BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Fachbereich VI Informatik und Medien

Master's Thesis

submitted by

Teodor-Constantin Chiaburu

for the degree of

Master of Science (M. Sc.)

in the Department of Data Science

Hierarchical Classification of Insect Species Using Wingbeat Signals

Supervisor: Prof. Dr. Frank Haußer

Examiner: Prof. Dr. Felix Bießmann

Submission date: 8.04.2021

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Abschlussarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Berlin, den Unterschrift: .....

Abstract

This thesis has been written within the research project *KInsekt (AI-based Insect Monitoring with Citizen Science)* at the Beuth University of Applied Sciences in Berlin. Its main objective is to investigate Deep Learning techniques to hierarchically classify insect species, based on photoacoustic signals generated from their wingbeats. The software presented here is developed in Python with TensorFlow, Keras, Scikit-Learn and is meant to be flexible and easy to adapt to changing aspects of the classification task such as new species added to the dataset or more taxonomic levels. The whole Machine Learning pipeline from data cleaning, handling of imbalanced data and hyperparameter tuning to careful evaluation of the prediction performance of various models is provided, including some tests with XAI methods. The networks conceived here can run efficiently on small devices like a Raspberry Pi and achieve performance scores comparable to the benchmarks published in the seminal papers cited [20] [21] [31]. Moreover, it will be shown that encoding the hierarchical structure of the insect taxonomy into the classifiers in the form of Hierarchy-based Class Embedders helps construct more reliable and robust models that successfully identify relevant wingbeat features.

Zusammenfassung

Die vorliegende Arbeit wurde im Rahmen des Projekts *KInsekt (KI-basiertes Insektenmonitioring mit Citizen Science)* an der Beuth Hochschule für Technik Berlin erstellt. Das Hauptziel ist, Methoden des Deep Learning zu untersuchen, um Insektenarten anhand der von ihren Flügelschlägen erzeugten fotoakustischen Signale hierarchisch zu klassifizieren. Die dafür erstellte Software ist in Python mit TensorFlow, Keras, Scikit-Learn geschrieben und kann flexibel und leicht an sich verändernde Aspekte der Aufgabenstellung angepasst werden wie z.B. das Hinzufügen neuer Insektenarten oder weiterer taxonomischer Stufen. Die gesamte Machine Learning Pipeline vom Data Cleaning, dem Umgang mit unbalancierten Daten, Hyperparameteroptimierung bis hin zur sorgfältigen Evaluation der Vorhersagepräzision der neuronalen Netze und dem Einsatz von XAI-Methoden zur Modellerklärung wird dargestellt. Die hier konzipierten Klassifikatoren laufen effizient auf kleinen Geräten wie einem Raspberry Pi und erreichen Präzisionswerte vergleichbar mit den Ergebnissen aus der Literatur [20] [21] [31]. Es wird außerdem gezeigt, dass die Berücksichtigung der hierarchischen Struktur des taxonomischen Baums in Form von Hierarchy-based Class Embedders zur Konstruktion von zuverläßigeren und robusteren Modellen beiträgt, die erfolgreich die relevanten Flügelschlagmerkmale identifizieren.

Contents

Introduction

In the past decade, modern society has witnessed an incredible spike in high-end technology. Increasingly intelligent systems, which people sometimes carry in their pockets every day, overtake a lot of arduous tasks making our lives invariably easier. It can only be expected that the ever-expanding fields of Artificial Intelligence and Machine Learning will become better and better at tackling difficult problems, till now out of humans' reach. In this respect, Deep Learning - a subfield of Machine Learning - has gained much attention lately, especially since higher computing power on GPUs and enormous amounts of data have become available.

The first milestone was reached in 2012 when a team of researchers broke all accuracy barriers and won the ImageNet classification challenge with their *AlexNet* model [24] - a Convolutional Neural Network (CNN). Since then, everyone resorted to Deep Learning and built their own CNNs to try and achieve even higher scores in the subsequent competitions. Nowadays, deep neural networks are the first choice not only when dealing with image classification tasks but also other complex problems ranging from text analysis, speech recognition, autonomous driving to even solving systems of partial differential equations to predict fluid dynamics [13].

Framing the Problem

A study published in 2017 [12] reported a 75% decline in insect biomass in German nature protected areas over the past 27 years, mainly because of pesticides, intensive farming and climatic changes. But not only the overall number of insects has dramatically sunken. The 33,000 different species that used to find their habitat in Germany are rapidly vanishing. The figures are shockingly high, so multiple organizations along with the German Ministry of Environment have joined forces to curb the dramatic decrease of insect diversity.

Steps have already been taken to determine which areas are primarily affected by insect depopulation and which species face a high risk of extinction [9] [2]. This

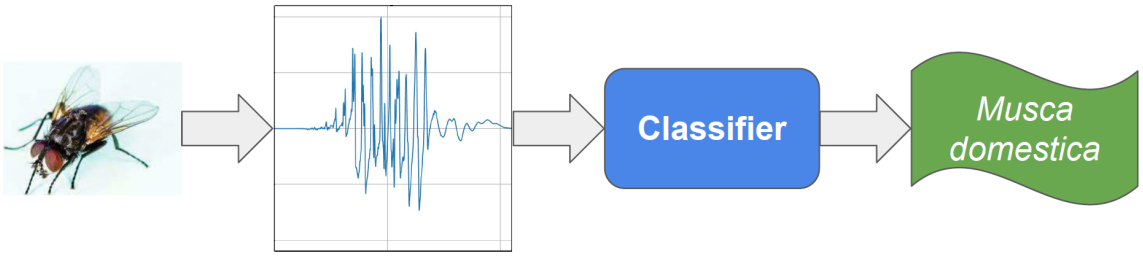


Figure 1.1. Outline of the classification process: The insect[3] flies through a light beam while a phototransistor translates the light perturbance into an optoacoustic signal. This is given as input into a classifier which then assigns one or more labels to the sample according to what taxonomic levels it has been taught. In this example, both genus - *Musca* - and species - *domestica* - are predicted.

is called *Insect Monitoring* and the project this thesis is part of, *KInsekt*[1], also contributes to the common efforts of understanding species variability and devising modern unharmful[2] methods of counting insects. These methods will be conceived and validated with the help of current state-of-the-art Machine Learning techniques.

The task at hand will be formulated as a *supervised classification problem*, where the input comes in form of a photoacoustic signal produced by a flying insect (Figure 1.1). In short, a light sensor detects the flight of the insect and translates it into a *wingbeat signal* (Figure 1.2). Usually, in a clean wingbeat signal, one can observe the high-frequency oscillations - responsible for the actual wingbeat - superimposed on a low-frequency tone - the trace of the body movement. Such samples are then classified with respect to one or more taxonomic levels, such as species or genus. Whether the wingbeat signals in their raw form are informative enough for the insects to be accurately classified will be investigated throughout the upcoming chapters. Alternative representations are given in Figure 1.2. Both the Fourier spectrum and the PSD transformation (see Section 3.2.1) emphasize particular frequencies through amplitude peaks. Ignoring the very first peak close to 0 Hz, which is caused by the body movement, the other peaks correspond to the *fundamental frequency* and its *higher harmonics*, which give the characteristic sound of a mosquito. This paper will focus on wingbeat signals solely, but the team at *KInsekt* is aiming at creating a multisensor classification system in the future, that also takes some other forms

[1]Part of the initiative *KI-Leuchttürme für Umwelt, Klima, Natur und Ressourcen* [22].

[2]A classical way of approximating insect biomass is to collect the insects in a *Malaise trap* and count them manually. In this process, insects most often die. Meanwhile, technology would allow automatic counting that spares insects' lives.

[3]Image source: By USDAgov, downloaded from https://de.wikipedia.org/wiki/Stubenfliege (license: https://commons.wikimedia.org/wiki/File:Common_house_fly,_Musca_domestica.jpg)

of input into account such as camera snapshots.

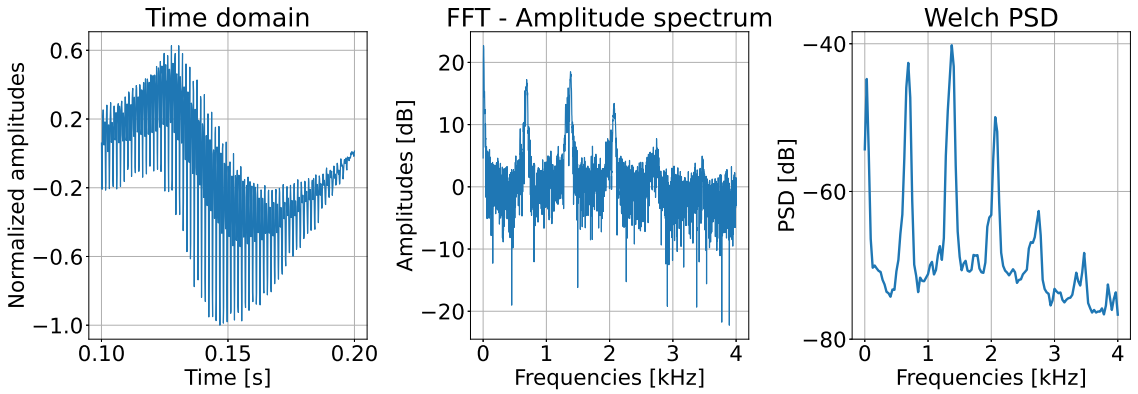


Figure 1.2. Visualizing wingbeat frequencies of a mosquito signal. From left to right: normalized raw wingbeat signal (originally 0.65 seconds long but zoomed in here to highlight the oscillations); Fourier spectrum (converted into dB), Power Spectral Density plot obtained through the Welch Transform (see Section 3.2.1) - values also converted into dB.

Related Work

Well-known wingbeat datasets from the established literature will be used as training and validation data: Potatamis et al. [31] and Keogh et al. [20] [21]. Both researcher teams have run wingbeat classification experiments and have achieved high-performance scores. Potatamis et al. report 96% accuracy on the species level by training CNNs on spectrograms. Keogh et al. have investigated standard ML techniques such as kNN, Random Forests or Support Vector Machines and also other feature sets like Mel-Frequency Cepstral Coefficients or circadian rhythm timetables. Their approaches have also reached accuracies beyond 90%.

This paper is inspired by their work, yet intends to branch out into other areas of research as well. While the datasets have short fixed hierarchies (Potatamis: 3 genera, 6 species; Keogh: 4 genera, 9 species), the software developed here should be easy to adapt to more ample taxonomies, too. In the long run, *KInsekt* plans to reach a classification volume of between 50 and 100 species. Given the sparsity of some of these species, teaching classifiers on little data will, in part, be a constraint to consider, so data augmentation techniques will inevitably come into play. Apart from being accurate, the models in this thesis are required to run efficiently on small devices such as a Raspberry Pi, so that they can be inserted outside in the field and recognize insects in real time. Lastly, some preliminary work will be done in trying to explain what the networks base their decisions on (known as *Explainable AI*).

Hierarchical Classification

It should be highlighted straight from the start that the 'hierarchical' part in the thesis title is not to be understood in classical ML terms. The reader might be tempted to think about methods devised to classify instances based on a hierarchical structure of their features, such as *Decision Trees* or *Dendrograms* in Hierarchical Clustering. In those situations, the assumption is that the dataset can efficiently be separated into subspaces of the feature space. Thus, the goal is to predict the final class labels by moving from coarse features (for instance, think of basic geometry figures in images such as lines and corners) to increasingly finer features (think of more complex irregular shapes that might define the face of a particular person).

This is not the case with the hierarchical classification of wingbeat signals. Insects, just like plants, are categorised according to a taxonomic tree (more on this in the next chapter). Similarly to a Decision Tree, at each level, a new set of features determines the division of the current node into subsequent nodes. For example, the *class* of flying insects is divided into multiple *orders* according to the type of wings they have: *Hymenoptera, Coleoptera, Diptera* and so on. Based on other morphological characteristics, the order *Diptera* is further split into flies, mosquitoes and so on [45]. Nevertheless, these are not the features taken into consideration by the classifiers described in this paper. The input features, as will more thoroughly be explained in later sections, are either amplitudes of sensor-recorded signals or other processed versions of them, such as spectrograms or Power Spectral Density profiles. What the classifiers do aim at is to map the biological tree as accurately as possible, namely to try to predict every taxonomic level known in the data in a top-to-bottom fashion, for example, first the order, then the family, the genus and, finally, the species. This kind of task is also known as *structured output prediction* and it requires data to have an inherent hierarchical semantic structure.

The developers of *YOLO* resorted to this concept in 2016 when they released the second version of their object detection network [35]. With the help of *WordNet*, a language database that structures concepts and how they relate, they managed to merge the labels from *COCO* (a dataset with more general classes used for detection) and *ImageNet* (which has more specific labels used for classification). Terms like 'dog' and 'Norfolk terrier' would not be mutually exclusive but connected now. The immediate benefit was that the model could still, for instance, reliably classify an image of a dog as being a dog, even if it was not sure what exact breed it was.

This thesis has also been conceived in this direction. The objective is to develop flexible models capable of predicting different levels of hierarchy. In the case of insects, some taxonomic levels may not be known for particular samples (for example, the order, family and genus could be determined, but the species is not available). In such a situation, the classifier should still be able to deliver predictions for all the known levels. And secondly, similar to the *YOLO* example, if the model is unsure regarding the species (maybe because the specimens are rare), then it is expected that at least predictions in the superior taxonomic levels are reliable. This way, every sample gets classified to a certain extent. A simple classifier with a classical architecture would not have that flexibility.

In this paper, hierarchical classification will be implemented with the help of *Hierarchy-based Class Embeddings* [1]. Other approaches involving Deep Conditional Generative Models can be found in [38].

Roadmap

This thesis contains four main chapters. Chapter 2 will present the data that has been worked with as well as the technicalities of acquiring it. Chapter 3 covers the theoretical background. Explaining the principles involved in data cleaning, data preprocessing, methods to avoid overfitting or hierarchical embeddings will be the central focus of this section. Chapters 4 and 5 will deal with the concrete implementation of the methods presented previously. The whole training pipeline will be unravelled here and broken down into three main stages: Hyperparameter Optimization and Transfer Learning (covered in Chapter 4) and Testing/Validation along with some preliminary work in the field of Explainable AI (in Chapter 5).

Data

This chapter will offer an overview of what kind of data the experiments in this paper rely on. The first section talks about the taxonomy of insects and defines relevant biological concepts connected to hierarchical classification. The subsequent section illustrates the data acquisition process, namely what wingbeat signals are and how they are recorded. Lastly, the two datasets utilised in this work are described and a first glimpse of what the insect recordings look like is given.

2.1 Insect Taxonomy

According to [44], the term *biodiversity* stands for the whole variability of life, referring not only to the sum of different categories of living beings (qualitative description), but also to other aspects such as where on the globe a category is observed (localization) and how well represented it is (quantization).

Biodiversity can be formalized on three levels:

1. **genetic diversity**: all individuals belonging to the same species and living simultaneously within the same area (also referred to as *population*)

2. **species diversity**: hierarchical categorization of individuals belonging to different species (also referred to as *taxonomy*)

3. **environmental diversity**: all the different biotopes/habitats along with the biological processes active within them; together they build *eco-systems*

This paper will be focusing predominantly on the second type of diversity, given that the main objective here is to conceptualize methods for differentiating species from one another. Also noteworthy is that the term *species* admits multiple definitions [44], such as:

1. **morphological definition**: A species represents all the individuals akin to one another based on their morphological features. In the case of insects, these may be the wing type, the number of feet, colorization and so on[1]. Sometimes labelling insects is not as straightforward as expected, mainly because of a phenomenon called *convergence*. This occurs when populations belonging to intrinsically different species live and evolve together in the same biotope, sharing the same resources and dealing with the same environmental conditions over a long period. Thereby, they tend to develop very similar characteristics (e.g. color pattern on the wings, defensive mechanisms, scent or sounds during flight), making it quite challenging to hold them apart.

2. **biological definition**: A species is collectively described as a community of individuals capable of reproducing among themselves and naturally tending to do so. They do not interact sexually with other communities.

3. **phylogenetic/evolutionary definition**: A species is a genetic branch sharing the same DNA sequence.

It is of great importance to the upcoming investigations to cast the concept of 'species' into its taxonomic frame. Taxonomy consists of the following hierarchical layers [45]: Domain, Kingdom, Root, Class, Order, Family, Genus, Species. Technically speaking, insects are a class - *Insecta* - so only the last four layers will be relevant for their categorisation. The focus in this paper will lie even more specifically on genus and species[2], but all the methods developed here are meant to be extended and applied to the next superior levels in future works. There are over 80,000 known insect species on the globe [45], which poses some serious difficulties on the task of classifying them with AI.

2.2 Wingbeat Signals

As stated in the title, the main focus of this work are the wingbeat patterns of flying insects. Therefore, as far as data acquisition is concerned, special equipment is needed to record insects' wingbeat frequencies during their flight. Since unrestricted

[1]From the point of view of Machine Learning, this paper will mainly revert to this definition because wingbeat and image data encompasses morphological characteristics. Other classification techniques such as DNA-Barcoding make use of the third definition.

[2]Please note that only the species exist as actual living beings; all the other superior layers are abstract representations.

