UCLouvain

École polytechnique de Louvain

Automated Detection of Bat Species in Belgium

Authors: **Mélanie Beauvois, Lucile Dierckx**
Supervisors: **Olivier Bonaventure, Siegfried Nijssen**
Reader: **Maxime Piraux**
Academic year 2020–2021
Master [120] in Computer Science and Engineering

Acknowledgements

We would first like to thank our Master's thesis supervisors, Prof. Siegfried Nijssen and Prof. Olivier Bonaventure, for their guidance and valuable advice as well as their time.

This thesis could not have been completed without the work and datasets of Batdetective, which served as a starting point for our automated detection of bats.

We would also like to express our gratitude to Natagora, and more specifically to Plecotus, for the large amount of labelled bat call recordings they shared with us.

Our sincere thanks go to Maxime Franco and Corrado Lipani for the work they accomplished last year. We are especially thankful for their data preprocessing and their prompt responses to our questions regarding their work.

A special thanks to Maxime Piraux for his help on technical aspects as well as for his advice and time.

Last but not least, many thanks to our families and friends for their support and encouragements.

Abstract

Many bat species are endangered, mostly by human activity. To address this, environmental organisations census bats, among others, by manually classifying bat call recordings which is a tedious and repetitive task requiring lots of experience. The objective of this thesis is to design an automated, robust and open-source bat call detection and classification tool for the twenty-three bat species in Belgium. Not only do we explore multi-class classification but also multi-label classification. We investigate the performance obtained when combining different machine learning algorithms, namely convolutional neural networks, support vector machines and extreme gradient boosting. The best multi-class classification performance is obtained by combining a CNN for the detection with another CNN followed by an XGBoost for the classification. This gives a precision of 78% and a recall of 75%. For the multi-label classification, our best performing model is composed of a CNN followed by an XGBoost and performs both detection and classification at once. This architecture has a global precision of 73% and a recall of 65%. During this project, we designed a robust tool that runs in real-time, making it possible to use it instead of performing manual classification of bat calls.

Contents

1 Introduction **1**

2 Bats **3**
2.1 Information on Bats . . . 3
2.2 Bats in Belgium . . . 5
2.3 Manual Classification of Bats . . . 8
2.4 Conclusion . . . 10

3 Machine Learning Techniques **13**
3.1 Metrics . . . 13
3.2 Balanced Class Weight . . . 15
3.3 Hard Negative Mining (HNM) . . . 15
3.4 Non Maximum Suppression (NMS) . . . 15
3.5 Support Vector Machine (SVM) . . . 16
3.6 Gradient Boosting . . . 17
3.7 Convolutional Neural Network (CNN) . . . 18
3.8 Losses . . . 20
3.9 Optimisers . . . 21
3.10 Early Stopping . . . 24
3.11 Architecture and Hyperparameter Tuning . . . 24
3.12 Conclusion . . . 26

4 State of the Art **27**
4.1 Batdetective . . . 27
4.2 Batmen . . . 29
4.3 Conclusion . . . 30

5 Bat Recordings Datasets **33**
5.1 Data Provider . . . 33
5.2 Data Collection . . . 34
5.3 Data Content . . . 35
5.4 Data Processing . . . 37
5.5 Data Superposition for Multi-Label Classification . . . 38
5.6 Conclusion . . . 40

6 Architectures for Bat Call Detection and Classification **43**
6.1 Double CNN Architecture . . . 43
6.2 Hybrid Models Using CNN Features . . . 44
6.3 Hybrid Models Using Call Features . . . 45

6.4 Conclusion 47

7 Multi-Class Classification **49**
7.1 Improvements for Multi-Class Classification 49
7.2 Evaluation for Multi-Class Classification 52
7.3 Experiments on the Original Batmen Architecture 55
7.4 Experiments on the Improved Batmen Architecture 57
7.5 Experiments on the Double CNN Architecture 58
7.6 Experiments on the Hybrid Model Using CNN Features 60
7.7 Experiments on the Hybrid Model Using Call Features 62
7.8 Overall Comparison 64
7.9 Classification Time on New Data 66
7.10 Conclusion 67

8 Multi-Label Classification **69**
8.1 Adaptations from Multi-Class to Multi-Label Classification 69
8.2 Evaluation for Multi-Label Classification 71
8.3 Experiments on the Improved Batmen Architecture 74
8.4 Experiments on the Double CNN Architecture 75
8.5 Experiments on the Hybrid Model Using CNN Features 76
8.6 Experiments on the Hybrid Model Using Call Features 78
8.7 Overall Comparison 80
8.8 Classification Time on New Data 81
8.9 Conclusion 83

9 Reproduction of the Results **85**
9.1 System Requirements 85
9.2 How to Train Our Classifiers on Your Own Data 86
9.3 How to Run Our Classifiers on Your Own Data 87
9.4 Conclusion 87

10 Conclusion **89**

Bibliography **93**

A Hyperparameters **99**

B Performance on All the Metrics **107**

C Basic Scenarios of the Multi-Label Confusion Matrices **115**

D Installation Problems and Solutions **117**

List of Figures

2.1 Effect of frequency division on a signal. 9
2.2 Bat call shapes. 9

4.1 Architecture of Batdetective's CNN. 28
4.2 Architecture of Batmen's CNN. 30

6.1 Double CNN architecture at train time. 44
6.2 Double CNN architecture at test time. 44
6.3 Hybrid model using CNN features. 45
6.4 Hybrid model using call features at train time. 46
6.5 Hybrid model using call features at test time. 47

7.1 Batmen's NMS. 50
7.2 Comparison between Batmen's confusion matrix and our improved confusion matrix on different scenarios. 54

8.1 Multi-label confusion matrices for different non-trivial scenarios. 73

List of Tables

2.1 Scientific and common names of bats in Belgium as well as their group and conservation status. . . . 6
2.2 Breeding, hunting and winter habitats of bats in Belgium. . . . 7

4.1 Performance of Batdetective's network on three datasets. . . . 29
4.2 Performance of Batmen's network. . . . 30

5.1 Location and year of the bat call recordings provided by Batdetective. . . . 34
5.2 Location and year of the bat call recordings provided by Natagora. . . . 35
5.3 Repartition of the files and calls in Batdetective's datasets. . . . 36
5.4 Repartition of the files and calls among the groups in Natagora's dataset for multi-class classification. . . . 37
5.5 Repartition of the calls from Natagora's dataset into a training and test set. . . . 37
5.6 Repartition of the files and calls among the groups in Natagora's dataset for multi-label classification. . . . 39
5.7 Number of call superpositions generated for each pair of groups to perform multi-label classification. . . . 39
5.8 Repartition of the calls for multi-label classification into a training and test set. 40

7.1 Influence of Hard Negative Mining on the precision and recall of the original Batmen architecture. . . . 56
7.2 Performance of the original Batmen architecture with our metrics. . . . 57
7.3 Multi-class classification: Influence of Hard Negative Mining on the precision and recall of the improved Batmen architecture. . . . 57
7.4 Multi-class classification: Performance of the improved version of Batmen's network. . . . 58
7.5 Multi-class classification: Influence of Hard Negative Mining on the precision and recall of the double CNN architecture. . . . 59
7.6 Multi-class classification: Performance of the double CNN architecture. . . . 59
7.7 Performance of the detection CNN with Batdetective's parameters. . . . 60
7.8 Multi-class classification: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and CNN features. . . . 60
7.9 Multi-class classification: Performance of the hybrid model using XGBoost and CNN features. . . . 61
7.10 Multi-class classification: Performance of the detection CNN followed by the hybrid model using XGBoost and CNN features. . . . 61
7.11 Multi-class classification: Performance of the hybrid model using an SVM and CNN features. . . . 62
7.12 Multi-class classification: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and call features. . . . 62

7.13 Multi-class classification: Performance of the hybrid model using XGBoost and call features. 63
7.14 Multi-class classification: Influence of Hard Negative Mining on the precision and recall of the hybrid model using an SVM and call features. 63
7.15 Multi-class classification: Performance of the hybrid model using SVM and call features. 64
7.16 Performance overview of the multi-class classification models. 65
7.17 Classification time on new data for our multi-class models using the CPU. . . 67
7.18 Classification time on new data for our multi-class models using the GPU. . . 67

8.1 Multi-label classification: Influence of Hard Negative Mining on the precision and recall of the improved Batmen architecture. 75
8.2 Multi-label classification: Performance of the improved Batmen architecture. . 75
8.3 Multi-label classification: Influence of Hard Negative Mining on the precision and recall of the double CNN architecture. 76
8.4 Multi-label classification: Performance of the double CNN architecture. 76
8.5 Multi-label classification: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and CNN features. 77
8.6 Multi-label classification: Performance of the hybrid model using XGBoost and CNN features. 77
8.7 Multi-label classification: Performance of the hybrid model using an SVM and CNN features. 78
8.8 Multi-label classification: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and call features. 79
8.9 Multi-label classification: Performance of the hybrid model using XGBoost and call features. 79
8.10 Multi-label classification: Influence of Hard Negative Mining on the precision and recall of the hybrid model using an SVM and call features. 80
8.11 Multi-label classification: Performance of the hybrid model using an SVM and call features. 80
8.12 Performance overview of the multi-label classification models. 81
8.13 Classification time on new data for our multi-label models using the CPU. . . 82
8.14 Classification time on new data for our multi-label models using the GPU. . . 82

Abbreviations

AdaGrad Adaptive Gradient Algorithm.

Adam Adaptive Moment Estimation.

BCR Balanced Classification Rate.

BGD Batch Gradient Descent.

CEN Confusion Entropy.

CF Constant Frequency.

CNN Convolutional Neural Network.

FM Frequency Modulated.

FN False Negative.

FP False Positive.

HNM Hard Negative Mining.

IoU Intersection over Union.

NMS Non Maximum Suppression.

QCF Quasi Constant Frequency.

RMSProp Root Mean Square Propagation.

SGD Stochastic Gradient Descent.

SVM Support Vector Machine.

TN True Negative.

TP True Positive.

TPE Tree-Structured Parzen Estimator.

XGBoost Extreme Gradient Boosting.

Chapter 1

Introduction

Bats are mammals that can be found all over the world except in polar regions and extreme deserts. Unfortunately, many bat species are endangered, mostly by human activity. Among others, humans destroy their habitats and poison their food. Many associations preserving biodiversity are concerned about the conservation of bats and set up actions to protect them, conduct census and raise awareness. A Belgian example of such an association is Natagora [40], which, among others, collects various data on bats.

To keep track of which species are present on a certain site and how many bats there are, different census techniques can be used. A first one is to visually track bats during their hibernation, which can wake them up and threaten their survival. Another possibility is to base the census on their ultrasound calls. However, the analysis of bat calls requires a lot of experience and time, especially nowadays with the very large amount of data collected every day.

Having highlighted the fact that bats are endangered as well as the fact that manual identification of bats is a tedious and repetitive task, the need for automated bat identification tools is understandable. This automated identification would ideally be acoustic to minimise the disturbance caused to bats during data collection. Some existing tools [37] perform automated bat call detection based on audio recordings, which means that they identify where the calls are located in a recording. Other automated tools [9, 25] perform both detection and classification of bat calls which means that, after having identified the location of a call, it is associated with a bat species or a bat group.

Most of these automated bat call identification tools are commercial, such as SonoChiro [9], but a few open-source projects exist. For bat call detection, a well-performing project called Batdetective [37] is made available, whereas for bat call classification there does not seem to exist any tool that is both open-source and competitive. Nonetheless, Batmen [25], a Master's thesis presented in academic year 2019-2020, started from Batdetective with the aim of designing an open-source classification tool for Belgian bat species. Batmen's tool can, however, still be improved in terms of performance.

Our thesis aims to develop an automated bat call detection and classification tool for bat species in Belgium and to make this tool available so that everyone can benefit from it and add their contribution. To achieve this, our work takes as a starting point Batmen's thesis and makes use of a large dataset of bat call recordings collected and labelled by

Natagora. Not only do we want to improve Batmen's tool to obtain a better performance, but we also want to try out other architectures to find the one best suited to the problem at hand. These other architectures are combinations of machine learning algorithms, namely deep convolutional neural networks, support vector machines as well as extreme gradient boosting.

Batmen's model performs multi-class classification, which means that it is designed for classifying recordings into one of three or more bat groups. We want to extend this to have models that can classify overlapping calls, which means that more than one bat group can be assigned to the same position. This is referred to as multi-label classification.

Thus, in terms of bat call identification, our goal is to improve the multi-class classification and to handle the multi-label classification. Our other objective is to compare different architectures to find out which one gives the best performance and is, therefore, the best suited for Belgian bat call detection and classification.

This thesis is divided into ten chapters. Chapter 2 gives general information on bats followed by specificities on Belgian species, which are at the core of our thesis. Before moving to the automated bat call classification, different manual classification techniques are described. Chapter 3 goes over the different machine learning techniques and algorithms that have been used throughout our work. Chapter 4 provides information on Batdetective and Batmen, and more specifically on their architecture and obtained results. The following chapter focuses on the provenance, the content and the preprocessing of the datasets used for the tuning, the training and the evaluation of our machine learning models.

The different architectures used to perform bat call detection and classification are described in Chapter 6. Chapter 7 first covers the improvements brought to the multi-class classification of Batmen as well as to the way Batmen evaluates the performance of its model. Then, the experiments made on our multi-class classification architectures are discussed and the different results are compared. Similarly, Chapter 8 first covers the changes brought to our multi-class classification models as well as to the way the performance is evaluated so that those are adapted to the multi-label context. Then, the experiments made on our multi-label classification architectures are discussed and the different results are compared. Before concluding, Chapter 9 gives information on how to train and run our models on new datasets.

Chapter 2

Bats

Since our work revolves around bats, and more specifically around their automated audio classification, it is important to first have some knowledge on bats.

This chapter begins with a general presentation of bats. It contains information on their calls, their seasonal activity pattern and the threats that they face. The focus is set on the bats in Belgium and more specifically on the Belgian association that monitors them, on the local species and on the locations where they can be found. Then, the manual classification of bats is discussed. Heterodyning, frequency division and time expansion, three ultrasound decoding systems, are described and the main bat call characteristics to manually identify bat species are addressed.

2.1 Information on Bats

Bats are nocturnal mammals of the Chiroptera order. They can be found all over the world except in polar regions and extreme deserts. They represent 20% of the mammals and are the only ones among them that can fly. Nowadays, more than 1200 species have been identified around the world [21, 58].

Calls

Bats emit calls that lie in the ultrasound domain. Certain species emit their calls through the nose, while others emit them through the mouth. Bat calls lie between 9kHz and 200kHz but the exact frequency range depends on the species. Most of their calls cannot be heard by humans as the latter can only hear sounds that have a frequency between 20Hz and 20kHz [33].

Among bat calls, two main categories can be distinguished based on the purpose of the call. The first category regroups the calls used for echolocation. Even though bats are not blind, they use echolocation calls to be able to travel and hunt at night. When a bat call encounters an obstacle or prey, it is reverberated and comes back to the bat that uses it to collect information on its environment [17, 33].

The second category encompasses the social calls, which are used to communicate with other bats. These calls are used in situations of conflict, distress, reproduction as well as to communicate with bats of the same species and to ward off enemies. The social calls

lie in lower frequencies than those used for echolocation and some of them can be heard by humans [17, 33].

Seasonal Activity Pattern

During winter, most bats hibernate so as not to starve, while a few migrate to warmer places. Indeed, most of them are insectivores and have thus no prey during that period. Bats hibernate in roosting sites that are quiet and that present a constant high humidity and low temperature. Common places that meet these characteristics are for instance caves, basements and trees. Bats need to adapt their metabolism during hibernation to save enough energy and survive the winter. They reduce their body temperature and slow down their heart rate and breathing. During that period, they should not be disturbed because waking up makes them use up a lot of energy [15, 24, 33].

When spring arrives, bats come out of hibernation and need to regain energy by hunting. Once they have recovered enough energy, they migrate to their summer roosting site, which is often only a few kilometres away from their winter home [15, 24, 33].

During summer, males and females stay in different roosting sites. Around May, females move to breeding colonies, which can count up to a few thousand female bats. These breeding colonies can for instance be trees, church spires and attics. This is where they give birth to one, rarely two, bat pups. Both males and females go hunting at dusk [15, 24, 33].

Autumn is the mating season during which bats gather by thousands at swarming sites. This is the period during which bats hunt to stock energy for the winter hibernation. They also start to migrate to their winter roosting site [15, 24, 33].

Threats

Even though bats are legally protected since 2012 in every country of the European Union, one out of three species suffers from a strong population decline. The main culprits are humans whose activities are harmful to bats [24].

A first domain in the life of bats that is impacted by human activity are the roosting sites. Humans destroy and close many places that bats usually take as their home. Among others, trees are cut down, underground cavities are closed, old buildings are demolished and access to buildings is made impossible by the isolation of roofing. Moreover, when bats still have access to roofing where the framework has been treated with toxic products, they can get poisoned. In addition to the closure and destruction of their habitats, bats are also disturbed by speleologists, tourists and sportsmen. Disturbing bats during winter wakes them up from hibernation and makes them consume a lot of energy which threatens their survival. When a breeding colony is disturbed, the engendered panic causes the abandonment of many bat pups [21, 24].

Human activity also has a threatening impact on the nutrition of bats. The modifications on their hunting territories, such as the removal of groves, the draining of wetlands and the channelling of rivers, make it difficult for bats to find food. Moreover, the use of chemical products like pesticides and insecticides kills lots of insects which reduces the

food available for bats. Bats can also get intoxicated if they eat too many preys that have been in contact with these chemicals [16, 33].

Other deadly threats are the infrastructures built by humans that hinder bats when they move around. When roads are constructed, they sometimes cut across places where bats often fly which disrupts their flight route. Collisions can happen with vehicles, especially when bats are used to the location and are thus less careful to possible obstacles. A further infrastructure that disrupts the flight route of bats and causes collisions are wind farms. These also generate important pressure variations, which cause fatal internal bleeding to bats [16, 21, 33].

2.2 Bats in Belgium

This section first presents Natagora, a Belgian association that, among others, collects data on bats. It also gives information on bats in Belgium, such as which species are found there and in which regions.

Natagora

Natagora [40] is a Belgian association protecting nature and collecting various data in Wallonia and Brussels. It aims to prevent further biodiversity loss and works on a balanced cohabitation between human activities and nature.

A specific department, called Plecotus [46], has been created to centralise all of Natagora's initiatives revolving around bats. One of their main goals is the observation and study of bats. Each year, hundreds of caves are visited to collect information on the hibernation of bats. During the reproduction period, Plecotus monitors the bat activity in shelters. When collecting information, Plecotus makes sure not to disturb the bats. In addition to protecting bats, Plecotus regularly updates inventories of the protected species across Belgium.

Since bats are not the animals humans appreciate the most, some activities are organised by Plecotus to raise public awareness. For example, families can participate in the yearly "Nuit Européenne des chauves-souris" where they have the opportunity to observe bats with domain experts.

Species

Among all bat species, twenty-three can be found in Belgium [45] and these are presented in Table 2.1. Their scientific and common names are listed and an indication of their conservation status is given [61]. This table also specifies the group to which the different species belong according to SonoChiro [13]. SonoChiro [9] is a paying software for automated bat analysis that is used by Natagora to get various information on their bat call recordings.

Scientific name	Common name	Group	Conservation status
Barbastella barbastellus	Western barbastelle	Barbar	Critically endangered
Eptesicus serotinus	Serotine bat	Envsp	Data deficient
Myotis alcathoe	Alcathoe bat	Myosp	Data deficient
Myotis bechsteinii	Bechstein's bat	Myosp	Data deficient
Myotis brandtii	Brandt's bat	Myosp	Least concern
Myotis dasycneme	Pond bat	Myosp	Endangered
Myotis daubentoni	Daubenton's bat	Myosp	Least concern
Myotis emarginatus	Geoffroy's bat	Myosp	Endangered
Myotis myotis	Greater mouse-eared bat	Myosp	Endangered
Myotis mystacinus	Whiskered bat	Myosp	Least concern
Myotis nattereri	Natterer's bat	Myosp	Endangered
Nyctalus lasiopterus	Greater noctule bat	Envsp	Data deficient
Nyctalus leisleri	Leisler's noctule	Envsp	Data deficient
Nyctalus noctula	Common noctule	Envsp	Data deficient
Pipistrellus kuhlii	Kuhl's pipistrelle	Pip35	Data deficient
Pipistrellus nathusii	Nathusius's pipistrelle	Pip35	Data deficient
Pipistrellus pipistrellus	Common pipistrelle	Pip50	Least concern
Pipistrellus pygmaeus	Soprano pipistrelle	Pip50	Data deficient
Plecotus auritus	Brown long-eared bat	Plesp	Vulnerable
Plecotus austriacus	Grey long-eared bat	Plesp	Vulnerable
Rhinolophus ferrumequinum	Greater horseshoe bat	Rhisp	Critically endangered
Rhinolophus hipposideros	Lesser horseshoe bat	Rhisp	Critically endangered
Vespertilio murinus	Parti-coloured bat	Envsp	Data deficient

Table 2.1: Scientific and common names of bats in Belgium as well as their group and conservation status.

Location

All the twenty-three species mentioned in the previous subsection can be found all over Belgium except for five of them. The alcathoe bat can only be found in Wallonia, while the greater noctule bat can only be found in Flanders. The kuhl's pipistrelle can be found everywhere in Belgium except in Flanders. The greater and lesser horseshoe bats can be found all over Belgium except for Brussels [61].

Depending on the species and on the time of the year, bats can be found in different types of habitats. Table 2.2 summarises the places occupied by bats during the breeding season and during the hibernation period as well as the places where they hunt [61].

Scientific name	Breeding habitats	Hunting grounds	Winter habitats
Barbastella barbastellus	Tree cavities	Forests	Tree cavities, Underground sites
Eptesicus serotinus	Roofing	Forest edges, Gardens, Grasslands	Underground sites, Building gaps
Myotis alcathoe	Tree cavities	Humid forests	Tree cavities
Myotis bechsteinii	Tree cavities	Forests, Clearings, Gardens	Tree cavities, Underground sites
Myotis brandtii	Trees, Nest boxes	Forests	Underground sites
Myotis dasycneme	Roofing, Building gaps, Tree cavities	Aquatic milieu, Grasslands, Forests	Underground sites
Myotis daubentoni	Tree cavities, Near water	Aquatic milieu, Forest edges	Underground sites
Myotis emarginatus	Roofing, Cellars, Stables	Alluvial valleys, Humid forests	Underground sites
Myotis myotis	Roofing	Parcs, Fields, Grasslands, Forests	Underground sites
Myotis mystacinus	Behind disjointed, flat, narrow spaces	Aquatic milieu, Grasslands, Forests	Underground sites
Myotis nattereri	Tree cavities	Fields, Forests	Underground sites
Nyctalus lasiopterus	Tree cavities	Forests	Tree cavities
Nyctalus leisleri	Tree cavities, Forest cabins	Aquatic milieu, Grasslands, Forests	Tree cavities
Nyctalus noctula	Tree cavities, Forest cabins	Aquatic milieu, Grasslands, Forests	Tree cavities
Pipistrellus kuhlii	Building gaps	Aquatic milieu, Gardens, Forests	Building gaps
Pipistrellus nathusii	Forest cabins, Tree cavities, Nest boxes	Aquatic milieu, Forests	Tree cavities
Pipistrellus pipistrellus	Building gaps	Aquatic milieu, Gardens, Forests	Building gaps
Pipistrellus pygmaeus	Building gaps	Aquatic milieu, Forests	Building gaps, Tree cavities
Plecotus auritus	Roofing, Tree cavities, Nesting boxes	Forests, Orchards, Grasslands, Hedges	Underground sites
Plecotus austriacus	Roofing, Tree cavities, Nesting boxes	Forests, Orchards, Grasslands, Hedges	Underground sites
Rhinolophus ferrumequinum	Roofing of churches and farms	Forests, Cliffs, Aquatic milieu	Underground sites
Rhinolophus hipposideros	Stairwells, Boiler rooms, Roofing	Hedges, Orchards, Forest edges, Parcs	Underground sites
Vespertilio murinus	Buildings, Cliffs	Aquatic milieu	Cliffs, Walls cracks

Table 2.2: Breeding, hunting and winter habitats of bats in Belgium.

2.3 Manual Classification of Bats

This section discusses the manual classification of bats. Three ultrasound decoding systems are described and the main bat call characteristics to manually identify bat species are addressed.

Ultrasound Decoding Systems

A first ultrasound decoding system that is frequently used is called heterodyning [5, 21, 60]. It is a real-time method where the user sets a frequency and the detector is only able to detect the frequencies close to the fixed one. Depending on the heterodyne detector, the audible frequency bandwidth around the chosen frequency can be larger or narrower.

Using this type of detector allows making abstraction of the various noises present on frequencies that lie out of the bandwidth. What is actually heard is the beat, which is the difference between the fixed frequency and the actual one, emitted by the bat. The closer the two frequencies are to each other, the lower is the sound emitted by the device. Theoretically, when the two frequencies are equal, no sound is emitted and it is called the zero beat.

The user varies the chosen frequency to find the one fitting best the heard call. The final zero beat frequency helps determine the species of the bat. A good initial frequency lies between 42 and 45 kHz as most bat species can be heard when the detector is set on such a frequency. Moreover, it is possible to use several detectors set on different frequencies to have a wider range of detection.

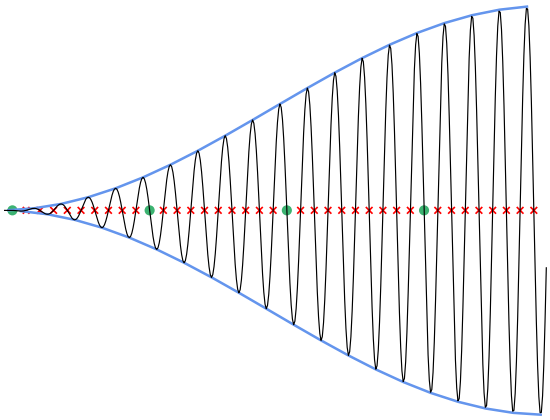
Another common ultrasound decoding system is the frequency division [5, 21, 59]. It is a real-time method that maps ultrasounds into the audible domain by dividing the frequency by a chosen factor, as illustrated in Figure 2.1. This method uses the envelop of the call as well as the zero amplitude points (Figure 2.1.a). Then, it recomposes the signal based only on the envelope and on a fraction of the zero amplitude points (Figure 2.1.b).

The most common division factor to analyse bats is 10. In that case, a 50 kHz call is heard as a 5 kHz call. Analogously, with a factor of 10, a call that is composed of 50 oscillations is mapped to a signal made of 5 oscillations.

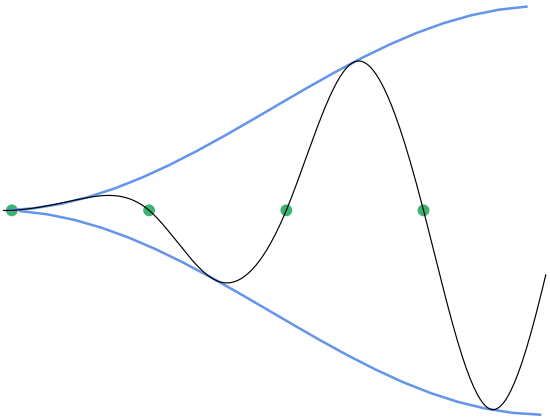
This method allows dividing the frequency while keeping the original amplitude and length of the call. Unlike heterodyning, the whole frequency domain is taken into consideration which avoids restricting the observation to a certain frequency band. However, it captures fewer details than heterodyning for a certain frequency band.

