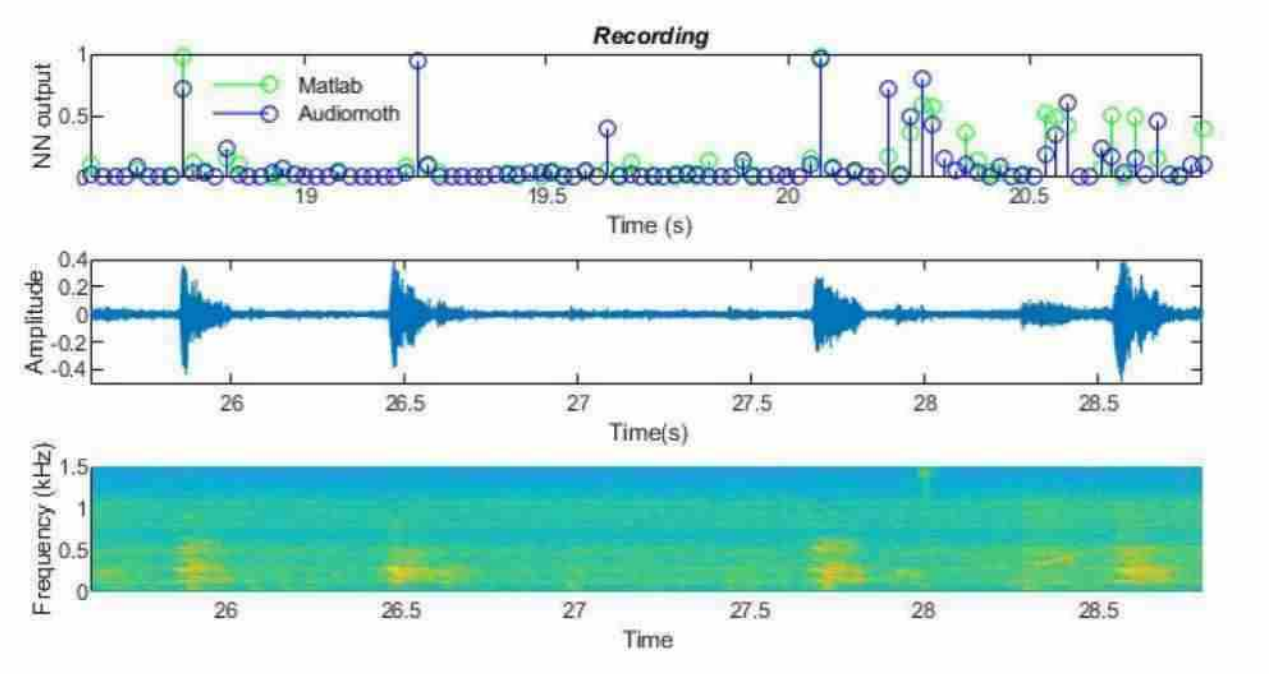


Figure 3.14: Outputs of neural networks and amplitude and spectrogram of a recording of a *Coloeus monedula* singing.

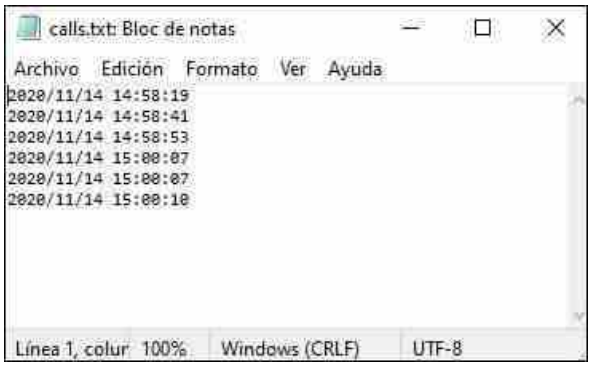


Figure 3.15: A text file with annotations of what time and date some calls of lesser kestrel were registered.

Figure 3.16: Configuration of a recording period of 24 hours. The energy consumption is shown above 'Configure AudioMoth' button.

Chapter 4

Conclusions

In this work, a method to distinguish vocalizations of *F. naumanni* in real time was presented, tested and implemented, with the aim to carry out field population studies.