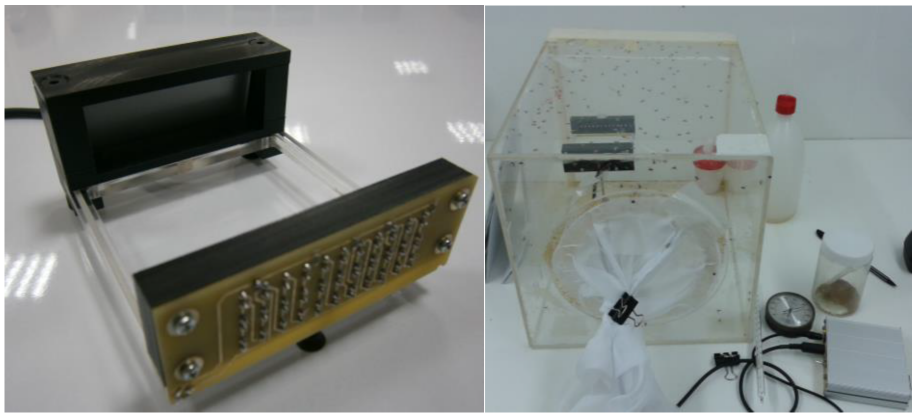


Figure 2.1. Left: array of diodes. Right: diode sensor placed in the insectary cage (Pictures from [32])

free flight is quite challenging to record, the wingbeat data meant for training models is gathered under controlled conditions in a laboratory. Concretely, insects of the same species are confined to the boundaries of a cage and a special device notices whenever they initiate a flying session.

A classical approach to record the wingbeats would be to use a microphone. Nonetheless, audio sensors would also record all the surrounding noise, making it difficult afterwards to discern the true insect flight from the background noise. In this respect, optical devices are much more efficient. Potatamis et al. used photodiodes (infrared LEDs) [32], which is what the *KInsekt* team also plans to do[3]. Whenever an insect flies through the light beam, the optical disturbance is recorded and translated into an audio signal (Figure 2.1). Such a device does not 'hear' anything outside the light beam[4] but is very sensitive to any light occlusions, even those caused by the soft wingbeats of small insects (i.e. moths or fruit flies) which a microphone could hardly detect.

After gathering enough ground truth data and training models up to satisfactory performance scores, the next step is to deploy the classifiers in real-life scenarios. For this, an automatic system is needed that attracts the insects and classifies them on the spot without requiring regular human intervention (like in the case of the Malaise traps mentioned in the Introduction).

[3]Keogh et al. used instead a laser sensor [20].

[4]The transistors do, however, record some background information such as vibrations from the device, but the noise intensity can be measured and subsequently subtracted.

2.3 Datasets

In the absence of own data, two well-known insect datasets from the existing literature have been utilized:

1. A mosquito wingbeat database published by Potatamis et al. [31] in 2018, which can be found on Kaggle[5] (referred to as *the Potatamis dataset* from now on)

2. A database of various insects' wingbeats published by Keogh et al. [20] in 2014, also available online[6] (referred to as *the Keogh dataset* from now on)

Following is a description of both.

2.3.1 Potatamis Dataset

The Potatamis dataset consists of 3 mosquito genera: *Aedes, Anopheles, Culex*, with two species each (Figure 2.2). According to [31] the data was recorded in a mosquito breeding establishment at *Biogents* in Regensburg, Germany. Every time an 'event' occurred, that is, an insect passed through the light beam, a snippet of 0,625 seconds would be cut from the transistor recording and stored in the database. At a sampling rate of 8 kHz, this meant that each sample would be 5000 amplitude points long (Figure 2.3). The dataset contains a total of 279,566 *wav*-files.

(a) **(b)** **(c)**

Figure 2.2. 3 species of mosquitoes in the Potatamis dataset[7]: (a) Aedes aegypti, (b) Aedes albopictus, (c) Anopheles gambiae

[5]https://www.kaggle.com/potamitis/wingbeats
[6]https://timeseriesclassification.com/description.php?Dataset=InsectWingbeat
[7]Sources of the mosquito images:
(a) By Muhammad Mahdi Karim, downloaded from https://en.wikipedia.org/wiki/Aedes_aegypti (license: https://www.gnu.org/licenses/old-licenses/fdl-1.2.html)
(b) By James Gathany, downloaded from https://simple.wikipedia.org/wiki/Asian_tiger_mosquito (license: https://commons.wikimedia.org/wiki/File:Aedes_Albopictus.jpg)
(c) By James Gathany, downloaded from https://en.wikipedia.org/wiki/Anopheles_gambiae (license: https://commons.wikimedia.org/wiki/File:AnophelesGambiaemosquito.jpg)

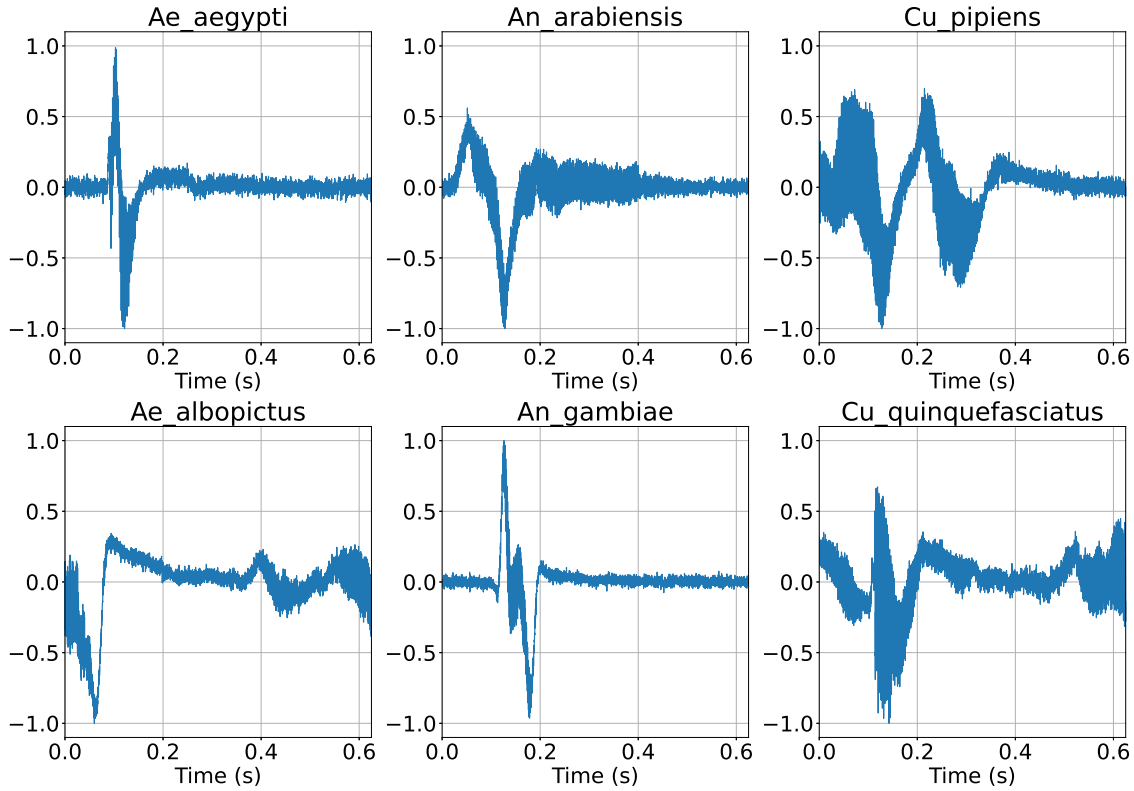


Figure 2.3. Examples of raw signals for each species in the Potatamis dataset. All samples are 5000 points long and have been normalised between [-1, 1]. Both genus and species are denoted by the title: the first two letters are the abbreviation for genus, the following term stands for species.

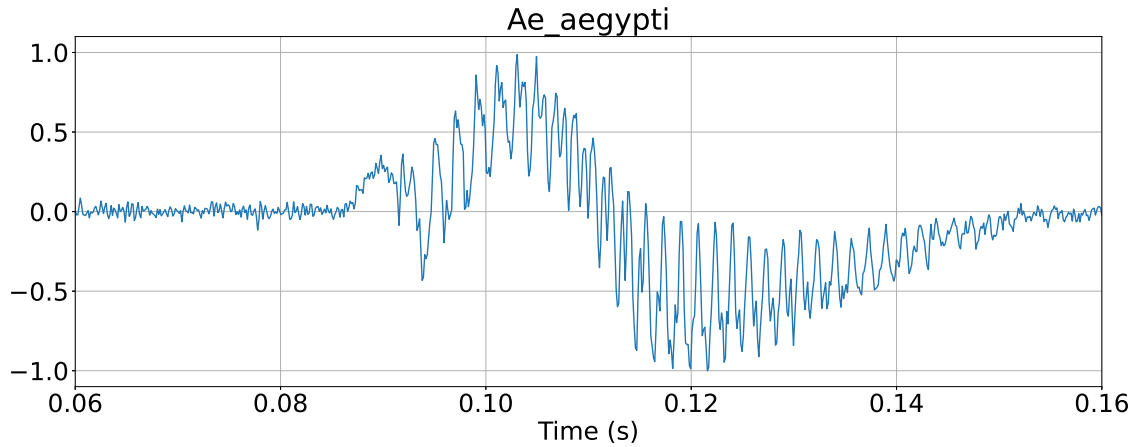


Figure 2.4. A zoomed-in view of the *aegypti* signal from Figure 2.3. You can approximately read the base wingbeat frequency from the plot itself by counting the peaks in an interval. For instance, there are 5 peaks from 0.10 to 0.11 seconds, which means the frequency is about $5/0.01 = 500$ Hz.

The wingbeat patterns are not visible in the side-by-side plots. For this, a close-up view is required like in Figure 2.4. Notice the repeating oscillations that start at around 0.09 seconds and fade away at 0.16 seconds. This is the mark of a flight event. One can even determine the approximate base frequency (see Section 3.3.1) by counting the peaks, for example, within 0.01 seconds of the 'eventful' part of the signal. There are 5 peaks between 0.10 and 0.11 seconds, which gives a frequency of 500 Hz.

2.3.2 Keogh Dataset

The Keogh dataset is made up of 9 insect species, some of them intersecting with the Potatamis dataset (Figure 2.5). The signals here were acquired differently: hours of audio material were first stored, after which a *Fourier detector*[8] [21] would swipe along the whole recording in search for wingbeats.

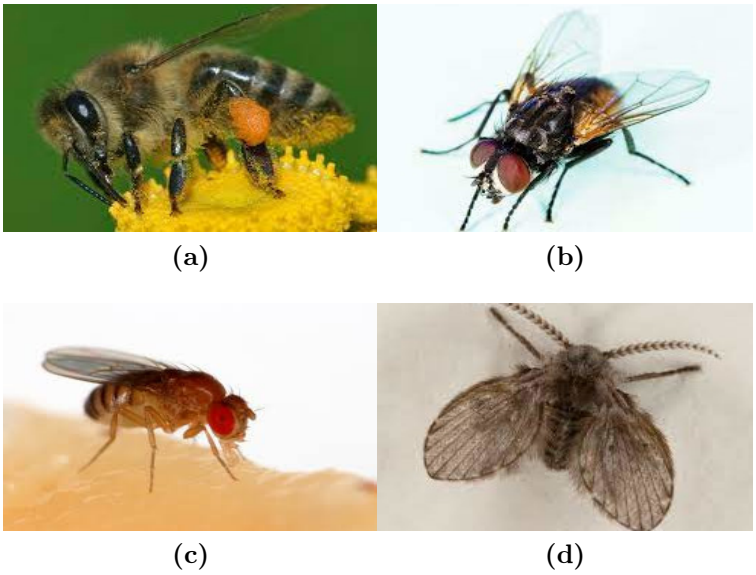


Figure 2.5. 4 insect species in the Keogh dataset[9]: (a) Apis mellifera, (b) Musca domestica, (c) Drosophila melanogaster, (d) Psychodidae diptera

[8] A similar detector will be devised for outlier removal in Section 3.1.4.

[9] Sources of the images:
(a) By Andreas Trepte, downloaded from https://en.wikipedia.org/wiki/Western_honey_bee
(license: https://creativecommons.org/licenses/by-sa/2.5/)
(b) By USDAgov, downloaded from https://de.wikipedia.org/wiki/Stubenfliege
(license: https://commons.wikimedia.org/wiki/File:Common_house_fly,_Musca_domestica.jpg)
(c) By Sanjay Acharya, downloaded from https://en.wikipedia.org/wiki/Drosophila_melanogaster
(license: https://creativecommons.org/licenses/by-sa/4.0/)
(d) By Sanjay Acharya, downloaded fromhttps://en.wikipedia.org/wiki/Psychodidae
(license: https://creativecommons.org/licenses/by-sa/4.0/)

The signals cut in this manner were unequal in length, so they had to be cropped down to a common length first. After cropping, all samples had a duration of roughly 0,1 seconds, which at a frequency rate of 16 kHz amounts to 1680 amplitude points.

There is a total of 18115 *wav*-files. The labels in Figure 2.6 have been taken directly from the dataset. For this thesis, the true taxonomic tree corresponding to the Keogh insects has been replaced by a simplified genus-species-hierarchy (Figure 3.26). While not biologically accurate, this representation still serves the purpose of investigating neural networks meant to recognize hierarchical structure in data. Ultimately, the goal here is to conceive and validate methods for classifying insects according to a taxonomic scheme, whichever that may be.

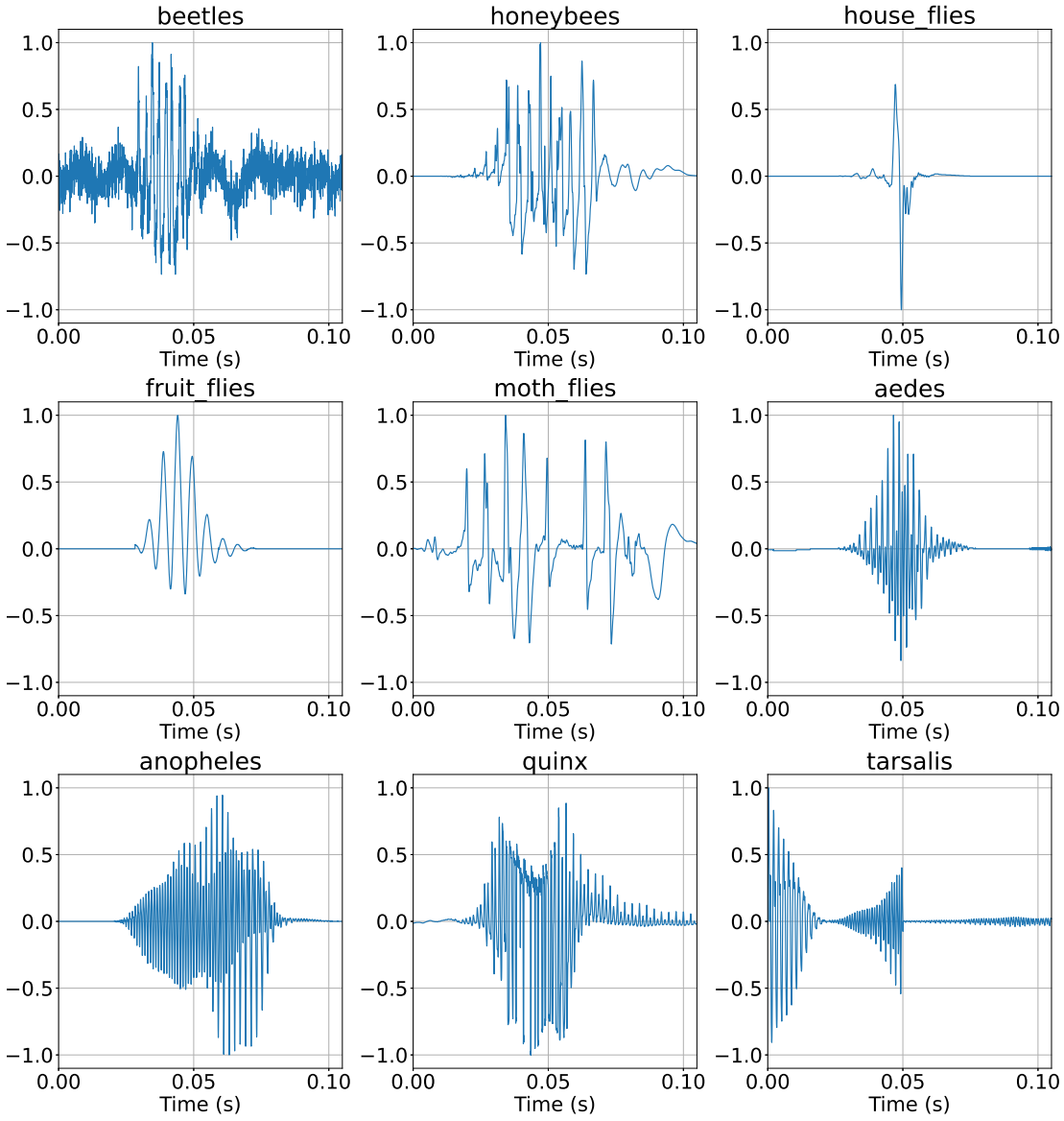


Figure 2.6. Examples of cropped raw signals for each species in the Keogh dataset.

Methods

This chapter is meant to lay the theoretical foundations needed by the whole ML framework constructed in this paper. The layout of the upcoming sections will follow the classical ML development pipeline. First and foremost, the *Data Cleaning* operations will be described. Most of the cleaning steps will be especially relevant for the Keogh dataset, since the signals there are unequal in length, noisy and may also contain potential outliers. Next, *Power Spectral Density* and *Spectrograms* are presented as alternative input formats for the plain raw signals. The question is whether the raw versions of the recordings have already captured all the necessary information needed for classification. The *Data Visualization* section talks about using the fundamental wingbeat frequency as a classifying discriminant and makes an attempt at clustering the data through the *t-SNE* transformation. The most extensive section, *Classification with CNNs*, will discuss the well-known problems of overfitting and imbalanced data and propose solutions to deal with them. Transfer Learning strategies are incipiently presented, since one important requirement from the models is to generalize well and be easily adjustable to new data. Lastly, the method for embedding class vectors, meant to integrate hierarchical structure into the classification process, is explained.