A further decoding system that is frequently used is called time expansion [5, 21]. It is not a real-time method. Indeed, the signals are recorded and replayed at a slower pace which allows hearing calls that would not be audible for humans otherwise. It is often used since it is the only method that keeps all the characteristics of the call. Another advantage is that the bat calls can be recorded with a classic recorder. However, only

short sequences can be recorded because the time expansion technique requires a high sampling rate.



2.1.a. Evolution of the signal as a function of time before the frequency division.



2.1.b. Evolution of the signal as a function of time after the frequency division.

Figure 2.1: Effect of frequency division on a signal. The signal envelope is illustrated in blue. The red and green dots represent the points of zero amplitude. Only the green ones are kept to perform a frequency division by 10.

Main Call Characteristics

When chiropterologists classify bats based on their calls, they take into consideration multiple call characteristics. A first one is the shape of the call. There exists 4 common shapes illustrated in Figure 2.2: Frequency Modulated (FM), Quasi Constant Frequency (QCF), FM-QCF and Constant Frequency (CF).

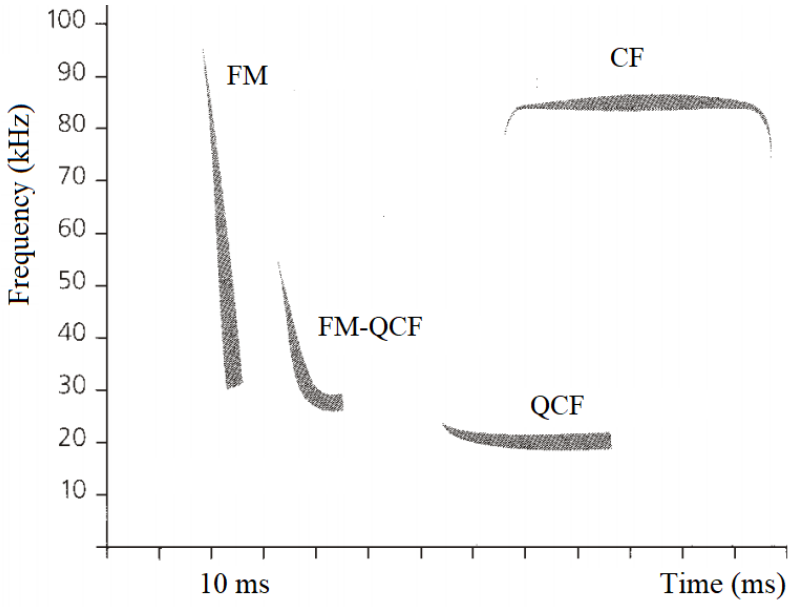


Figure 2.2: Bat call shapes. FM stands for frequency modulated, QCF for quasi constant frequency and CF for constant frequency [60].

The frequency modulated structure covers a wide frequency bandwidth but lasts only for a few milliseconds because it is very steep. The species using this type of calls are present in forested areas.

Unlike the FM shape, the quasi constant frequency shape covers a narrow frequency bandwidth of less than 5 kHz. The species travelling in open spaces use this structure of calls.

When a species alternates between forest environment and open spaces, it emits calls presenting an FM-QCF structure. It is composed of a descending FM followed by a QCF. In that way, it lasts longer than an FM and has a larger frequency range than a QCF.

The constant frequency shape is only observable in calls emitted by horseshoe bats. It always starts with a short ascending FM followed by a long constant part and ends with a short descending FM. Therefore, it has a narrow frequency bandwidth but has a longer duration than an FM or an FM-QCF [5, 60].

Other call features that are relevant to manually classify bat calls are the minimal and maximal frequencies as well as the bandwidth, which is the difference between these two frequencies. Moreover, the frequency of maximum energy, also called peak frequency, as well as the frequency at the beginning and end of the call, are relevant characteristics. Finally, the duration of the call, the interval between two calls and the sonority, which can be either nasal or whistled, are useful to identify the bat species [5, 60].

2.4 Conclusion

Bats emit ultrasound calls between 9 and 200 kHz. These are either social or echolocation calls. During winter, they hibernate and when spring arrives, they need to regain the energy they lost during their hibernation. Then, bats migrate to their summer breeding colonies. Autumn is the mating season as well as the time to stock energy for the winter. The main threat to bats are humans. They destroy bats' roosting and hunting sites, poison the insects bats eat and build infrastructures that cause collisions with bats.

In Belgium, there are twenty-three species divided into seven groups. Bats present in Belgium are protected and monitored by Plecotus, the bat department of Natagora. Natagora is a Belgian association protecting nature and collecting various data in Wallonia and Brussels, such as the conservation status of bats, their hunting grounds and their breeding and winter habitats which are all summarised in this chapter.

This chapter also discusses the manual classification of bats based on their calls. Heterodyning, frequency division and time expansion are three ultrasound decoding systems. Heterodyning is a real-time method where the user sets a frequency and the detector is only able to detect the frequencies close to the fixed one. The user varies the chosen frequency until finding the zero beat frequency. Frequency division is another real-time method that maps ultrasounds into the audible domain by dividing the frequencies by a chosen factor. Unlike the other two methods, time expansion is not a real-time method. The signals are recorded and replayed at a slower pace to make the bat calls audible for humans.

The main call characteristics used by chiropterologists are the shape, the frequency properties, the sonority, the duration of the call and the interval between two calls. The four common bat call shapes are called frequency modulated (FM), quasi constant frequency (QCF), FM-QCF and constant frequency (CF). The commonly used frequencies are the minimal, maximal, peak, initial and final frequencies as well as the frequency bandwidth.

Chapter 3

Machine Learning Techniques

This chapter presents the different machine learning techniques that are used throughout our work to perform multi-class and multi-label classification. First, some general machine learning concepts are described. Among those are the metrics used in the following chapters, the principle of balanced class weight as well as the Hard Negative Mining and Non Maximum Suppression algorithms. Then, an explanation of the Support Vector Machine model and of some Gradient Boosting algorithms is given. In a second phase, the focus is set on the neural network domain with the presentation of Convolutional Neural Networks, losses, optimisers, early stopping as well as hyperparameter and architecture tuning.

3.1 Metrics

A lot of metrics have been designed over the years to assess and compare the quality of models. The following subsections present the classical ones as well as the way to use them in a multi-class context.

Accuracy

Accuracy [26], represents the fraction of examples that are correctly classified. A perfect system has thus an accuracy of 1. The accuracy can be computed as

$$\frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

where
True Positive (TP) is the number of examples for which the predicted class corresponds to the considered class.
True Negative (TN) is the number of examples that are predicted as not belonging to the considered class and that do not belong to that class.
False Positive (FP) is the number of examples that are predicted as belonging to the considered class while they do not belong to that class.
False Negative (FN) is the number of examples that are predicted as not belonging to the considered class while they do belong to that class.

Precision

Precision [26] is the fraction of examples that are correctly classified to a class among all the examples that are classified to that class. It is defined as

$$\frac{TP}{TP + FP} \tag{3.2}$$

Recall

Recall [26], also called sensitivity, is the fraction of examples that are correctly classified to a class among all the examples that actually belong to that class. Its formula is given by

$$\frac{TP}{TP + FN} \tag{3.3}$$

Precision-Recall Curve

The precision-recall curve can be used in binary problems to show the relation between precision and recall. This curve is built by varying the probability threshold above which samples are considered as belonging to the positive class. The threshold takes values between 0 and 1. For each threshold, the precision is plotted in function of the recall to form the precision-recall curve. From the latter, it is for example possible to retrieve the recall at 95% of precision, which is sometimes used as a metric to evaluate models [11].

Micro- and Macro-Averages

Since the metrics presented in the previous subsections are intended for binary classification, some adjustments need to be made to use them in a multi-class context. The most common way of achieving this is to compute the average of the metric.

A first possible version is the macro-average [26], which computes the arithmetic mean of the metric over all the N classes

$$\frac{\sum_{n=1}^{N} metric_n}{N} \tag{3.4}$$

Another variant is the micro-average [26], which weighs each sample equally. This gives the same value for accuracy, precision and recall:

$$\frac{\sum_{n=1}^{N}(TP_n + TN_n)}{\sum_{n=1}^{N}(TP_n + TN_n + FP_n + FN_n)} \quad \text{(Accuracy)} \tag{3.5}$$

$$= \frac{\sum_{n=1}^{N} TP_n}{\sum_{n=1}^{N}(TP_n + FP_n)} \quad \text{(Precision)} \tag{3.6}$$

$$= \frac{\sum_{n=1}^{N} TP_n}{\sum_{n=1}^{N}(TP_n + FN_n)} \quad \text{(Recall)} \tag{3.7}$$

The main difference between the macro- and micro-average is that the macro-average gives the same weight to every class while the micro-average gives a weight proportional to the number of examples for each class. Thus, with the latter, the majority classes have a bigger impact on the metric than the minority classes.

3.2 Balanced Class Weight

In the case of an imbalanced dataset, it can be useful to balance the importance of the different classes so that each class has the same impact during training, no matter the number of training samples available for each class. The weight of each class y is defined as [12]

$$\frac{\text{total samples}}{\#\text{classes} * \#\text{samples(y)}} \tag{3.8}$$

where
total samples is the number of samples used for the training.
#classes is the number of different classes that are used as labels.
#samples(y) is the number of samples in the training set that are labelled as class y.

3.3 Hard Negative Mining (HNM)

When facing an object detection problem, the data is often composed of few positive examples and an enormous amount of negative examples. To build the training set, all the positive examples are taken. Since using the same number of positive and negative examples is preferable, the negative examples are not all taken. Instead, negative examples are picked randomly from the training files until reaching the number of positive examples. However, these negative examples might not be the most relevant to get a well-performing classifier. To obtain negative examples that are more difficult to classify, the Hard Negative Mining method can be used [55].

Before applying Hard Negative Mining (HNM), the network needs to be trained on the training data once. Then, the network is used to predict a classification on the training files, which contain all the training examples and all the negative examples that were not picked for the training set. The principle of HNM is to retrieve among all the training files the examples that were classified as positive but are actually negative. These are called false positives and are added to the training set as hard negative examples. The new training set is then used to train the network again.
This will help increase the performance of the model since it will train on the negative examples that were wrongly classified during the previous training.

HNM can be applied several times in a row until no improvement is observed anymore. However, each iteration of HNM takes a lot of time because of the training and testing. Besides, the training set keeps getting larger with the addition of the hard negatives examples which slows down the training and testing.

3.4 Non Maximum Suppression (NMS)

It is common in sound classification to decompose the file into overlapping windows that are then fed as input to the classifier in a sliding window fashion. The network associates a probability of containing a call to each window. Several windows can contain the same call and thus each of them will be given a high probability. However, only one window should be kept per call since each call should be associated with a single position.

That is when Non Maximum Suppression (NMS) [27] comes into action. The algorithm starts by selecting the window with the highest probability among all the windows of the recording. This window will be part of the final set of windows and is thus removed from the set of windows that still need to be filtered. The Intersection over Union (IoU) between the selected window and all the other ones is computed by

$$\frac{\text{Area of overlap}}{\text{Area of union}} = \frac{A \cap B}{A \cup B} \tag{3.9}$$

where A and B are two windows.

Every window that has an IoU above a chosen threshold is removed from the set of windows that still need to be filtered. Then again, the window with the highest probability is chosen among the set of windows to be filtered. All the steps are repeated until the latter is empty. The windows that are present in the final set correspond to the positions with the highest probability for each call.

3.5 Support Vector Machine (SVM)

The Support Vector Machine (SVM) [49] is a classification algorithm that tries to determine the hyperplane separating best the data samples coming from different classes. The hyperplane is chosen such as to maximise the margin, which is the distance between the hyperplane and the closest data points to it. Those points that lie on the margin are called support vectors. They are essential data samples because, without them, the position of the separating hyperplane would change. The margin is maximised so that the support vectors are as far as possible from the hyperplane while separating the data samples from different classes as best as possible. Having a large margin enables the SVM model to have more confidence when classifying data samples that have not been encountered during training. Indeed, the classification confidence increases the further a data point is from the hyperplane.

Since the SVM always determines a hyperplane that does a linear separation for any problem, it needs the kernel trick in case of non-linear separation problems. When a problem is not linearly separable in the original dimensional space, the data needs to be mapped into higher dimensions in order to become linearly separable. However, adding dimensions is computationally expensive. Using a kernel allows to achieve the same results as the dimensional augmentation but in a computationally efficient way. There exist several kernels among which the linear, polynomial, radial basis function (RBF) and sigmoid kernels [49].

Like other classifiers, the SVM has advantages and drawbacks. The SVM works well with smaller and cleaner datasets while it performs poorly on larger and noisier datasets. Another advantage is that the SVM is memory-efficient because it only uses the support vectors in the decision function and not all the training points. A drawback is that no probability can directly be retrieved from the classification since data samples are classified depending on their position with respect to the separating hyperplane.

3.6 Gradient Boosting

Before explaining the concept of Gradient Boosting, it is important to understand what boosting is. Boosting [32, 64], also called Hypothesis Boosting, is an ensemble method. The principle of ensemble methods is to combine a large number of weak learners so that, all together, they form a strong model. Weak learners are models that perform only somewhat better than random models.

Boosting is a sequential process in which a given number of weak learners are created one after the other. The performance of the previous model influences the training of the following weak learner. Indeed, the next model either receives only the data examples that were wrongly classified by the previous model or it receives the whole dataset, but with more weight on those examples that were wrongly classified. In this way, the models created towards the beginning of the sequence are more adapted to the common data samples, while the models created towards the end of the sequence have been specialised for difficult examples. Since the first model has no previous one, it receives the whole dataset with a weight of one for each example. In addition, each model is associated with a weight that is computed based on its classification error at train time. The weight of a model determines the impact that it has when doing a prediction with the final strong model.

Once the boosting model is trained, it can be used to make predictions on new data in a straightforward way. To classify an example, the latter is fed to all the weak models individually. In the case of regression, the global prediction is the weighted average of the individual predictions, while for classification the final prediction is based on a majority vote.

Using a boosting model instead of a single model has several advantages. A first one is that it improves the stability as well as the performance and reduces the risk of overfitting. Another advantage is that if a model makes a bad prediction, its impact can be reduced if most of the other models make a good prediction.

Now that the boosting concept has been described, Gradient Boosting [32] can be explained. The general ideas are the same as in boosting and, typically, the weak learners are decision trees.

A first difference lies in the fact that the whole training set is used to build every new tree no matter the performance of the previous model. This means that, unlike in boosting, no sample is removed from the training set and no weight is associated with the examples either.

Another difference is that the tree that is being built does not predict the label of the sample, but instead predicts what is called the residual. A residual is the prediction error made by the combination of all the previous trees on a training example. Since the first tree has no previous one, a first value is computed, for instance by taking the mean of the training labels, to get an initial rough prediction. This initial prediction is used to compute the residuals that the first tree has to predict. Thus, the initial prediction is refined by the addition of each tree. However, adding too many trees can lead to overfitting.

A last difference with boosting concerns the weight assigned to each model. In gradient boosting this weight does not represent how well a certain tree performs on the training set, but is a fixed chosen value called the learning rate. The latter is the same for each tree except for the initial prediction, which is not weighted. During training and testing, the prediction of a sample is the sum of the initial prediction and the weak prediction of each tree, weighted by the learning rate.

Gradient Boosting has been implemented in multiple libraries, among which the Python Extreme Gradient Boosting (XGBoost) library [14]. This open-source project puts emphasis on computational efficiency.

Several hyperparameters can be chosen when using XGBoost. There is the number of estimators, the maximum depth of each estimator and the learning rate. When the algorithm decides whether to partition a leaf node, it verifies if the loss reduction induced by the split would be greater than gamma, the minimum loss reduction. It also verifies if the weights of all observations of a leaf node sum up to at least the value of the minimum child weight hyperparameter. Moreover, the ratio of training samples that are used to grow each tree is defined by the subsample hyperparameter. For unbalanced classification problems, the scale positive weight hyperparameter helps tackle the imbalance issue by giving more weight to the minority class. It is also possible to use XGBoost with the GPU by specifying it in the tree method hyperparameter [18].

3.7 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) [30, 44], also called ConvNet, is a deep learning neural network designed to process structured arrays of data that is spatially dependent. Such data can be one-dimensional signals or two-dimensional images and audio spectrograms. Even though CNNs were initially designed for image classification, they can also be used to classify other types of data.

The principle of a CNN is to transform the input into a set of features that is easier to classify. CNNs are designed to learn and keep features that are essential for a good prediction while giving little or no weight to features that are not useful for the classification. Moreover, the data given as input to a CNN does not need to be pre-processed and CNNs are scalable to large datasets.

The first part that composes a CNN are the convolutional and pooling layers that tackle the feature extraction. It is followed by dense layers, which handle the classification. The last dense layer has as many nodes as the number of classes with which the data can be labelled. Each convolutional and dense layer is associated with an activation function. The activation function as well as the pooling layer use a function that is fixed and does not need to be learned. On the contrary, the convolutional and dense layers are composed of neurons, which have weights and biases. These parameters need to be learned during training using gradient descent.

The different layers are sequentially combined and their characteristics are described in the following subsections.

Convolutional Layer

The sequence of one or more convolutional layers [30, 44] is responsible for the feature extraction and retrieves high-level features from the data. These features are computed using the convolution operation between the input and a filter at each convolutional layer. The convolution operation consists in taking the dot product between the filter and a small square of the input having the same size as the filter. To do the dot product with the whole input, the filter is moved in a sliding fashion over the input. A given convolutional layer often contains more than a single filter. Having multiple filters allows learning multiple features in parallel.

The convolution preserves the spatial relationship of the data. When the convolution reduces the dimensionality of the input, the term called valid padding is used. Keeping the dimensionality unchanged is referred to as same padding [10].

Since a CNN can be composed of several convolutional layers, the first one extracts low-level features while the convolutional layers added to the first one take care of higher-level features. This allows the network to have an overall as well as a refined understanding of the data.

Pooling Layer

The pooling layer [10, 44], also called the downsampling layer, is often used after a convolutional layer. Its objective is to reduce the size of the extracted features so that the number of connections between the layers is decreased. This is useful in the case of large inputs to reduce the computational cost. Another advantage of this layer is the reduction of overfitting. Even though the dimensionality is decreased, the pooling layer keeps important features.

The input is divided into non-overlapping pooling regions on which a pooling operation is applied. The most commonly used pooling operation is max pooling, which takes the maximum value of the considered pooling region. This pooling operation also serves as a noise suppressant. Indeed, in addition to dimensionality reduction, the max pooling performs some denoising. Besides max pooling, average pooling can be applied to keep the average value of each pooling region. Average pooling also reduces noise but only by reducing the dimensionality of the feature map. A less common pooling operation is the sum pooling, which computes the sum of all values in the considered pooling region.

Dense Layer

Dense layers [4, 44], also called fully connected layers, usually follow the convolutional and pooling layers. As the name indicates, the neurons in a fully connected layer are all connected to the neurons of the next layer. To avoid overfitting during training, a certain random fraction of the links between the dense layers can be dropped out.

The dense layers are responsible for the classification process. The first dense layer receives as input the features that are extracted by the convolutional and pooling layers of the network. There can be more than one dense layer but the last one always has as many neurons as the number of classes with which the data can be labelled.

Activation Functions

An activation function [23, 41] is used after each convolutional and dense layer to introduce non-linearity to the CNN. This allows extracting complex information out of the data which would not be possible with a simple linear function. Various activation functions can be used depending on the task and the layer on which the function is applied.

A first activation function is the Rectified Linear Unit (ReLU), which is commonly used for the convolutional and dense layers except for the output dense layer. It is defined as

$$max(0, x) \tag{3.10}$$

where x is the value of a neuron of the layer on which the ReLU function is applied.

Another activation function is the softmax function, which computes the relative probability of each class so that they sum up to 1. This activation function is used for the output layer of the CNN in case of multi-class classification. The softmax function is defined as

$$\frac{e^{x_j}}{\sum_k e^{x_k}} \tag{3.11}$$

where x_i is the value of the ith neuron of the layer on which the softmax function is applied.

The sigmoid activation function is also frequently used and is applied in multi-label contexts for the output layer of the CNN. For each class, it computes a probability between 0 and 1, independently of the value of the other classes. Therefore, the probabilities of all classes do not necessarily sum up to 1. This independence between classes is required for the multi-label classification so that more than one class can be associated with a high probability. The sigmoid function is defined as

$$\frac{1}{1 + e^{-x}} \tag{3.12}$$

where x is the value of a neuron of the layer on which the sigmoid function is applied.

3.8 Losses

During the training of a machine learning model, it is common to use a loss as an indication of how close the predictions are to the actual data. Various losses have been designed for that purpose.

Cross-Entropy Loss

The cross-entropy loss [30, 54] is a loss that is frequently used for detection and classification tasks. The categorical cross-entropy loss is used for the multi-class context while the binary cross-entropy loss is chosen for binary tasks.

For one sample, the categorical cross-entropy loss can be computed by

$$-\sum_{i=1}^{\mathrm{N}} t_i \log(p_i) \tag{3.13}$$

where N is the number of classes, t_i is 1 if class i is the actual class and 0 otherwise, and p_i is the probability that the example belongs to class i.

For one sample, the binary cross-entropy loss can be computed by

$$-\sum_{i=1}^{2} t_i \log(p_i) = -t_1 \log(p_1) - (1 - t_1) \log(1 - p_1) \tag{3.14}$$

where t_i is 1 if class i is the actual class and 0 otherwise, and p_i is the probability that the example belongs to class i.

Focal Loss

The focal loss [34] was initially designed for imbalanced binary classification. Unlike the cross-entropy loss, this loss weighs the contribution of each sample depending on how well it was classified. To do so, the modulating factor $(1 - p_i)^\gamma$ is added.

For one sample, the focal loss is defined as [35]

$$-\sum_{i=1}^{\mathrm{N}} (1 - p_i)^\gamma \; t_i \log(p_i) \tag{3.15}$$

where N is the number of classes, p_i is the probability that the example belongs to class i, γ is the focusing parameter (≥ 0) and t_i is 1 if class i is the actual class and 0 otherwise.

For each example, only the probability of belonging to the actual class is considered. The lower that probability is, the higher is the impact of that training example on the loss. Thus, the examples that are correctly classified with a high confidence level will have almost no influence on the focal loss and the focus will be set on the difficult examples.

3.9 Optimisers

While the loss indicates how well a model performs, an optimiser is required to interpret the value of the loss and to modify the parameters of the classifier in order to reduce the loss.

Gradient Descent Algorithms

In machine learning, gradient descent is used to minimise the loss of a network by iteratively updating the parameters of the network using the back-propagation algorithm.

Gradient descent [10, 52] consists in iteratively following the direction of the steepest descent to find a local minimum of the objective function, which can for instance be the loss of a network. The direction of the steepest descent is obtained by taking the opposite of the gradient of the objective function. The size of the step taken in that direction is defined by the learning rate, which influences the speed and the precision of the optimisation.

The formula of gradient descent is defined as

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{3.16}$$

where θ are the values of the trainable parameters, η is the learning rate and $\nabla_\theta J$ is the gradient of the objective function.

A low learning rate induces smaller changes to the parameters, which implies that more iterations are needed to reach a minimum and that the risk of getting stuck in a local minimum is greater. On the other hand, a high learning rate induces larger changes to the parameters, which implies that fewer iterations are needed to reach a minimum but that the risk of reaching a suboptimal solution very fast is greater. This shows that the choice of the learning rate can strongly influence the quality of the final local minimum.

Different variants of the gradient descent algorithm have been designed for use in machine learning. A first one is Batch Gradient Descent (BGD) [52], which computes the gradient for each training example and uses the mean of the gradients to update the parameters. Having to compute the gradient of every training example at each step takes some time. Thus, the computation time required for a single iteration of BGD increases proportionally with the size of the training set.

Stochastic Gradient Descent (SGD) [10, 52] is another variant of gradient descent which tackles the time issue of BGD. Instead of using the entire training set to update the parameters, SGD computes the gradient for a single training example chosen at random. Each iteration is thus extremely faster and the random behaviour decreases the risk of getting stuck in a poor local minimum. The drawback of SGD is that the update is made based on a single random sample which may not be representative enough of the global trend of the training set.

Mini-Batch Gradient Descent [10, 52] is a trade-off between BGD and SGD. Sometimes, mini-batch gradient descent is referred to as SGD too. At each iteration, it considers a certain number of random training examples, which form a batch. Like in BGD, the mean of the gradients of the batch is used to update the parameters. The batch size needs to be adapted for the problem at hand since the speed of each iteration and the stability of the convergence depend on the batch size. If the batch has a size of one, it comes back to the SGD algorithm. Similarly, if the batch size is equal to the training size, it comes back to the BGD algorithm.

Adaptive Moment Estimation (Adam)

Adaptive Moment Estimation (Adam) [31] is a gradient descent algorithm that combines the strong points of both AdaGrad [22] and RMSProp [57], which are two improved versions of SGD.

Unlike SGD, which has a single learning rate for the whole set of parameters, Adaptive Gradient Algorithm (AdaGrad) keeps an adaptive learning rate per parameter in order to obtain better results in case of problems having sparse gradients. Each learning rate is decreased at each iteration according to all the previous encountered gradients.

The following formula [10] illustrates how parameter i is updated in AdaGrad:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\epsilon + \sum_{\tau=1}^{t}(\nabla_{\theta_{\tau,i}} J(\theta_{\tau,i}))^2}} \nabla_{\theta_{t,i}} J(\theta_{t,i}) \tag{3.17}$$

where
$\theta_{t,i}$ is the value of parameter i at time t.
η is the fixed part of the learning rate.
ϵ is a small value (usually $1{\cdot}10^{-8}$) used to make sure that no division by 0 ever occurs.
$\sum_{\tau=1}^{t}(\nabla J(\theta_{\tau,i}))^2$ is the contribution of the past gradients of parameter i in the reduction of the learning rate.

Root Mean Square Propagation (RMSProp) also keeps a learning rate per parameter, but, unlike AdaGrad, the learning rates are not monotonously decreasing. Since the most recent gradients have more weight in the update of the learning rates than the gradients from older steps, the value of the learning rates can increase and decrease along the iterations. Due to this improvement, RMSProp is known to be well adapted for non-stationary problems.

The following formula [10] illustrates how parameter i is updated in RMSProp:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{\epsilon + v_{t+1,i}}} \nabla_{\theta_{t,i}} J(\theta_{t,i}) \tag{3.18}$$

$$\text{with } v_{t+1,i} = (1-\gamma)(\nabla_{\theta_{t,i}} J(\theta_{t,i}))^2 + \gamma\ v_{t,i} \tag{3.19}$$

where
$\theta_{t,i}$ is the value of parameter i at time t.
η is the fixed part of the learning rate.
ϵ is a small value (usually $1{\cdot}10^{-8}$) used to make sure that no division by 0 ever occurs.
γ is the momentum term that reduces the influence of older gradients along the iterations.
$v_{t,i}$ is the contribution of the past gradients of parameter i in the reduction of the learning rate.