After a bibliographical research, it was decided to employ Mel-scale Frequency Cepstral Coefficients (MFCCs) to do feature extraction of signal windows and a neural network to classify them in *F. naumanni* segments or segments resulting from background noise or other species. The probability of belonging to the *F. naumanni* class is summed for the time that a call lasts, and compared to a threshold. Different parameters have been tested to set those that achieve better results. Triangular filters with area normalization and excluding zeroth coefficient turned out to cause less error after training. In addition, incrementing the size of the neural network does not involve a better performance. On the other hand, the threshold has been chosen comparing the differences in accuracy in each case. This has been validated through a *35 min 45 seg* dataset, built from Xeno-Canto recordings of 10 cohabiting birds.

The performance of the method achieves an accuracy of 98,99%, slightly above or similar to systems reported in the literature.

In terms of hardware implementation, specifically in AudioMoth, the firmware developed by the company has been modified to include the recognition system, adding the aforementioned capability.

Finally, it is worth noting that, because of the amount of species and the diverse range of vocalizations, a larger dataset is needed to obtain more definitive results, collecting data from the locations where AudioMoth will be deployed.

4.1 Future work

In spite of changes done in the system after programming AudioMoth, there are another issues that would improve the implementation. Some proposals are working with integer values during all the processing instead of normalized float values (between -1 and 1), or padding the signal before applying DCT to be able to use CMSIS library.

As commented before, more species and different vocalization should be included in the dataset, specially those whose bandwidth or other features are similar. A first step to carry out at the field could be continuously recording for recognizing all species that cohabit there and use these data to train the network. Moreover, it is often more successful if the training data are based on the same type of recorder as will be used in practice. The reason for this is partly because the dataset could contain recordings of directional microphones (used for manual recording), whereas automated recorders use omni-directional microphones.

Related to the MFCC technique, it would be interesting to extend the duration of the window to make it of the same length of a *F. naumanni* call. This way, the procedure of call score would be direct.

Regarding the high probability of windy or rainy weather, it is important to study how these phenomena affect the recognition and suggest a noise reduction algorithm, for example, by wavelet transform.

Finally, AudioMoth must be installed near a local colony of *F. naumanni* to try to relate the number of occurrences to the number of breeding pairs or their reproductive success.

Chapter 5

Bibliography

[CL11] Bruce Campbell and Elizabeth Lack. *A dictionary of birds*. Volume 108. A&C Black, 2011 (cited on page 19).

[d'A+05] Christophe d'Alessandro et al. "The speech conductor: gestural control of speech synthesis". In: *eINTERFACE'05-Summer Workshop on Multimodal Interfaces*. 2005 (cited on page 25).

[DBD] Howard Demuth, Mark Beale, and Howard B. Demuth. *Deep Learning Toolbox: For Use with MATLAB*. URL: `https://es.mathworks.com/help/pdf_doc/deeplearning/nnet_ug.pdf` (visited on 05/23/2020) (cited on pages 34, 35).

[Dem+14] Howard B Demuth et al. *Neural network design*. Martin Hagan, 2014 (cited on pages 30–34).

[DM80] Steven Davis and Paul Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences". In: *IEEE transactions on acoustics, speech, and signal processing* 28.4 (1980), pages 357–366 (cited on page 25).

[Gra+10] Martin Graciarena et al. "Acoustic front-end optimization for bird species recognition". In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2010, pages 293–296 (cited on page 28).

[Hil+19] Andrew P Hill et al. "AudioMoth: A low-cost acoustic device for monitoring biodiversity and the environment". In: *HardwareX* 6 (2019), e00073 (cited on pages 43, 44).

[IB10] A Iñigo and B Barov. "Action Plan for the lesser kestrel Falco naumanni in the European Union". In: *SEO| BirdLife and BirdLife International for the European Commission* 55 (2010) (cited on page 18).

[PMC18] Nirosha Priyadarshani, Stephen Marsland, and Isabel Castro. "Automated birdsong recognition in complex acoustic environments: a review". In: *Journal of Avian Biology* 49.5 (2018), jav–01447 (cited on pages 19–21, 23, 24).

[PMH01] Roger Tory Peterson, Guy Mountfort, and Philip Arthur Dominic Hollom. *A field guide to the birds of Britain and Europe*. Volume 8. Houghton Mifflin Harcourt, 2001 (cited on pages 18, 19).

[Pot+14] Ilyas Potamitis et al. "Automatic bird sound detection in long real-field recordings: Applications and tools". In: *Applied Acoustics* 80 (2014), pages 1–9 (cited on pages 18, 19).

[Pri+20] Nirosha Priyadarshani et al. "Wavelet filters for automated recognition of birdsong in long-time field recordings". In: *Methods in Ecology and Evolution* 11.3 (2020), pages 403–417 (cited on pages 20, 21).