3.1 Data Cleaning

3.1.1 Normalization

It is common knowledge that *Stochastic Gradient Descent (SGD)* converges faster when the features are roughly on the same scale. In the case of the raw signals, the features are the amplitude values recorded by the sensor. While the amplitude scales do not differ greatly from sample to sample, insects that fly closer to the sensors are perceived as being louder than others flying slightly farther away. This may confuse

the classifiers, so, to equalize the loudness throughout the dataset, each sample is L_∞-normalized[1] by dividing it by its maximum absolute amplitude (Equation 3.1).

$$\hat{x} = \frac{x}{||x||_\infty} = \frac{x}{max(|x|)} \tag{3.1}$$

Throughout this paper, whenever a raw signal is described or plotted, its normalized version from Equation 3.1 is implied. Later in the section *Data Preprocessing* all transformations are also applied to the normalized signals.

At this point, an important implementation detail needs to be mentioned, which raised a few question marks during the early stages of the experiments. Both datasets consist of wav-files. This means one needs to first read the audio signals into a format that allows further manipulations within a Python environment (e.g. Numpy arrays). There are two well-known methods to do this: *scipy.io.wavfile.read()* and *librosa.load()*. However, there is a noteworthy difference between them: while the former loads the amplitudes as *int16*-points, the latter converts them into *float*-points by mapping the integers into the interval [-1, 1]. It is also the reason why the *librosa* method is much slower than its *scipy* counterpart. Beware that the conversion applied by *librosa* is a different transformation than the one in Equation 3.1 because the denominator is not the highest occurring absolute value, but the highest possible value in the data type range, namely $2^{16} - 1$ for *int*16. To draw a line, irrespective of the loading method, one still needs to apply the formula in 3.1 to normalize each signal individually.

3.1.2 Cropping

As opposed to the signals in the Potatamis dataset, those in the Keogh dataset are not of equal length. As one can see in Figure 3.1, signals may last as little as a tenth of a second or be longer than four seconds. One requirement of the classifiers is that all inputs have the same shape. Therefore, a method needed to be devised to crop the signals down to a common length. One such common length may simply be the number of points in the shortest signal found in the whole dataset (left plot

[1]Some signals in the Keogh dataset exhibit what is called a *Delta-peak* - a relatively high amplitude value that drastically reduces any contribution of other amplitudes (the signal then resembles a *Dirac-Delta impulse*, hence the name of the peak). An L_∞-normalization divides the vector by a large number. Its small components lead to small Fourier coefficients for the relevant frequencies between 100 and 1000 Hz, which further leads to the signal being discarded as an outlier, although a wingbeat pattern can still be observed in the spectrum (see section Figure 3.7). Alternatively, one may consider using other norms, such as L_1 or L_2.

in Figure 3.1). At a sampling rate of 16000 samples per second, a signal with a duration of roughly 0,105 seconds will be composed of $16000 \cdot 0{,}105 = 1680$ points. This is the fixed length that all the samples will be cut down to.

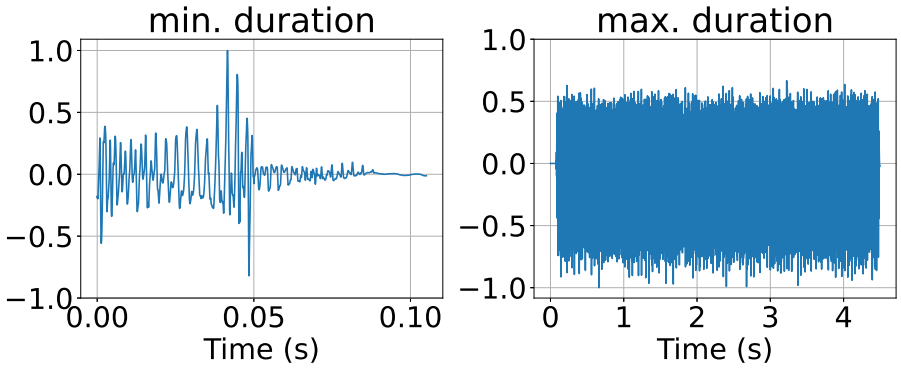


Figure 3.1. Extreme signal lengths in the Keogh dataset. The shortest signal lasts only about 0.1 seconds, while the longest has 4.5 seconds.

The cropping method is fairly straightforward and depends on three parameters:

- l_{min}: the length the signal will be resized to
- n: length of the (overlapping) segments the signal will be divided into
- h: the amount each segment is shifted through the signal

For the upcoming experiments, the parameters were chosen as follows: $l_{min} = 1680$, $n = \frac{l_{min}}{2}$ and $h = \frac{l_{min}}{16}$. Naturally, the segments will be overlapping each other by $n - h = \frac{7 \cdot l_{min}}{8}$. Within each frame s of length n the root mean squared value of the amplitudes is computed:

$$RMS(s_i) = \frac{||s_i||}{\sqrt{n}} \tag{3.2}$$

Afterwards, the frame with the greatest RMS is selected as the center of the new cropped signal. To reach the desired l_{min}, the frame is subsequently extended by a total of $\frac{l_{min}}{2}$ points to its left and/or to its right, depending on how many points on both sides of the frame are still available from the original signal (see Figure 3.2).

3.1.3 Denoising

One species in the Keogh dataset that appears to have a greater amount of noise than the others are the beetles - see Figure 2.6. The question that immediately

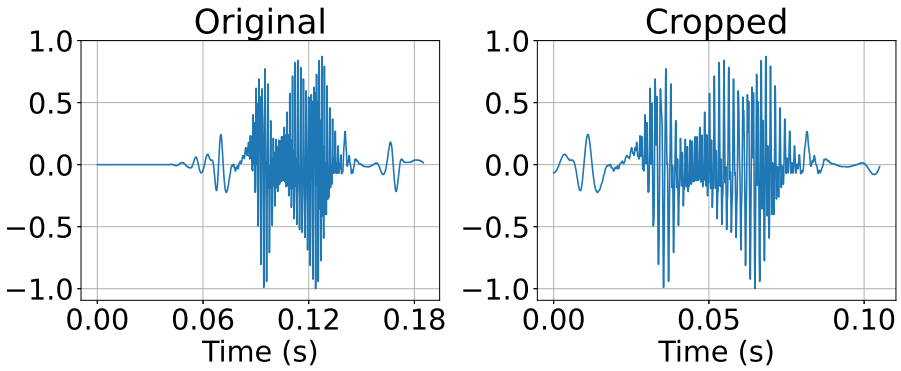


Figure 3.2. Comparison between a signal before (left) and after being cropped (right).

arises from this observation is whether the classifiers are going to learn to recognize the beetles based on this noise rather than their inherent features (a so-called *Clever Hans Effect* [4]). However, on a closer inspection, not all beetle samples are that noisy (Figure 3.3). Those that are though exhibit higher Fourier and PSD coefficients above 2 kHz (compare the FFT and PSD plots in the first two rows in Figure 3.3). These prominent coefficients also diminish the contrast between the characteristic wingbeat frequencies and background noise that one would expect a spectrogram to highlight. Notice it is harder to distinguish the fundamental frequencies in the second spectrogram, while in the first one they stand out.

In such situations, one may consider applying a low-pass filter to denoise the 'unclean' samples. In [20] the signals were denoised by applying a technique called *spectral subtraction*. Replicating this process was not possible because a 'baseline' signal containing only the background noise (without any insect flying through the sensor) would have been needed, whose frequencies were to be subtracted from the noisy samples. As an alternative, the perturbed signals could be passed through a low-pass Butterworth filter (Figure 3.4). While this does remove some noise and reduces the contribution of the high-frequency artifacts to some extent (3rd row in Figure 3.3), the improvement is not considerable. Harsher 3rd and 4th-degree filters have also been tested but it turned out that they were artificially marking the signals by cutting off high frequencies too abruptly, which brings back the same problem of the Clever Hans Effect[2].

Considering all the above arguments and, additionally, given that it is bad practice to apply a preprocessing step only to a subset of the data, not to the whole dataset, no filter has been applied on any of the species.

[2]Chapter 5 will further investigate whether the trained models are sensitive to this effect.

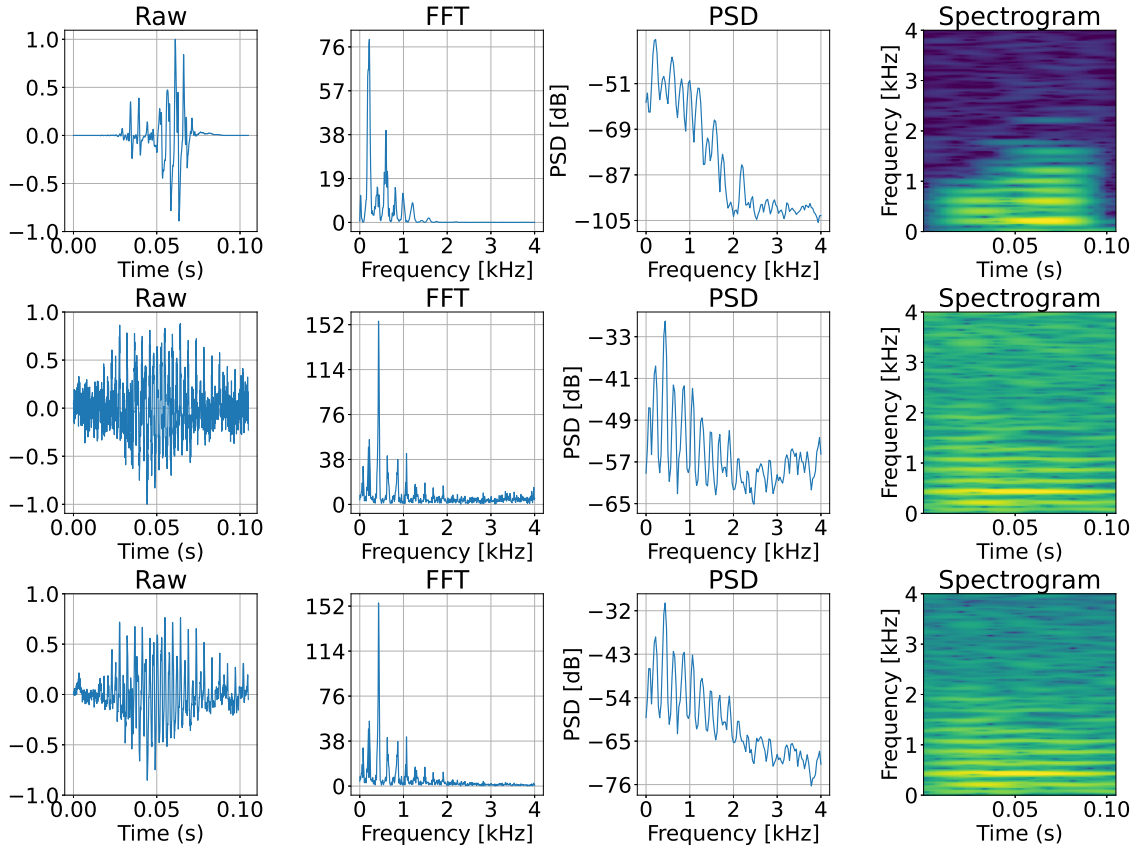


Figure 3.3. Inspection of beetle samples over four signal formats: original raw amplitudes (1st column), amplitude of the Fourier spectrum (2nd column), Welch-PSD transformation (3rd column, see Section 3.2.1) and spectrogram (4th column, see Section 3.2.2). Frequencies above 4 kHz have been dropped and the colormaps of the three spectrograms have been rescaled to the same min-max-scale. The first row depicts a 'clean' beetle signal. The second row shows a noisy beetle and the third one its filtered version (a Butterworth filter like in Figure 3.4 has been applied).

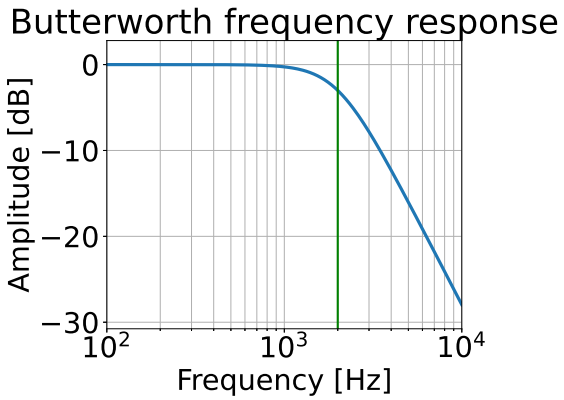


Figure 3.4. Second-degree low-pass Butterworth filter evaluated on frequencies ranging from 100 to 10000 Hz. The cutoff frequency is at 2000 Hz (marked in green).

3.1.4 Outlier Detection

Another important function of any data cleaning toolbox is the outlier detection and removal. The large number of samples makes it impractical to search for anomalies manually, hence the need to develop an automatic detector. The algorithm used in this paper is inspired by [20] and [21] and described below:

1. Each normalised signal is Fourier-transformed.

2. In the amplitude spectrum the maximum Fourier coefficient between 100 and 1000 Hz is calculated. According to [21], this is the range where insect wingbeat frequencies are most likely found.

3. For each species, a histogram of these maximum Fourier coefficients is built.

4. The signals that delivered the coefficients corresponding to the first histogram bin (or first few bins) are further inspected. These will most often be good outlier candidates.

Figures 3.5 and 3.6 are depictions of this algorithm. The red line is the *log-normal* curve fitted to each histogram, strengthening the hypothesis that the maximum Fourier coefficients have a logarithmic distribution (see Equation 3.3). For each fitted curve, the mean value μ and standard deviation σ of the logarithm of the maximum coefficients were needed. Notice how well the histograms for the Potatamis dataset resemble a true log-normal distribution for all species. Of course, one might argue that the Keogh dataset is in an unfair disadvantage here by lacking the great volume of data that the Potatamis dataset disposes of.

$$p(x) = \frac{1}{\sigma x \sqrt{2\pi}} e^{-\frac{(ln(x)-\mu)^2}{2\sigma^2}} \tag{3.3}$$

Usually, whenever the left tail of the histogram stands out as having higher bins than the log-normal curve would have predicted them to be, it is a sign that those samples may be anomalies. Figure 3.7 shows examples of obvious outliers derived from the histograms on the Keogh dataset. Mostly, these are either very thin signals (resembling a Dirac impulse) or wrongly cut signals (with abrupt changes in amplitudes). Their Fourier spectrum offers further evidence that these samples are out-of-the-ordinary since the coefficients of the relevant frequencies (100 to 1000 Hz) are very small or not significantly greater than those of higher (noise-related)

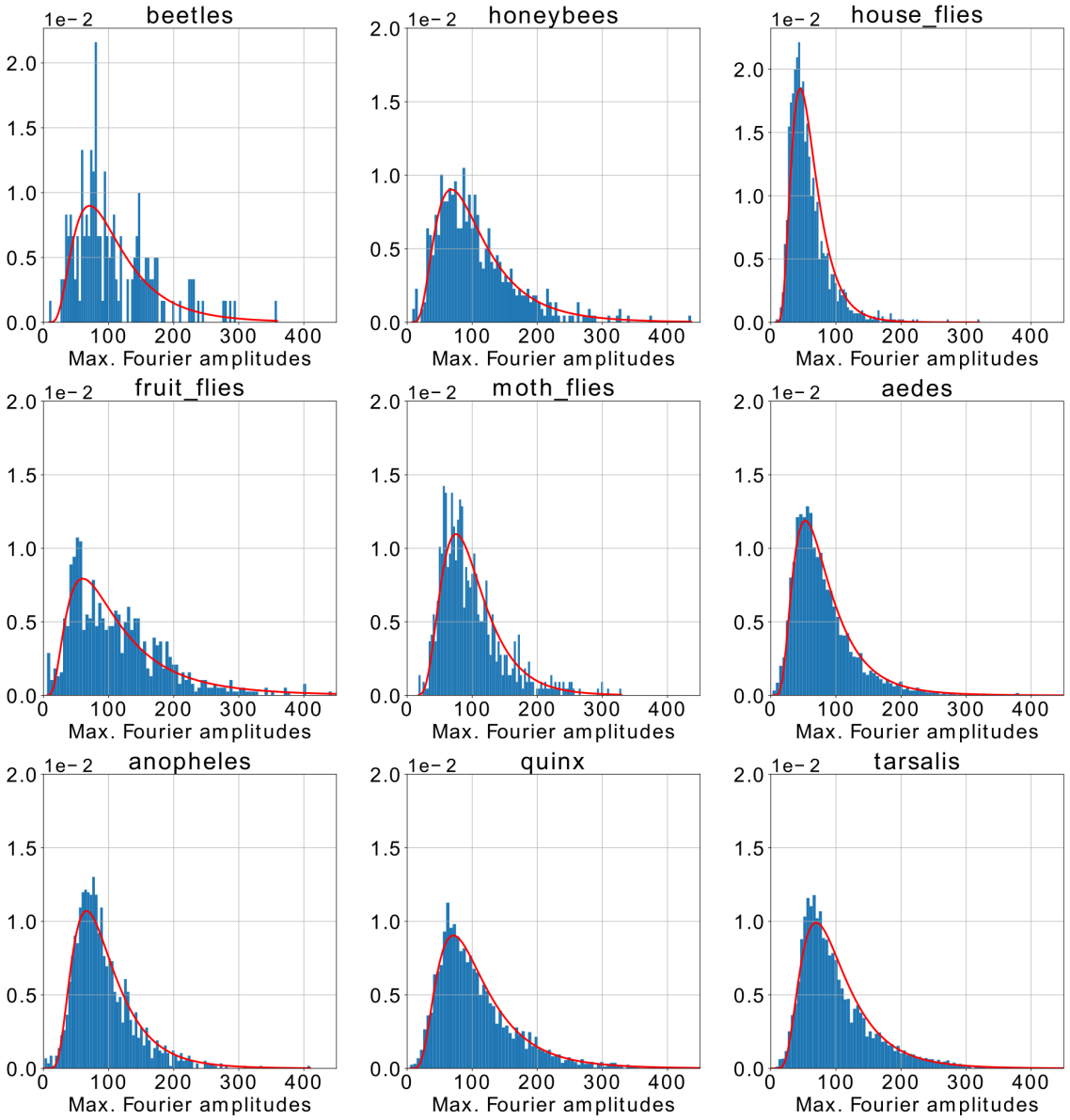


Figure 3.5. Histograms of the maximum Fourier spectrum amplitudes between 100 and 1000 Hz on the Keogh dataset (100 bins).