Adam is an improvement of the two gradient descent algorithms presented above. Like AdaGrad and RMSProp, Adam keeps an adaptive learning rate per parameter. One of Adam's differences with RMSProp lies in the fact that it takes into account both the first and the second moments of the gradients. Moreover, it is not the first and second moment estimates that are directly injected in the update of the parameters but their bias-corrections.

The following formula [10, 31] illustrates how parameter i is updated in Adam:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta \cdot \hat{m}_{t+1,i}}{\sqrt{\hat{v}_{t+1,i}} + \epsilon} \tag{3.20}$$

where
$\theta_{t,i}$ is the value of parameter i at time t.
η is the fixed part of the learning rate.
ϵ is a small value (usually $1{\cdot}10^{-8}$) used to make sure that no division by 0 ever occurs.

To update the parameters, it is necessary to compute the biased (3.21) and bias-corrected (3.22) first moment estimates:

$$m_{t+1,i} = (1 - \beta_1)\nabla_{\theta_{t,i}} J(\theta_{t,i}) + \beta_1 m_{t,i} \tag{3.21}$$

$$\hat{m}_{t+1,i} = \frac{m_{t+1,i}}{1 - (\beta_1)^t} \tag{3.22}$$

where β_1 is the exponential decay rate for the first moment estimate and $\nabla_{\theta_{t,i}} J(\theta_{t,i})$ is the gradient of parameter i at time t.

In addition, it is required to compute the biased (3.23) and bias-corrected (3.24) second raw moment estimates:

$$v_{t+1,i} = (1 - \beta_2)(\nabla_{\theta_{t,i}} J(\theta_{t,i}))^2 + \beta_2 v_{t,i} \tag{3.23}$$

$$\hat{v}_{t+1,i} = \frac{v_{t+1,i}}{1 - (\beta_2)^t} \tag{3.24}$$

where β_2 is the exponential decay rate for the second moment estimate and $\nabla_{\theta_{t,i}} J(\theta_{t,i})$ is the gradient of parameter i at time t.

The improvements included in Adam allow it to combine both the advantages of AdaGrad and RMSProp. Indeed, Adam can handle sparse gradients as well as non-stationary problems. Moreover, Adam performs computations efficiently and is thus pretty fast to converge.

3.10 Early Stopping

The goal of early stopping [48] is to avoid overfitting and underfitting during the training of a model and to reduce the training time. Indeed, early stopping interrupts the training when the chosen metric does not improve enough for a certain number of epochs. Moreover, instead of keeping the weights of the last iteration, the best weights found during the whole training are retrieved.

Early stopping can be customised by choosing the value of different parameters. A first parameter is the metric that needs to be monitored. It is possible to indicate whether this metric should be minimised or maximised. Another parameter is the minimum amount by which the metric should vary to be considered as an improvement. A further parameter is the number of consecutive epochs that the system will wait before interrupting the training when no sufficient improvement is observed.

3.11 Architecture and Hyperparameter Tuning

The performance of a classifier varies significantly depending on the values chosen for its hyperparameters. Therefore, it is highly recommended to tune them. Besides the hyperparameters, it is also possible to tune the architecture of certain classifiers such as CNNs.

Common methods to perform tuning include the random and grid search. To use these methods, a set of values needs to be manually specified for each hyperparameter. The random search takes, at each iteration, a random combination of the values made available and stops when the maximum number of iterations is reached. The grid search iterates over the chosen hyperparameter space until all the combinations have been tried out. The random and grid search evaluate the performance of a classifier at each iteration according to a chosen metric, called the objective function. They return the combination of hyperparameter values that optimised the objective function during the search.

The main drawback of these methods lies in the fact that they do not learn from previous iterations and are, therefore, slow to find hyperparameter values that improve significantly the performance. There exist other search algorithms, such as the Tree-Structured Parzen Estimator, that take into account the information of previous evaluations when selecting the next hyperparameter values.

The Tree-Structured Parzen Estimator (TPE) [6] approach is adapted for problems where the objective function is computationally expensive. This is the case of classifiers that first need to undergo training and prediction of class labels. Only then can their performance be evaluated to obtain the value of the objective function. The principle of TPE is thus to use a surrogate function, less expensive than the objective function, to select the next combination of hyperparameters. This surrogate is used to compute the expected improvement of the objective function for the different combinations of hyperparameters. The combination that maximises the expected improvement is selected and the value of its objective function is computed. The obtained value is used during the following iterations to improve the approximation made by the surrogate function.

The TPE allows setting the focus on combinations of hyperparameters that are more likely to improve the performance of the classifier. Hence, this search algorithm usually requires fewer iterations than the random or grid search to find hyperparameter values that improve significantly the performance.

Hyperopt [8] is a Python library, which performs hyperparameter and architecture tuning using, among others, the TPE as a search algorithm. A useful feature of Hyperopt is that it allows to easily interrupt the tuning and restart from where it stopped thanks to the Trials object, which keeps track of all the iterations that have already been completed [7].

Other libraries have been developed to tune hyperparameters and/or architectures. Among others there is AutoKeras [28], Keras Tuner [43], Model Search [50] and MLBox [51]. We decided to use Hyperopt because, unlike some of these libraries, Hyperopt does not only provide a random search algorithm. Indeed, it allows the use of other search algorithms, such as TPE, that consider the previous iterations when choosing the combination of hyperparameters for the next iteration. Moreover, with Hyperopt it is possible to perform both hyperparameter and architecture tuning, while with some other libraries only hyperparameter tuning is possible. Furthermore, Hyperopt is easy to integrate into an already existing implementation and a lot of documentation is available.

3.12 Conclusion

Metrics are important to assess and compare the quality of models. The classical ones, which are accuracy, precision and recall, are presented. To adapt them to multi-class classification, the micro- and macro-averages can be taken.

A technique that can be used to tackle the problem of imbalanced datasets is called the balanced class weight and consists in giving the same importance to each class.

Then, Hard Negative Mining is introduced. It consists in adding the false positives that are more difficult to classify to the training set in order to improve the performance of the model.

When working with audio recordings, the files are decomposed into overlapping windows. Non Maximum Suppression removes certain windows from the predictions so that every object in the file is associated with the window having the locally highest probability.

The first classification algorithm that is introduced is the Support Vector Machine. Its objective is to determine the hyperplane that separates best the data samples using a kernel.

Another classification algorithm is an ensemble method called Gradient Boosting, which often uses decision trees as weak learners. The next tree is trained to predict the residuals of the predictions made by the combination of the previous trees, weighted by a learning rate.

A further algorithm used in our thesis is the Convolutional Neural Network, which processes structured arrays of data that is spatially dependent. It is composed of two parts. The first one is made of convolutional and pooling layers, which are responsible for the feature extraction, while the second part uses dense layers to classify the data based on the features.

The covered losses are the categorical cross-entropy loss, used for multi-class classification, and the focal loss, designed for imbalanced binary classification.

Well-known optimisers, based on gradient descent algorithms, are presented. The three basic ones are Batch Gradient Descent, Stochastic Gradient Descent and Mini-Batch Gradient Descent. A detailed description of Adaptive Moment Estimation, which is an improved version of SGD, is given.

Early stopping is a way to avoid overfitting and underfitting by interrupting the training when the chosen metric does not improve enough for a certain number of epochs.

Finally, the architecture and hyperparameter tuning is addressed. Exploring a subspace of hyperparameter values can be done by different search algorithms, such as the Tree-Structured Parzen Estimator, which are presented in this chapter. A Python library that can be used for tuning is Hyperopt.

Chapter 4

State of the Art

In the previous chapter, we laid out the foundations for the different machine learning techniques that are used throughout this report.

This chapter presents the existing projects on which our work is based. Indeed, our thesis is the continuation of another thesis project that was presented in the academic year 2019-2020 by Maxime Franco and Corrado Lipani. This project, entitled Batmen [25], is itself based on Batdetective [37], an open-source project completed by Mac Aodha et al. in 2018.

The following sections present these two projects by describing their goal, their approach, the machine learning techniques that they use as well as the results that they obtained.

4.1 Batdetective

Batdetective [37] is a project led by Mac Aodha et al. at University College London in 2018. Their objective was to develop an open-source program to detect the positions of bat calls in audio recordings.

In their opinion, too much attention is paid to the classification of bat species and not enough to the detection even though the latter can be very complex in the case of noisy audio samples. With their work, they want to show that it is possible to use deep learning to obtain good performance on audio monitoring. The whole program was built in such a way that only a small fraction of it is specific to bats. Thus, with only a few adaptations, the code could be reused for the audio monitoring of other types of animals. Moreover, with their code, they wanted to fill a void in the domain of open-source tools for bat call detection. Indeed, the existing tools that are reliable are also quite expensive.

Approach

The first step in the detection process is to generate features from the recordings. Batdetective opted for spectrograms. To do so, the audio samples are decomposed into windows of 230 milliseconds having an overlap of 75% with their two neighbours. Then, a Hanning filter is applied on the windows before performing a Fast Fourier Transform to convert them to the frequency domain. The magnitude of the obtained spectrogram is computed and a bandpass filter is applied to only keep the frequency bands between 5kHz and

135kHz. This removes frequencies that are not emitted by bats but that originate from interference noises. To complete the generation of the features, the mean amplitude in each frequency band is subtracted to reduce the impact of background noise and a Gaussian filter is applied to smooth the spectrogram.

These features are fed in a sliding window fashion to a CNN. The latter is composed of three convolutional layers, each followed by a max pooling layer. The network ends with two drop and dense layers where the last dense layer is composed of two nodes corresponding to the "not a call" class and the "call" class. The architecture is illustrated in Figure 4.1 with the parameters associated with each layer.

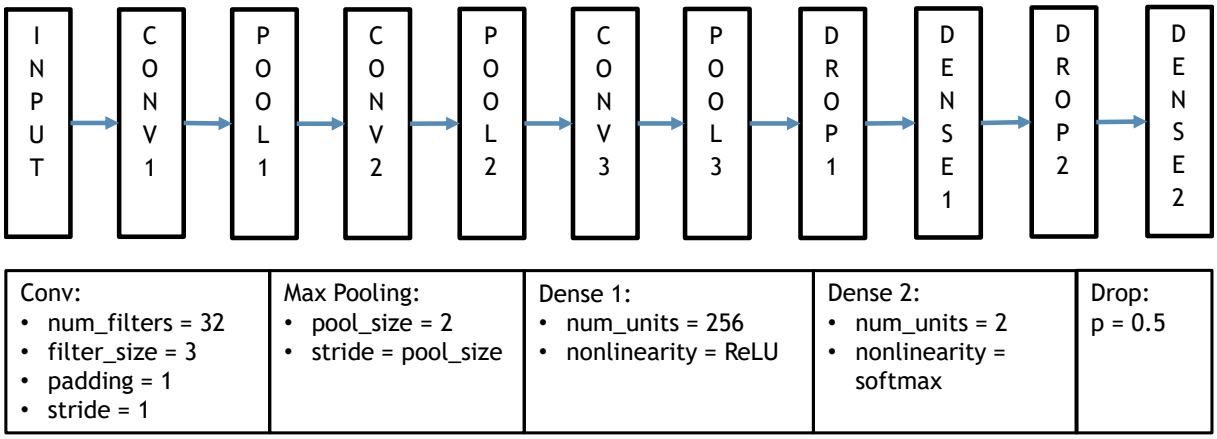


Figure 4.1: Architecture of Batdetective's CNN.

To obtain better performance, two iterations of HNM are applied during the training. At test time, the predictions of the network are smoothed with a Gaussian filter. Then, an NMS algorithm is applied on the smoothed predictions to retrieve precise starting positions of calls by removing the overlapping windows having lower probabilities.

Performance

Batdetective evaluated the performance of its detection model on three different test sets: `Bulgaria`, `UK` and `Norfolk`. These are collections of data from all across Europe, recorded over several years.

For the evaluation of the performance, Batdetective starts by removing the ground truth and predicted positions that are too close to the end of the file. Then, the number of true positives is computed by considering the predicted positions that overlap with a ground truth position. Batdetective considers that two calls overlap if their starting positions are less than 10 milliseconds away from each other. Since more than one predicted position can overlap with a ground truth call, duplicate detections need to be removed. Among the duplicates, only the prediction with the highest probability is kept.

Batdetective expresses the performance of its detection model in terms of the average precision and the recall at 95% of precision. These measurements are obtained from the precision-recall curve as presented in Section 3.1. To build this curve, the detection probability is thresholded from 0 to 1 and, at each step, the precision and recall are recorded. The performance are presented in Table 4.1.

	Bulgaria	UK	Norfolk
Average Precision	0.895	0.866	0.882
Recall at 0.95 precision	0.818	0.670	0.754

Table 4.1: Performance of Batdetective's network on three datasets.

4.2 Batmen

Batmen [25] is a Master's thesis project that was led by Maxime Franco and Corrado Lipani at Université Catholique de Louvain during the academic year 2019-2020. Their objective was to perform multi-class classification of bat calls by starting from the call detection program implemented by Batdetective.

Approach

Before being able to perform multi-class classification, Batmen needed data that is adapted to the problem at hand. Batdetective's datasets contain only information on the starting position of calls and not on the bat groups that emitted these calls. Thus, a new dataset provided by Natagora was used. This dataset still required a lot of preprocessing that has been done by Batmen and is described in Chapter 5.

As in Batdetective, the first step is to generate features from the recordings. Batmen opted for the same approach, which consists in computing a log magnitude spectrogram from the audio recordings, using Fast Fourier Transform on 230 milliseconds Hanning windows having an overlap of 75%. The same bandpass filter, denoising method and Gaussian filter as Batdetective are then applied.

These features are fed in a sliding window fashion to a CNN. As in Batdetective, the latter is composed of three convolutional layers, each followed by a max pooling layer. The network ends with two drop and dense layers where the last dense layer is composed of eight nodes corresponding to the "not a bat" class and to the seven Belgian bat groups.

The difference between Batmen's and Batdetective's CNNs lies in the number of output nodes in the last dense layer. Indeed, Batdetective has two output nodes since it performs detection, while Batmen has eight output nodes since it performs multi-class classification between eight classes. Batmen's architecture is illustrated in Figure 4.2 with the parameters associated to each layer.

To obtain better performance, it is possible to apply HNM and to choose the number of iterations. As in Batdetective, only the false negative examples of class "not a bat" are added to the training set during HNM. At test time, NMS allows retrieving precise starting positions of calls by removing the overlapping windows having lower probabilities.

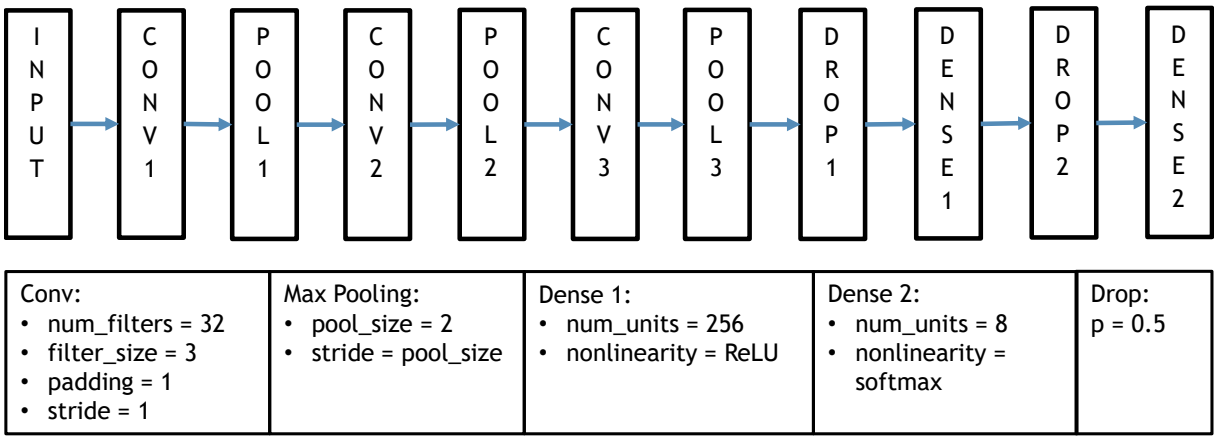


Figure 4.2: Architecture of Batmen's CNN.

Performance

Batmen uses Natagora's dataset while Batdetective uses the Bulgaria, UK and Norfolk datasets. In Batdetective's article, the detection performance is given for the Bulgaria, UK and Norfolk datasets. To compare the detection performance on Natagora's dataset with the detection performance of the three other datasets, Batmen decided to run Batdetective's detection tool on Natagora's dataset. This gives an average precision of 0.873 and a recall at 0.95 precision of 0.625. By comparing these values with the ones obtained for Batdetective's datasets (Table 4.1), it can be noticed that the precision is approximately equal, while the recall at 0.95 precision is worse.

After verifying how Batdetective performs on Natagora's dataset, Batmen evaluated the performance of its classification network. The accuracy, precision and recall were computed on each of the eight classes and the macro-average was taken as a mean of comparison. To assess the impact of HNM on the classifier, the performance was computed for a version with two iterations of HNM and a version without HNM. From Table 4.2, it can be observed that applying two iterations of HNM only improves the performance slightly.

	Average accuracy	Average precision	Average recall
Without HNM	0.9110	0.6930	0.4747
Two HNM iterations	0.9096	0.7177	0.4927

Table 4.2: Performance of Batmen's network.

4.3 Conclusion

Batdetective is a program that detects bat calls in audio files. The detector computes spectrograms from the recordings and feeds them to a CNN ending with two output classes that correspond to the presence and absence of a call. Two iterations of HNM are applied to improve the performance and, at test time, NMS retrieves the precise predicted call positions. Batdetective's algorithm has an average precision of about 88% and a recall at 0.95 precision of approximately 75%.

Batmen is a classification program that detects bat calls and associates them to a bat group. The classifier computes spectrograms from the recordings and feeds them to a CNN ending with eight output classes that correspond to the absence of a call and to the seven Belgian bat groups. Several iterations of HNM can be applied to improve the performance and, at test time, NMS retrieves the precise predicted call positions. Batmen's algorithm has an average accuracy of about 91%, an average precision of about 70% and an average recall of approximately 48%.

Since our main tasks consist in improving Batmen's multi-class classification of bats as well as in implementing an efficient multi-label classifier, our thesis is the continuation of Batmen and thus also an extension of Batdetective.

Chapter 5

Bat Recordings Datasets

Data is an essential component without which detecting and classifying bat calls would not be possible. In machine learning approaches, especially in deep learning, a large amount of data is needed to obtain accurate detectors and classifiers.
Our approach for identifying bats is based on audio recordings of ultrasound calls, which present different patterns depending on the species of bat.

In this chapter, we present the different datasets used for detection and classification throughout our thesis. The chapter describes who provided the datasets, when and where the recordings were made, the content of the datasets as well as the preprocessing of the data. These datasets are used for detection and multi-class classification but are not sufficient for multi-label classification. The last section thus presents the manipulations made to Natagora's dataset to generate new data adapted for multi-label classification.

5.1 Data Provider

This section presents who provided the different datasets used throughout our work.

Batdetective

When detection and classification are performed separately, the data used for the detection comes from the datasets made available by Batdetective [37]. These datasets are composed of recordings taken from 2005 to 2011 by citizen scientists across Europe as part of the Indicator Bats Programme (iBats) [29].

To perform bat call detection using deep learning, the data they collected needed to be annotated with the starting position of each call. Batdetective uploaded all its audio recordings on the Zooniverse [38] web portal so that citizen scientists could analyse and add labels to the files. These labels discern three types of call: social calls, search-phase echolocation and terminal feeding buzzes. Each label is associated with a bounding box placed around the call.

Since classifying bats manually is a difficult art that requires years of experience, few users made accurate annotations. Indeed, Batdetective verified some labels and noticed many errors. However, they observed that their most active user was very accurate which is why they only kept recordings that were annotated by that user. However, in some cases,

the calls were ambiguous and the assistance of chiropterologists was needed. Therefore, Batdetective contacted two experts to disambiguate the calls.

Natagora

Over the years, Natagora collected a large number of bat calls and gathered their analysis of the different recordings in datasets. They shared some of this data with our faculty so that we could conduct experiments to achieve automated recognition of bats.

In the academic year 2019-2020, Maxime Franco and Corrado Lipani, two students from our faculty, worked on the same subject for their thesis [25]. They were in direct contact with Natagora through the intermediary of Claire Brabant who gave them all the additional information they needed to avoid any misinterpretation of the data.

5.2 Data Collection

This section presents when, where and how the bat call recordings were collected.

Batdetective

The data provided by Batdetective is a collection of labelled recordings that were taken in four different regions and are made available on Batdetective's website. The different locations and years of recording are summarised in Table 5.1. This table also contains the total duration and size of the recordings per location. All the recordings were made using the Tranquility Transect recorder except for the Norfolk dataset which was recorded using the Song Meter SM2BAT+ developed by Wildlife Acoustics [2, 37].

Location	Year	Total duration (min)	Total size (MB)
Bulgaria & Romania	2006-2011	212	1070
UK	2005-2011	28	140
Norfolk	2015	33	146

Table 5.1: Location and year of the bat call recordings provided by Batdetective. For each location, the total duration and size of the recordings are indicated.

Most of the data was recorded in Romania and Bulgaria because a large diversity of bat species can be found in these countries. The recordings are representative of the variety of species that can be found in Europe.

Natagora

The data that Natagora provided has been divided into several folders according to the location and year of the recordings. The latter were made using either the Song Meter SM2 or the Song Meter SM4BAT, which are two devices developed by Wildlife Acoustics [2, 25]. The different locations and years of recording are summarised in Table 5.2. This table also contains the total duration and size of the recordings per location.

Location	Year	Total duration (min)	Total size (MB)
Belgian Luxembourg	2016-2018	41	1092
Brussels' ecoduc	2015	1	100
Brussels' Natura 2000 area	2018	0.4	6.2
Caves Pahaut	2013	5	42
Escaut basin	2015	17.5	2048
Fagne-Famenne's Natura 2000 area	2012-2018	207	2166
Gueule valley (Pays de Herve)	2015	22.5	1024
Natura 2000 area	2014	31.7	184
Plateau Engeland (Uccle, Brussels)	2017	17.5	89
Train line 161 (Brussels)	2016-2017	33.5	619
Vesdre's Natura 2000 area	2014	19.5	200
Walloon Branbant's Natura 2000 area	2014	36.5	600
Woluwe Saint Pierre (Brussels)	2019	31.5	400

Table 5.2: Location and year of the bat call recordings provided by Natagora. For each location, the total duration and size of the recordings are indicated.

The recordings containing calls from more than one bat group were left out. Natagora indicated all the groups present in one file but did not associate each call to the corresponding group. Since we do not work with a specialist who could manually classify, we only used the recordings with a single group.

5.3 Data Content

This section presents the content of the datasets used for the detection and classification of bat calls.

Batdetective

From all the recordings collected by Batdetective, only the ones containing search-phase echolocation calls were kept. The labels that are associated with the recordings do not specify the species but only consist of the start times of the bat calls.

After having kept only the search-phase calls, a total of 4246 files are remaining, containing 8573 calls. Batdetective has split the data into one training set and three test sets. The training set is composed exclusively of recordings from Bulgaria and Romania because the majority of the European bat species can be found there. The twenty-three bat species of Belgium can be found in Romania and/or in Bulgaria [39, 47].

One of the test sets contains the remaining recordings coming from Bulgaria and Romania, while the two others contain either data from Norfolk only or from all across the UK. The test set containing bat calls from Bulgaria and Romania allows assessing the performance on bat calls of the same regions as the ones used during training. The Norfolk and UK test sets are there to verify that the detection performs well on bat calls recorded in other regions.

The exact number of files and calls for each training and test set is presented in Table 5.3.

Data set	Number of files	Number of calls
Training set: Bulgaria & Romania	2812	4782
Test set: Bulgaria & Romania	500	1604
Test set: UK	434	842
Test set: Norfolk	500	1345

Table 5.3: Repartition of the files and calls in Batdetective's datasets.

Natagora

Natagora's dataset is composed of several projects that were led by Plecotus, the bat department of Natagora. Each project contains recordings in time expansion with a factor of 10. Moreover, Natagora's dataset provides information such as the exact time and place of recording, the number of calls and the medium frequency. There is also an indication on the group to which the bat calls belong, as predicted by SonoChiro [9], a commercial software for automated bat call analysis. These automated predictions are verified and corrected by experts of the Plecotus team.

To build a classifier, the exact starting position and the group of each call are needed. However, Natagora does not state the positions of the calls but only indicates the different groups present in the recording. The absence of the call positions implies that a call detection needs to be made before being able to perform bat call classification. The files containing more than one group cannot be used since there is no way to link the different calls to their respective group.

To solve this issue, Batmen first trained Batdetective's network on Batdetective's training set and then ran the network on Natagora's dataset to get the starting time of each call in the recordings. The drawback of this approach is that Batdetective does not give perfect predictions. Indeed, it finds fewer calls than what was predicted by SonoChiro and Kaleidoscope [1], which are widely recognised tools in the study of bat calls.

Because of this imprecision in the position labels, the data used for the training and testing of our bat call identification models contains two types of inconsistencies. The first one is that some calls, that are present in the recordings, were not found by Batdetective and are thus labelled as "not a bat" during training and testing. Thus, if the model assigns a bat class to such a call, the model will be misled during training and will get a lower performance even though the model's prediction is actually correct. The second inconsistency is that some noises were considered as calls by Batdetective. This will also penalise the model.

We are aware that these inconsistencies could affect the final performance. However, no other reasonable solution was found to solve the issue of the absence of call positions in Natagora's data.

After having sorted the data and removed the unusable files, a total of 2575 files are remaining, containing 57705 calls. The number of files and calls per group is presented in Table 5.4.

Class	Group	Number of files	Number of calls
1	Barbar	18	678
2	Envsp	588	17462
3	Myosp	1079	21167
4	Pip35	178	4398
5	Pip50	396	8841
6	Plesp	82	1359
7	Rhisp	237	3800

Table 5.4: Repartition of the files and calls among the groups in Natagora's dataset for multi-class classification.

It can be noticed from Table 5.4 that some groups have way more calls than others which means that we are facing an imbalanced dataset. The minority class with the smallest amount of calls is Barbar. It is followed by the Plesp and the Rhisp groups. Envsp and Myosp are the majority classes.

To train and evaluate a model, the data needs to be split into a training and test set. It is common to use 90% of the data for the training and the rest for the testing.
Batmen decided to separate the data according to the number of files available for each group. However, the calls are not uniformly distributed among the files. Thus, we decided to base our repartition on the number of calls to get closer to the wanted percentage. Table 5.5 displays the repartition resulting from the two methods.

Class	Group	Training set		Test set	
		Batmen	New division	Batmen	New division
1	Barbar	625	569	53	109
2	Envsp	15726	15717	1736	1745
3	Myosp	18903	19047	2264	2120
4	Pip35	3863	3916	535	482
5	Pip50	7815	7959	1026	882
6	Plesp	1259	1188	100	171
7	Rhisp	3394	3404	406	396

Table 5.5: Repartition of the calls from Natagora's dataset into a training and test set. In case of Batmen, the repartition of calls is based on the number of files, while our new division is based on the number of calls.