[RG10] Tobias Riede and Franz Goller. “Peripheral mechanisms for vocal production in birds–differences and similarities to human speech and singing”. In: *Brain and language* 115.1 (2010), pages 69–80 (cited on page 18).

[Roo+19] Matías Roodschild et al. “Optimización de Scaled Conjugate Gradient para Froog Neural Networks”. In: *XX Simposio Argentino de Inteligencia Artificial (ASAI 2019)-JAIIO 48 (Salta)*. 2019 (cited on page 35).

[RP20] Colin Richardson and Richard Porter. *Birds of Cyprus*. Bloomsbury Publishing, 2020 (cited on page 19).

[RV14] K Sreenivasa Rao and Anil Kumar Vuppala. *Speech processing in mobile environments*. Springer, 2014 (cited on page 25).

[SP14] Dan Stowell and Mark D Plumbley. “Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning”. In: *PeerJ* 2 (2014), e488 (cited on page 21).

[TS04] R. Triay and M. Siverio. *Cernícalo primilla, Falco Naumanni*. Dirección General para la Biodiversidad-SEO/BirdLife, 2004 (cited on page 18).

Appendices

Appendix A

Xeno-canto references

In this appendix, the identification code[1] of all recordings are shown, indicating when they were used during the realization of this thesis.

[1] They can be searched in `https://www.xeno-canto.org`.

Species	Training, validation & test (NN)	Syllable detection performance
Falco naumanni	XC141117 XC149085 XC237758 XC240179 XC308080 XC308081 XC308083 XC308084	XC308077 XC308089
Streptopelia decaocto	XC337620 XC270454 XC165045	XC308063 XC269719
Sturnus unicolor	XC286796 XC431986	XC179785 XC192088 XC343499
Falco tinnunculus	XC288407 XC353863 XC290332 XC290331 XC295532	XC282760 XC264073 XC360599 XC463430
Coloeus monedula	XC269706 XC270245 XC286393	XC289360
Apus Apus	XC466668 XC466677	XC418869 XC466671
Emberiza calandra	XC179875 XC269396 XC278143 XC301725 XC361586	XC263307 XC299528
Upupa epops	XC263714 XC278153 XC279736 XC291953 XC291955 XC316704 XC279735	XC278154 XC202569 XC181659
Columba livia	https://www.youtube.com/watch?v=nc0oh8ktcuY	XC246586
Passer Domesticus	XC178488 XC271733 XC293932 XC381732 XC295541	XC381732

Appendix B

Code

In this appendix, MATLAB and AudioMoth codes are presented. They are not shown here owing to their extension, but they are available at `https://github.com/ponseta/AutomatedBirdsongRecognition`. They are organised in two directories: matlab and AudioMoth.

B.1 MATLAB code

The files are organized in the following way:

- Data preparation
  - *create_inputs_NN.m*: This script opens every MP3 file and calls *data_extract.m* to organize each species in a cell with the required information.
    - *data_extract.m*: This script saves the sampling frequency and the MFCC coefficients in the corresponding cell. It uses the function described in $mfccs.m$ which calculates the MFCCs with the filters calculated in $melfilterbanks.m$.
- Training
  - *trainNNmat.m*: This script is a modification of the auto-generated code that MATLAB creates when using the Deep Learning Toolbox. It loads the cell created by *create_inputs_NN.m* and organizes the MFCCs as inputs in a random way (always with the same proportion of 0s and 1s in training, validation and test). Then, the script fixes the parameters of the neural network and performs its training. Subsequently, it saves the function with the structure of the network in $finalNeuralNetworkFunction15khz.m$
- Performance
  - *main.m*: It loads MP3 files, applies $mfccs.m$, then $finalNeuralNetworkFunction15khz.m$ and accumulates to generate figures in order to manually count FPs, FNs, TPs and TNs.

B.2 AudioMoth code

The code that can be found in the link above is only a modification and an extension of *main.c* of the project developed by Open Acoustic Devices. To see or download the rest of it, please visit `https://github.com/OpenAcousticDevices/AudioMoth-Firmware-Basic/releases/tag/1.3.0`.

The C file available on GitHub contains:

- The definition of necessary constants for the added functionality.
- A modification of *makeRecording* function (included in *main.c*). It calls the added functions:
  - *MFCC* function, which calculates the MFCCs. In turn, it uses:
    - *DCTII* function, which calculates the DCT, as part of the steps needed for the feature extraction (see Subsec. 2.2.1).

- *deltas* function, which calculates the deltas of the MFCCs.
- *neuralNetwork* function, which processes the MFCCs and deltas by the neural network.