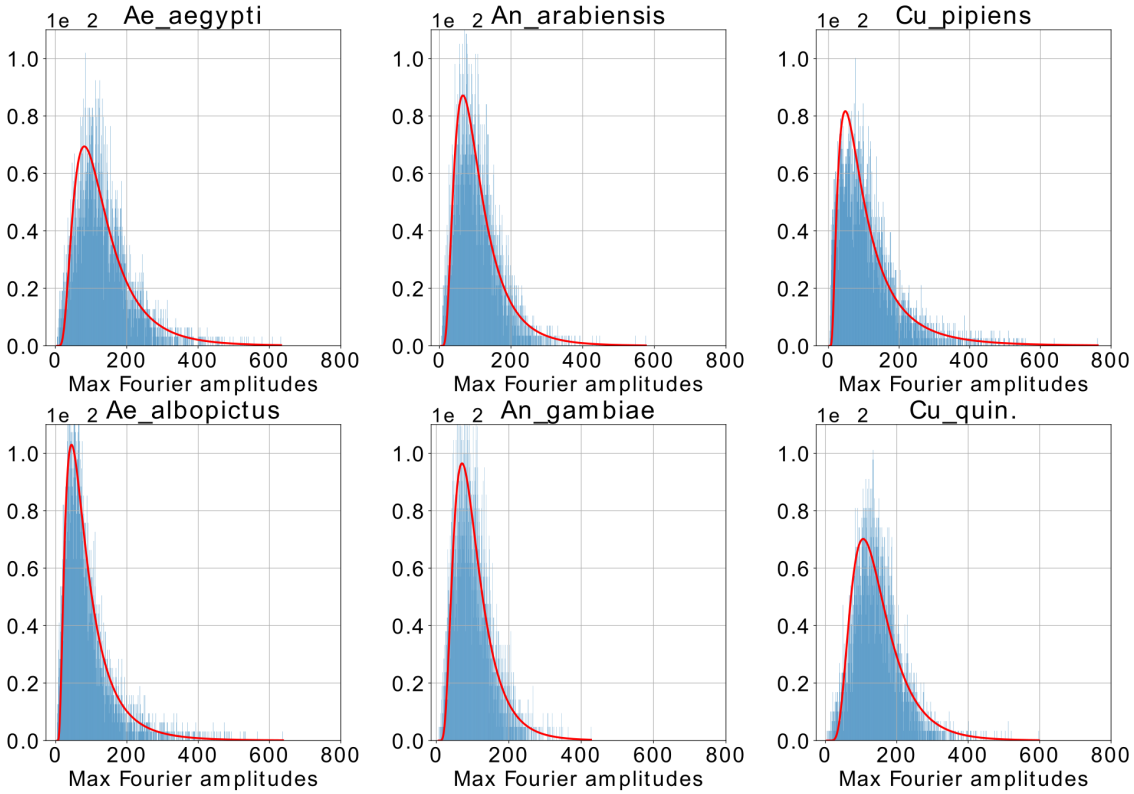


Figure 3.6. Histograms of the maximum Fourier spectrum amplitudes between 100 and 1000 Hz on the Potatamis dataset (1000 bins).

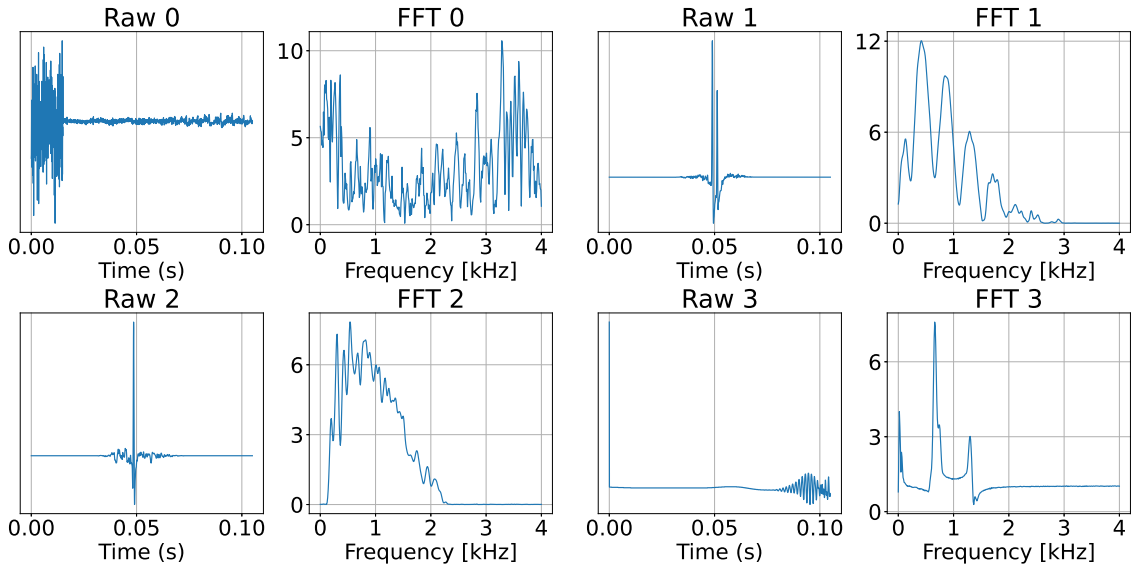


Figure 3.7. Examples of outlier candidates in the Keogh dataset as inferred from the histograms in Figure 3.5. As pointed out in Section 3.1.1, there are signals such as Raw 1 which are still labeled as outliers because of their very low Fourier coefficients, even if their spectrum does show a wingbeat pattern.

frequencies[3]. On this account, all samples that meet this criterion will be removed from the Keogh dataset[4].

When it comes to outlier detection, there are two facets of the problem. Firstly, one may want to maintain a clean dataset devoid of anomalies that could lead to confusions in the classification. These outliers are static and, once removed, never come back. This step is done to try to increase the performance of the classifiers during *batch learning* [11]. It is what this thesis also deals with. Secondly, when the models are fully trained and ready to be put to work in real life, one may also want to identify outliers that do not belong to any of the learnt classes as soon as possible, ideally before inference. Since the classifiers have only been trained on the given labels, they will always assign one of those labels to their input. Thus, situations may occur where something completely different from an insect (a leaf or a wind blow) passes through the sensor and it gets classified as an insect. Such scenarios fall into the category of *online learning* [11] and they exceed the scope of the thesis. One solution to this problem though may be to create a similar outlier detector operating in the Fourier spectrum as described above, but this time the histogram is computed on the whole dataset. Again, a threshold is chosen for the lowest acceptable maximal coefficient. The assumption is that a new sample that does not meet the threshold is not a flying insect. Another solution would be to change the design of the models so that they also predict a 'non-insect' class. But then the challenge is to come up with good non-insect training examples.

[3]Arguably, normalizing the Fourier spectrum could also be a good idea. Spectra of different signals could then be more easily compared.

[4]For the Potatamis dataset, outlier removal was not really necessary.

3.2 Data Preprocessing

As already mentioned, this thesis will consider a total of three types of input for the neural networks: raw (normalised) amplitudes, PSD transformations and spectrograms. The PSD vectors were preferred over the FFT spectrum because of the smaller number of features (129 for PSD vs 2500 and 840 for FFT in the Potatamis and Keogh dataset, respectively). There is also considerably less noise in the PSD vectors (refer back to Figure 1.2). As far as spectrograms are concerned, it is expected that they will outperform the PSD inputs thanks to the extra temporal information they contain.

3.2.1 Power Spectral Density

To better understand where the 129 values of a PSD vector come from, consider the following computation steps on the Potatamis dataset [39]:

1. Let $N = 5000$ be the length of one raw signal (number of discrete points). Then set $n = 256$ (length of one segment of data to be transformed with DFT) and $p = 192$ (number of overlapping points between the segments). Implicitly, each segment will be shifted by $h = 256 - 192 = 64$ points.

2. Divide the whole signal into k overlapping segments of length n, where

$$k = \left\lfloor \frac{N-n}{h} \right\rfloor + 1 = 75 \tag{3.4}$$

3. Multiply each segment s_i, $1 \leq i \leq k$ with a window function w (here *Hanning*).

4. Apply DFT on every $w(s_i)$. This returns as many (complex) Fourier coefficients as discrete points per segment (256). Due to their symmetry, only $\frac{256}{2} + 1 = 129$ of them will be actually needed for the next steps (the additional term corresponds to the 0 Hz frequency).

5. Compute for each segment a *modified periodogram* (weighted power spectrum):

$$P(s_i) = \frac{1}{W}|DFT(seg_i)|^2 \tag{3.5}$$

where $W = ||w||_2^2$ is the squared L_2-norm of the discrete window function and constant for every segment. Note that P is a 129-dimensional vector.

6. Average periodograms to yield a vector

$$PSD = \frac{1}{k}\sum_{i=1}^{k} P(s_i) \tag{3.6}$$

which again holds 129 components.

Figures 3.8 and 3.9 show the PSD vectors of the signals plotted in Figures 2.3 and 2.6, respectively. Given that the samples in the Potatamis dataset are longer and cleaner, the fundamental frequency and its harmonics can be easier identified than in the Keogh dataset.

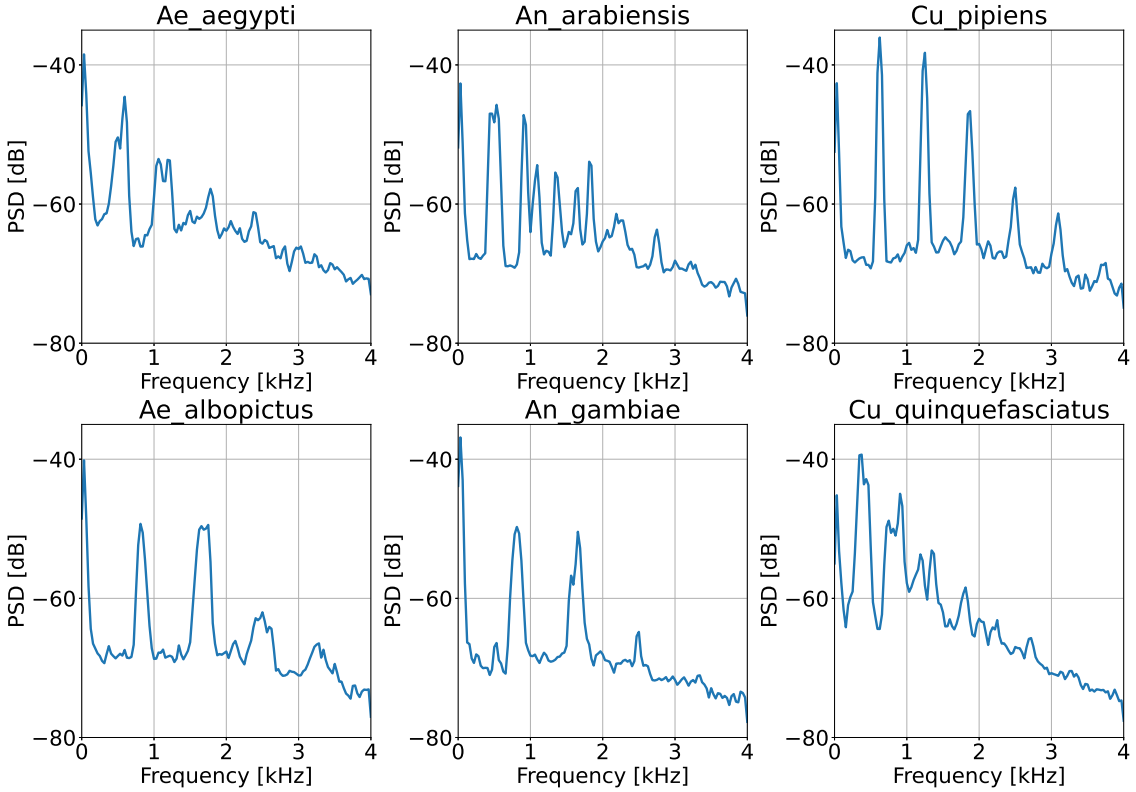


Figure 3.8. PSD plots of the same signals from the Potatamis dataset as in Figure 2.3 ($n = 256, p = \frac{3}{4} \cdot n$). Since the mosquito signals from the Potatamis dataset are longer and cleaner, one can easily read the base wingbeat frequency from the plots by identifying the first peak after 100 Hz. All the other peaks afterwards are the higher harmonics and occur approximately at multiples of the base frequency.

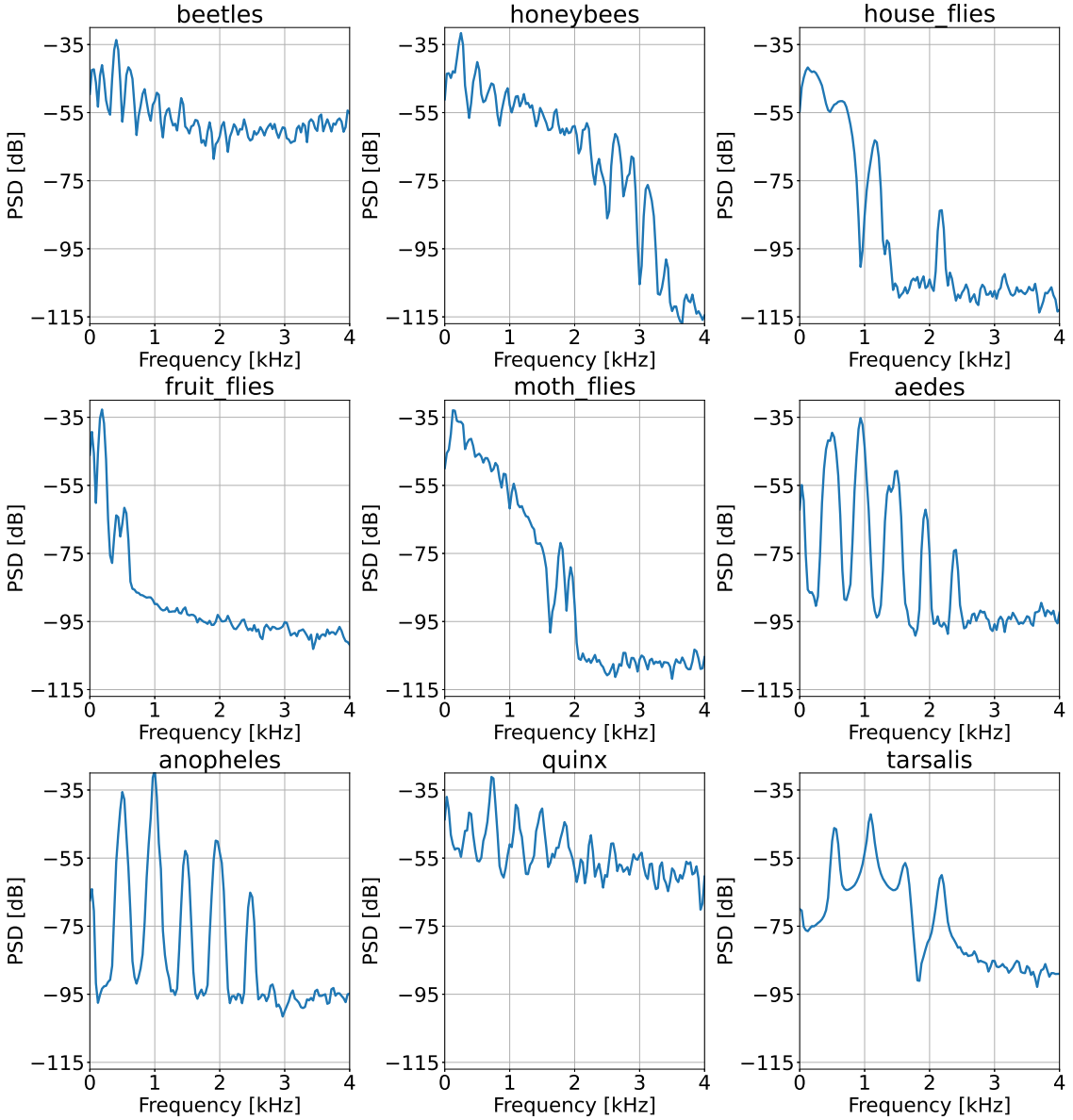


Figure 3.9. PSD plots of the same signals from the Keogh dataset as in Figure 2.6 ($n = 512, p = \frac{3}{4} \cdot n$). By doubling the length of the segments that FFT operates on from 256 to 512 points, the resulting PSD vectors now have 129 values corresponding to frequencies up to 4 kHz, which ensures comparability between the PSD profiles of both datasets. Note that the actual PSD signal has 256 values for the whole 8 kHz range according to the Welch algorithm but only the first half is stored for further computations.

3.2.2 Spectrograms

Another input format used in the experiments (this time two-dimensional) are the *spectrograms*. The idea of a spectrogram is to localize frequencies in time by applying the so-called *Short-Time Fourier Transform* (STFT) - a series of Fourier transforms of a windowed signal [19]. This method is especially suitable for signals whose frequency components are time-variant. A simple Fourier transform of the whole signal would just average out the temporal information.