5.4 Data Processing

The first data processing step is to remove all the recordings in Natagora's dataset that contain calls from more than one bat group, as presented in the previous section.

Since recordings do not all have the same sampling rate, the raw audio from Batdetective's and Natagora's dataset cannot be used directly for detection and classification. To deal with the varying sampling rates, Batdetective decided to use spectrograms, which they computed from the raw audio. This technique is applied to all the data used throughout our work.

To compute spectrograms, the audio samples are decomposed into windows. Before converting them to the frequency domain by performing a Fast Fourier Transform, a Hanning filter is applied. Applying a Hanning filter on the windows is important because the Fast Fourier Transform introduces leakage in the case of non-periodic signals, such as bat call recordings.

Leakage [36] means that the energy is scattered over a wider frequency range than the range that it would normally be smeared over in absence of leakage. Another effect of leakage is the reduction of the frequency amplitudes of the signal. To reduce leakage, the starting and ending points of the signal have to be the same, which is why the Hanning filter modifies the signal so that its starts and ends at zero. Moreover, the Hanning filter increases the centre amplitudes of the signal to keep the original average amplitude of the signal.

After the conversion into the frequency domain, the magnitude of the obtained spectrogram is computed and a band-pass filter is applied to only keep the frequency bands between 5kHz and 135kHz. This removes frequencies that are not emitted by bats but originate from interference noises. To complete the preprocessing of the recordings, the mean amplitude in each frequency band is subtracted to reduce the impact of background noise and a Gaussian filter is applied to smooth the spectrogram.

Once the raw audio recordings have been completely preprocessed, the windows containing a call are selected as training positions. The same number of windows that are devoid of calls are randomly selected and added to the training positions.

5.5 Data Superposition for Multi-Label Classification

Performing multi-label classification requires recordings containing superimposed bat calls. However, the files kept from Natagora's dataset do not contain calls from more than one group each. Thus, if two calls overlap in these files, they both belong to the same group, while superpositions of bat calls from different groups are also needed to train a multi-label classification tool.

To have a dataset with overlapping calls from different bat groups, we artificially created recordings using calls from Natagora's dataset. When creating the new recordings, we verified whether it is possible to hear bat calls of any two pairs of bat groups on the same site. We did not find any statistics or probabilities about this. Therefore, we looked into Natagora's folders, which are organised by observation site, whether each pair of groups can be found in at least one site. This analysis has shown that it is indeed the case, and it was then confirmed by Plecotus.

Moreover, we decided to select the calls in such a way that the distribution of calls between groups is representative of the minority and majority groups observed in Natagora's dataset. The respect of the minority and majority repartition of groups was achieved by repeatedly taking two recordings from Natagora's dataset at random and picking a call randomly from each of the two recordings. These two calls are overlapped with a random shift between the starting position of the two calls. The duration of this shift ranges between zero and the size of a window. The shift is not fixed but is chosen randomly so that the artificial overlaps are more representative of real ones. Moreover, the maximum duration of the shift was defined to ensure that there is an overlap between the calls.

For multi-label classification, not only superimposed bat calls are needed but also some recordings with no superposition and some others with no call at all. Thus, we decided to keep half of the recordings that are used for multi-class, which results in the repartition presented in Table 5.6. The other half of the multi-label dataset is composed of the generated recordings containing superposition. Table 5.7 presents the number of superpositions that were made between each pair of groups. When combining the calls from Natagora with the ones from the generated recordings, a total of 85990 calls are used. The training set is composed of 77143 calls and the test set of 8847 calls. The exact repartition of calls among the different groups can be found in Table 5.8.

Class	Group	Number of files	Number of calls
1	Barbar	8	393
2	Envsp	291	8371
3	Myosp	592	11040
4	Pip35	91	2189
5	Pip50	196	3977
6	Plesp	37	842
7	Rhisp	116	1936

Table 5.6: Repartition of the files and calls among the groups in Natagora's dataset for multi-label classification.

	Barbar	Envsp	Myosp	Pip35	Pip50	Plesp	Rhisp
Barbar	20	71	159	35	55	11	29
Envsp	71	1028	4765	847	1777	377	1062
Myosp	159	4765	5546	1967	3904	827	2500
Pip35	35	847	1967	180	691	161	423
Pip50	55	1777	3904	691	757	228	902
Plesp	11	377	827	161	288	35	189
Rhisp	29	1062	2500	423	902	189	265

Table 5.7: Number of call superpositions generated for each pair of groups to perform multi-label classification.

Class	Group	Number of calls for training	Number of calls for testing
1	Barbar	663	130
2	Envsp	17381	1945
3	Myosp	32619	3635
4	Pip35	5446	727
5	Pip50	11793	1315
6	Plesp	2474	291
7	Rhisp	6767	804

Table 5.8: Repartition of the calls for multi-label classification into a training and test set. Half of the calls comes from Natagora's dataset and the other half comes from the generated recordings that contain calls superposition.

5.6 Conclusion

The dataset used for the detection is made available by Batdetective and is a collection of recordings made between 2005 and 2011 in Bulgaria, Romania, Norfolk and the UK. From these recordings, only those containing search-phase calls have been kept, which gives a total of 8573 calls. The recordings have been labelled by a citizen scientist and were verified by an expert. The labels only consist of the starting position of the call and do not specify the corresponding species. The data has been split into a training set and three test sets. The training set only contains calls from Romania and Bulgaria because the species that can be found there are representative of the species that can be found all over Europe. One test set contains the remaining data from Romania and Bulgaria, another one is composed of data from Norfolk and the last test set is formed by data from the UK.

For the classification, we use a dataset created by Plecotus, the bat department of Natagora. It contains recordings that were made during their observations across Belgium from 2012 to 2019. Natagora uses SonoChiro to produce the labels, which are then verified by experts of Plecotus. For a given file, the labels indicate only the bat groups that are present but not the starting position of the calls, which are however required to perform bat call classification. Therefore, only the recordings containing a single group are kept, and the starting positions are obtained by running Batdetective's code on these files. Since the timings of the calls are not specified by experts but instead are found with a code that is not 100% accurate, the labels used to train our models contain some errors that may impact the performance of our architectures. The final dataset for the multi-class classification contains 57705 calls with the fewest calls by far for the Barbar group and the most calls for the Myosp group.

Since the sampling rate is not the same for all audio recordings, the raw audio cannot be directly used. Thus, some preprocessing is applied to the detection and classification datasets. The raw audio is decomposed into windows on which a Hanning filter and a Fast Fourier Transform are applied to convert the signal to the frequency domain. Then, a band-pass filter removes the too low and too high frequencies. The last preprocessing steps are the denoising and the application of a Gaussian filter.

Finally, a dataset containing recordings of superpositions of calls is needed to perform multi-label classification. These recordings cannot be taken from Natagora's dataset since,

for recordings with more than one group, there is not enough information to associate each call with its corresponding group. Therefore, new recordings containing superpositions of calls between pair of groups are created. The final dataset used for multi-label classification contains half of Natagora's recordings as well as the newly created recordings. This gives a total of 85990 calls.

Chapter 6

Architectures for Bat Call Detection and Classification

In the previous chapters, the only architecture that was addressed is the single CNN. In the case of Batdetective, it performs solely detection and for Batmen it is used for both detection and classification.

Since the architecture that performs best for a given classification problem is not predefined, the choice of the architecture is essential as it will greatly impact the performance. Therefore, we decided to implement several multi-class and multi-label architectures with various characteristics. This chapter presents all of them, while later chapters will compare them to determine which one performs best for our problem.

6.1 Double CNN Architecture

In Batmen, the classification and detection tasks are tackled by a single CNN. Since Batmen has the same architecture and parameters as Batdetective, except for the last layer that has eight nodes instead of two, we studied whether the performance could be improved if detection and classification were performed separately. Therefore, we decided to implement an architecture that we call the double CNN architecture. The principle behind the double CNN architecture is the separation of the detection and classification tasks into two networks.

As in the previously presented models, the features used are spectrograms that are fed in a sliding window fashion to the two CNNs. The first CNN detects whether or not a bat call is present in a window. If this is the case, the window is fed to a second CNN, which assigns one of the seven bat groups to the call. Like in Batdetective and Batmen, the two CNNs alternate between convolutional and max pooling layers and end with drop and dense layers. However, the number of each type of layer and their specific hyperparameters are optimised using Hyperopt.

The two CNNs are trained separately so that the different hyperparameters can be optimised for their respective task, as illustrated in Figure 6.1.

For the detection, we use Batdetective's dataset instead of Natagora's. Indeed, Natagora's dataset was initially exclusively labelled with the bat group present in each file and not

with the starting position of the calls. To solve this issue, Batmen used Batdetective's algorithm to add the starting positions to the dataset. Thus, using Natagora's dataset would not be appropriate to train our detection CNN and could bias the results since Natagora's dataset was labelled using a similar detection CNN. This is why we opted for Batdetective's dataset, for which the starting positions were labelled by hand.

For the classification task, only Natagora's dataset can be used considering that the groups are not indicated in Batdetective's dataset. Only the windows containing a call are used to train the classification CNN since it solely has to predict the groups and not whether the windows contain a call or not.

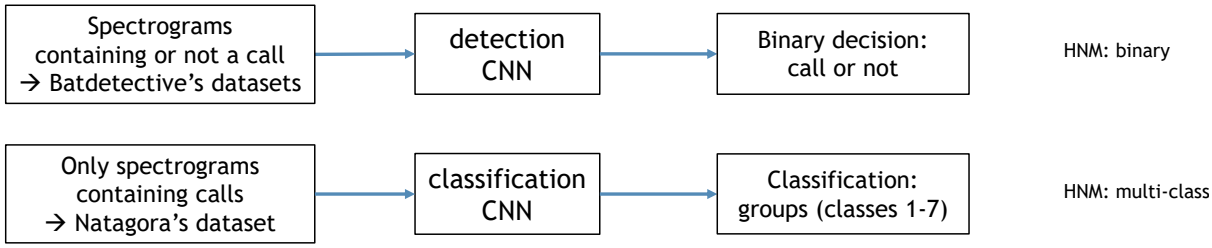


Figure 6.1: Double CNN architecture at train time.

At test time, all the windows are fed to the detection CNN and those for which a call is predicted are forwarded to the classification network. Because of the classification part, only Natagora's dataset is usable. The testing is illustrated in Figure 6.2.

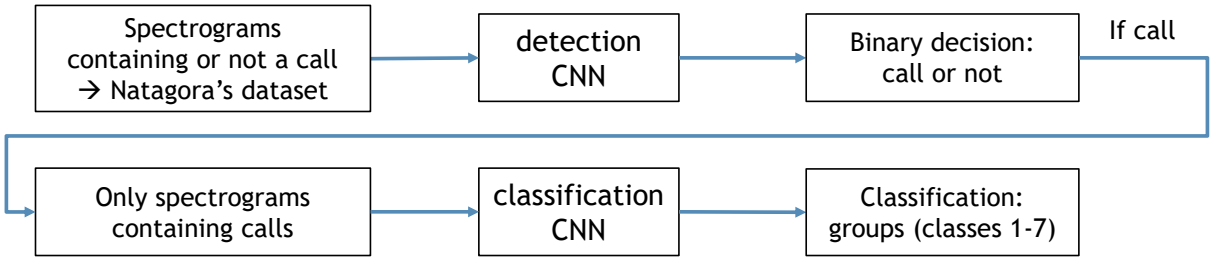


Figure 6.2: Double CNN architecture at test time.

6.2 Hybrid Models Using CNN Features

Besides experimenting with different types of architectures, we also wanted to try out different kinds of features and not restrict ourselves to spectrograms. A way to obtain other features than spectrograms is analysed by Neha Singh in her thesis [53] on the classification of animal calls. She compares the performance of a CNN using spectrograms with the ones obtained by hybrid models that use features computed by a CNN, which itself receives spectrogram features as input. Such hybrid models could be adapted to the classification of bat groups and are presented in this section.

This architecture is denoted as hybrid because it is composed of two models, the first one being a CNN and the second one being a model other than a neural network. This second model can for instance be an SVM or an XGBoost and receives as input features

the output of a layer of the CNN.

The architecture that results from the application of such a hybrid model to our problem is presented in Figure 6.3. As in Batmen's architecture, the CNN performs both detection and classification and receives spectrograms as input. The second model also performs detection and classification but does not use a spectrogram as input features. Instead, it takes features computed by the CNN. These features correspond to the output of the second to last dense layer of the CNN.

The CNN is trained on spectrograms and maps them to one of the eight classes. Once it is fully trained, it is used to compute the features given to the second model so that it, in turn, can train.

At test time, the spectrograms are fed to the CNN which returns predictions. However, these predictions are not taken into account since only the output of its second to last layer matters to provide features to the second model. The latter's predictions are used to compute the performance.

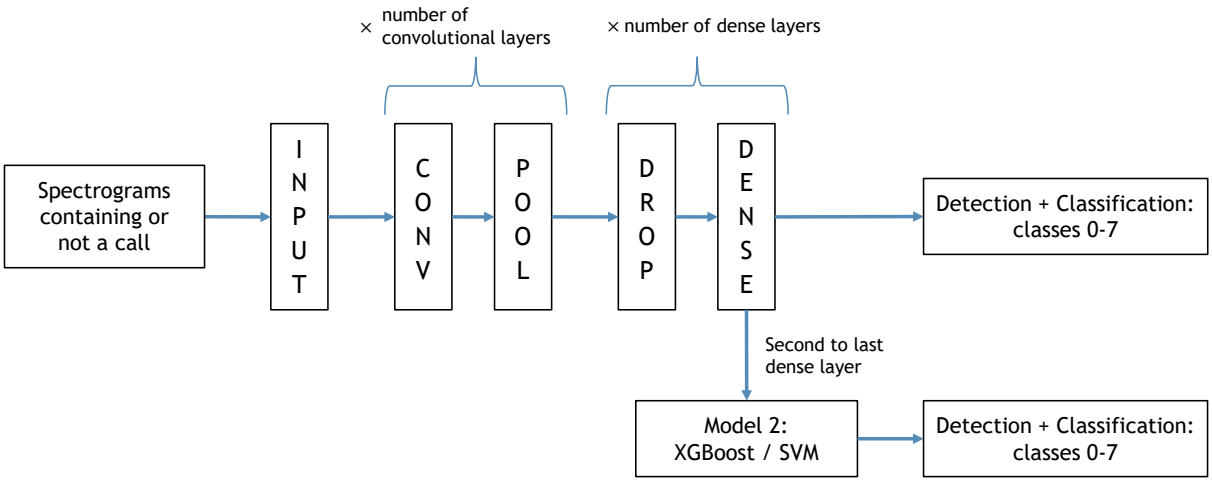


Figure 6.3: Hybrid model using CNN features. The number of convolutional and dense layers is tuned. These values differ for multi-class and multi-label classification and can be found in their respective experiment section.

6.3 Hybrid Models Using Call Features

Taking the principle of hybrid models, described in the previous section, as a starting point, we decided to replace the CNN features with new features related to bat calls.

Bat Call Features

Bat calls have different characteristics depending on the species of bat that emits the call. Among the large variety of possible characteristics, we selected 18 to use as features. Some of them are inspired from the features used by BatScope [42], an application for bat call processing.

Our selection is composed of the following features:

- The mean, central, minimal and maximal frequencies.
- The centred bandwidth containing 90% (resp. 50%) of the total energy such that 5% (resp. 25%) of the total energy is contained in the frequencies lower than the band and the other 5% (resp. 25%) is contained in the frequencies higher than the band.
- The frequencies at which 5%, 25%, 50%, 75% and 90% of the total energy of the call is reached when starting at a frequency of 0 Hz.
- The peak frequency and its associated magnitude.
- The minimal and central magnitudes as well as their associated frequency.
- The mean magnitude.

Architecture

As in the previously presented hybrid model, this model is separated into two parts. The first one is a CNN and the second one is a model other than a neural network, such as an XGBoost or an SVM. In this case, however, the CNN is used solely for the detection and not to produce features for the second model. Instead, the latter receives new features based on bat call characteristics to perform the classification.

The two models are trained separately so that the different hyperparameters can be optimised for their respective task, as illustrated in Figure 6.4.

For the detection, we use Batdetective's dataset instead of Natagora's for the same reasons as described in Section 6.1. Moreover, since the first model is a CNN, spectrogram features are used, as it was the case in any of our other architectures.

For the classification task, only Natagora's dataset can be used considering that the groups are not indicated in Batdetective's dataset. Only the spectrogram windows containing a call are kept to compute the call features that are used to train the classification model.

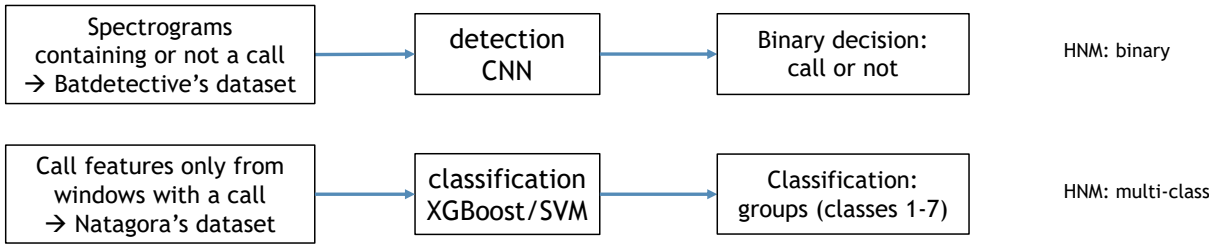


Figure 6.4: Hybrid model using call features at train time.

At test time, all the windows are fed to the detection CNN and those for which a call is predicted are forwarded to the classification network. Because of the classification part, only Natagora's dataset is usable. The testing is illustrated in Figure 6.5.

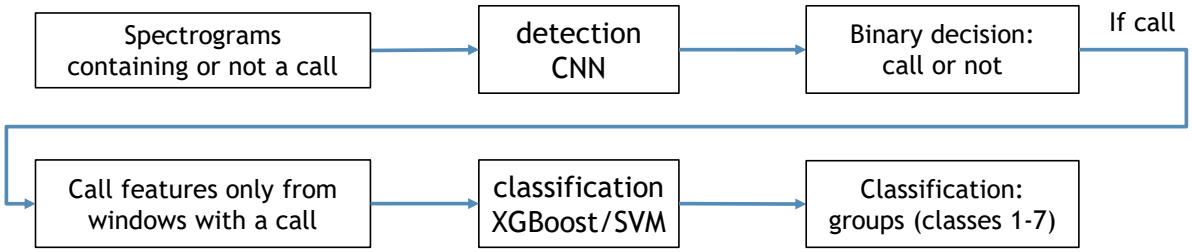


Figure 6.5: Hybrid model using call features at test time.

6.4 Conclusion

In addition to the single CNN model introduced by Batmen, we implemented an architecture that we call the double CNN architecture. Its name comes from the fact that it is composed of two CNNs, one for the detection and the other for the classification. Both use spectrograms as input, as is the case for the single CNN.

To experiment with features other than spectrograms, we implemented hybrid models which are composed of two parts. The first one is still a CNN which receives spectrograms as input. The second model is not a CNN but is for instance a Support Vector Machine (SVM) or an Extreme Gradient Boosting (XGBoost). In this case, the ultimate goal of the CNN is neither detection nor classification. Indeed, the output of the second to last layer is fed to the second model as input features. It is thus the second model that does the detection and the classification.

Another type of features that can be used for classification are features related to bat calls, like the central or maximal frequency. A large variety of such characteristics exists and we selected 18 of them. These features are given as input to a model that is not a CNN but is for instance an SVM or an XGBoost. This model performs solely classification. There is also a CNN that receives spectrograms as input and performs the detection. These two models form what we call a hybrid model using call features.

Chapter 7

Multi-Class Classification

This chapter presents all the design choices and experiments that we did for multi-class classification.

The first two sections present the modifications that have been brought to Batmen's classification and evaluation processes. The following sections contain a description and an analysis of the various experiments that were conducted on the architectures introduced in the previous chapter. Then, an overall comparison of the architectures is made and the time needed for each model to classify new data is presented.

All the measurements of this work have been made on a desktop with an i7-9800X CPU @ 3.80GHz, 32 GB RAM and an RTX 2080 SUPER GPU on CentOS 8.1.

7.1 Improvements for Multi-Class Classification

To implement the architectures presented in Chapter 6, we decided to start from Batmen's code that we reimplemented and improved. The different subsections present the modifications that we did to improve the performance of Batmen's architecture and to correct some aspects that were not adapted to the multi-class problem. These changes are applied to all the multi-class architectures used throughout this work.

Environment

Batmen's code is written in Python 2.7 and uses the Theano and Lasagne libraries to build a CNN for the classification. However, Python 2.7 is deprecated since January 2020 and will thus not be fixed nor changed anymore. Therefore, we translated the whole Batmen code into Python 3.6 to start from a code that is up to date. We also decided to replace Theano and Lasagne with TensorFlow and Keras [20].

Multi-Class Non Maximum Suppression

The implementation of NMS in Batmen is similar to the one used for the binary detection in Batdetective. The classifier associates each window to a vector containing a probability for each of the eight classes. The highest probability of each window is then kept, without taking into account the probability of class 0. A Gaussian filter is applied to smooth

these highest probabilities over time. For every window, NMS receives the filtered highest probability as well as the corresponding class. It keeps the positions and the related classes corresponding to the different local maximums of probability.

The problem with Batmen's approach is that it keeps some positions where class 0 is actually the most likely one, while the highest probability among the probabilities of classes 1 to 7 is extremely low. This situation is depicted by the red frame in Figure 7.1.

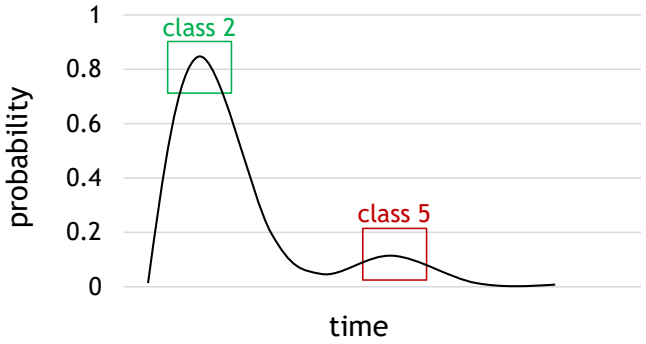


Figure 7.1: Graph of the probabilities over time. The frames are the positions selected by Batmen's NMS. The green frame corresponds to a position for which class 0 does not have the highest probability since the probability of class 2 is already 85%. The red frame corresponds to a position for which class 0 has the highest probability. Indeed, class 5 has the highest probability among classes 1 to 7, while only having a probability of 10%. Batmen's implementation keeps both positions while ours removes the red one.

To solve the issue of low probability peaks, we decided to remove the positions selected by NMS where the probability of belonging to class 0 is actually higher than the probability of the selected class.

Multi-Class Hard Negative Mining

To perform HNM, Batmen only considers the examples that are classified as any of the seven groups of bats (classes 1 to 7) but are actually not corresponding to a call (class 0). These false negatives of class 0 are added to the training set. Taking only this type of misclassification into consideration is analogous to a binary case where the seven bat groups correspond to the positive class and the absence of call corresponds to the negative class.

Since we are in a multi-class context, the misclassifications between bat groups as well as the false positives of class 0 are important examples that help improve the performance. The misclassification between bat groups covers the predicted positions that match with ground truth positions but are not of the correct class. The false positives of class 0 are the ground truth positions that are not overlapped by any prediction.

To take those examples into account, the problematic positions cannot simply be added as it was done in binary HNM. Indeed, the additional misclassifications correspond to call positions that are already in the training set. Adding the exact same positions would create

duplicate entries which would not bring any new relevant information for the classification.

Therefore, we decided to add the positions of the calls that were missed or misclassified but shifted left and right by a small amount of time, defined by Batdetective. This allows to represent the difficult calls in slightly different windows and to avoid duplication. If the shift is too large, the new positions are too far away from the original one to still contain enough information about the call. If it is too small, the variations are too close to the original position to have an impact.

Architecture and Hyperparameter Tuning with Hyperopt

Batmen keeps the same values as Batdetective for the different hyperparameters of the network as well as the same architecture. However, these were fine-tuned for the binary classification task and may not be optimal for multi-class classification. Thus, we implemented a hyperparameter and architecture tuning using Hyperopt [7, 8].

For CNNs, the objective function that is optimised is the sparse categorical cross-entropy loss. The tuned hyperparameters and architecture components of the CNNs are the number of convolutional layers, the number and size of the convolutional filters, the number of dense layers and their associated number of nodes, the dropout probability of the dropout layers as well as the batch size. Since our CNNs use Adam as an optimiser, the learning rate, the β_1, the β_2 and the ϵ hyperparameters are tuned. For early stopping, the tuned hyperparameters are the minimum delta, which is the minimum improvement of the loss, as well as the patience, which is the number of iterations after which the training is interrupted if the minimum improvement is not reached.

For XGBoost, the objective function that is optimised is the cross-entropy loss. The tuned hyperparameters are the number of estimators, the maximum depth of each estimator, the learning rate, gamma, the minimum child weight, the subsample ratio and the scale positive weight.

For SVMs, the objective function that is optimised is the cross-entropy loss. The tuned hyperparameters are the kernel, the degree in case of a polynomial kernel, the regularisation parameter C and the maximum number of iterations. Whether or not to use balanced class weights is also part of the tuning. Moreover, the value of the kernel coefficient gamma, which is used in the case of RBF, polynomial and sigmoid kernels, is tuned.

All the models presented in Chapter 6 were tuned and the performance presented in the following experiment sections are obtained with the tuned architectures and hyperparameters. Appendix A.1 presents the hyperparameter values of the different models obtained after tuning.

Further Improvements

In addition to all the previous changes made to Batmen's project, we modified some other aspects of the training. A first one concerns the optimiser. We decided to use Adam, which is an improvement of the initially used SGD with mini-batches.

To avoid underfitting and overfitting, we included early stopping to the training. This reduces the total training time since, instead of having to train for a fixed number of epochs, early stopping can interrupt the training earlier when not enough improvement is observed on the validation set anymore.

Another modification that considerably reduces the total running time is the saving of the features. Indeed, during one run of the program, the features are computed once during the training, once during the testing and twice at each iteration of HNM. Since these features remain the same during the whole run, we decided to save them to make sure that they are not computed more than once. The features still need to be loaded from memory but this takes much less time than computing them.

Since Batmen's network uses the sparse categorical cross-entropy loss, which is adapted for multi-class classification but not for imbalanced datasets, we decided to try out two alternatives to tackle the problem of imbalanced classes. The first one is the use of the focal loss, which is designed specially for imbalanced cases. The other possibility is the combination of the sparse categorical cross-entropy loss with class weights, called the balanced categorical cross-entropy loss. We decided to keep the latter since it gives better results for our task.

Finally, Batmen's model solely works with recordings that contain bat calls from a single group per recording. This is constraining because recordings will often contain calls from more than one group. Thus, we modified Batmen's model so that it can process recordings having calls from any number of groups, as long as they do not overlap since that would fall into the multi-label classification.

7.2 Evaluation for Multi-Class Classification

The following subsections present the modifications brought to Batmen's evaluation method and to the way the confusion matrix is filled. In addition, the metrics used to evaluate the performance are presented.