Appendix C

Improving performance

C.1 Feature variations

In order to obtain the best feature characteristic, the network was trained three times with the same initial weights and the same order of the inputs. No normalization of the inputs was applied. The modifications were:

- Shape of the Mel filter banks.
- Incorporation of the first MFCC.

C.1.1 Shape of the Mel filter banks

As stated in subsection 2.2.1, there is not a defined Mel filter bank for a specific problem. In this work, a triangular shape is used. However, normalization respect to the area of the triangle has been tested (refer to Figs. 2.6 and 2.7). This validation has been done without including zeroth MFCC.

The confusion matrices for both cases are presented in Figs. C.1 and C.2.

Test performance measured with cross-entropy results to be 0.0509 and 0.0249, for linear scale and for area normalization, respectively.

C.1.2 Incorporation of the first MFCC

In addition, as commented in subsection 2.2.1, the zeroth coefficient is often excluded. To decide if doing so, training has been performed with both options. This validation has been done with a triangular-shaped Mel filter bank with normalization.

The confusion matrices for both cases are presented in figures C.3 and C.2.

Test performance measured with cross-entropy results to be 0.0280 and 0.0249, for 13 and 12 coefficients (with and without the first one), respectively.

C.2 Preprocessing before training

A typical option is to normalize inputs between -1 and 1 before feeding a neural network. Both implementations have been performed in order to choose the best one. This validation has been done with a triangular-shaped Mel filter bank with normalization and not including the zeroth coefficient.

Test performance measured with cross-entropy results to be 0.0249 and 0.0250, for no normalization and normalization, respectively.

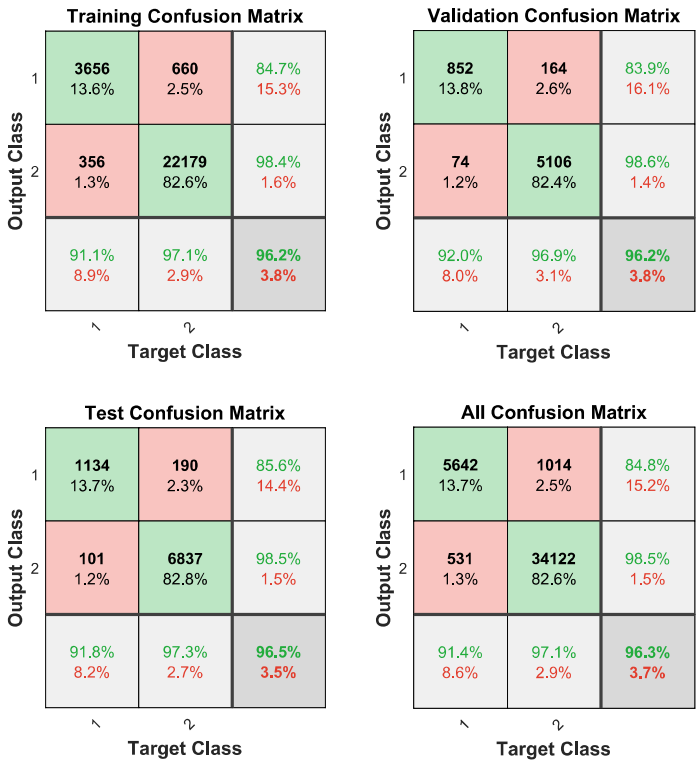


Figure C.1: Confusion matrix after training with triangular Mel filter bank (linear scale).