Consider a discrete signal x of length N partitioned into overlapping segments of length L. Let h be the 'hopping distance', the amount by which each segment is shifted to the right. Inherently, the overlapping will then be given by $L - h$ and the number of segments M is computed similar to 3.4. The starting index of each segment is $m \cdot h$, $m = 0, ..., M - 1$. Let g be a window function discretized into L points. Equation 3.7 shows how to compute the frequency spectrum of the m-th frame within signal x. Figure 3.10 is a visualization of this equation.

$$X(m, n) = \sum_{k=0}^{L-1} x[m \cdot h + k] \cdot g[k] \cdot e^{-i2\pi nk/L} \tag{3.7}$$

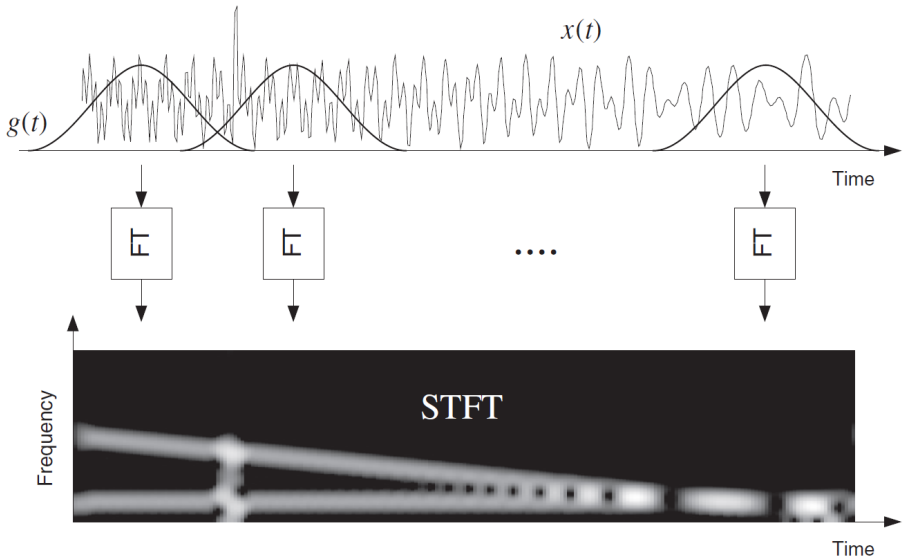


Figure 3.10. Short-Time Fourier Transform. Each frame is multiplied by the window function g and individually Fourier transformed. Picture taken from [19].

Index m can be interpreted as a time index, while $n = 0, ..., N$ is the frequency index (according to the DFT algorithm, there are as many frequencies as discrete data points in the time domain). Therefore, $X(m, n)$ can be visualized as an in-

tensity plot[5], where each point (m, n) gives the intensity of the n-th frequency at time point m. Usually, the spectrogram is preferred for plots, since it can deal with complex spectra by squaring the absolute value of $X(m, n)$ (Equation 3.8).

$$spectrogram\{x\}(m, n) = |X(m, n)|^2 \tag{3.8}$$

Figures 3.11 and 3.12 show the spectrograms of the same signals plotted in Figures 2.3 and 2.6, respectively. One aspect to mention here is that the PSD plots in the preceding section are really just averaged spectrograms along the time axis. A striking difference between the two datasets is that the spectrograms corresponding to the Potatamis dataset have more time variation in their frequencies (notice the wave-like shapes). In contrast, given that the recordings in the Keogh dataset are much shorter, there is little to no temporal variation in their spectrograms (notice the fairly parallel lines). In this respect, one could argue that spectrograms do not offer more information than the PSD vectors when the original signals are too short. This idea will be addressed again in Chapter 5.

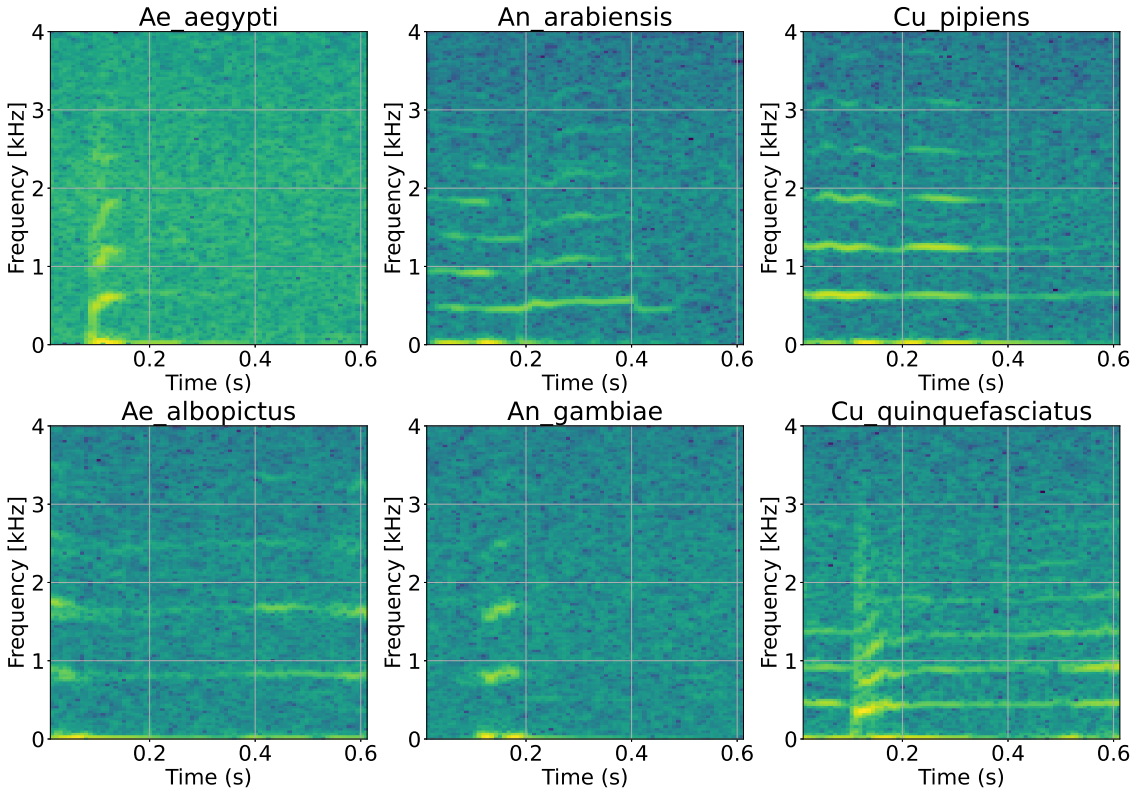


Figure 3.11. Spectrograms of the raw signals in Figure 2.3. The fundamental frequencies and their higher harmonics can be clearly seen in the plots.

[5]Intensity images do not have RGB-channels. The spectrograms plotted further are only colored thanks to a mapping from the gray scale values to a color bar done in *matplotlib*.

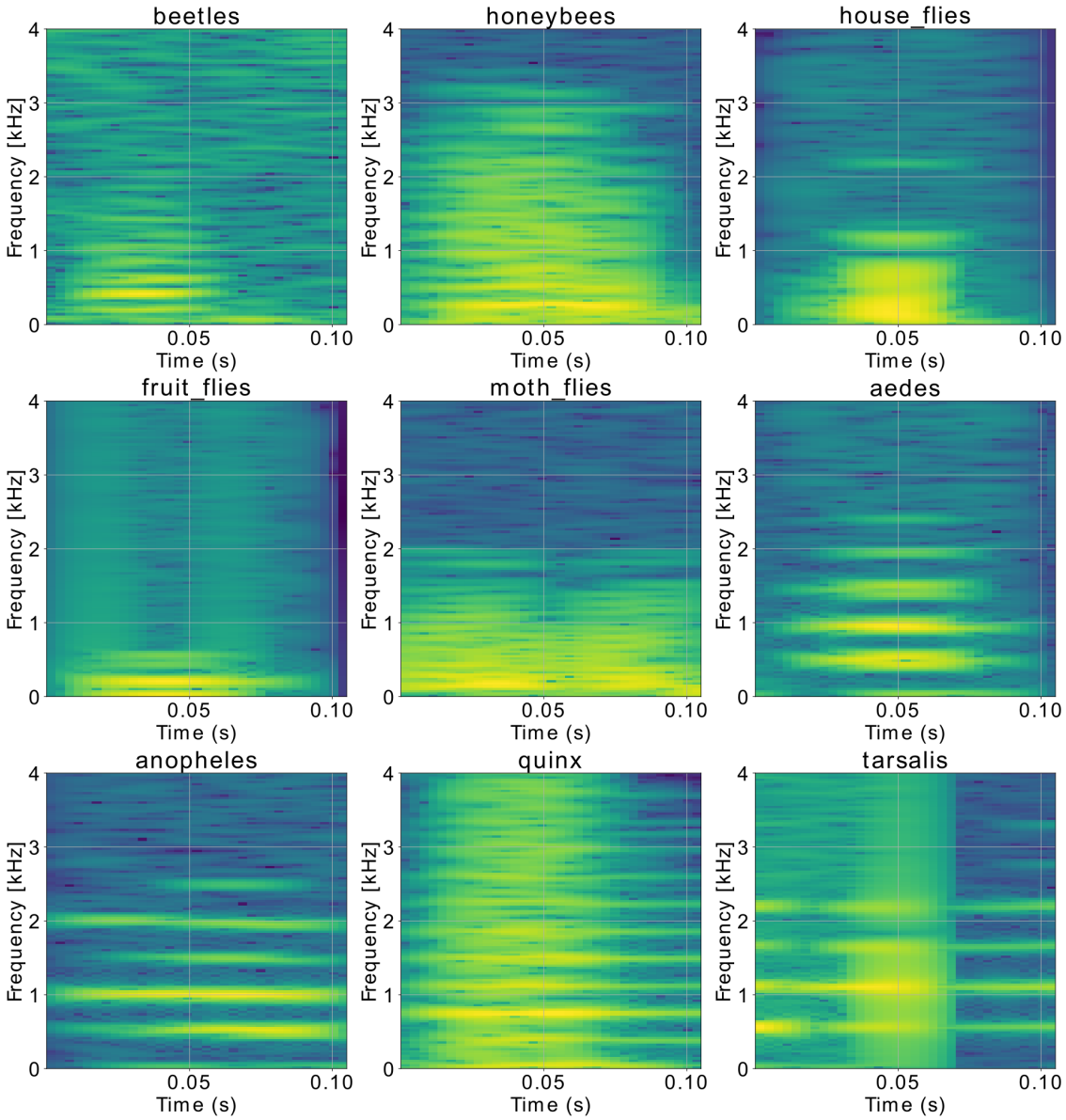


Figure 3.12. Spectrograms of the raw signals in Figure 2.6. As opposed to Figure 3.11, the spectral lines barely vary with time.

3.3 Data Visualization

3.3.1 Fundamental Frequency

According to [21], insect species tend to have characteristic fundamental wingbeat frequencies. The *fundamental frequency* f_0 is the lowest frequency of a periodic signal; the assumption is that insects could be recognized based on it. Refer to the PSD plots in Figure 3.8 again; f_0 is marked by the highest peak above 100 Hz. The peaks afterwards are the higher harmonics and occur roughly at multiples of f_0.

For determining f_0, the *cepstrum* of the signal may be used[6]. This is calculated through the inverse Fourier transform of the logarithm of the Fourier spectrum[7] (Equation 3.9 [5]) and is meant to describe the periodicity in the frequencies of the original signal. Figure 3.13 shows an example of a Fourier spectrum and its equivalent cepstrum. Notice that the horizontal axis is now labeled *quefrency*, which is to be understood as a measure of time since the transformation maps frequencies back into the time domain. The quefrency of the highest peak in the cepstrum will determine the fundamental frequency. In the figure, this happens at about 5 milliseconds, which corresponds to a frequency of 200 Hz[8].

$$C = \left| \mathcal{F}^{-1} \left\{ log\left(|\mathcal{F}\{f(t)\}| \right) \right\} \right| \tag{3.9}$$

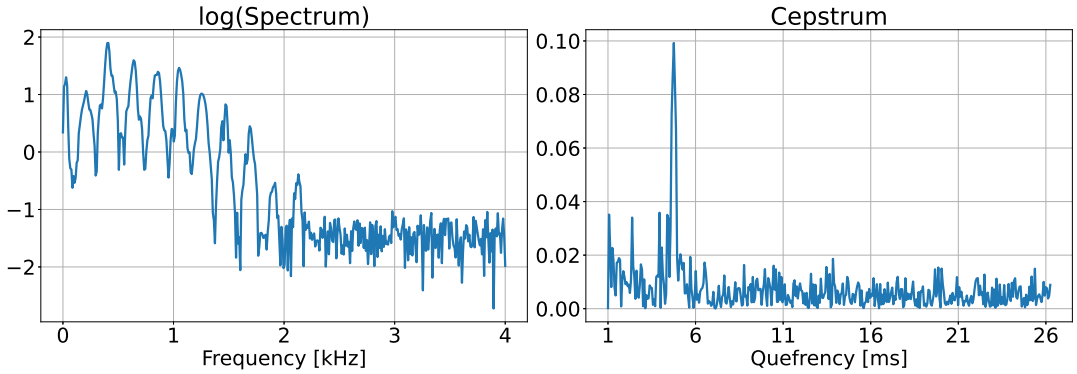


Figure 3.13. Example of a cepstrum plot (right) along with the logarithm of the original Fourier spectrum (left). The first few samples in the cepstrum have been skipped because their peaks would make it difficult to see the actually relevant amplitudes.

[6] Alternative methods are maximal autocorrelation or simply examining peaks in a PSD plot.

[7] Different cepstrum formulas can be found in literature [29] that use the power spectrum (the square of the spectrum) and the direct Fourier transform instead of the inverse.

[8] It follows from $f = 1/T$ that $200 = 1/0.005$.

By calculating the base frequency of every signal in the Keogh dataset, the histograms in Figure 3.14 were created. It is quite obvious that the frequency distributions alone could not be used to classify wingbeat signals. Firstly, the distributions are far from being normal. Secondly, median values are not unique for species (in the first three plots, all species seem to have their f_0 around 200 Hz).

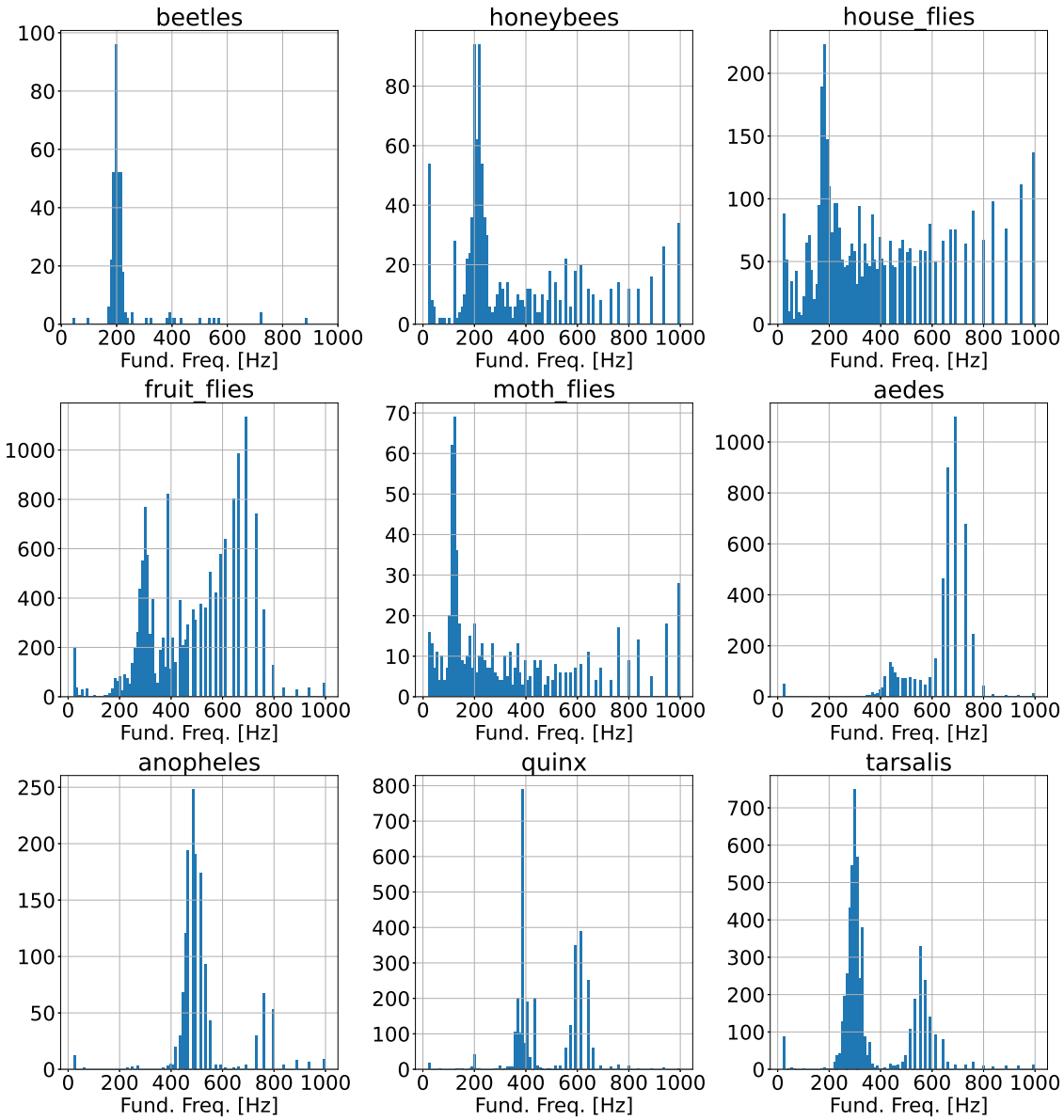


Figure 3.14. Histogram of fundamental frequencies on the Keogh training set calculated as shown in Equation 3.9.