Multi-Class Confusion Matrix

Batmen's confusion matrix is composed of classes 1 to 7, which represent the seven bat groups that are studied. For each file, Batmen's evaluation iterates over the predicted calls and their associated predicted class to fill the confusion matrix. The element `(actual class, predicted class)`, which represents the entry in the row of the actual class and the column of the predicted class in the confusion matrix, is incremented by one. Note that, in Batmen, the actual class is associated with the file as a whole and not to a specific call position, as explained in Section 5.2.

Batmen's version of the confusion matrix is not complete since it does not consider the predictions related to class 0 nor the overlap between the actual and the predicted call positions to build the matrix. Among all the cases illustrated in Figure 7.2, only those where the predicted call overlaps with the ground truth call are handled correctly. The predicted class associated with the predicted call can either be the same as the ground truth class or be different from it. Both cases are well accounted in Batmen's confusion

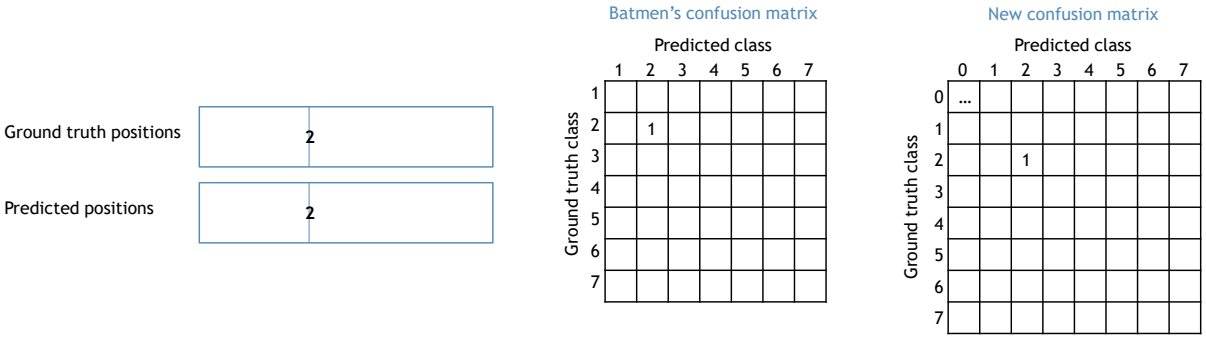
matrix as illustrated in Figures 7.2.a and 7.2.b.

However, some cases are not taken into account in Batmen's confusion matrix. We integrated these missing cases when building our own confusion matrix.

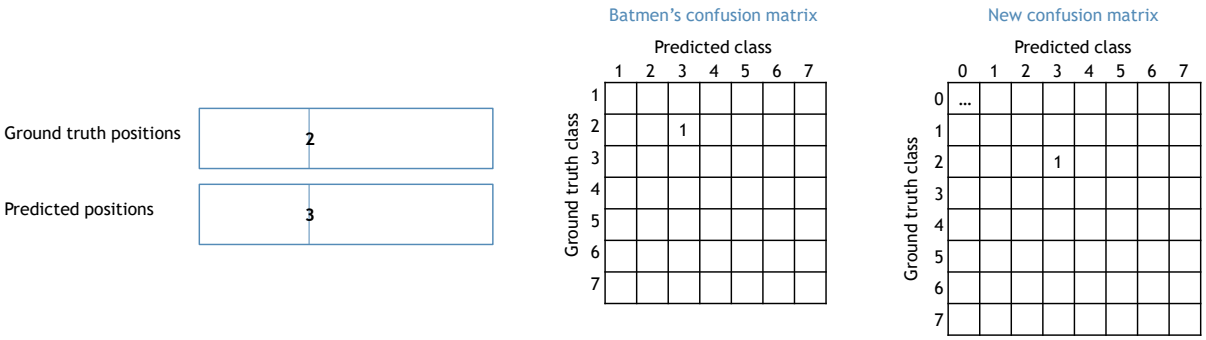
The first case, presented in Figure 7.2.c, is when no calls are predicted but there are some ground truth positions in the recording. No penalty is given in Batmen's confusion matrix, while we increase the entry corresponding to `(ground truth class, class 0)` by the same amount as the number of ground truth positions since they were all missed.

The second case, illustrated in Figure 7.2.d, is when a call is predicted while there is no call overlapping with the predicted position. Batmen does not check whether or not the position of the predicted call and the ground truth position overlap. Batmen simply adds 1 to the entry `(ground truth class, predicted class)` even though the predicted position is not the position of a ground truth call. Instead of doing this, we increment the entry corresponding to `(class 0, predicted class)` by one.

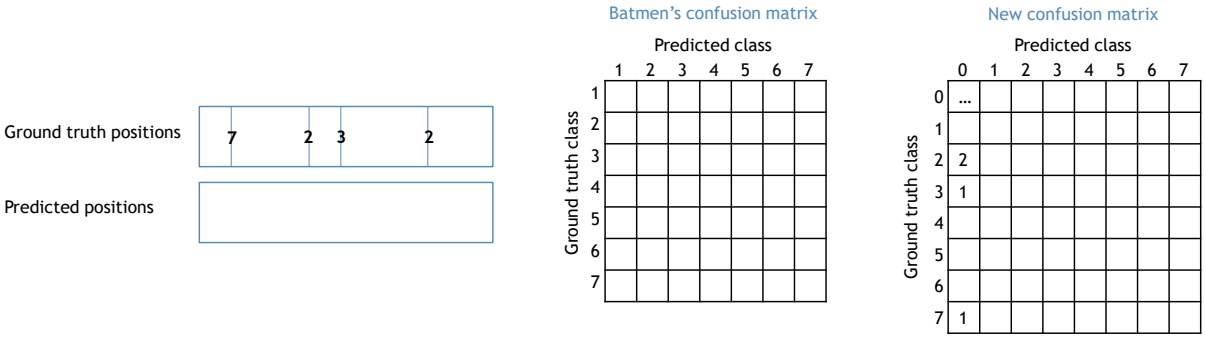
The last case, presented in Figure 7.2.e, is when there are ground truth positions that are not overlapped by any predicted calls. No penalty is given in Batmen's confusion matrix, while we increment the entry corresponding to `(ground truth class, class 0)` by one.



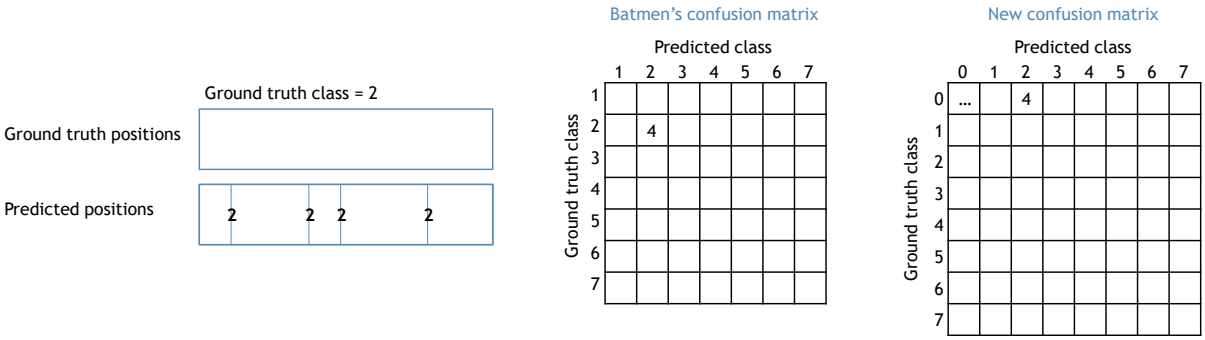
7.2.a. First well handled case: overlap and correct predicted class. The empty entries contain the value 0. The three dots in the `(0,0)` entry represents the fact that there is a large amount of correctly classified class 0 (i.e. no call).



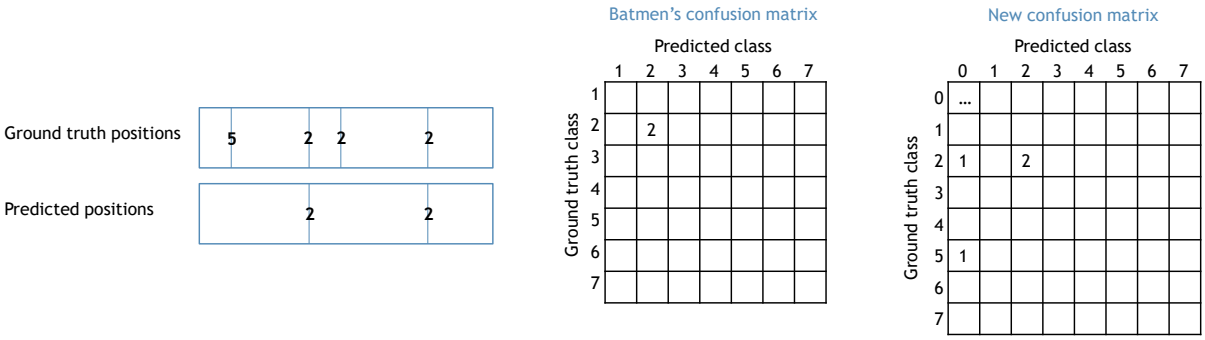
7.2.b. Second well handled case: overlap and wrongly predicted class. The empty entries contain the value 0. The three dots in the `(0,0)` entry represents the fact that there is a large amount of correctly classified class 0 (i.e. no call).



7.2.c. First incorrect case: no call predicted while there are some calls. The empty entries contain the value 0. The three dots in the `(0,0)` entry represents the fact that there is a large amount of correctly classified class 0 (i.e. no call).



7.2.d. Second incorrect case: call predicted where there is no ground truth call. The empty entries contain the value 0. The three dots in the `(0,0)` entry represents the fact that there is a large amount of correctly classified class 0 (i.e. no call).



7.2.e. Third incorrect case: some ground truth positions are not found by the predictions. The empty entries contain the value 0. The three dots in the `(0,0)` entry represents the fact that there is a large amount of correctly classified class 0 (i.e. no call).

Figure 7.2: Comparison between Batmen's confusion matrix and our improved confusion matrix on different scenarios.

In addition to these corrections, we made some design choices. It can happen that one predicted position overlaps with several ground truth positions or that one ground truth position overlaps with several predicted positions. Therefore, when the ground truth position is overlapped by several predicted positions of the correct class, we decided to increment the entry `(ground truth class, predicted class)` only once. When the ground truth position is overlapped by several predicted positions of the incorrect class,

we decided to increment the entry `(ground truth class, predicted class)` as many times as the number of incorrect predictions.

Metrics

To evaluate the performance of a model, Batmen works with accuracy, precision and recall. They are first computed for each class and then the macro-average is taken to perform the comparison.

Since the problem at hand is an imbalanced multi-class problem, we added the following metrics to Batmen's evaluation metrics: the F_1 score, the balanced classification rate, the Kappa score and the confusion entropy. The first two metrics are computed for each class and the macro-average is then taken to perform the comparison. This kind of adaptation is not necessary for the Kappa score and the confusion entropy because they are specially designed for the multi-class context.

The macro-average was chosen over the micro-average because, in our problem, the minority classes are as important as the majority classes. Indeed, even though certain groups of bats appear less often than others, it is still important to have a model that is able to identify them in a recording.

The metric that influences our comparison the most is the recall. A high recall indicates that a large proportion of the actual calls were found by the model. Due to the precision-recall trade-off, increasing the recall can lead to a decrease in the precision. However, in our opinion, in the case of bat call identification, it is more important to reach a higher recall than to have a higher precision. Indeed, when chiropterologists need to validate the predictions made by a classifier, it is easier for them to discard or change invalid predictions than to go through the whole recording to find the calls that were missed by the model.

In some situations, it is interesting to compare the model not only on the global task, which regroups detection and classification, but also on these two subtasks individually. For detection, the same metrics are applied but on two classes only. Class 0 represents the absence of a call and class 1 represents the presence of a call of any group of bats. The classification task is evaluated in the same way as the global task except that class 0 is not taken into account.

When the results of our experiments are presented, we only indicate the accuracy, precision and recall. All our metrics are given in Appendix B in order to not overload the reader with numerical information.

7.3 Experiments on the Original Batmen Architecture

As mentioned in the previous section, the evaluation method used by Batmen contains a lot of inconsistencies. Therefore, we would like to find out what performance the original Batmen architecture can achieve with our evaluation method. Another question is

whether or not HNM can improve the performance of this architecture.

Before diving into these questions, we reproduced Batmen's results to make sure that we obtain a similar performance even after changing Batmen's environment. As stated in Batmen's paper, reproducing the results without HNM gives an average precision of about 69% and an average recall of about 48%.

After that, we computed the performance of Batmen's architecture using our new metrics and confusion matrix. This allows comparing our improved model with the original Batmen network based on the same evaluation method. The performance was computed without HNM as well as with one and two iterations of HNM and is presented in Table 7.1.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.25	0.94	0.86	0.76	0.36	0.75
1	0.25	0.94	0.86	0.76	0.40	0.75
2	0.25	0.94	0.86	0.75	0.33	0.74

Table 7.1: Influence of Hard Negative Mining on the precision and recall of the original Batmen architecture. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

From the precision and recall of the global performance in Table 7.1, it can be drawn that one iteration of HNM should be used for this model. A surprising observation is that, even though the detection and classification performance are the same with none or one iteration of HNM, the global precisions are different.

This comes from the fact that having the same percentage of precision and recall in detection does not mean that the same bat calls have been found by both models. If a model detects fewer calls from a minority class and instead more calls from a majority class than the other model, the performance for the minority class will drop more than the increase observed in the majority class. Since the global performance is computed as the macro-average over all classes, the minority classes have as much impact as the majority classes. The model using one HNM iteration has thus a higher performance for minority classes and a slightly lower performance for some majority classes than the model without HNM. Subsequently, when the performance of the original Batmen architecture comes up, it refers to the performance of this architecture using one iteration of HNM.

The performance of the original Batmen is presented in Table 7.2 and the hyperparameters that are used are listed in Table A.1.

It can be noticed in Table 7.2 that the recall is quite high for the detection, the classification and the global task, while the precision is very low for the detection and the global task. The opposite is observed when Batmen's confusion matrix is used to compute the performance. Indeed, in that case, the precision is high and the recall is lower, as shown in Table 4.2. This is mainly because Batmen's confusion matrix does not take the overlap between ground truth and predicted calls into account. It only checks whether

the predicted class is the same as the ground truth class of the file without verifying the timings.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.25	0.94	/
Classify	0.96	0.86	0.76	/
Global	0.99	0.40	0.75	117min

Table 7.2: Performance of the original Batmen architecture with our metrics. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

7.4 Experiments on the Improved Batmen Architecture

In this section, we try to determine whether the adaptations brought to the original Batmen and presented in Section 7.1 have a positive impact on the performance. Another question is whether or not HNM can improve the performance of this architecture. Finally, this section determines whether saving the features brings a gain of time that justifies the memory space needed to save the features.

After applying all our improvements to the original Batmen, we computed the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 7.3.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.68	0.92	0.88	0.76	0.74	0.74
1	0.78	0.90	0.86	0.77	0.76	0.74
2	0.69	0.92	0.85	0.76	0.71	0.74

Table 7.3: Influence of Hard Negative Mining on the precision and recall of the improved Batmen architecture. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

It can be noticed in Table 7.3 that doing one iteration of HNM increases a lot the precision of the detection while the recall is only slightly decreased. Since the recall is very high, it enables to bring more balance between the precision and the recall. Because of this considerable increase in the precision of the detection, the global precision increases too. Thus, when the performance of the improved Batmen architecture comes up, it refers to the performance of this architecture using one iteration of HNM.

The performance of the improved Batmen architecture is presented in Table 7.4. The associated tuned hyperparameters are listed in Table A.2.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.78	0.90	/
Classify	0.96	0.86	0.77	/
Global	0.99	0.76	0.74	87min

Table 7.4: Performance of the improved version of Batmen's network. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

It can be observed from Table 7.4 that the precision and the recall are very balanced and high for the global performance.

When comparing the performance of the original Batmen, computed with our metrics, with the performance of the improved version of Batmen's code, it can be noticed that the recall is approximately the same. However, the detection precision is way better in the case of the improved Batmen. This is mostly because improved Batmen uses the multi-class NMS that we implemented, while original Batmen uses a version of NMS that was designed for binary detection. Indeed, the multi-class NMS gets rid of windows whose highest assigned probability is the one of not being a bat call, as explained in Section 7.1.

Another improvement lies in the total training time, which is much faster than for the original Batmen's code. This difference in speed comes from the fact that we introduced the saving of the training features and that we added early stopping to the training.

7.5 Experiments on the Double CNN Architecture

In this section, we would like to find out whether separating the detection and classification tasks instead of having a network that performs both can improve the performance. As for the two previous models, we want to observe whether or not HNM can improve the performance of this architecture. Another point of interrogation is whether the tuning of the detection network helped find better hyperparameters than those used by Batdetective.

After having decomposed the algorithm into a detection network and a classification network, we computed the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 7.5.

It can be noticed in Table 7.5 that the precision globally increases while the recall decreases. Since the recall is already lower than the precision without HNM, this gap increases when HNM is applied. Therefore, when the performance of the double CNN architecture comes up, it refers to the performance of this architecture without using HNM.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.85	0.84	0.85	0.73	0.77	0.68
1	0.82	0.81	0.88	0.70	0.79	0.63
2	0.91	0.76	0.87	0.70	0.82	0.61

Table 7.5: Influence of Hard Negative Mining on the precision and recall of the double CNN architecture. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

The double CNN's performance is presented in Table 7.6. The associated tuned hyperparameters are listed in Table A.3 for the detection CNN and in Table A.4 for the classification CNN.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.85	0.84	/
Classify	0.96	0.85	0.73	/
Global	0.99	0.77	0.68	18min

Table 7.6: Performance of the double CNN architecture. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

From Table 7.6, it can be observed that the precision and the recall are quite high and well-balanced for the detection. However, for the classification and the global performance, there is a considerable gap between the precision and the recall.

Assigning the detection and classification tasks to two networks instead of a single one, like in improved Batmen, results in a more balanced detection in terms of precision and recall. However, it cannot be clearly said that one model is better in detection than the other. Indeed, the precision of the double CNN has increased while the recall has decreased compared to the improved Batmen. Which model does a better detection depends on whether the addressed problem requires a high detection recall or rather a good balance between precision and recall. As observed for the detection, the recall of the classification of the double CNN is lower than the one of the improved Batmen, while the precisions are identical. Therefore, the global precision of the double CNN is a little higher compared to improved Batmen's global precision but the recall is considerably lower.

A last point that is interesting to address is whether the tuned detection network of this architecture gives a better detection performance than Batdetective's network. This interrogation comes from the fact that there is no indication in Batdetective's article about the choice and the tuning of the hyperparameters. When training and testing our detection network with Batdetective's hyperparameters on our data, we obtain the detection performance presented in Table 7.7.

	Accuracy	Precision	Recall
Detect	0.99	0.80	0.84

Table 7.7: Performance of the detection CNN with Batdetective's hyperparameters. For `detect`, two classes are considered and the metrics are the ones of class "bat".

From Tables 7.7 and 7.6, it can be observed that the recall is identical but the precision is 5% higher with the double CNN's tuned hyperparameters. Thus, it can be concluded that the hyperparameters of the detection CNN found with our tuning are better suited for detection than the ones used by Batdetective.

7.6 Experiments on the Hybrid Model Using CNN Features

In the previous sections, the different architectures are solely composed of CNNs. Therefore, we try to determine whether using hybrid models that combine a CNN and another model that is not a CNN can achieve a better performance than using only CNNs. Again, the impact of HNM on the performance is analysed.

Extreme Gradient Boosting

One of the two algorithms that we use as a component of the hybrid model is XGBoost. The first experiment is to compute the performance without HNM as well as with one and two iterations of HNM. However, the number of examples added during an iteration of HNM is too large for the XGBoost algorithm which raises an out of memory exception.

To solve this issue while still being able to use HNM, a random fraction of all the examples generated with HNM is selected until reaching the memory limit. The memory limit is reached when taking 15.1% of the total number of examples that would normally be added by performing two iterations of HNM. The results obtained with the reduced number of examples added with HNM are presented in Table 7.8.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.60	0.94	0.85	0.81	0.64	0.79
1	0.61	0.94	0.84	0.80	0.63	0.79
2	0.60	0.94	0.84	0.81	0.62	0.79

Table 7.8: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and CNN features. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

From Table 7.8, it can be noticed that the precision and recall of the detection and classification vary by at most one percent. This smaller variation compared to other models is due to the fact that fewer examples are added for this model than for any of the others, because of the memory issue. When the performance of the hybrid model using XGBoost

and CNN features comes up, it refers to the performance of this architecture without using HNM, as it gives the best global performance.

The performance of the hybrid model using XGBoost is presented in Table 7.9. The associated tuned hyperparameters are listed in Table A.2 for the features CNN and in Table A.5 for the classification XGBoost.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.60	0.94	/
Classify	0.96	0.85	0.81	/
Global	0.99	0.64	0.79	44min

Table 7.9: Performance of the hybrid model using XGBoost and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

The results in Table 7.9 indicate that the classification is particularly efficient and that the recall of the detection is very high. The weak point of this model comes from the detection precision which is very low and is responsible for the low global precision.

Given that the classification is particularly good while the detection could need some improvement, it would be interesting to combine this architecture with another one that has a good detection performance. Since the detection CNN of the double CNN architecture has a good detection performance, it can be used to handle the detection. The XGBoost hybrid model can therefore be used solely for the classification. The performance of the combination of the binary detection CNN with the XGBoost hybrid model can be found in Table 7.10.

	Accuracy	Precision	Recall
Detect	0.99	0.85	0.84
Classify	0.97	0.85	0.83
Global	0.99	0.78	0.75

Table 7.10: Performance of the detection CNN followed by the hybrid model using XGBoost and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

From Table 7.10, it can be noticed that the combination of the detection CNN and the XGBoost hybrid model does increase the performance. The detection precision is much higher and closer to the detection recall than it is the case for the XGBoost hybrid model. The same holds for the global performance, for which the precision has increased by 14% while the recall has only decreased by 4%.

Support Vector Machine

Another model that is used as a component of the hybrid model is the SVM. The performance of the hybrid model that uses an SVM and CNN features is presented in Table

7.11. The associated tuned hyperparameters are listed in Table A.2 for the features CNN and in Table A.6 for the classification SVM.

	Accuracy	Precision	Recall	Time
Detect	0.98	0.17	0.60	/
Classify	0.93	nan	0.52	/
Global	0.99	0.40	0.47	3.5h

Table 7.11: Performance of the hybrid model using an SVM and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

From Table 7.11, it can be observed that the classification performance is not as good as with XGBoost. Moreover, the Rhisp group is never correctly recognised. In addition to the poor classification, the SVM is extremely slow. Since using HNM is very time-consuming and will surely not improve the performance enough to make this model usable, we did not analyse the effect of HNM on this hybrid SVM model.

7.7 Experiments on the Hybrid Model Using Call Features

In the previous sections, the features given as input to the models are always spectrograms. Therefore, we try to determine whether using features related to characteristics of bat calls can achieve a better performance than using spectrograms. Again, the impact of HNM on the performance is analysed.

Extreme Gradient Boosting

One of the two models that we use as a component of the hybrid model is XGBoost. The first experiment is to compute the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 7.12.

Number of	Detection		Classification		Global	
HNM iterations	Precision	Recall	Precision	Recall	Precision	Recall
0	0.85	0.84	0.75	0.70	0.68	0.65
1	0.90	0.74	0.76	0.70	0.72	0.60
2	0.90	0.76	0.76	0.71	0.73	0.61

Table 7.12: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and call features. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

It can be noticed in Table 7.12 that the precision globally increases while the recall decreases. This creates a significant gap between the precision and the recall. Therefore, when the performance of the hybrid model using XGBoost and call features is mentioned,

it refers to the performance of this architecture without using HNM.

The performance of the hybrid model that uses XGBoost and call features is presented in Table 7.13. The associated tuned hyperparameters are listed in Table A.3 for the detection CNN and in Table A.7 for the classification XGBoost.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.85	0.84	/
Classify	0.95	0.75	0.70	/
Global	0.99	0.68	0.65	5min

Table 7.13: Performance of the hybrid model using XGBoost and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

From Table 7.13, it can be observed that the global precision and recall are well-balanced but quite low. This comes from a poor classification, which indicates that using call features does not allow XGBoost to generate a strong classification model. The detection performance is well-balanced and quite high. However, this is neither due to the call features nor to XGBoost. Indeed, the detection is performed by the binary CNN of the double CNN architecture and uses spectrograms. Another point to note is that the training of this model is especially fast compared to the other architectures.

Since call features do not perform well on their own, it could be interesting to analyse an XGBoost hybrid model that takes as input a combination of both the call features and the CNN features. This would allow seeing whether the call features could help improve the already well-performing model that uses CNN features.

Support Vector Machine

Another model that we use as a component of the hybrid model is the SVM. The first experiment is to compute the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 7.14.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.85	0.84	0.67	0.51	0.63	0.51
1	0.88	0.81	0.67	0.49	0.64	0.49
2	0.89	0.80	0.64	0.48	0.63	0.47

Table 7.14: Influence of Hard Negative Mining on the precision and recall of the hybrid model using an SVM and call features. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

It can be noticed in Table 7.14 that the increase of the precision and the decrease of the recall with the number of HNM iterations is not as clearly visible as for the other mod-

els. Even so, the smallest gap between the global precision and recall is when no HNM is used. Therefore, when the performance of the hybrid model using an SVM and call features is mentioned, it refers to the performance of this architecture without using HNM.

The performance of the hybrid model that uses an SVM and call features is presented in Table 7.15. The associated tuned hyperparameters are listed in Table A.3 for the detection CNN and in Table A.8 for the classification SVM.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.85	0.84	/
Classify	0.89	0.67	0.51	/
Global	0.99	0.63	0.51	21min

Table 7.15: Performance of the hybrid model using SVM and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

As it is the case for the XGBoost hybrid model using call features, Table 7.15 indicates that the model considered has a good and balanced detection performance because it uses the detection CNN of the double CNN architecture.

The classification performance is even worse than when using XGBoost. It can even be concluded that this model is worse than a random classifier since its Kappa score for the classification is 0.45, while a random classifier has a Kappa score of 0.5. The poor performance of the SVM could be explained by the fact that only 18 features are used, which might be too few. As for XGBoost, it could be interesting to analyse an SVM hybrid model that takes as input a combination of both the call features and the CNN features to see whether or not it could improve the performance.

In addition to the poor classification, SVM is slow considering that it only has to process 18 features, which is way less than the 130x20 spectrogram features used in the models of the previous sections.

7.8 Overall Comparison

In the previous sections, the performance of the studied models has been presented in detail. In this section, we discuss which architectures are stronger in detection, classification and/or globally. To support our analysis, a global overview is given in Table 7.16, where the performance of the different models are regrouped in terms of precision and recall. The only model that is not considered in this comparison is the SVM hybrid model using CNN features because it performs too poorly.

The accuracy is not included in the table because every model has an accuracy of almost 99%. This is due to the fact that there is a very large number of windows that do not contain any call and are correctly classified. This leads to a very high number of TPs for class 0, and thus to a high accuracy.