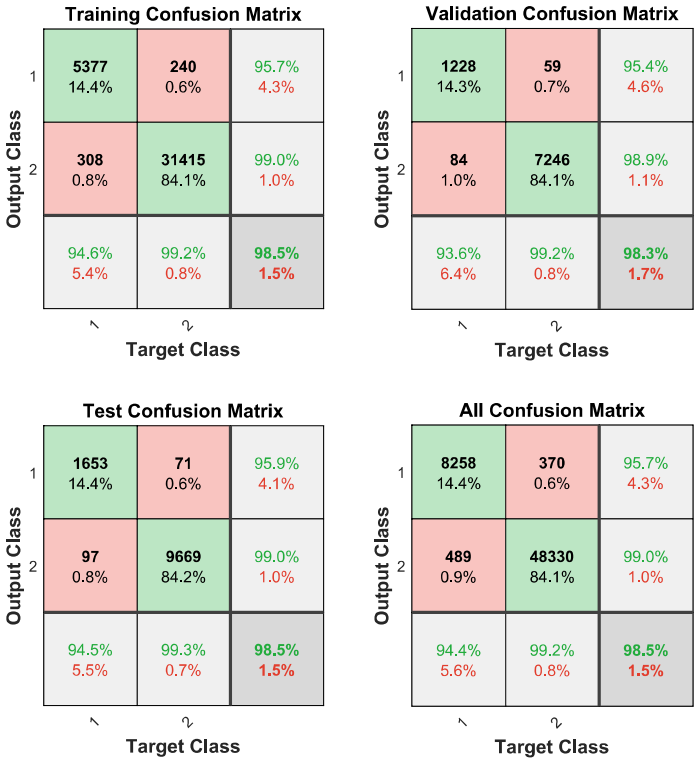


Figure C.2: Confusion matrix after training with triangular Mel filter bank (area normalized).

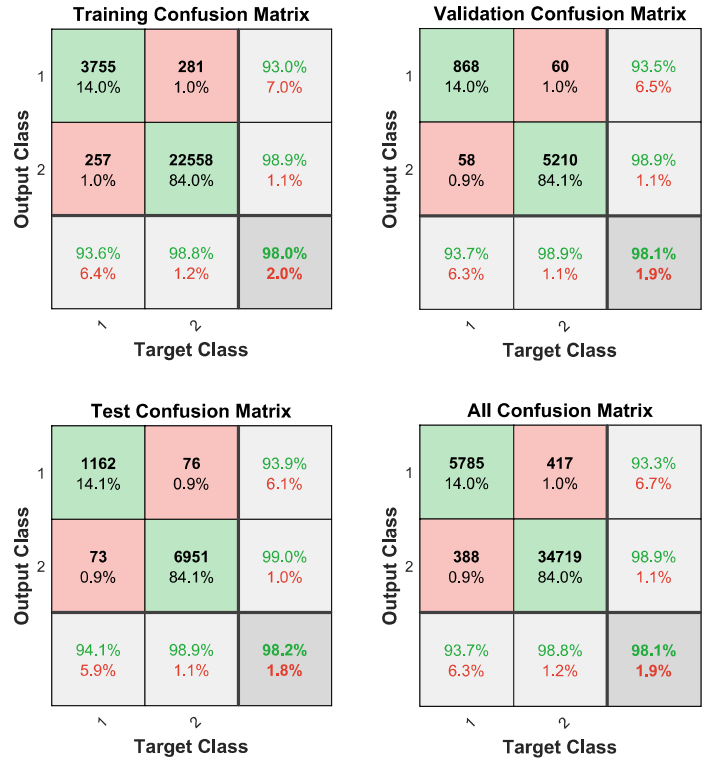


Figure C.3: Confusion matrix after training with inputs including the zeroed MFCC.

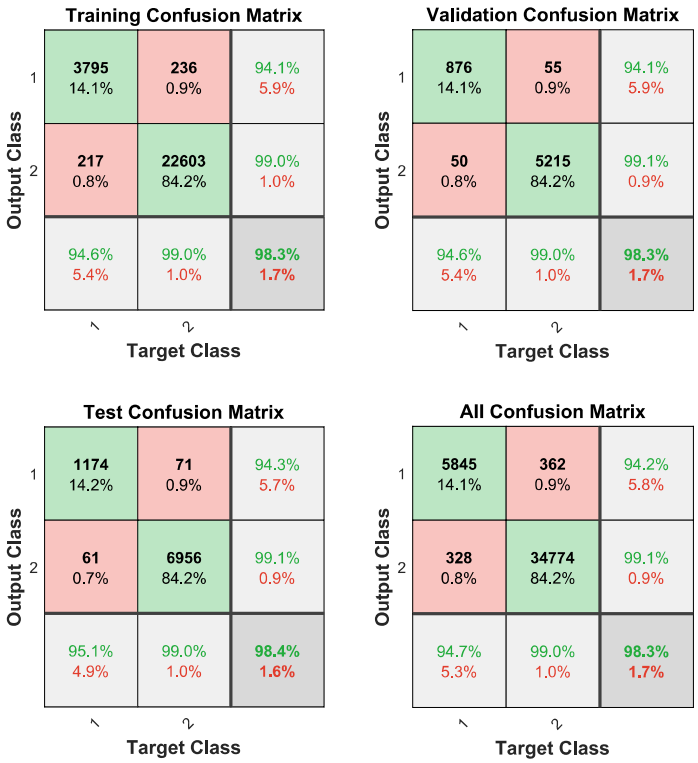


Figure C.4: Confusion matrix after training with input normalization.