3.3.2 t-SNE

To visualize the structure of both datasets, *t-SNE* (t-Distributed Stochastic Neighbor Embedding) was applied [27]. This technique aims at highlighting possible clusters in the data by keeping similar instances close together while pulling dissimilar ones farther apart from one another (Figures 3.15 and 3.16). According to [14]:

"It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data."

In Figure 3.15 the PSD vectors from the Keogh training set are projected onto the xy-plane. Figure 3.16 shows the projection of the features extracted from these PSD samples by the CNN of a pre-trained *Simple Classifier* (Figure 6.5). Since t-SNE is quite computationally expensive, dimensions have been in both cases additionally reduced with *PCA*[9], leaving only the top 50 most relevant principal components as inputs for t-SNE.

It does appear that clustering works somewhat better on the CNN features than on the PSD inputs themselves (cluster boundaries are better defined), which may be an indication that convolutions are, indeed, helpful at finding more relevant characteristics in the signals. However, simply using these projections in combination with a standard algorithm such as *kNN* would not suffice for satisfactory classification results.

[9]The feature vectors computed by the CNN have 1024 components.

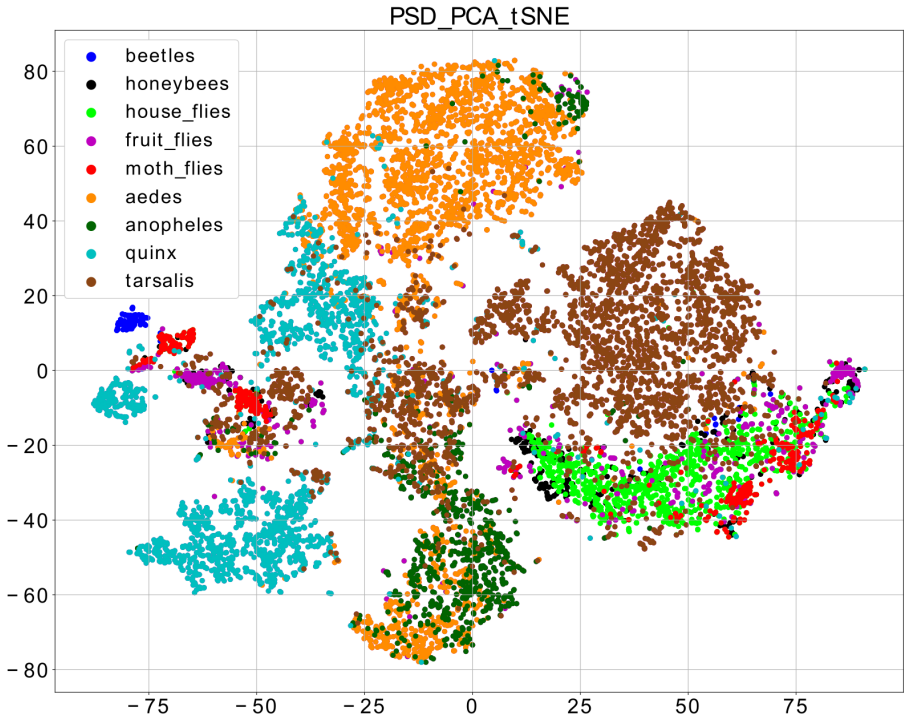


Figure 3.15. Two-dimensional t-SNE on the Keogh training set, after applying Welch-PSD and PCA with 50 components.

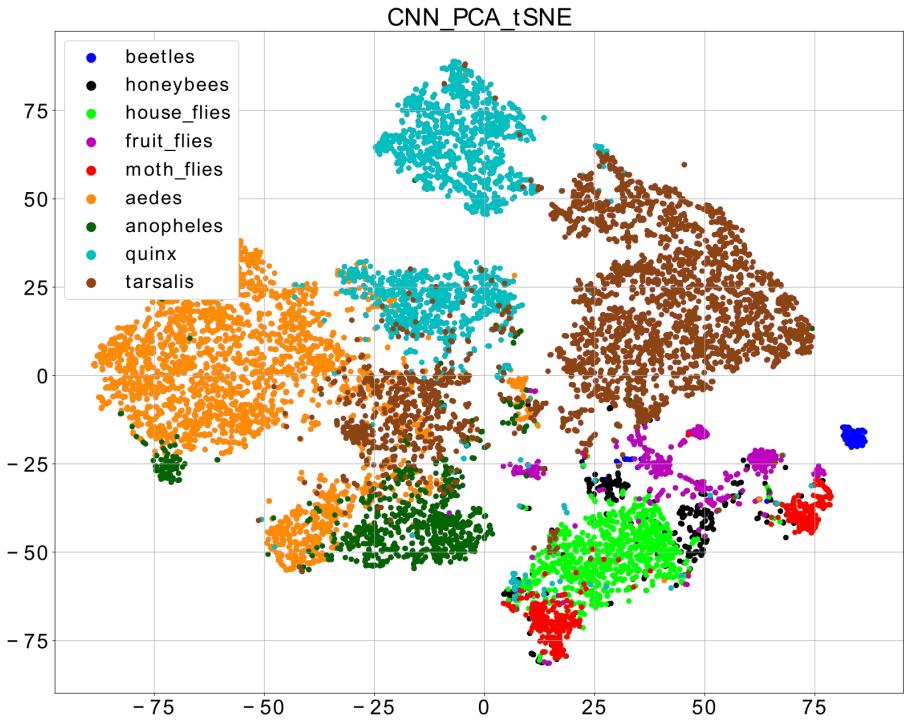


Figure 3.16. Two-dimensional t-SNE on the Keogh training set, after extracting features with the CNN body of a pre-trained *Simple Classifier* and applying PCA with 50 components.

3.4 Classification with CNNs

The classification task in this work will require a more complex approach than just fitting standard classifiers from the literature (see Table 4.1). A good feature extractor is a must when dealing with wingbeat patterns, so the CNNs will constitute a central part of the model architectures. Furthermore, given the hierarchical structure of the data, it is expected that encoding this information into the network design will yield better results than training a simple CNN. The requirements and issues with this modelling path will be exposed in the following sections.

3.4.1 Overfitting

In Machine Learning there is a fine line between simplicity and complexity when it comes to designing the appropriate model for solving a task, be it regression, classification or some other type of problem. A good model needs to be complex enough to consider all the relevant factors involved in the learning setup, but also so simple that it captures the structure of the objects it processes as precisely as possible. This fine line is what is called *the bias-variance trade-off*.

According to [3], the *expected mean squared generalization error (EMSGE)* can be decomposed into three terms (Equation 3.10); bias and variance are two of the contributors[10]. Notice that to minimize the loss, all three terms need to be at their minimum. Given that they are all non-negative quantities, that minimum would be 0. Unfortunately, it is practically impossible to achieve zero-contribution from all terms.

$$EMSGE = (bias)^2 + variance + noise \tag{3.10}$$

To begin with, the noisy remainder will never disappear completely because there will always be some inaccuracies present either in the data collection phase (here, in the sensor measurements) or in the model building process. It is the reason why noise is also called *the irreducible error*[11].

Secondly, bias and variance are not independent terms. Low variance models tend to be highly biased and underfit the training data. This is usually caused by wrong assumptions regarding the structure of the data when a relatively complicated

[10]The equation was originally devised for regression problems [16], but a similar relationship between the EMSGE and bias and variance was proven for probabilistic classification tasks [28].

[11]What can be done, to attenuate the noise's share of the error, is data cleaning.

problem is approximated through too simple methods (for instance, assuming linearity when, in fact, the data is non-linear). Conversely, low bias models, although more flexible, tend to exhibit high variance and be prone to overfitting. This happens due to the classifier's sensitivity to the given training data. When fitting the model on a different training set, a new approximation of that model is produced. However, a robust predictor should not change much when small variations are applied to the data (such as the different training-validation splits during k-fold cross-validation).

The challenge is, therefore, to find a balance between bias and variance. In this paper, the focus will primarily lie on addressing the problem of overfitting, as is often the case with deep neural networks that rely on hundreds of thousands of parameters. Concretely, the following sections will discuss different regularization techniques employed in the training process.

L_2-Regularization

A straightforward way to keep model complexity in check is to add a penalty term to the loss function, just like in Equation 3.11. The cost function J, which is the optimization target during training, is composed of the loss function[12] and a regularizing term expressed as the sum of the squared model weights θ_i. This is what is called L_2- or *Ridge-Regularization*. The factor λ is the *regularization parameter* and determines how strong the weights should be penalized. λ is a hyperparameter and will be optimized in the hypertuning pipeline along with the learning rate.

$$J(\boldsymbol{\theta}) = loss(\boldsymbol{\theta}) + \frac{\lambda}{2}\sum_{i=1}^{n}\theta_i^2 \tag{3.11}$$

As an alternative, the squared weights could be replaced with their absolute value, which would give the *Lasso-Regularization*. The main difference is that the latter tends to shrink less important weights to 0 during SGD and, thus, outputs a sparse model. The former method also brings weights closer and closer to 0 but they never get eliminated. For this work, small efficient models that can run on devices such as Raspberry Pi are a goal to keep in mind. Model sparsity would, therefore, be a desirable characteristic. However, for the experiments in this paper, Ridge penalty was preferred[13] and model sparsity would be addressed in later phases of the project, where techniques such as *Post-training Quantization* (Section 5.1.2) are employed to convert the models into an appropriate format for small devices.

[12] Different kinds of losses will be used depending on the model architecture (Section 3.4.5).
[13] Only Dense layers were regularized, convolutional layers not.

Dropout and Monte Carlo Dropout

Another very simple yet efficient regularization technique is randomly *dropping out* neurons during training [41]. This way they will not rely too heavily on their neighboring neurons and be as useful as possible on their own. Moreover, the layer connected after the dropout affected layer will learn to pay attention to all possible inputs, not just particular ones. The result is a more robust classifier with a higher generalization power.

A Dropout layer is characterized by its *dropout rate*, namely the ratio of neurons that are going to be randomly deactivated in the training phase. Since the networks developed for this paper are fairly prone to overfitting, the dropout rate was set to $p = 0.5$. Also, irrespective of model architecture, the Dropout layer would always be placed right between the averaged output of the feature extractor and the first Dense layer (see Section 4.1).

A particularity of these layers is that they behave differently during inference as opposed to training time. When in testing mode, Dropout does not set any neurons to 0 anymore, since it is the unregularized model that needs to make the predictions. Instead, Dropout just passes its inputs unmodified to the next layer[14]. Nevertheless, leaving the random neuron deactivation turned on during testing might be desirable.

A fully trained classifier with now constant weights can only deliver a point-wise approximation of each class probability, which offers no information about how uncertain the prediction is. This is one of the arguments that motivated the development of *BNNs (Bayesian Neural Networks)* [18]. In short, a BNN outputs a distribution approximation for each class probability, making the prediction far more informative regarding its uncertainty. The drawback with this kind of networks is that they are built on weight distributions, not single-valued weights as in a classical architecture, which dramatically increases computation time and complexity. Luckily, there is a shortcut method to simulate a BNN from a standard CNN, namely by keeping Dropout active during inference, too. This technique is called *Monte Carlo Dropout* or *MC-Dropout*. It has been proven in [10] that training MC-Dropout models is mathematically equivalent to approximate Bayesian inference. In this paper, MC-Dropout layers will be used for inspecting and validating models in Section 5.2.

[14]For $p = 0.5$, a Dropout neuron at inference time would be linked to twice as many neurons as it was on average during training. To address this issue, one would need to divide these neurons' outputs by $1-p$ when training, otherwise performance is likely to decrease. Fortunately, the Keras implementation of the Dropout layer takes care of this aspect automatically.

Learning Schedules

The choice of the learning rate α can greatly impact the training process in terms of convergence. One simple approach for finding an appropriate value is to train multiple models with different α's and look at how the loss behaves. Usually, the loss will be found to decrease fairly rapidly at the beginning until it reaches a plateau. One can identify this transition point through the *elbow method* and the α at that point will often be an acceptable default value to run experiments.

Nonetheless, according to [11], applying a constant learning rate throughout the whole training session is not necessarily a good strategy, unless the cost function has a convex landscape (like MSE). Unfortunately, the cost functions used in this paper are very irregular and a fixed α might make SGD[15] bounce around the solution or, even worse, diverge. To help the algorithm settle at a solution, it is advisable to gradually reduce α as training progresses. The function that returns the learning rate for each iteration is called *learning schedule*.

There are still two dangers to consider when integrating learning schedules in the training process. On the one hand, if α is reduced too quickly, the optimizer may get trapped in a local minimum or freeze on its way to the global minimum. On the other hand, if α is reduced too slowly, the algorithm could jump around the minimum and provide a suboptimal solution if training is stooped too early.

For this thesis, three learning schedules were considered:

1. *Exponential*: The learning rate is subject to an exponential decay between a maximum value $\alpha_0 \in \mathbb{R}$ and a minimum value $\alpha_1 \in \mathbb{R}$ over an interval of length T. After reaching α_1, the rate stays constant (Figure 3.17):

$$\alpha(t) = \begin{cases} \alpha_0 \cdot e^{rt} & 0 \leq t < T \\ \alpha_1 & t \geq T \end{cases} \tag{3.12}$$

   with $r = \dfrac{ln(\alpha_1) - ln(\alpha_0)}{T}$ as the decay rate, T as the number of epochs for which the decay continues and $t \in \mathbb{N}$.

[15] Some authors differentiate further between *Stochastic Gradient Descent*, which picks one random instance at every iteration to compute the gradients and *mini-batch Gradient Descent*, which picks random batches of instances (see [11]). In this paper, the term SGD refers to the latter definition.

2. *Triangle Cycle*: A type of *cyclical learning schedule* [37]. The idea is to periodically increase and decrease α between a minimal value α_1 and a maximal value α_0 over an interval of length T. A period is defined by a step size s. After T iterations/epochs, the rate stays constant at α_1 (Figure 3.17):

$$\alpha(t) = \begin{cases} \alpha_1 + (\alpha_0 - \alpha_1) \cdot \dfrac{max(0, 1-x)}{2^{c-1}} & 0 \leq t < T \\ \alpha_1 & t \geq T \end{cases} \tag{3.13}$$

with $c = \left\lfloor 1 + \dfrac{t}{2s} \right\rfloor$ as the current cycle/triangle and $x = \left| \dfrac{t}{s} - 2c + 1 \right|$ as the local time point in the current triangle.

3. *ReduceLROnPlateau*: An adaptive strategy to reduce the learning rate when a chosen metric has stopped improving. Apart from the usual parameters α_0 and α_1, this schedule also needs the following: $\Delta_{min} \in (0, 1)$ as the amount by which the metric should improve at end of every iteration to pass as significant, $r \in (0, 1)$ as the decay rate and $p \in \mathbb{N}$ as the number of consecutive below-Δ_{min} iterations to wait until to reduce α (also called 'patience'). Again, after reaching α_1, the rate stays constant.

To decide on a learning schedule, all three methods have been tested on the Keogh dataset. The results are summarised in Figure 3.18[16]. By comparing the mean species accuracy and loss curves, it appears that *ReduceLROnPlateau* and *Exponential* are the best strategies for the task at hand. It was only by a slight margin that the former method exceeded the latter in the average validation accuracy score: 91.88% vs 91.52%. *ReduceLROnPlateau* was selected for the training pipeline but *Exponential* would have been just as good of a choice.

[16]The statistics were created through 4-fold cross-validation with a *Simple Classifier* receiving PSD inputs and default hyperparameters. Other schedule parameter combinations yielded similar plots. When training a similar spectrogram classifier the balance was even more in favor of *ReduceLROnPlateau* in terms of loss and accuracy. More on model architectures in Chapter 4.

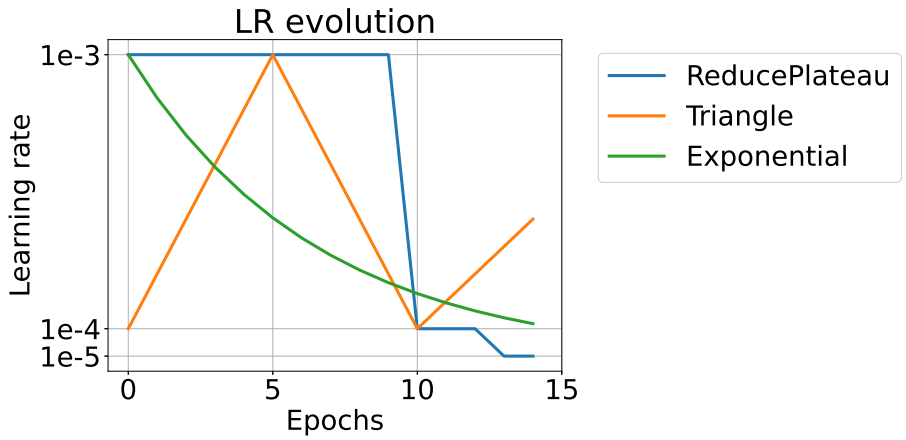


Figure 3.17. Visualization of the learning rate trends for all three schedules during a 4-fold cross-validation on the Keogh dataset. For *ReduceLROnPlateau*: $\Delta_{min} = 10^{-2}$, $r = 10^{-1}$ and $p = 3$. For *Triangle*: $T = 30, s = 5$. For *Exponential*: $T = 30$.