Moreover, we address the influence of HNM on the performance. As observed in the previous sections, using one or two iterations of HNM generally decreases the recall and increases the precision. Indeed, with HNM, many positions where a call is wrongly predicted are added to the training set. By doing so, the network becomes stricter on what it considers to be a call. Therefore, it finds fewer calls, which reduces the recall, but it also predicts fewer calls on positions where there is actually no call, which increases the precision.

The original and improved Batmen are the only two models for which using HNM improves the global performance. For these two models, using one iteration of HNM increases the global precision without decreasing the recall.

Model	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
Original Batmen	0.25	0.94	0.86	0.76	0.40	0.75
Improved Batmen	0.78	0.90	0.86	0.77	0.76	0.74
Double CNN	0.85	0.84	0.85	0.73	0.77	0.68
XGB CNN	0.60	0.94	0.85	0.81	0.64	0.79
XGB call	0.85	0.84	0.75	0.70	0.68	0.65
SVM call	0.85	0.84	0.67	0.51	0.63	0.51
Detection CNN & XGB CNN	0.85	0.84	0.85	0.83	0.78	0.75

Table 7.16: Performance overview of the multi-class classification models.

From Table 7.16, it can be observed that the original Batmen has an extremely low detection precision. This is also the weak point of the XGBoost hybrid model using CNN features but to a lesser extent than for the original Batmen.

The two models that use call features have the lowest classification performance among all the models. As suggested in Section 7.7, it could be interesting to analyse the performance of a hybrid model that takes as input a combination of both the call features and the CNN features.

A further point to note is that, when the same features are used for the hybrid models using XGBoost or an SVM, XGBoost gives a better performance than the SVM.

For the detection, the best performance is obtained with the CNN that solely performs detection. This CNN is used in the double CNN architecture and in the two hybrid models using call features. For the classification, the best performance is obtained with the XGBoost hybrid model using CNN features.

Combining the best detection model, namely the detection CNN, with the best classification model, namely the XGBoost hybrid model using CNN features, gives a model having the best global performance. The second best model is the improved Batmen, which is close to the best one in terms of global performance but has a less balanced detection

and classification. Therefore, the combined model is the best among all the studied models, not only in terms of global performance but also due to its well-balanced recall and precision.

7.9 Classification Time on New Data

In this section, we would like to find out whether or not our trained models can be used on new bat call recordings in real-time. It is also interesting to discover which step of the classification process is the most time-consuming. The only model that is not considered in this experiment is the SVM hybrid model using CNN features because it performs too poorly.

To find an answer to these questions, we ran all our models on the same audio recording, which has a duration of 110 seconds. This audio file has been recorded by ourselves with an AudioMoth version 1.1.0 [19] and is thus not part of any dataset that has been used for our other experiments.

The measurements are made with and without using the GPU. When the models are run on the GPU, only the CNN and XGBoost models can take advantage of it. The SVM, the features computation and the NMS always run on the CPU.

The measurements obtained when running the models on the CPU are presented in Table 7.17, while Table 7.18 presents the results on the GPU. From these tables, it can be observed that every model is able to classify the audio file faster than in real-time. This is the case for both CPU and GPU even if the models are slower when running on the CPU.

The step in the classification process that takes the most time on the GPU is the feature computation. When running on the CPU, the slowest step is the computation done by the first CNN. Moreover, for the architectures composed of two models, the first model is much slower than the second one. Indeed, the first CNN either filters the data samples or reduces the dimensions of the features, depending on whether it is a detection or a feature CNN. Thus, the second model either does its computations on fewer data samples or on features having smaller dimensions.

Another observation is that the hybrid models using call features for the classification are slower than the ones using CNN features. This is because the latter only have to compute the spectrogram features, while the former have to compute both the spectrogram and call features.

Classification step	Improved Batmen	Double CNN	XGB CNN	XGB call	SVM call
Feature computation	22.4	22.4	22.8	32.5	31.6
Detection CNN	/	37.4	/	47.4	46.8
Features CNN	/	/	46.8	/	/
NMS	0.9	0.8	1.3	0.8	0.8
Classification model	47.5	5.3	4.1	4.2	4.0
Total for file	70.9	66.1	75.0	84.8	83.2

Table 7.17: Classification time on new data for our multi-class models using the CPU. The time is expressed in seconds.

Classification step	Improved Batmen	Double CNN	XGB CNN	XGB call	SVM call
Feature computation	22.4	22.4	22.8	32.5	31.6
Detection CNN	/	13.7	/	20.8	20.8
Features CNN	/	/	14.5	/	/
NMS	0.9	0.8	1.3	0.8	0.8
Classification model	14.8	5.0	4.2	4.3	4.0
Total for file	38.1	42.0	42.8	58.4	57.2

Table 7.18: Classification time on new data for our multi-class models using the GPU. The time is expressed in seconds.

7.10 Conclusion

Various improvements have been brought to the original Batmen. The first changes concern the development environment, where we updated the Python version as well as some libraries. Moreover, the Hard Negative Mining algorithm, the Non Maximum Suppression algorithm as well as the confusion matrix, have been adapted for the multi-class classification.

Other modifications brought to the original Batmen are the use of the Adam optimiser, the introduction of early stopping, the saving of the features to gain time and the tuning of the hyperparameters and architectures.

Concerning the metrics, the macro-average is taken. The different metrics are computed for the detection task, the classification task and the global task, which regroups both the detection and classification.

Experiments have been conducted on every architecture presented in Chapter 6 in order to determine the strengths and weaknesses of each of them for the detection, the classification and the global task. These experiments are also intended to establish whether or not using HNM can improve the performance of the models.

The first conclusion to draw from the experiments is that the modifications brought to the original Batmen have drastically improved the detection precision and thus also the global precision.

A further point to note is that the SVM hybrid model using CNN features performs so poorly that one of the classes is never correctly predicted and this model is extremely slow to train compared to the other ones.

Moreover, the original Batmen and the XGBoost hybrid model using CNN features have a low detection precision. The two models that use call features have the lowest classification performance, while it is the highest for the XGBoost hybrid model using CNN features. For detection, the best performance is obtained with the CNN that solely performs detection. Combining the detection CNN with the XGBoost hybrid model using CNN features gives the best global performance.

Concerning HNM, one iteration is used for the original and improved Batmen, while all the other models perform better without HNM.

The last experiment that was conducted has shown that all the models are faster than real-time when running on the CPU or the GPU.

Chapter 8

Multi-Label Classification

This chapter presents all the design choices and experiments that we did for multi-label classification. The first two sections present the modifications that have been brought to the multi-class classification and evaluation processes in order to adapt them to the multi-label context. The following sections contain a description and an analysis of the different experiments that were conducted on the different architectures introduced in Chapter 6. Then, an overall comparison of the different architectures is made and the time needed for each model to classify new data is presented.

8.1 Adaptations from Multi-Class to Multi-Label Classification

To adapt the architectures presented in Chapter 6 to the multi-label context, we decided to start from our multi-class implementation and modify it. Most of the design choices that we made for the multi-class classification are still valid for the multi-label task. The following subsections present the different aspects that required an adaptation to perform multi-label classification.

Binary Relevance

In multi-label classification, several classes can be assigned to the same instance which is not the case in multi-class classification. For bat call classification, this corresponds to a situation where several calls overlap, and thus the same position can be associated with several species.

Binary relevance [63] is the most intuitive method to handle the classification of multi-label examples. The principle is to consider each class as an independent binary problem having its own binary classifier. The latter predicts whether or not its associated class label is present in the final multi-label classification.

This method comes with several advantages. A first one is its conceptual simplicity. Having one binary classifier per class label allows having a complexity linear to the number of classes and allows making use of well-known binary classification techniques. Moreover, since each class label is predicted independently of the others, a decision threshold can be applied for each binary problem. Furthermore, binary relevance works well with

macro-averaged metrics. This is due to the fact that, with binary relevance, each class is optimised separately which is well suited for macro-averaged metrics that take the average performance of each class independently.

Loss, Activation Function and Classifier

For multi-class classification with CNNs, the categorical cross-entropy loss is used. However, for multi-label classification, each class needs to be treated independently. Thus, the binary cross-entropy loss is used in CNNs to consider the eight classes as eight independent binary problems.

For multi-class classification, the activation function used in the CNN is the softmax activation function. The latter distributes the probabilities over the eight classes so that their probabilities sum up to 1. Since in multi-label classification, the same position can be associated with several classes, each class needs to receive a probability independently of the other classes. This is achieved with the sigmoid activation function.

The libraries we use for the SVM and XGBoost do not support multi-label classification. Therefore, it is not possible to perform binary relevance using a single SVM or XGBoost. To overcome this issue, we decided to use one model per class, each of them performing binary classification. Thus, every class is treated as a binary problem where the considered class is distinguished from all the other classes.

Non Maximum Suppression

The Non Maximum Suppression (NMS) algorithm used for multi-class classification cannot be left unchanged. Indeed, in the multi-class case, NMS is applied on the predominant class of each window. This is not possible in the multi-label context since each window is not necessarily associated with a single class but can contain calls from several classes.

For multi-label classification, instead of applying NMS on the predominant class, NMS is performed on each class independently. This is what allows the same position to be associated with several classes. For each class, a position found by NMS is kept if the probability of the class is higher than the probability of class 0 for that position.

Architecture and Hyperparameter Tuning with Hyperopt

The hyperparameters and architecture components that are tuned to perform multi-label classification are the same as the ones presented in Section 7.1 for multi-class classification. The objective function that is optimised for the CNNs, XGBoost and SVMs is the binary cross-entropy loss.

All the models presented in Chapter 6 were tuned using Hyperopt. The performance presented in the following experiment sections are obtained with the tuned architectures and hyperparameters. Appendix A.2 presents the hyperparameter values of the different models obtained after tuning.

8.2 Evaluation for Multi-Label Classification

The following subsections present the modifications brought to our multi-class evaluation method and to the way the confusion matrix is filled. In addition, the metrics used to evaluate the performance are presented.

Confusion Matrices

The confusion matrix used for the multi-class classification cannot be used as it is for the multi-label evaluation. A first reason is that certain cases appear in the multi-label context, while they do not occur in the multi-class case. Another reason is that in the multi-class context a single 8x8 confusion matrix is used, while in the multi-label case one binary confusion matrix is needed for each of the eight classes. Indeed, in the multi-class classification, the probability of each class depends on the probability of the others, while in the multi-label classification the probability of each class is independent.

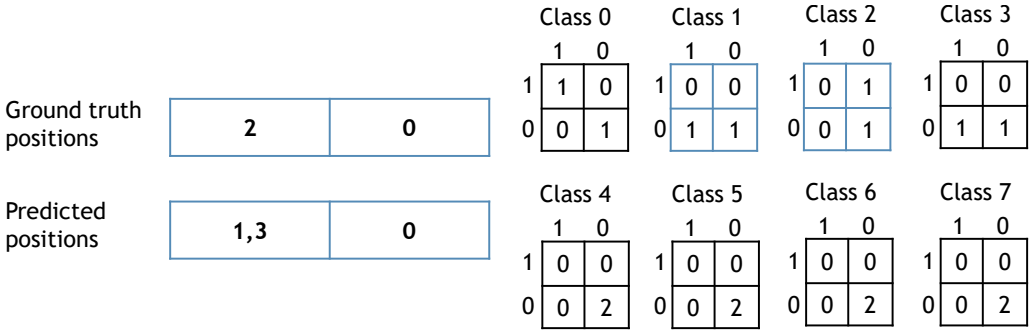
The basic cases that were encountered in the multi-class evaluation can still occur in the multi-label context. A first scenario is when a prediction is made at the correct timing and the correct class is predicted. A second scenario is when a prediction is made at the correct timing but the wrong class is predicted. A third case is when a call is predicted even though there is no call. The last basic scenario that can happen is when no call is predicted even though there is a call. These four basic cases are illustrated in Appendix C, Figure C.1.

Four other scenarios are specific to the multi-label classification and are illustrated in Figure 8.1. The first one is when a prediction is made at the correct timing and several classes are predicted but none are correct. This case is illustrated in Figure 8.1.a. In this scenario, the evaluation of classes 1 and 2 is analysed, which is why their confusion matrix are highlighted in blue.

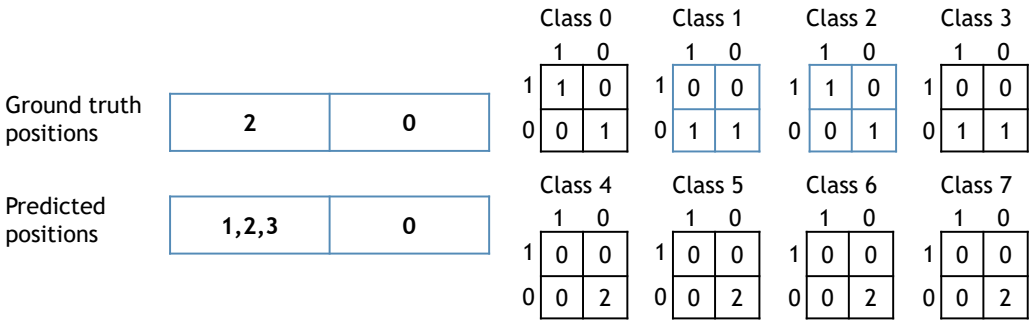
Class 1 has one FP because, in the first window, class 1 is predicted but the ground truth class is actually class 2. Class 2 has one FN because, in the first window, the ground truth class is class 2 but no prediction is made for this class in that window. Class 1 (resp. class 2) has one TN because, in the second window, class 1 (resp. class 2) is not predicted and the ground truth class is indeed not 1 (resp. 2).

The second scenario is when a prediction is made at the correct timing and several classes are predicted among which the correct one. This case is illustrated in Figure 8.1.b. In this scenario, the evaluation of classes 1 and 2 is analysed, which is why their confusion matrix are highlighted in blue.

For class 1, it can be noticed that the confusion matrix is the same as in the previous scenario. Indeed, even though the correct class is among the predictions for the first window, we decided to still penalise the fact that class 1 is wrongly predicted. Moreover, class 2 has one TP instead of the FN from the previous scenario because class 2 is among the predictions and it corresponds to the ground truth class. The fact that, for the first window, other predictions are of the incorrect class does not impact the performance of class 2.



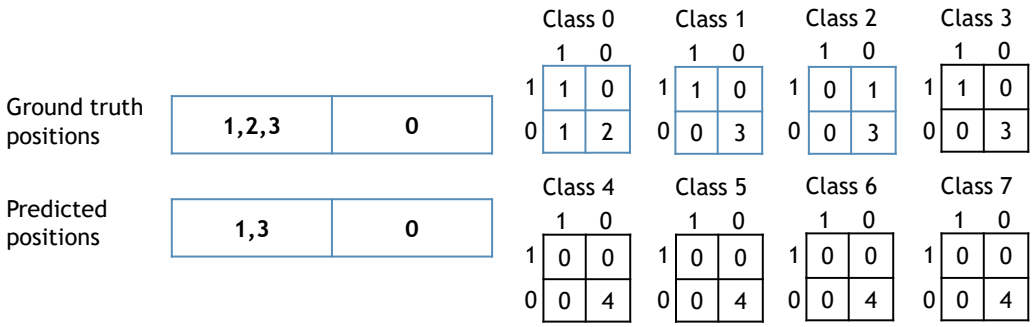
8.1.a. First case: Correct timing and several incorrectly predicted classes. This is a small example of two windows, represented by rectangles. The numbers in the windows represent the classes of bat calls. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.



8.1.b. Second case: Correct timing and several predicted classes among which the correct one. This is a small example of two windows, represented by rectangles. The numbers in the windows represent the classes of bat calls. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.

The third scenario is when several ground truth calls are overlapping and one of them is not predicted. This case is illustrated in Figure 8.1.c. In this scenario, the evaluation of classes 0, 1 and 2 is analysed, which is why their confusion matrix are highlighted in blue.

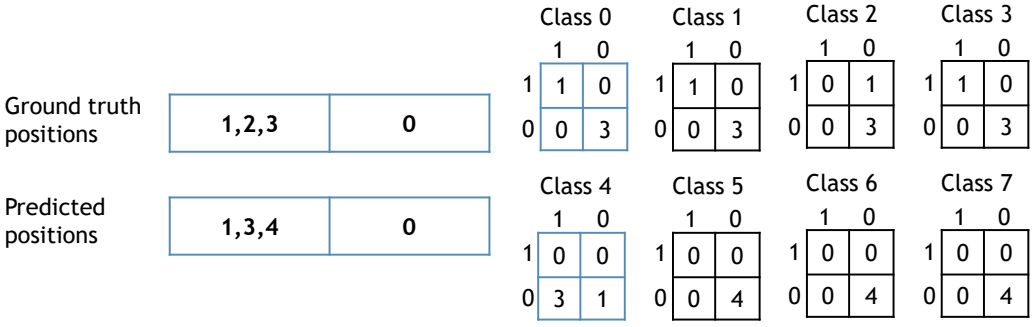
For class 1, there is one TP because, in the first window, one of the predictions is of class 1 and one of the ground truth positions is of class 1 too. There are three TNs because class 1 is not predicted for the two other ground truth calls in the first window as well as for the second window.
For class 2, there is one FN because one of the ground truth classes is 2 but 2 is not predicted. For the two other ground truth calls, class 2 is not predicted and they are indeed not of class 2. Thus, two TNs are added to the TN from the second window.
For class 0, there is one TP due to the second window. Moreover, there is one FP and two TNs because, in the first window, one of the three ground truth calls is not identified and is thus considered as a prediction of class 0.



8.1.c. Third case: Several overlapping calls among which one is not predicted. This is a small example of two windows, represented by rectangles. The numbers in the windows represent the classes of bat calls. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.

The fourth scenario is when several ground truth calls are overlapping and one of them is wrongly predicted. This case is illustrated in Figure 8.1.d. In this scenario, the evaluation of classes 0 and 4 is analysed, which is why their confusion matrix are highlighted in blue.

For class 0, there is one TP due to the second window. In the first window, there are three ground truth calls and three calls are predicted. Thus, there are three TNs because no call is predicted as belonging to class 0.
For class 4, there is one TN due to the second window. In the first window, class 4 is predicted but none of the three ground truth calls belongs to that class which gives three FPs. Indeed, even though two of the three calls are correctly predicted, the performance of class 4 is not influenced by it.



8.1.d. Fourth case: Several overlapping calls among which one is wrongly predicted. This is a small example of two windows, represented by rectangles. The numbers in the windows represent the classes of bat calls. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.

Figure 8.1: Multi-label confusion matrices for different non-trivial scenarios.

Since multi-class classification is a particular case of multi-label classification, the multi-label confusion matrices should give the same performance as the multi-class confusion matrix of Section 7.2 for multi-class classification. This was verified on each of the multi-class architectures and the same performance is indeed obtained for both the multi-class and multi-label confusion matrices.

Evaluation

Some aspects of the multi-class evaluation are also used for multi-label evaluation. A first one concerns the metrics. All the metrics presented for the multi-class classification are also computed for the multi-label classification except for the confusion entropy and the Kappa score, which are not designed to perform a multi-label evaluation. When the results of our experiments are presented, we only indicate the accuracy, precision and recall. The totality of our metrics are given in Appendix B in order to not overload the reader with numerical information. Moreover, as for multi-class classification, the performance is computed for the global task as well as for the two subtasks individually, namely the detection and classification.

In the multi-class classification, the predicted call is associated with the class having the highest predicted probability for that call. However, in the multi-label classification, several calls can overlap which means that the same position can be associated with several classes. Thus, in the multi-label classification, it is wrong to only choose the class with the highest probability as it might not be the only class present at that position. Instead, a threshold is used to decide whether or not a prediction has a high enough probability. For a predicted position, if the probability of a class is above the threshold, this class is taken into account for that position, otherwise, this class is ignored.

Since some species are easier to recognise than others, the optimal threshold is different for each class. Therefore, there is no global threshold common for all classes, but instead, a threshold between 0% and 100% is defined for each class. To find the optimal threshold of each class, we compute the performance while varying the thresholds from 0% to 100% with a step of one. For each class, the chosen threshold is the one that gives the best F_1 score for that class [3]. To make sure that the results are not biased, a validation set is used to determine the thresholds while the final performance is computed on a test set using those thresholds. The thresholds of all our architectures are given in Appendix B.

8.3 Experiments on the Improved Batmen Architecture

In this section, we try to determine whether the adaptations brought to the improved Batmen multi-class architecture allow obtaining an efficient model for multi-label classification. Another question is whether or not HNM can improve the performance of this architecture.

After having adapted the multi-class architecture of Section 7.4, we computed the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 8.1.

It can be noticed in Table 8.1 that after one iteration of HNM the detection and classification recalls are slightly improved. Moreover, the precision of the detection is increased a lot, while the precision of the classification is only slightly decreased. Because of this considerable increase in the precision of the detection, the global precision increases too. Thus, when the performance of the improved Batmen architecture comes up, it refers to the performance of this architecture using one iteration of HNM.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.74	0.75	0.80	0.80	0.69	0.62
1	0.81	0.76	0.78	0.81	0.71	0.62
2	0.81	0.75	0.78	0.81	0.72	0.61

Table 8.1: Influence of Hard Negative Mining on the precision and recall of the improved Batmen architecture. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

The performance of the improved Batmen architecture is presented in Table 8.2. The associated tuned hyperparameters are listed in Table A.9.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.81	0.76	/
Classify	0.94	0.78	0.81	/
Global	0.99	0.71	0.62	84min

Table 8.2: Performance of the improved Batmen architecture. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

It can be observed from Table 8.2 that the precision is higher than the recall for the global performance. Furthermore, the performance of this multi-label architecture is not as good as the one of the multi-class architecture. The metric for which there is the biggest gap between the two classification contexts is the recall. Indeed, there is a difference of 14% between the two recalls while there is only a gap of 5% between the precisions.

8.4 Experiments on the Double CNN Architecture

In this section, we try to determine whether the adaptations brought to the double CNN multi-class architecture allow obtaining an efficient model for multi-label classification. We would also like to find out whether separating the detection and the classification tasks, instead of having a network that performs both, can improve the performance for multi-label classification. As for the previous model, we want to observe whether HNM can improve the performance of this architecture.

After having adapted the multi-class architecture of Section 7.5, we computed the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 8.3.

It can be noticed in Table 8.3 that using HNM decreases the precision. The recall slightly increases for the detection but decreases for the classification. On the global performance, using one iteration of HNM allows gaining 2% of recall but decreases the precision too much for it to be worthwhile. Therefore, when the performance of the double CNN

architecture comes up, it refers to the performance of this architecture without using HNM.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.86	0.69	0.75	0.79	0.72	0.57
1	0.84	0.72	0.70	0.78	0.66	0.59
2	0.85	0.70	0.69	0.74	0.67	0.58

Table 8.3: Influence of Hard Negative Mining on the precision and recall of the double CNN architecture. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

The performance of the double CNN architecture is presented in Table 8.4. The associated tuned hyperparameters are listed in Table A.10 for the detection CNN and in Table A.11 for the classification CNN.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.86	0.69	/
Classify	0.93	0.75	0.79	/
Global	0.99	0.72	0.57	24min

Table 8.4: Performance of the double CNN architecture. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

From Table 8.4, it can be observed that this architecture has an even better detection precision than the one of the corresponding multi-class architecture but that the recall has decreased, and is way too low. The inverse situation is observed for the classification and a positive point to notice is that the precision and the recall are well-balanced. Concerning the global performance, the recall is very low and requires improvement.

Assigning the detection and classification tasks to two networks, instead of a single one as is the case for the improved Batmen architecture, results in a better precision but worse recall for the detection. For the classification, both metrics are worse for this architecture. From the results, it can be inferred that the improved Batmen architecture is globally better at performing multi-label bat call classification than the double CNN architecture.

8.5 Experiments on the Hybrid Model Using CNN Features

As in the previous sections, we try to determine whether the adaptations brought to this multi-class architecture allow obtaining an efficient model for the multi-label classification. We also want to find out whether using hybrid models that combine a CNN and another model that is not a CNN can achieve a better performance than using only CNNs. Again, the impact of HNM on the performance is analysed.

Extreme Gradient Boosting

One of the two models that we use as a component of the hybrid model is XGBoost, as is the case for the multi-class classification in Section 7.6. The first experiment is to compute the performance without HNM as well as with one and two iterations of HNM. However, the number of examples added during an iteration of HNM is too large for the XGBoost algorithm which raises an out of memory exception.

To solve this issue while still being able to use HNM, a random fraction of all the examples generated with HNM is selected until reaching the memory limit. The memory limit is reached when taking 4.1% of the total number of examples that would normally be added by performing two iterations of HNM. The results obtained with the reduced number of examples added with HNM are presented in Table 8.5.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.72	0.73	0.84	0.82	0.72	0.65
1	0.69	0.74	0.83	0.80	0.70	0.64
2	0.78	0.74	0.83	0.81	0.73	0.65

Table 8.5: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and CNN features. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

From Table 8.5, it can be noticed that using two iterations of HNM gives a better performance than when applying one or no iteration of HNM. Thus, when the performance of the hybrid model using XGBoost and CNN features comes up, it refers to the performance of this architecture using two iterations of HNM.

The performance of the hybrid model using XGBoost is presented in Table 8.6. The associated tuned hyperparameters are listed in Table A.9 for the features CNN and in Table A.12 for the classification XGBoost.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.78	0.74	/
Classify	0.96	0.83	0.81	/
Global	0.99	0.73	0.65	5h

Table 8.6: Performance of the hybrid model using XGBoost and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

The results in Table 8.6 indicate that the classification is particularly efficient. The weak point of this model is the detection, especially the recall that is lower than for the classification and is responsible for the low global recall. When comparing the performance with the one of the corresponding multi-class classification model, it can be noticed that the classification performance is approximately the same. Furthermore, the detection

performance is more balanced for the multi-label architecture. This architecture has a high global precision but a lower global recall which is the opposite of the corresponding multi-class architecture.

Another point to note is that the training of this model is slow compared to the other models. This is partly because, to perform multi-label classification, one XGBoost classifier per class is needed. Another reason is that two iterations of HNM are executed.

Support Vector Machine

Another model that is used as a component of the hybrid model is the SVM, as is the case for the multi-class classification in Section 7.6.

The performance of the hybrid model that uses an SVM and CNN features is presented in Table 8.7. The associated tuned hyperparameters are listed in Table A.9 for the features CNN and in Table A.13 for the classification SVM.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.44	0.36	/
Classify	0.86	0.35	0.46	/
Global	0.99	0.35	0.26	5.5h

Table 8.7: Performance of the hybrid model using an SVM and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

From Table 8.7, it can be observed that the classification performance of this model is by far not as good as with XGBoost. The hybrid model using an SVM and CNN features performs poorly for both the detection and the classification. When looking in more detail at the performance, one can find out that the model never recognises any call of the Pip50 group.

In addition to the poor performance, the SVM is extremely slow. Since using HNM is very time-consuming and will surely not improve the performance enough to make this model usable, we did not analyse the effect of HNM on this hybrid SVM model.

8.6 Experiments on the Hybrid Model Using Call Features

As in the previous sections, we try to determine whether the adaptations brought to this multi-class architecture allow obtaining an efficient model for the multi-label classification. We also want to find out whether using features related to characteristics of bat calls can achieve a better performance than when using spectrograms. Again, the impact of HNM on the performance is analysed.