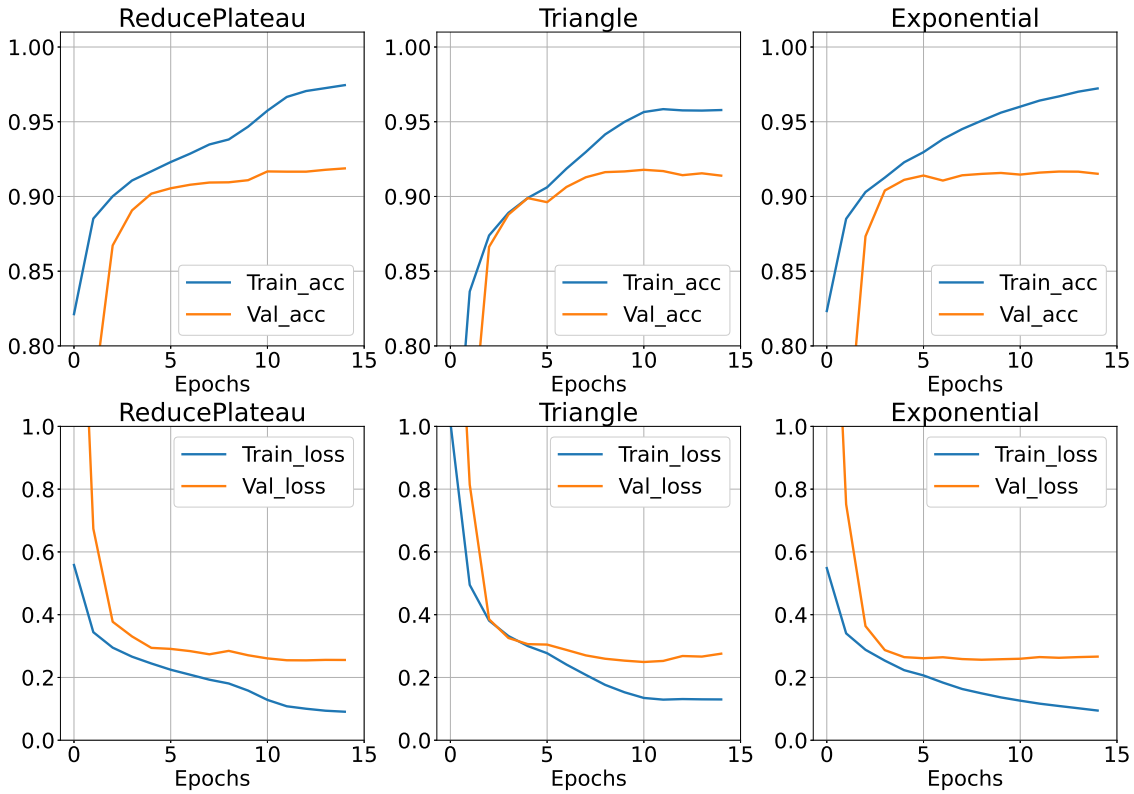


Figure 3.18. Mean accuracy and loss curves for all three schedules after a 4-fold cross-validation on the Keogh dataset.

Early Stopping

The plots in Figure 3.18 show that training could have been halted before reaching the maximal epoch since the models had already converged. Notice that after reaching the highest possible score, the validation accuracy stays fairly constant, whereas the training accuracy continues rising. This is an indication of the network starting to overfit the training data. To prevent this from happening, *Early Stopping* will be integrated.

Early Stopping is characterized by two parameters: Δ_{min} as the minimal improvement the metric needs to achieve and the patience p. In other words, after each epoch, the current validation loss[17] is checked to see whether it is continually improving. If not, training is stopped and its history is rolled back to the model weights that delivered the best metric.

3.4.2 Imbalanced Data

Very often datasets comprise unequally represented classes, that is, some classes outweigh the others in the number of samples. In that case, the dataset is said to be *imbalanced*. The problem with imbalanced datasets is that the classifier may learn to recognize the majority classes very well while having difficulties in recognizing the minorities.

Broadly speaking, strategies to fight against class imbalance can be *data specific*, where the goal is to adjust the ratio between minority and majority classes through resampling (concretely, undersampling or oversampling), or *algorithm specific*, where the learning algorithm is directly modified to alleviate the bias towards the majority (i.e. class weighting) [23]. For this paper, three methods were considered:

1. *Undersampling*, usually performed by randomly discarding samples from the dominant classes. Although easy to implement, it has the obvious disadvantage of not making use of the whole dataset.

2. *Oversampling*, in which the underrepresented classes are augmented, either by just randomly copying samples already present in the dataset or by generating new artificial ones (i.e. time shifts or adding noise). When oversampling, one

[17] Alternatively, one could set the validation accuracy as the observed metric. Still, given that it is a sort of metric that summarises much information, such as prediction uncertainty, into a single number, the validation loss was deemed as the better option. The lower the validation loss, the more certain the model is on its decisions.

needs to keep in mind that computation time and the space needed to store the data will increase.

3. *Class weighting*, which keeps all the available samples in the dataset, but assigns different class weights depending on how well each category is represented. Minority classes get higher weights so that the model does not overlook them during training. Finding a good list of weights becomes then an important part of hyperparameter optimization though.

The Potatamis dataset is a very ample one: 280k audio files in total, each species with at least 10k samples. While not perfectly balanced, there was no real need to apply any of the techniques described in this section on the Potatamis dataset. It was the Keogh dataset that required intervention to control the imbalance, as two species - the beetles and the honey bees - were in clear inferiority (172 and 511 respectively against an average of over 1000 for the other species). Moreover, these two categories only had one species per genus in the database, while the flies and the mosquitoes had 3 and 4, lending their genus branch a wider variety of samples the models could learn from. Experiments with undersampling, different class weight lists and also time shifts or noise injection (Figure 3.19) led to little to no increase in accuracy scores for the underrepresented species. On the contrary, operations such as the last two mentioned sometimes caused a drop in performance, when the newly generated samples were too different from the normal ones that the classifier used to see (time shifts sometimes end up cutting the relevant amplitudes into half at the start and end of the recording, while the *SNR* parameter (Signal to Noise Ratio) proved to be very sensitive when injecting noise - adding too little means the new samples are not new at all, but too much noise means the wingbeats can hardly be recognized by the classifier and the accuracy drops). What did turn out useful was an oversampling technique called *SMOTE*, described below.

SMOTE

First proposed by [6], *SMOTE* (Synthetic Minority Oversampling Technique) is a simple method of creating new samples through convex combinations[18] (Figure 3.20). First, a positive scalar k is chosen (originally set to 5 in [6]). Depending on how much the minority set needs to be augmented, some of the k nearest neighbors are selected at random (for instance, if the set has to increase its size by a factor of 3,

[18] A convex combination of n vectors $x_1, ..., x_n$ is a weighted sum of the form $\sum_{i=1}^{n} \lambda_i x_i$, where $\sum_{i=1}^{n} \lambda_i = 1$

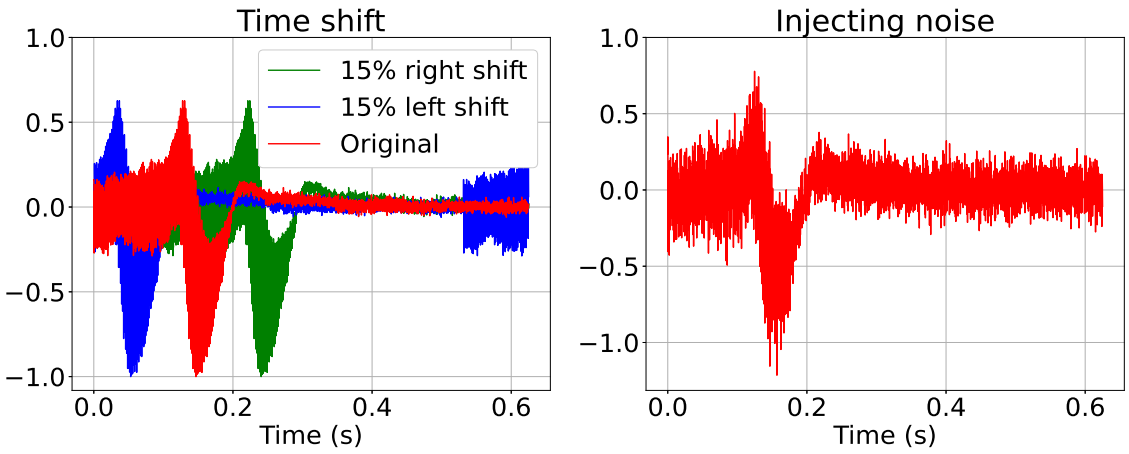


Figure 3.19. Left: Example of shifting a signal. The original signal (in red) is moved by 15% of its length to the left and right along the time axis. Right: Adding noise to the same signal in the first plot (SNR = 5).

then 2 such neighbors are chosen). The difference (connecting vector) between the reference sample and each of its selected nearest neighbors is computed, multiplied by a random scalar between 0 and 1 and then added to the reference sample. The resulting vector is then appended to the set under consideration.

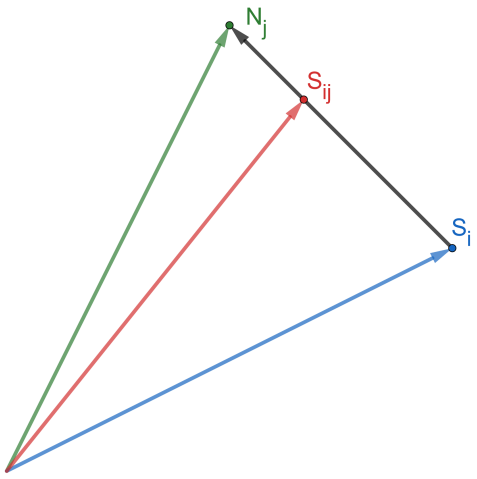


Figure 3.20. SMOTE algorithm. Vector S_{ij} is selected at random on the line between the real sample S_i and one of its nearest neighbors N_j.

Depending on whether it is nearer to the reference signal or the neighboring one, the new synthetic sample is going to look more similar to one or the other (Figure 3.21). It is common practice in signal processing to first *align* two signals before linearly combining them. Alignment means shifting one signal in time until it reaches the highest cross-correlation with respect to the other [7] (Figure 3.22). The motivation for this is that adding two signals together that share similar frequency

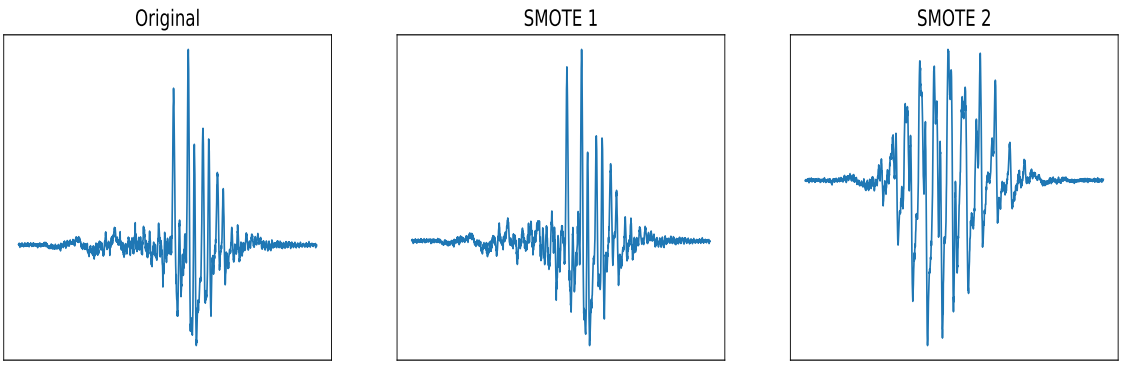


Figure 3.21. Example of synthetic signals produced by SMOTE. The new sample may look more similar to the original one (SMOTE 1) or the neighbor (SMOTE 2).

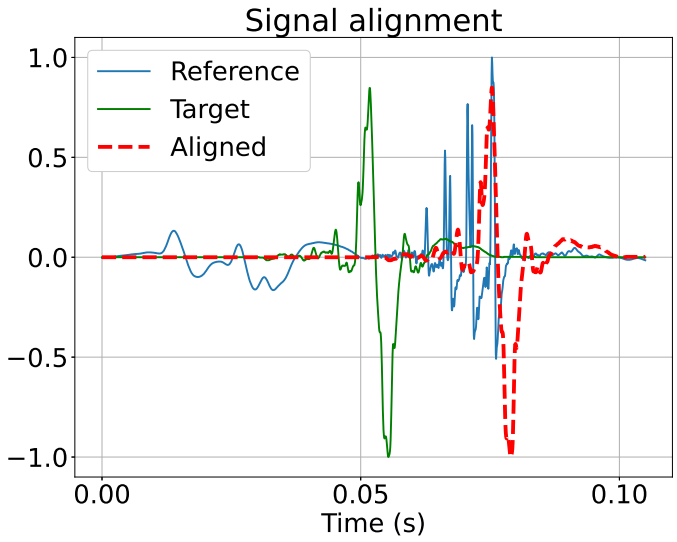


Figure 3.22. Example of aligning two honey bee signals. The target is shifted in the position with the highest correlation to the reference.

information but have shifted peaks can produce a quite different new signal that does not share the same frequencies with the old ones. With SMOTE the goal is to create new samples while preserving as much spectral information as possible. That is why an extra preprocessing step such as alignment is worth looking closer at. Out of time and implementation reasons though, this step is left out here.

As it turns out, SMOTE is quite sensitive to the input parameters for augmenting the data. The first (naive) approach was to simply augment every species until it reached the number of samples in the dominant class, namely the tarsalis mosquito with 5309 samples. The dataset was, thereby, perfectly balanced but the overwhelming number of fake signals in some categories led to significant drops in accuracy. It became clear that more trials needed to be done with different SMOTE strategies, in order to find the right classes to augment and also the right augmenting factor.

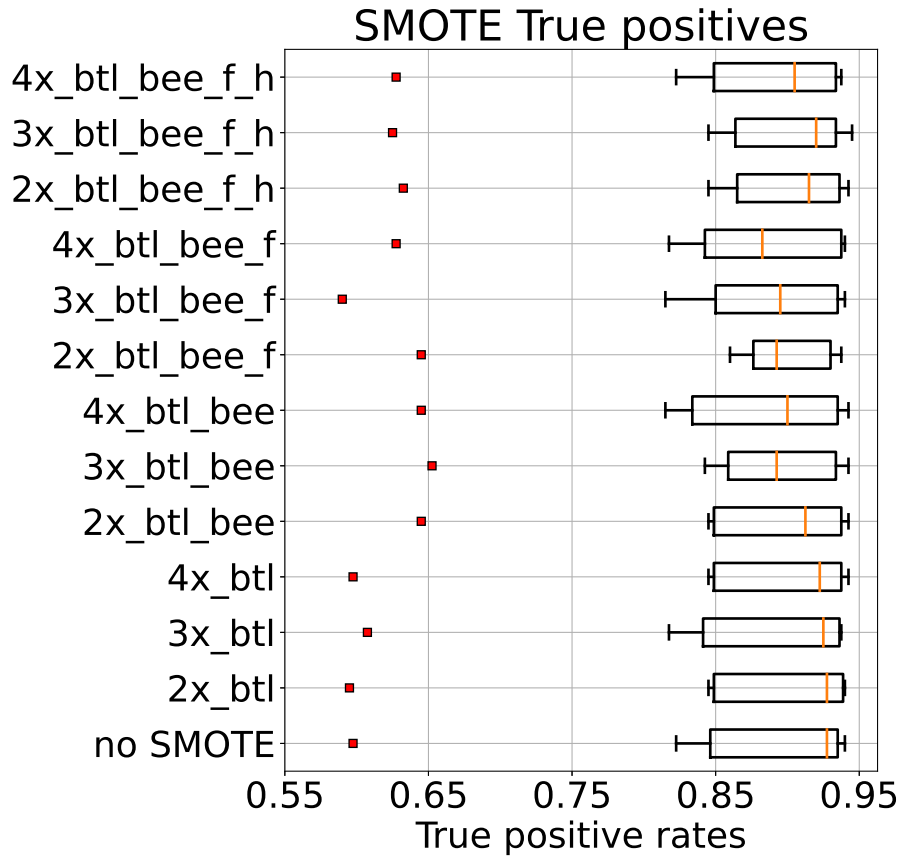


Figure 3.23. Boxplots with true positive prediction rates for 7 of the Keogh species (moths and anopheles mosquitoes were left out). Different SMOTE strategies are highlighted on the y-axis: the initial number is the augmentation factor of the species written in shorthand afterwards: beetles - btl, honey bees - bee, fruit flies - f and house flies - h. The species were chosen in increasing order of their sample size. The outliers marked with red dots are the accuracies for the bees. The orange line marks the median.

Strategy	**2x**	**3x**	**4x**
btl_ bee_ f_ h	92.31	92.22	91.89
btl_ bee_ f	91.88	91.65	91.71
btl_ bee	92.18	91.84	91.88
btl	92.17	92.05	92.07
no SMOTE	92.05		

Table 3.1. Accuracy scores (in percentages) for each of the SMOTE strategies listed in Figure 3.23. Each strategy is labelled according to the augmented species: *btl* - beetles, *bee* - honey bees, *f* - fruit flies, *h* - house flies. Column labels **2x, 3x, 4x** specify the augmenting factor.

Figure 3.23 summarises these trials[19]. Notice how different parameter combinations lead to a different spread of the true positive rates per species. It is also not necessarily the case that the more a class is augmented, the better its accuracy becomes. This information would be missing if only overall accuracy scores were considered (Table 3.1). But the goal here is to strike a balance between the performance on the whole dataset and the performance on every class individually. A model with a very high validation accuracy that does not do better than chance in recognizing bees is, of course, of little use. Therefore, only the beetles and the bees would be augmented, namely by a factor of 2.