Extreme Gradient Boosting

One of the two models that we use as a component of the hybrid model is XGBoost, as is the case for the multi-class classification in Section 7.7. The first experiment is to compute the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 8.8.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.85	0.73	0.52	0.63	0.52	0.52
1	0.87	0.71	0.56	0.62	0.56	0.51
2	0.88	0.69	0.52	0.64	0.54	0.51

Table 8.8: Influence of Hard Negative Mining on the precision and recall of the hybrid model using XGBoost and call features. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

It can be noticed in Table 8.8 that the recall generally decreases while the precision increases. When using one iteration of HNM instead of none, the global performance gains 4% of precision while only losing 1% of recall. Therefore, when the performance of the hybrid model using XGBoost and call features is mentioned, it refers to the performance of this architecture using one iteration of HNM.

The performance of the hybrid model that uses XGBoost and call features is presented in Table 8.9. The associated tuned hyperparameters are listed in Table A.10 for the detection CNN and in Table A.14 for the classification XGBoost.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.87	0.71	/
Classify	0.89	0.56	0.62	/
Global	0.99	0.56	0.51	81min

Table 8.9: Performance of the hybrid model using XGBoost and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

From Table 8.9, it can be observed that the global precision and recall are well-balanced but very low. This comes mainly from a poor classification, which indicates that using call features does not allow XGBoost to generate a strong classification model. The detection has a high precision but the recall has room for improvement. However, this is neither due to the call features nor to the XGBoost. Indeed, the detection is performed by the detection CNN of the double CNN architecture and uses spectrograms.

Since the model using call features does not perform well, it could be interesting to analyse an XGBoost hybrid model that takes as input a combination of both the call features and the CNN features. This would allow seeing whether the call features could help improve the better performing model that uses CNN features.

Support Vector Machine

Another model that we use as a component of the hybrid model is the SVM, as is the case for the multi-class classification in Section 7.7. The first experiment is to compute the performance without HNM as well as with one and two iterations of HNM. These results are presented in Table 8.10.

Number of HNM iterations	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
0	0.64	0.76	0.20	0.66	0.28	0.56
1	0.69	0.76	0.20	0.58	0.28	0.52
2	0.66	0.78	0.22	0.55	0.29	0.51

Table 8.10: Influence of Hard Negative Mining on the precision and recall of the hybrid model using an SVM and call features. For the detection, two classes are considered and the metrics are the ones of class "bat". For the classification (resp. the global task), the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

It can be noticed in Table 8.10 that the precision slightly increases, while the classification and global recalls strongly decrease with the number of HNM iterations. Therefore, when the performance of the hybrid model using an SVM and call features is mentioned, it refers to the performance of this architecture without using HNM.

The performance of the hybrid model that uses an SVM and call features is presented in Table 8.11. The associated tuned hyperparameters are listed in Table A.10 for the detection CNN and in Table A.15 for the classification SVM.

	Accuracy	Precision	Recall	Time
Detect	0.99	0.64	0.76	/
Classify	0.51	0.20	0.66	/
Global	0.99	0.28	0.56	26min

Table 8.11: Performance of the hybrid model using an SVM and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7). The time represents the total training time of the model.

From Table 8.11, it can be observed that the classification performance is even worse than when using XGBoost. The poor performance of the SVM could be explained by the fact that only 18 features are used, which might be too few. As for XGBoost, it could be interesting to analyse an SVM hybrid model that takes as input a combination of both the call features and the CNN features to see whether it could improve the performance.

8.7 Overall Comparison

In the previous sections, the performance of the multi-label models has been presented in detail. In this section, we discuss which architectures are stronger in detection, classifi-

cation and/or globally. To support our analysis, a global overview is given in Table 8.12, where the performance of the different models are regrouped in terms of precision and recall. The only model that is not considered in this comparison is the SVM hybrid model using CNN features because it performs too poorly. Moreover, we address the influence of HNM on the performance.

The experiments of the previous sections show that there is no clear pattern of the effect of HNM on the performance. Indeed, for some models, using HNM increases the precision and decreases the recall, while for other models it is the other way around. The models for which using HNM improves the global performance are the improved Batmen architecture as well as the two hybrid models that use XGBoost.

Model	Detection		Classification		Global	
	Precision	Recall	Precision	Recall	Precision	Recall
Improved Batmen	0.81	0.76	0.78	0.81	0.71	0.62
Double CNN	0.86	0.69	0.75	0.79	0.72	0.57
XGB CNN	0.78	0.74	0.83	0.81	0.73	0.65
XGB call	0.87	0.71	0.56	0.62	0.56	0.51
SVM call	0.64	0.76	0.20	0.66	0.28	0.56

Table 8.12: Performance overview of the multi-label classification models.

From Table 8.12, it can be observed that the SVM hybrid model using call features has the lowest detection performance as well as the lowest classification performance. As suggested in Section 8.6, it could be interesting to analyse the performance of a hybrid model that takes as input a combination of both the call features and the CNN features.

A further point to note is that, when the same features are used for the hybrid models using XGBoost or an SVM, XGBoost gives a better performance than the SVM.

Moreover, the detection performance of the double CNN architecture and of the two hybrid models that use call features are not equal, even though they use the same detection CNN. This is due to the evaluation thresholds, which vary from one architecture to the other. Depending on the thresholds, certain predictions will or will not be taken into consideration during the evaluation.

For the detection, the best performance is obtained with the improved Batmen architecture. For the classification, the best performance is obtained with the XGBoost hybrid model using CNN features. The best global performance is also obtained with this model. Among all our architectures, it is the one having the best classification performance while also having a reasonable detection.

8.8 Classification Time on New Data

In this section, we would like to find out whether our trained models can be used on new bat call recordings in real-time. It is also interesting to discover which step of the classification process is the most time-consuming. The only model that is not considered

in this experiment is the SVM hybrid model using CNN features because it performs too poorly.

To find an answer to these questions, we ran all our models on the same audio recording, which has a duration of 110 seconds. This audio file has been recorded by ourselves with an AudioMoth version 1.1.0 [19] and is thus not part of any dataset that has been used for our other experiments.

The measurements are made with and without using the GPU. When the models are run on the GPU, only the CNN and XGBoost models can take advantage of it. The SVM, the features computation and the NMS always run on the CPU.

The measurements obtained when running the models on the CPU are presented in Table 8.13, while Table 8.14 presents the results on the GPU.

Classification step	Improved Batmen	Double CNN	XGB CNN	XGB call	SVM call
Feature computation	22.5	22.5	35.8	32.0	31.4
Detection CNN	/	37.3	/	37.3	46.9
Features CNN	/	/	96.0	/	/
NMS	0.4	0.8	0.8	2.2	0.8
Classification model	41.0	5.3	156.3	23.8	3.0
Total for file	64.0	66.0	289.0	95.8	82.3

Table 8.13: Classification time on new data for our multi-label models using the CPU. The time is expressed in seconds.

Classification step	Improved Batmen	Double CNN	XGB CNN	XGB call	SVM call
Feature computation	22.4	22.5	22.9	32.5	31.5
Detection CNN	/	13.9	/	14.0	20.6
Features CNN	/	/	14.8	/	/
NMS	0.4	0.8	0.6	1.8	0.8
Classification model	16.1	4.9	21.4	23.0	2.9
Total for file	37.9	42.2	59.7	71.2	56.0

Table 8.14: Classification time on new data for our multi-label models using the GPU. The time is expressed in seconds.

From Table 8.14, it can be observed that every model is able to classify the audio file faster than in real-time when running on the GPU. However, Table 8.13 shows that the XGBoost hybrid model using CNN features is not able to classify the recording in real-time when running on the CPU. All the other models run in real-time, even though they are slower on the CPU than on the GPU.

The step in the classification process that takes the most time on the GPU is the feature computation. When running on the CPU, the slowest step is the computation done by the first CNN.

For the architectures composed of two models, the first model is much slower than the second one. Indeed, the first CNN either filters the data samples or reduces the dimensions of the features, depending on whether it is a detection or a feature CNN. Thus, the second model either does its computations on fewer data samples or on features having smaller dimensions.

These observations do not apply to the XGBoost hybrid model using CNN features. This is because it uses one XGBoost classifier per class which slows down the classification, making it the slowest step for this architecture.

Concerning the XGBoost hybrid model using call features, it is indeed its first model that is slower when running on the CPU. However, it is not the case with the GPU because the detection CNN gains more time by using the GPU than the seven XGBoost classifiers.

8.9 Conclusion

Various modifications have been brought to the multi-class algorithms and architectures to adapt them for multi-label classification. The first difference lies in the use of the binary relevance technique to decompose the multi-label classification into eight independent binary classification problems.

Moreover, to perform multi-label classification, the binary cross-entropy loss and the sigmoid activation function are used. In the case of the SVM and XGBoost, one model per class needs to be built. Furthermore, the tuning of the hyperparameters and architectures is done using Hyperopt. The Non Maximum Suppression algorithm and the confusion matrix have also been adapted for the multi-label classification.

Concerning the metrics, the macro-average is taken. The different metrics are computed for the detection task, the classification task and the global task, which regroups both the detection and the classification. For the evaluation, one threshold is computed for each class. If the probability of a predicted class is above its associated threshold, the prediction is taken into account, otherwise, it is ignored.

Experiments have been conducted on every architecture presented in Chapter 6 in order to determine the strengths and weaknesses of each of them for the detection, the classification and the global task. These experiments are also intended to establish whether or not using HNM can improve the performance of the models.

A first conclusion is that the SVM hybrid model using CNN features performs so poorly that one of the classes is never correctly predicted and it is extremely slow to train compared to the other models. Moreover, the SVM hybrid model using call features has the lowest detection performance as well as the lowest classification performance. The precision and recall of the detection are the highest for the improved Batmen model. The best classification and global performance are obtained with the XGBoost hybrid model using CNN features.

Concerning HNM, one iteration is used for the improved Batmen architecture as well as for the XGBoost hybrid model using call features. Two iterations of HNM are used for

the XGBoost hybrid model using CNN features, while the other models perform better without HNM.

The last experiment that was conducted has shown that all the models are faster than real-time when running on the CPU or the GPU except for the XGBoost hybrid model using CNN features, which is slower than real-time on the CPU.

Chapter 9

Reproduction of the Results

This chapter presents the environment required to run our code, which is available on `https://github.com/MelanieBeauvois/batML`. It also contains an explanation on how to train our different architectures with your own data and finally how to use the different models to classify new recordings.

9.1 System Requirements

The totality of the libraries present in our environment can be found in the `environment` file, which is available in our GitHub project. The more specific libraries are the following:

- Python 3.6
- CUDA 10.2
- cuDNN 7.6.5
- Cython 0.29.21
- hyperopt [8] 0.2.5
- joblib 1.0.0
- numpy 1.16.4
- pandas 1.1.5
- scikit-image 0.17.2
- scikit-learn [12] 0.24.1
- scikit-multilearn 0.2.0
- scipy 1.4.1
- tensorflow 2.1.0
- xgboost [14] 1.4.0

The execution times presented throughout this work were obtained by running the code on a computer with the following specifications

- OS: CentOS 8.1
- GPU: NVIDIA GeForce RTX 2080 SUPER
- CPU: Intel® Core™ i7-9800X CPU @ 3.80GHz
- RAM: 32 GB

9.2 How to Train Our Classifiers on Your Own Data

This section presents the data required to train our classifiers as well as the parameters to be set by the user.

Gather the Necessary Data

The data necessary for the detection can be found on Batdetective's website [37]. The data we use for the classification belongs to Natagora and is therefore not made available online. However, you can use your own labelled recordings to train our models. To do so you will need to create a .npz file with the following fields: `train_files`, `train_durations`, `train_pos`, `train_class`, `test_files`, `test_durations`, `test_pos` and `test_class`.

- `train_files`: array containing the filename of the recordings, without the extension.
- `train_durations`: array containing the duration of each file present in `train_files`.
- `train_pos`: array where each line is an array of the call positions in the respective file.
- `train_class`: array where each line is an array containing the classes of the calls in the respective file.

The same holds for the test fields.

For multi-label classification, when two calls overlap at the same position, they need to be both inserted in the arrays as separate entries having the same position.

Run the Training and Evaluate the Model

The `run_training.py` file allows training and evaluating any of our available architectures with your own data. At the beginning of the main section of the file, you can define several parameters such as the name of the model you want to train. The trained model will be evaluated on the detection, the classification and on both tasks combined. The performance metrics are written in a text file and the trained model is saved in order to use it to classify new recordings.

The tuned hyperparameters we obtained when training our different architectures on Batdetective's and Natagora's data are available in the `data/` folder and will be used by default. In case you want to change the hyperparameters, you should modify the values in the .csv files of the `data/` folder.

The `data_set_params.py` file is another important file where certain parameters need to be chosen. It is possible to perform tuning by setting the appropriate model variables to True and choosing the desired tuning time. Note that when the tuning is interrupted it will automatically resume to its last iteration when started again. To perform Hard Negative Mining during the training, the number of iterations should be indicated in the `num_hard_negative_mining` variable. By default, no HNM is performed. To gain time, the `save_features_to_file` variable can be set to True so that the features are computed and saved once. The next time the features are needed by the models, the features will be loaded if the `load_features_from_file` variable is set to True.

9.3 How to Run Our Classifiers on Your Own Data

The `run_classifier.py` file allows running an already trained model on new data to find calls and predict the corresponding groups. At the beginning of the main section of the file, you can define various parameters such as the name of the model you want to use.

Our already trained models are available in the `data/models/` directory. Your own models will also be saved in this folder after training. To use one of your models, you have to change the value of the `date` and `hnm_iter` variables so that they correspond to the date present in the name of your model and to the number of HNM iterations used during training.

9.4 Conclusion

All the measurements of this work have been made on a desktop with an i7-9800X CPU @ 3.80GHz, 32 GB RAM and an RTX 2080 SUPER GPU on CentOS 8.1. The different packages necessary to run the code have been presented in this chapter.

The dataset used for detection is available on Batdetective's website but the user needs to procure labelled data to perform classification.

It is possible to train our models on your own data using the `run_training.py` code. The `run_classifier.py` file allows running an already trained model on new data to detect and classify bat calls. These two files contain several parameters that can be modified to customise the training and the classification.

Chapter 10

Conclusion

This thesis aimed to build a robust bat call detection and classification tool for the twenty-three Belgian bat species. To achieve this, our work takes as a starting point two open-source projects, called Batdetective and Batmen. Batdetective worked on the preprocessing of audio files and the detection of bat calls, while Batmen modified Batdetective's code to adapt it to the multi-class classification.

Our first step was to improve Batmen's model in order to obtain an enhanced performance for the multi-class classification. Then, we combined different machine learning algorithms, receiving various types of features as input, in hope of designing a model that is even better suited to the problem at hand. Some of our architectures perform the detection and classification separately, while the others perform both tasks at once. The features fed as input to the models are either spectrograms, characteristics related to bat calls or features computed by a CNN. As a final step, we adapted each of our multi-class classification architectures and algorithms for them to be able to perform multi-label classification.

Concerning the multi-class classification, not only have we improved the performance of Batmen's model, but we also designed an architecture that gives even better performance. The latter detects bat calls using a CNN that solely performs detection, and classifies the calls using a model composed of a CNN followed by an XGBoost. This architecture has a global precision of 78% and a recall of 75%.

Concerning the multi-label classification, our best performing model is composed of a CNN followed by an XGBoost and performs detection and classification at once. This architecture has a global precision of 73% and a recall of 65%. These are not as high as for multi-class classification, especially the recall. This comes from the fact that the detection task has a lower performance in the multi-label context.

In the perspective of enhancing our bat call identification tool, several areas for improvement can be explored. A first one concerns the datasets used for the multi-class and multi-label classification. Since the dataset provided by Natagora only indicates the group of bat present in the file but not the position of the calls, Batdetective's tool was used to add the call positions to the data labels. However, given that Batdetective has not a perfect performance, the labels contain errors. Moreover, the dataset does not contain any file where overlapping calls have been labelled which is why calls had to be artificially

overlapped to train our multi-label classification models. Considering these shortcomings, a possible way to better train our models would be to replace these datasets with another one containing overlapping calls as well as indications of the call positions and the respective groups. This would require validation by an expert in manual bat call classification.

Another possible improvement on the datasets would be to label the calls according to the species instead of the group. This would allow having a classification of the twenty-three Belgian bat species instead of only their respective group. Even for a use case where only the group is needed, it would still be useful to have the species indicated in the data labels in order to verify that all the species of each group are well represented in the dataset.

A second area for improvement concerns the Hard Negative Mining algorithm. Using this technique only improves the performance of some of our models. It might be possible to make the best out of HNM by adding some of its hyperparameters to the Hyperopt tuning, such as the number of iterations and the shift used to create the HNM examples. Given that HNM is very slow, it should be verified whether or not tuning these hyperparameters is time-effective.

A third possibility to improve the performance is to try out other architectures to discover one that is even more suited to the problem at hand. For instance, it would be interesting to see the performance when combining the call features with the features computed by a CNN. Furthermore, the temporal characteristics related to bat calls, such as the interval between the calls and the duration of each call, could be exploited. These temporal characteristics could be taken into consideration by using a Long Short-Term Memory (LSTM) architecture, which is designed to process sequences of data.

A last area for improvement lies in the way the multi-label classification is handled. In our work, we use the binary relevance concept that considers one binary problem per class, making a total of eight independent binary classifiers. However, other techniques exist such as the label powerset method. The latter transforms the multi-label classification into a multi-class classification problem with one label per possible combination of the output classes. In our case, this would amount to 128 classes as the seven bat groups can all be combined. Instead of making adaptations to the labels, some preprocessing could be done on the audio recordings to tackle the challenging multi-label classification. Indeed, some preprocessing step could isolate the different calls that overlap in the recordings. In this way, multi-class classification algorithms could be applied as no overlapping calls would be left in the dataset.

These further research ideas show that there is still room for improvement, especially for our multi-label classification tool. Even so, its performance is reasonable and we obtain high performance for the multi-class classification. Up to now, there is a lack of such bat call classification tools that are both open-source and well-performing. Our work addresses this need by providing a competitive open-source tool for bat call detection and classification. This is useful for everyone who is passionate about bats and/or machine learning.

Being open-source, our code allows anyone to contribute in order to improve or add new functionalities to our architectures. A possible application of our work is to integrate our

best classifier in a portable recorder to see in real-time which groups of bats are present in the surrounding area. Another conceivable application is to leave the recorder someplace and to make it send its recordings over the network to a computer, which then processes the files using our model. A further possibility is to develop a website where users can upload their bat call recordings and get an indication of the position and group of the calls.

Our tool is helpful for people who are interested in bats but do not have the necessary knowledge to detect and classify bat calls. It is also valuable for chiropterologists who usually have to perform the classification of bat call recordings by hand, which is a tedious and repetitive task requiring lots of experience. Moreover, today's recorders allow collecting large amounts of data every day, rendering the manual classification of all the recordings infeasible. These highlighted facts show that our automated tool, which is both fast and competitive, will be of great help to teams that monitor bats in their pursuit of protecting endangered species.

Bibliography

[1] Wildlife Acoustics. Kaleidoscope: Bioacoustics sound analysis, 2018. URL: https://www.wildlifeacoustics.com/products/kaleidoscope.

[2] Wildlife Acoustics. Wildlife audio recording equipment, 2021. URL: https://www.wildlifeacoustics.com/.

[3] Reem Al-Otaibi, Peter Flach, and Meelis Kull. Multi-label classification: A comparative study on threshold selection methods. 2014. URL: https://www.researchgate.net/publication/277887001_Multi-label_Classification_A_Comparative_Study_on_Threshold_Selection_Methods.

[4] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi:10.1109/ICEngTechnol.2017.8308186.

[5] Michel Barataud. *Ecologie acoustique des Chiroptères d'Europe, identification des espèces, étude de leur habitats et comportements de chasse*. Biotope Editions, 3 edition, 2015. ISBN: 978-2-36662-142-6.

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, page 2546–2554. Curran Associates Inc., 2011.

[7] James Bergstra, Brent Komer, Chris Eliasmith, Daniel Yamins, and David Cox. Hyperopt: A python library for model selection and hyperparameter optimization. *Computational Science Discovery*, 8, 2015. doi:10.1088/1749-4699/8/1/014008.

[8] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123. PMLR, 17–19 Jun 2013. URL: http://proceedings.mlr.press/v28/bergstra13.html.

[9] Biotope. Sonochiro®, le logiciel d'analyse automatique d'ultrasons de chauves-souris, 2017. URL: https://sonochiro.biotope.fr/.

[10] Giuseppe Bonaccorso. *Mastering Machine Learning Algorithms : Expert Techniques to Implement Popular Machine Learning Algorithms and Fine-Tune Your Models*. Packt Publishing, 2018.

[11] Michael Buckland and Fredric Gey. The relationship between recall and precision. *Journal of the Association for Information Science and Technology*, 45, 1994. URL: https://doi.org/10.1002/(SICI)1097-4571(199401)45:1<12::AID-ASI2>3.0.CO;2-L.

[12] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013. URL: https://scikit-learn.org/stable/modules/classes.html.

[13] Florian Cantori and Alexandre Haquart. Sonochiro: User guide, 2017. URL: http://sonochiro.biotope.fr/doc/SonoChiro_Manual_EN.pdf.

[14] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016. URL: http://doi.acm.org/10.1145/2939672.2939785, doi:10.1145/2939672.2939785.

[15] Plan National d'Actions Chiroptères. Cycle de vie. URL: https://plan-actions-chiropteres.fr/les-chauve-souris/biologie/cycle-de-vie.

[16] Plan National d'Actions Chiroptères. Menaces. URL: https://plan-actions-chiropteres.fr/les-chauve-souris/menaces.

[17] Plan National d'Actions Chiroptères. Émissions sonores / vocalises. URL: https://plan-actions-chiropteres.fr/les-chauve-souris/biologie/emissions-sonores.

[18] XGBoost Developers. Xgboost documentation: Xgboost parameters, 2021. URL: https://xgboost.readthedocs.io/_/downloads/en/release_1.4.0/pdf/.

[19] Open Acoustic Devices. Audiomoth. URL: https://www.openacousticdevices.info/audiomoth.

[20] Lucile Dierckx and Mélanie Beauvois. Automated detection of bat species in belgium: Code, 2021. URL: https://github.com/MelanieBeauvois/batML.

[21] Christian Dietz and Andreas Kiefer. *Chauves-souris d'Europe. Connaître, identifier, protéger*. Delachaux, 2015.

[22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011. URL: https://www.researchgate.net/publication/220320677_Adaptive_Subgradient_Methods_for_Online_Learning_and_Stochastic_Optimization.

[23] Pierre Dupont. Lecture 5: Deep learning. *INGI2262 - Machine Learning : classification and evaluation*, 2020.

[24] Bruxelles Environnement. Les chauves-souris: connaître et protéger, 2019. URL: https://document.environnement.brussels/opac_css/elecfile/BRO_20190819_Chauves_souris_FR.

[25] Maxime Franco and Corrado Lipani. Automated monitoring of bat species in belgium. 2020. URL: http://hdl.handle.net/2078.1/thesis:25168.

[26] Margherita Grandini, E. Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *ArXiv*, abs/2008.05756, 08 2020. URL: https://arxiv.org/pdf/2008.05756.pdf.

[27] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. A convnet for non-maximum suppression. In *Pattern Recognition*, pages 192–204, Cham, 2016. Springer International Publishing. URL: https://arxiv.org/pdf/1511.06437.pdf.

[28] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-Keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019. URL: https://autokeras.com/.

[29] Kate Jones, Jon Russ, Andriy-Taras Bashta, Zoltálan Bilhari, Colin Catto, István Csősz, Alexander Gorbachev, Páleter Győrfi, Alice Hughes, Igor Ivashkiv, Natalia Koryagina, Anikó Kurali, S.D. Langton, Alanna Collen, Georgiana Mărginean, Ivan Pandourski, Stuart Parsons, Abigél Szodoray-Parádi, and Oleg Zavarzin. Indicator bats program: A system for the global acoustic monitoring of bats. *Biodiversity Monitoring and Conservation: Bridging the Gap between Global Commitment and Local Action*, 2013. doi:10.1002/9781118490747.ch10.

[30] Phil Kim. Convolutional neural network. In *MATLAB Deep Learning: With Machine Learning, Neural Networks and Artificial Intelligence*, pages 121–147. Apress, 2017. doi:10.1007/978-1-4842-2845-6_6.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL: https://arxiv.org/pdf/1412.6980.pdf, arXiv:1412.6980.

[32] Alok Kumar and Mayank Jain. *Ensemble Learning for AI Developers: Learn Bagging, Stacking, and Boosting Methods with Use Cases*. Apress, 2020. doi:10.1007/978-1-4842-5940-5.

[33] Michèle Lemaire and Laurent Arthur. *Les chauves-souris de France, Belgique, Luxembourg et Suisse*. Biotope Editions, 2009.

[34] Tsung-Yi Lin, Priyal Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP:1–1, 07 2018. URL: https://arxiv.org/abs/1708.02002, doi:10.1109/TPAMI.2018.2858826.

[35] Wei Liu, Lin Chen, and Yajun Chen. Age classification using convolutional neural networks with the multi-class focal loss. *IOP Conference Series: Materials Science and Engineering*, 428:012043, 10 2018. doi:10.1088/1757-899X/428/1/012043.

[36] Douglas Lyon. The discrete fourier transform, part 4: Spectral leakage. *Journal of Object Technology*, 8:23–34, 11 2009. doi:10.5381/jot.2009.8.7.c2.

[37] Oisin Mac Aodha, Rory Gibb, Kate Barlow, Ella Browning, Michael Firman, Robin Freeman, Briana Harder, Libby Kinsey, Gary Mead, Stuart Newson, Ivan Pandourski, Stuart Parsons, Jon Russ, Abigel Szodoray-Paradi, Farkas Szodoray-Paradi, Elena Tilova, Mark Girolami, Gabriel Brostow, and Kate. E. Jones. Bat detective - deep learning tools for bat acoustic signal detection. 2018. URL: http://visual.cs.ucl.ac.uk/pubs/batDetective/, doi:https://doi.org/10.1371/journal.pcbi.1005995.

[38] Fingerprint Digital Media. Zooniverse, 2009. URL: https://www.zooniverse.org/.

[39] Z. L. Nagy, L. Barti, A. Dóczy, CS. Jére, T. Postawa, L. Szántó, A. Szodoray-Parádi, and F. Szoroday-Parádi. Survey of romania's underground bat habitats: Status and distribution of cave dwelling bats. 2005. URL: https://www.conservationleadershipprogramme.org/media/2014/11/zoltan-romania-bats-final-report.pdf.

[40] Natagora. Qui est natagora. URL: https://www.natagora.be/qui-est-natagora.

[41] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. 2020. URL: https://arxiv.org/pdf/1811.03378.pdf.

[42] Martin Obrist and Ruedi Boesch. Batscope manages acoustic recordings, analyses calls and classifies bat species automatically. *Canadian Journal of Zoology*, 96(9):939–954, 02 2018. doi:10.1139/cjz-2017-0103.

[43] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Keras Tuner, 2019. URL: https://github.com/keras-team/keras-tuner.

[44] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015. arXiv:1511.08458.

[45] Plecotus. Les différentes espèces de chauves-souris. URL: https://plecotus.natagora.be/index.php?id=700.

[46] Plecotus. Plecotus étudie les chauves-souris. URL: https://plecotus.natagora.be/index.php?id=707.

[47] Vasil V. Popov. Bats in bulgaria: Patterns of species distribution, richness, rarity, and vulnerability derived from distribution models. In *Bats*. IntechOpen, 2018. doi:10.5772/intechopen.73623.

[48] Lutz Prechelt. Early stopping — but when? In *Neural Networks: Tricks of the Trade: Second Edition*, pages 53–67. Springer Berlin Heidelberg, 2012. doi:10.1007/978-3-642-35289-8_5.

[49] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *An Introduction to Machine Learning*. Springer, Cham, 2019. URL: https://link.springer.com/content/pdf/10.1007%2F978-3-030-15729-6.pdf, doi:https://doi.org/10.1007/978-3-030-15729-6.

[50] Google Research. Model Search, 2021. URL: https://github.com/google/model_search.

[51] Axel Aronio De Romblay. MLBox, Machine Learning Box. 2017. URL: https://mlbox.readthedocs.io/en/latest/.

[52] Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016. URL: https://arxiv.org/pdf/1609.04747.pdf.

[53] Neha Singh. Classification of animal sound using convolutional neural network. 2020. URL: https://arrow.tudublin.ie/cgi/viewcontent.cgi?article=1222&context=scschcomdis, doi:https://doi.org/10.21427/7pb8-9409.

[54] Oleksandr Sosnovshchenko and Oleksandr Baiev. *Machine Learning with Swift : Artificial Intelligence for iOS*. Packt Publishing, 2018.

[55] Zhuojin Sun, Yong Wang, and Robert Laganière. Hard negative mining for correlation filters in visual tracking. *Machine Vision and Applications*, 2019. doi:10.1007/s00138-019-01004-0.

[56] A. Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2020. URL: https://www.emerald.com/insight/content/doi/10.1016/j.aci.2018.08.003/full/html.

[57] T Tieleman and G Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. 2012. URL: http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf.

[58] Bat Conservation Trust. About bats. URL: https://www.bats.org.uk/about-bats.

[59] Bat Conservation Trust. Frequency division bat detectors. URL: https://www.bats.org.uk/about-bats/bat-detectors-1/frequency-division-bat-detectors.

[60] Sven Verkem, Ben Van der Wijden, and Pierrette Nyssen. Manuel d'utilisation du détecteur d'ultrasons hétérodyne pour débutants, 2014. URL: https://plecotus.natagora.be/fileadmin/Plecotus/Documentation/Manuel_heterodyne_versionMars2014.pdf.

[61] Biodiversité Wallonie. Chauves-souris. URL: http://biodiversite.wallonie.be/fr/chauves-souris.html?IDC=325.

[62] Jin-Mao Wei, Xiao-Jie Yuan, Qing-Hua Hu, and Shu-Qin Wang. A novel measure for evaluating classifiers. *Expert Systems with Applications*, 37(5):3799–3809, 2010. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.708.1668&rep=rep1&type=pdf, doi:https://doi.org/10.1016/j.eswa.2009.11.040.

[63] Min-Ling Zhang, Yu-Kun Li, Xu-Ying Liu, and Xin Geng. Binary relevance for multi-label learning: an overview. *Frontiers of Computer Science*, 12, 11 2017. doi:10.1007/s11704-017-7031-7.

[64] Zhi-Hua Zhou. Ensemble learning. *Encyclopedia of Biometrics*, pages 270–273, 2009. URL: http://129.211.169.156/publication/springerEBR09.pdf.

Appendix A

Hyperparameters

All the architectures used for the multi-class and multi-label classification have been tuned with Hyperopt to improve their performance. This appendix presents the optimal hyperparameters that have been found during the tuning for each of the architectures used during this thesis.

A.1 Multi-Class Classification

This section presents the tuned hyperparameters of the different multi-class classification models.

Original Batmen

CNN	number of convolutional layers	3
	number of filters	32
	filter size	3
	number of dense layers	2
	number of dense nodes	256
	max pooling size	2
	dropout probability	0.5
	number of epochs	50
SGD	learning rate	0.01
	moment	0.9
	batch size	256
	nesterov	True
HNM	number of iterations	1

Table A.1: Hyperparameters of the original Batmen network.

Improved Batmen

CNN	number of convolutional layers	3
	number of filters	56
	filter size	3
	number of dense layers	3
	number of dense nodes	384
	max pooling size	2
	dropout probability	0.7
	batch size	320
Adam	learning rate	0.00140
	beta 1	0.95
	beta 2	0.999
	epsilon	1×10^{-8}
Early stopping	minimum delta	0.0005
	patience	20
HNM	number of iterations	1

Table A.2: Hyperparameters of the improved version of Batmen's network.

Double CNN

CNN	number of convolutional layers	4
	number of filters	48
	filter size	3
	number of dense layers	2
	number of dense nodes	384
	max pooling size	2
	dropout probability	0.5
	batch size	128
Adam	learning rate	0.00032
	beta 1	0.95
	beta 2	0.999
	epsilon	1×10^{-8}
Early stopping	minimum delta	0.005
	patience	15
HNM	number of iterations	0

Table A.3: Hyperparameters of the detection CNN of the double CNN architecture.

CNN	number of convolutional layers	3
	number of filters	24
	filter size	4
	number of dense layers	3
	number of dense nodes	320
	max pooling size	2
	dropout probability	0.6
	batch size	192
Adam	learning rate	0.00373
	beta 1	0.9
	beta 2	0.999
	epsilon	1×10^{-8}
Early stopping	minimum delta	5×10^{-5}
	patience	20
HNM	number of iterations	0

Table A.4: Hyperparameters of the classification CNN of the double CNN architecture.

Hybrid Models Using CNN Features

This hybrid model is composed of a feature CNN over eight classes followed by either XGBoost or an SVM. The hyperparameters of this feature CNN are the same as the hyperparameters of the classification CNN used in the improved Batmen architecture and can be found in Table A.2. The hyperparameters of XGBoost are listed in Table A.5, while the hyperparameters of the SVM are listed in Table A.6.

XGBoost	number of estimators	349
	maximum depth	14
	learning rate	0.08474
	gamma	0.01
	minimum child weight	3
	subsample	0.7
	scale positive weight	1.5
	tree method	gpu_hist
HNM	number of iterations	0

Table A.5: Hyperparameters of XGBoost in the hybrid model using CNN features.

SVM	kernel	linear
	degree	5
	C	10
	gamma	auto
	maximum number of iterations	2500
	class weight	None
HNM	number of iterations	0

Table A.6: Hyperparameters of the SVM in the hybrid model using CNN features.

Hybrid Models Using Call Features

This hybrid model is composed of a detection CNN followed by either XGBoost or an SVM. The hyperparameters of this detection CNN are the same as the hyperparameters of the detection CNN used in the double CNN architecture and can be found in Table A.3. The hyperparameters of XGBoost are listed in Table A.7, while the hyperparameters of the SVM are listed in Table A.8.

XGBoost	number of estimators	1463
	maximum depth	11
	learning rate	0.12915
	gamma	0.0001
	minimum child weight	1
	subsample	0.9
	scale positive weight	0.5
	tree method	gpu_hist
HNM	number of iterations	0

Table A.7: Hyperparameters of XGBoost in the hybrid model using call features.

SVM	kernel	rbf
	degree	13
	C	10
	gamma	100
	maximum number of iterations	2500
	class weight	None
HNM	number of iterations	0

Table A.8: Hyperparameters of the SVM in the hybrid model using call features.

A.2 Multi-Label Classification

This section presents the tuned hyperparameters of the different multi-label classification models.

Improved Batmen

CNN	number of convolutional layers	3
	number of filters	48
	filter size	4
	number of dense layers	2
	number of dense nodes	320
	max pooling size	2
	dropout probability	0.6
	batch size	320
Adam	learning rate	0.00085
	beta 1	0.9
	beta 2	0.999
	epsilon	1×10^{-8}
Early stopping	minimum delta	0.005
	patience	20
HNM	number of iterations	1

Table A.9: Hyperparameters of the improved version of Batmen's network.

Double CNN

CNN	number of convolutional layers	4
	number of filters	48
	filter size	3
	number of dense layers	2
	number of dense nodes	384
	max pooling size	2
	dropout probability	0.5
	batch size	128
Adam	learning rate	0.00032
	beta 1	0.95
	beta 2	0.999
	epsilon	1×10^{-8}
Early stopping	minimum delta	0.005
	patience	15
HNM	number of iterations	same as for its associated classification model

Table A.10: Hyperparameters of the detection CNN of the double CNN architecture.

CNN	number of convolutional layers	3
	number of filters	40
	filter size	4
	number of dense layers	3
	number of dense nodes	512
	max pooling size	2
	dropout probability	0.5
	batch size	320
Adam	learning rate	0.002276
	beta 1	0.9
	beta 2	0.999
	epsilon	1×10^{-8}
Early stopping	minimum delta	0.0005
	patience	5
HNM	number of iterations	0

Table A.11: Hyperparameters of the classification CNN of the double CNN architecture.

Hybrid Models Using CNN Features

This hybrid model is composed of a feature CNN over eight classes followed by either XGBoost or an SVM. The hyperparameters of this feature CNN are the same as the hyperparameters of the classification CNN used in the improved Batmen architecture and can be found in Table A.9. The hyperparameters of XGBoost are listed in Table A.12, while the hyperparameters of the SVM are listed in Table A.13.

XGBoost	number of estimators	2000
	maximum depth	15
	learning rate	0.1968
	gamma	0.001
	minimum child weight	1
	subsample	0.9
	scale positive weight	1.5
	tree method	gpu_hist
HNM	number of iterations	2

Table A.12: Hyperparameters of XGBoost in the hybrid model using CNN features.

SVM	kernel	RBF
	degree	6
	C	10
	gamma	1
	maximum number of iterations	1500
	class weight	None
HNM	number of iterations	0

Table A.13: Hyperparameters of the SVM in the hybrid model using CNN features.

Hybrid Models Using Call Features

This hybrid model is composed of a detection CNN followed by either XGBoost or an SVM. The hyperparameters of this detection CNN are the same as the hyperparameters of the detection CNN used in the double CNN architecture and can be found in Table A.10. The hyperparameters of XGBoost are listed in Table A.14, while the hyperparameters of the SVM are listed in Table A.15.

XGBoost	number of estimators	2000
	maximum depth	15
	learning rate	0.1968
	gamma	0.005
	minimum child weight	1
	subsample	0.8
	scale positive weight	1.5
	tree method	gpu_hist
HNM	number of iterations	1

Table A.14: Hyperparameters of XGBoost in the hybrid model using call features.

SVM	kernel	RBF
	degree	14
	C	10
	gamma	100
	maximum number of iterations	2500
	class weight	None
HNM	number of iterations	0

Table A.15: Hyperparameters of the SVM in the hybrid model using call features.

Appendix B

Performance on All the Metrics

This appendix presents theoretical information on some metrics, the performance on the totality of the metrics that we selected as well as the thresholds used for multi-label classification.

B.1 Metrics

This section presents some theoretical information on the metrics that were used to evaluate the models but that were not shown in the sections presenting our experiments.

F_1 Score

The F_1 score [26] represents the harmonic mean of the recall and the precision. It can be computed as

$$\frac{2 * recall * precision}{precision + recall} \tag{B.1}$$

Balanced Classification Rate (BCR)

The Balanced Classification Rate (BCR) [56], also called balanced accuracy, is the mean between sensitivity and specificity:

$$\frac{sensitivity + specificity}{2} \tag{B.2}$$

where $sensitivity$ is the recall and $specificity$ is given by $\frac{TN}{TN+FP}$.

In the case of an imbalanced dataset, the BCR gives a better indication of the classification performance than the classical metrics such as the accuracy.

Cohen's Kappa

The Cohen's Kappa score [26] measures the degree of agreement between two random categorical variables: the predicted and the actual labels. It thus indicates how similar the model is to a random classifier.

The Kappa score is defined as

$$\frac{c * s - \sum_k^K p_k * t_k}{s^2 - \sum_k^K p_k * t_k} \tag{B.3}$$

where
K is the number of classes.
c is the number of correctly predicted elements.
s is the total number of elements.
p_k is the number of items that are classified as class k.
t_k is the number of items whose actual class is k.

The closer the score is to 1, the more the predicted and actual labels match together. A model with a Kappa score of 0 behaves as a random classifier. A negative Kappa score indicates that the model performs worse than classifying randomly.

Confusion Entropy (CEN)

The Confusion Entropy (CEN) [62] is a metric that was specially designed for the purpose of evaluating multi-class classifiers. It is based on the distribution of the misclassification of the different classes.

Throughout the computation of the confusion entropy, N is the number of class labels and $C_{i,j}$ is the value of the entry at row i and column j in the confusion matrix.

The first step needed to obtain the global confusion entropy is to compute the misclassification probabilities

$$\begin{cases} P_{i,j}^j = \dfrac{C_{i,j}}{\sum_{k=1}^N (C_{j,k} + C_{k,j})}, & i \neq j \quad i,j = 1,...,N \\ P_{i,j}^i = \dfrac{C_{i,j}}{\sum_{k=1}^N (C_{i,k} + C_{k,i})}, & i \neq j \quad i,j = 1,...,N \\ P_{i,i}^i = 0 \end{cases} \tag{B.4}$$

$P_{i,j}^u$ represents the probability of labelling to class j the examples that have i as actual class, subject to class u, which means with respect to the amount of TP, FP and FN of class u. The per-class confusion entropy is computed by

$$CEN_j = -\sum_{k=1,k\neq j}^{N} P_{j,k}^j \log_{2(N-1)} P_{j,k}^j \; - \sum_{k=1,k\neq j}^{N} P_{k,j}^j \log_{2(N-1)} P_{k,j}^j \tag{B.5}$$

If $P_{j,k}^j$ (resp. $P_{k,j}^j$) equals 0, then the first term (resp. the second term) is zero.

The per-class confusion probability is defined as

$$P_j = \frac{\sum_k (C_{j,k} + C_{k,j})}{2 \sum_{k,l} C_{k,l}} \tag{B.6}$$

Finally, the global confusion entropy is the following

$$CEN = \sum_j P_j CEN_j \tag{B.7}$$

Unlike the usual metrics, the closest the CEN is to 0, the better the model performs. Indeed, for a perfect model with a confusion matrix filled with zeros except on the diagonal, the CEN is equal to zero. The worst situation is when the diagonal is filled with zeros, which corresponds to a CEN of 1.

B.2 Multi-Class Performance

This section contains the performance on all the metrics for the different multi-class classification models.

Original Batmen

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.99	0.25	0.94	0.39	0.97	/	/
Classify	0.96	0.86	0.76	0.80	0.87	0.81	0.195
Global	0.99	0.40	0.75	0.48	0.87	0.36	0.017

Table B.1: Performance of Batmen's network. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

Improved Batmen

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.99	0.78	0.90	0.84	0.95	/	/
Classify	0.96	0.86	0.77	0.79	0.87	0.82	0.194
Global	0.99	0.76	0.74	0.73	0.86	0.78	0.004

Table B.2: Performance of the improved version of Batmen's network. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

Double CNN

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.99	0.85	0.84	0.84	0.92	/	/
Classify	0.96	0.85	0.73	0.76	0.85	0.79	0.210
Global	0.99	0.77	0.68	0.70	0.83	0.78	0.004

Table B.3: Performance of the double CNN architecture. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

Hybrid Models Using CNN Features

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.99	0.60	0.94	0.73	0.97	/	/
Classify	0.96	0.85	0.81	0.82	0.89	0.83	0.192
Global	0.99	0.64	0.79	0.69	0.89	0.69	0.006

Table B.4: Performance of the hybrid model using XGBoost and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.99	0.85	0.84	0.84	0.92	/	/
Classify	0.97	0.85	0.83	0.83	0.90	0.84	0.186
Global	0.99	0.78	0.75	0.76	0.86	0.80	0.004

Table B.5: Performance of the detection CNN followed by the hybrid model using XGBoost and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.98	0.17	0.60	0.26	0.79	/	/
Classify	0.93	nan	0.52	nan	0.73	0.63	0.239
Global	0.99	0.40	0.47	nan	0.71	0.23	0.020

Table B.6: Performance of the hybrid model using an SVM and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

Hybrid Models Using Call Features

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.99	0.85	0.84	0.84	0.92	/	/
Classify	0.95	0.75	0.70	0.71	0.83	0.74	0.275
Global	0.99	0.68	0.65	0.66	0.81	0.76	0.004

Table B.7: Performance of the hybrid model using XGBoost and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

	Accuracy	Precision	Recall	F1 score	BCR	Kappa	CEN
Detect	0.99	0.85	0.84	0.84	0.92	/	/
Classify	0.89	0.67	0.51	0.55	0.72	0.46	0.380
Global	0.99	0.63	0.51	0.54	0.75	0.68	0.005

Table B.8: Performance of the hybrid model using an SVM and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics, except Kappa and CEN, are marco-averages over classes 1 to 7 (resp. 0 to 7).

B.3 Multi-Label Performance

This section contains the performance on all the metrics that we selected for the different multi-label models. It also presents the threshold used for each class.

Improved Batmen

	Accuracy	Precision	Recall	F1 score	BCR
Detect	0.99	0.81	0.76	0.78	0.88
Classify	0.94	0.78	0.81	0.79	0.88
Global	0.99	0.71	0.62	0.65	0.79

Table B.9: Performance of the improved version of Batmen's network. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

Thresholds						
1	**2**	**3**	**4**	**5**	**6**	**7**
98	37	33	64	39	81	60

Table B.10: Threshold of each class for the improved version of Batmen's network.

Double CNN

	Accuracy	Precision	Recall	F1 score	BCR
Detect	0.99	0.86	0.69	0.77	0.85
Classify	0.93	0.75	0.79	0.76	0.87
Global	0.99	0.72	0.57	0.61	0.77

Table B.11: Performance of the double CNN architecture. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

Thresholds						
1	**2**	**3**	**4**	**5**	**6**	**7**
99	15	21	36	30	84	26

Table B.12: Threshold of each class for the double CNN architecture.

Hybrid Models Using CNN Features

	Accuracy	Precision	Recall	F1 score	BCR
Detect	0.99	0.78	0.74	0.76	0.87
Classify	0.96	0.83	0.81	0.81	0.89
Global	0.99	0.73	0.65	0.68	0.81

Table B.13: Performance of the hybrid model using XGBoost and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

Thresholds						
1	**2**	**3**	**4**	**5**	**6**	**7**
98	81	96	82	70	92	92

Table B.14: Threshold of each class for the hybrid model using XGBoost and CNN features.

	Accuracy	Precision	Recall	F1 score	BCR
Detect	0.99	0.44	0.36	0.36	0.68
Classify	0.86	0.35	0.46	nan	0.67
Global	0.99	0.35	0.26	nan	0.59

Table B.15: Performance of the hybrid model using an SVM and CNN features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

Thresholds						
1	**2**	**3**	**4**	**5**	**6**	**7**
83	14	32	14	0	18	14

Table B.16: Threshold of each class for the hybrid model using an SVM and CNN features.

Hybrid Models Using Call Features

	Accuracy	Precision	Recall	F1 score	BCR
Detect	0.99	0.87	0.71	0.78	0.86
Classify	0.89	0.56	0.62	0.54	0.76
Global	0.99	0.56	0.51	0.51	0.74

Table B.17: Performance of the hybrid model using XGBoost and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

Thresholds						
1	**2**	**3**	**4**	**5**	**6**	**7**
92	16	4	8	4	7	12

Table B.18: Threshold of each class for the hybrid model using XGBoost and call features.

	Accuracy	Precision	Recall	F1 score	BCR
Detect	0.99	0.64	0.76	0.69	0.88
Classify	0.51	0.20	0.66	0.27	0.54
Global	0.99	0.28	0.56	0.33	0.76

Table B.19: Performance of the hybrid model using an SVM and call features. For `detect`, two classes are considered and the metrics are the ones of class "bat". For `classify` (resp. `global`), all the metrics are marco-averages over classes 1 to 7 (resp. 0 to 7).

Thresholds						
1	**2**	**3**	**4**	**5**	**6**	**7**
22	17	27	6	6	3	4

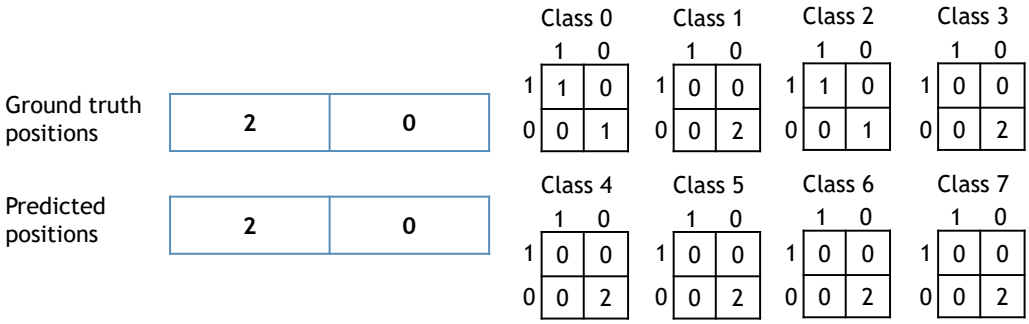
Table B.20: Threshold of each class for the hybrid model using an SVM and call features.

Appendix C

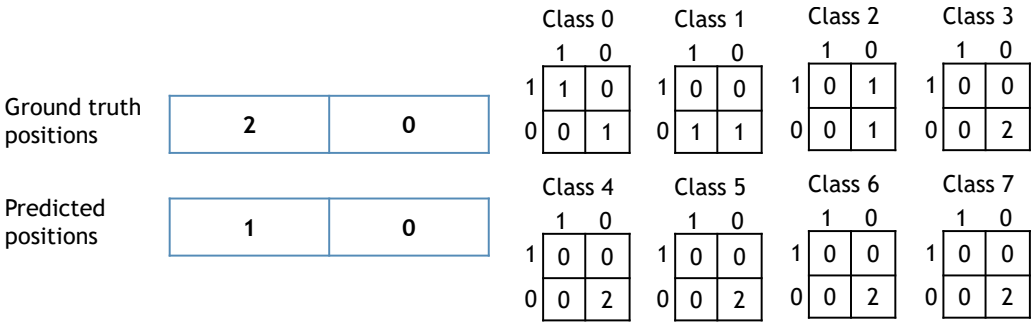
Basic Scenarios of the Multi-Label Confusion Matrices

Four basic cases can occur when evaluating a model that performs multi-label classification of bat calls. These cases are presented in Figure C.1.

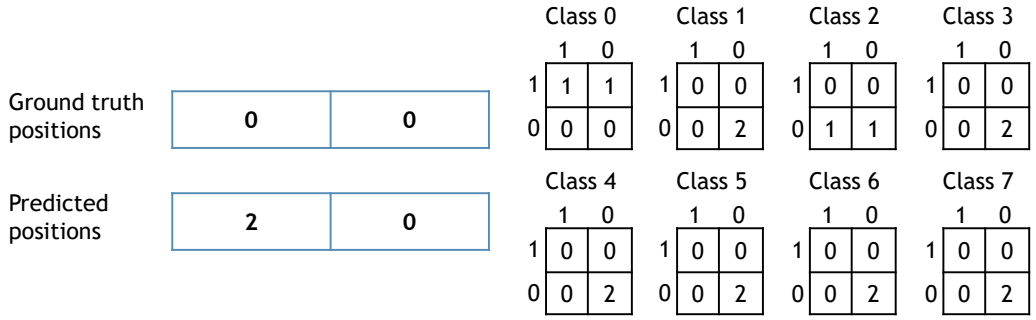
A first scenario, which is illustrated in Figure C.1.a, is when a prediction is made at the correct timing and the correct class is predicted. A second scenario, which is illustrated in Figure C.1.b, is when a prediction is made at the correct timing but the wrong class is predicted. A third case, which is illustrated in Figure C.1.c, is when a call is predicted even though there is no call. The last basic scenario that can happen, which is illustrated in Figure C.1.d, is when no call is predicted even though there is a call.



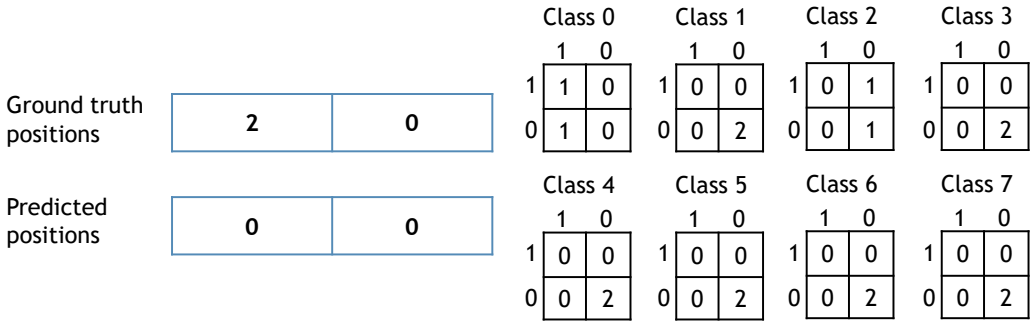
C.1.a. First basic case: Correct timing and correctly predicted class. This is a small example of two windows, represented by rectangles. Each number represents the class of a call that is present in the window. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.



C.1.b. Second basic case: Correct timing and incorrectly predicted class. This is a small example of two windows, represented by rectangles. Each number represents the class of a call that is present in the window. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.



C.1.c. Third basic case: Prediction of a call when there is no call. This is a small example of two windows, represented by rectangles. Each number represents the class of a call that is present in the window. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.



C.1.d. Fourth basic case: No prediction even though there is a call. This is a small example of two windows, represented by rectangles. Each number represents the class of a call that is present in the window. Each class is associated with a binary confusion matrix where the lines represent the ground truths and the columns represent the predictions. The (1,1) entry contains the number of TPs, (1,0) the number of FNs, (0,1) the number of FPs and (0,0) the number of TNs.

Figure C.1: Multi-label confusion matrices on different basic scenarios.

Appendix D

Installation Problems and Solutions

When trying to run the code on the GPU we encountered two different installation problems with TensorFlow.

At first, we installed tensorflow-gpu version 2.0.0b1 but the code was not using the GPU and the following message was displayed:

```
I tensorflow/stream_executor/platform/default/dso_loader.cc:53]
Could not dlopen library 'libcudart.so.10.0'; dlerror:
libcudart.so.10.0: cannot open shared object file: No such file
or directory; LD_LIBRARY_PATH:
/usr/local/cuda-10.2/lib64:/usr/local/cuda-10.2/lib64
```

This was solved by uninstalling tensorflow-gpu since in the newer versions of TensorFlow the GPU is already included without needing to install a specific library.

After uninstalling tensorflow-gpu, we installed tensorflow 2.1.0. However, this caused another error:

```
Could not create cudnn handle: CUDNN_STATUS_INTERNAL_ERROR
tensorflow.python.framework.errors_impl.UnknownError: Failed to
get convolution algorithm. This is probably because cuDNN
failed to initialize, so try looking to see if a warning log
message was printed above.
```

To deal with this issue, the following lines were added to the `run_training.py` and `run_detector.py` files:

```
config = tf.compat.v1.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.compat.v1.InteractiveSession(config=config)
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**