3.4.3 Transfer Learning

Leveraging the benefits of Transfer Learning is a key piece in this work and investigations in this respect will be carried out on both datasets.

As mentioned earlier, two species have been set aside from the Keogh dataset: the moth flies and the anopheles mosquitoes[20]. All networks will first be trained and optimized on the clipped Keogh dataset. Dividing the data like this instead of feeding it all to the classifiers has practical reasons. Future applications within this thesis' project will require models to learn online; that is, to be able to adapt to any new insect species that may be introduced into the database. A competent online learner should not be retrained on the whole dataset every time a new class is added. So the interest hereby lies in examining the models' robustness towards adding new categories.

Secondly, one would like to know how sensitive the networks are towards data gathered from different sensors. The phenomenon of *data shift* [8] is quite common in Data Science and usually occurs whenever multiple data sources are not *i.i.d.* (independent and identically distributed), but *o.o.d.* (out of distribution). Models trained on one distribution cannot perform as well on a very different one. Since the datasets used here have been acquired by different sensors, it is interesting to check how much of an impact this can have on the performance of the pre-trained classifiers. Not to mention that the Potatamis dataset comes with a new insect taxonomy, so models with broader knowledge about various species of insects are now evaluated on a narrower subject, namely recognizing mosquitoes.

[19] Like in the selective process of the learning schedules, these statistics were also created through a 4-fold cross-validation with a *Simple Classifier* receiving PSD inputs and default hyperparameters. Roughly the same results could be observed when training a similar spectrogram classifier.

[20] Only flies and mosquitoes had more than one species.

3.4.4 Hierarchy-based Class Embeddings

The approach taken here towards integrating hierarchical structure into the models' learning process is inspired by [1] and involves *Hierarchy-based Class Embeddings* (HCEs). The central idea is to represent the label vectors in a more efficient way that mirrors the underlying taxonomic tree. The problem with canonical one-hot-encodings is that they do not take similarities between species into account. When represented as orthogonal unit vectors, aedes mosquitoes are as similar to tarsalis mosquitoes as to beetles, although they belong to different genera. HCEs address this issue by offering an alternative n-dimensional basis to represent the label vectors, which preserves the structure information derived from the taxonomic tree. As a consequence, mosquitoes, for instance, will be 'closer' to one another than to any other species from a different genus. Following is the algorithm for computing HCEs from a given hierarchy.

HCE algorithm

For explaining the HCE algorithm, an imaginary 4-level tree will be constructed, which comprises the levels for species, genus, family and order (Figure 3.24). The following notations, formulas and the algorithm are taken from [1].

Let $G = (V, E)$ be the directed acyclic graph (DAG) with nodes V and edges $E \subseteq V \times V$, corresponding to the pseudo-tree. Let $C = \{c_1, ..., c_n\} \subseteq V$ be the set of labels corresponding to the terminal leaves, namely to the species. The *dissimilarity* of two labels/classes from the graph G is then defined as follows:

$$d_G : C \times C \to \mathbb{R}, \quad d_G(u, v) = \frac{height(lcs(u, v))}{max_{w \in V}\ height(w)} \tag{3.14}$$

where $(u, v) \in E$ and $lcs(u, v)$ is the *lowest common subsummer* of the classes u and v, namely the ancestor of both nodes that does not have any child being an ancestor of both nodes as well. The height of a node is defined as the longest path from that node to a leaf. Note that $lcs(u, u) = u$, which means that $d_G(u, u) = 0$ because the height of a terminal node is 0.

One can derive a similarity measure s_G from d_G as well[21]:

$$s_G(u, v) = 1 - d_G(u, v) \tag{3.15}$$

[21] This works because of $0 \leq d_G \leq 1$.

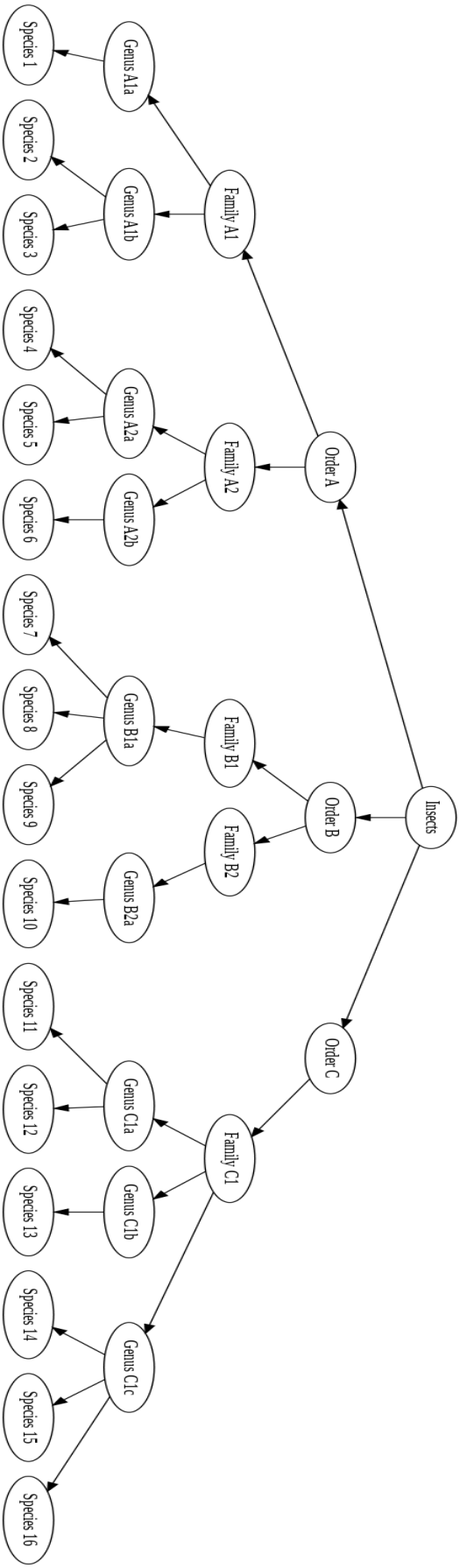


Figure 3.24. Pseudo-tree containing all 4 taxonomic sub-levels of the class *Insects*.

In order to preserve the structural information from G, the HCE algorithm will aim at finding for each of the n classes an embedding $\varphi(c_i) \in \mathbb{R}^n$, such that:

$$\varphi(c_i)^T \varphi(c_j) = s_G(c_i, c_j), \ \ \forall i, j \in \{1, ..., n\} \tag{3.16}$$

From the fact that $s_G(u, u) = 1$ it follows that the class embeddings should also be L_2-normalized, so $||\varphi(c_i)|| \stackrel{!}{=} 1$. Hence, the class embeddings will all lie on the n-dimensional unit hypersphere.

Algorithm 1 *Hierarchy-based Class Embeddings*

Input: DAG $G = (V, E)$, with edges $E \subseteq V \times V$ and set of n classes $C = \{c_1, ..., c_n\} \subseteq V$

Output: class embeddings $\varphi(c_i), \quad \forall i \in \{1, ..., n\}$

1: $\varphi(c_1) \leftarrow e_1$
2: **for** $i = 2$ to n **do**
3: $\quad \hat{\varphi}(c_i) \leftarrow$ solution to 3.17
4: $\quad x \leftarrow$ positive solution to 3.18
5: $\quad \varphi(c_i) \leftarrow (\hat{\varphi}(c_i), x, 0, ..., 0)$
6: **end for**

Algorithm 1 presents the steps for computing the HCEs. The choice of $\varphi(c_1)$ in step 1 of the algorithm is arbitrary. One may start with any point on the unit hypersphere but for simplification, e_1 is selected. For a fixed class c_i ($i \geq 2$), Equation 3.17 gives a system of $i - 1$ linear equations, where the coordinates of $\varphi(c_i)$ are the unknown variables. By setting the first embedding equal to e_1, only the first j dimensions of all $\varphi(c_j)$ are non-zero. This will eventually lead to a triangular matrix of embedded vectors. The solution $\hat{\varphi}(c_i)$, which has $i - 1$ coordinates, is not L_2-normalized yet, so an extra coordinate is needed to ensure unit length (Equation 3.18). The positive solution of the square root is preferred so that all class embeddings lie in the positive hyperoctant.

$$\varphi(c_i)^T \varphi(c_j) = s_G(c_i, c_j), \ \ \forall j \in \{1, ..., i - 1\} \tag{3.17}$$

$$\varphi(c_i)_i = +\sqrt{1 - ||\hat{\varphi}(c_i)||^2} \tag{3.18}$$

The associated matrix of embedded vectors computed from the graph of the pseudo-tree is visualized as a color map in Figure 3.25. Notice the block structure corresponding to each of the three insect orders. The spans of their embedded

vectors define orthogonal subspaces in $\mathbb{R}^9$. Families can also be recognized within each order block by looking at the repeating patterns that start with squares colored most closely to yellow.

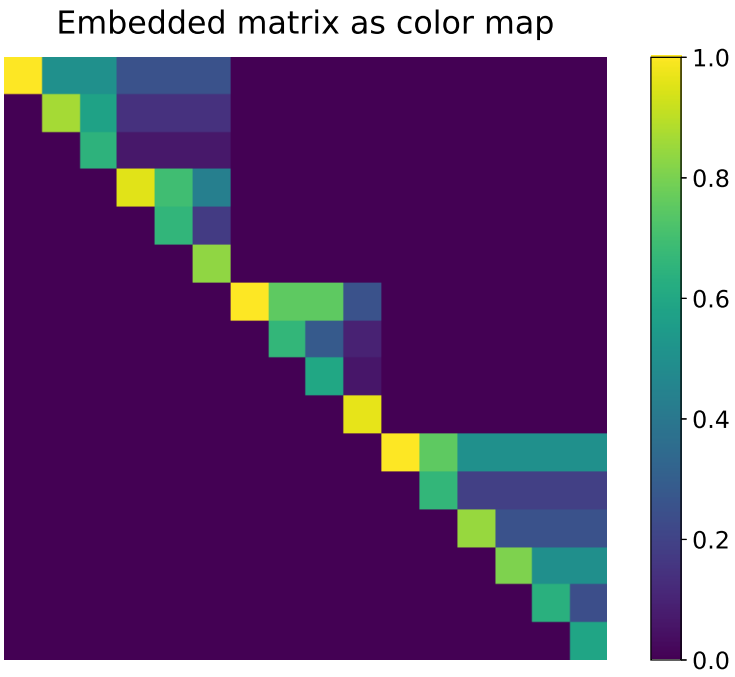


Figure 3.25. Colormap visualization of the embedding matrix corresponding to the tree in Figure 3.24.

As stated in Section 2.3.2, a simplified representation of the true Keogh taxonomy will be used (Figure 3.26). The class embeddings resulting from its graph are shown in 3.27. Non-zero coordinates corresponding to embedded vectors of the same genus have been accordingly colorized. The step-by-step computation that led to this result is available in Appendix A.

Dimensionality Reduction

As pointed out in [1], the algorithm described above would become tedious to implement if the number of classes increased significantly. In that case, one may consider replacing the n-dimensional embedding vectors $\varphi(c_i)$ with lower-dimensional vectors $\varphi^*(c_i)$, such that the pairwise dot products $\varphi^*(c_i)^T \cdot \varphi^*(c_j)$ only give the *best approximations* of the cross similarities $s(c_i, c_j)$. Such a set of vectors can easily be obtained via eigenvalue decomposition.

Let $\Phi \in \mathbb{R}^{n \times n}$ be the matrix of embedded vectors, that is, its i-th column is $\varphi(c_i)$. By multiplying Φ with itself, a new matrix S is constructed, which holds at every

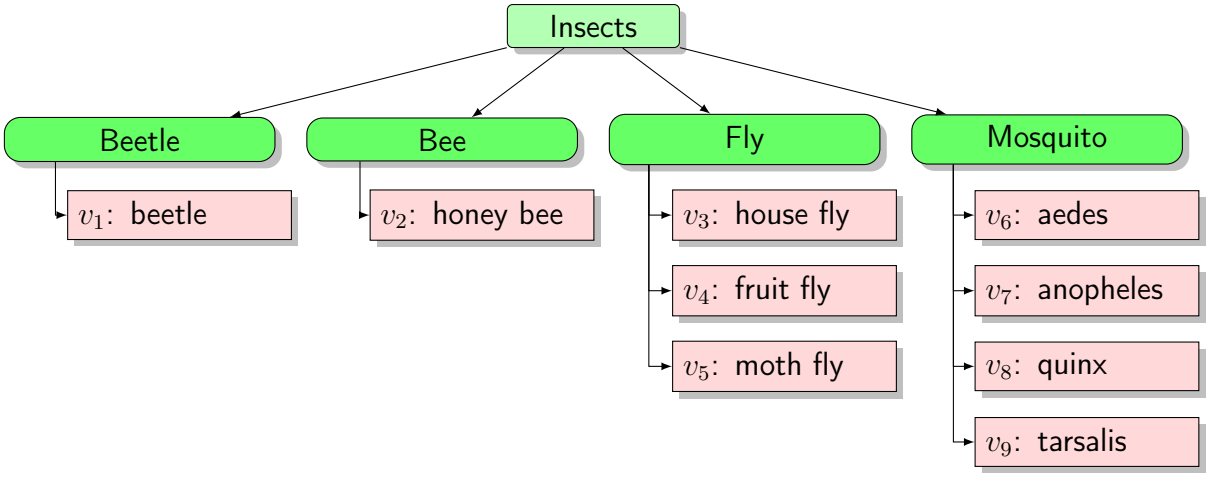


Figure 3.26. Simplified taxonomic tree for the Keogh dataset

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \sqrt{3}/2 & \sqrt{3}/6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \sqrt{6}/3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1/2 & 1/2 & 1/2 \\
0 & 0 & 0 & 0 & 0 & 0 & \sqrt{3}/2 & \sqrt{3}/6 & \sqrt{3}/6 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{6}/3 & \sqrt{6}/12 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{10}/4
\end{pmatrix}$$

Figure 3.27. HCEs in the Keogh dataset - green: Beetle, red: Bee, blue: Fly, orange: Mosquito

(i, j)-position the similarity value $s(c_i, c_j)$. Given that S is real and symmetric, it is also diagonalizable by an orthogonal matrix Q (see 3.19):

$$\Phi^T\Phi = \begin{pmatrix} — & \varphi(c_1)^T & — \\ & \vdots & \\ — & \varphi(c_n)^T & — \end{pmatrix} \begin{pmatrix} | & & | \\ \varphi(c_1) & \dots & \varphi(c_n) \\ | & & | \end{pmatrix} = S = QDQ^T \tag{3.19}$$

Q contains the eigenvectors of S and D its eigenvalues in descending order. The eigenvalues are all non-negative because $\Phi^T\Phi$ is positive semi-definite. Since D is a diagonal matrix, it can be rewritten as $D^{1/2}D^{1/2}$. This is convenient because it now follows from 3.19 that $\Phi = D^{1/2}Q^T$. Hence, lower-dimensional embedded vectors can be obtained by simply dropping some of the final eigenvectors in Q along with their

corresponding eigenvalues in D (the smallest eigenvalues account for less variability and can, therefore, be discarded, without losing a significant amount of information).

Suppose the first m eigenvectors are chosen ($m < n$). Then, Q and D are replaced by $Q^* \in \mathbb{R}^{n \times m}$ and $D^* \in \mathbb{R}^{m \times m}$, which leads to a new embedding matrix $\Phi^* \in \mathbb{R}^{m \times n}$. Figure 3.28 illustrates this dimensionality reduction in form of a commutative diagram.

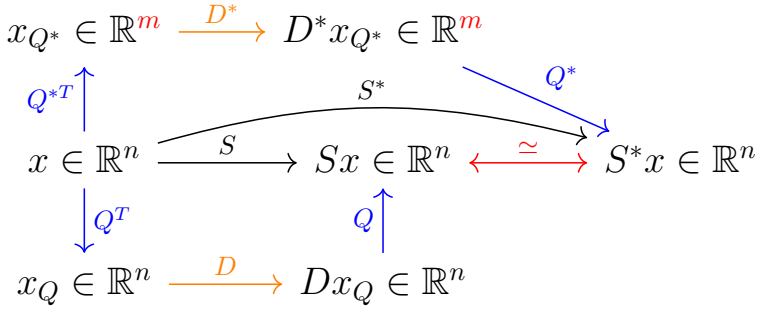


Figure 3.28. Commutative diagram of the dimensionality reduction process. By replacing Q with Q^* and D with D^* an approximation $S^* = Q^* D^* Q^{*T}$ of the true similarity matrix is computed.

3.4.5 Performance Measures

Loss Functions

Two types of network outputs were considered: class probabilities for genus or species and embeddings. The different possible combinations of these outputs are what defined the model architectures tested in this work (Section 5). The loss functions are directly associated with the type of output(s) delivered by the classifier.

For class probabilities, the *categorical cross-entropy* was utilized (Equation 3.20). B denotes a batch of m samples. A sample x_i can be of any input format (raw, PSD or spectrogram). y_i is the corresponding true class label (genus or species) for x_i, while $p : X \to [0, 1]^n$ is the mapping from the input space into the n-dimensional unity-hypercube that the model learns during training. The hypercube is normalized through the softmax activation.

$$CE(B) = -\frac{1}{m} \sum_{i=1}^{m} log(p(x_i)_{y_i}) \tag{3.20}$$

As an alternative to the classical approach, the so-called *focal loss* [26] was also investigated. Equation 3.21 shows that it is, basically, an extension of the cross-

entropy loss. By adding the probability of the false classes as a factor in front of the logarithm, the focal loss penalizes the model less for well-classified instances but more for misclassified examples (see Figure 3.29). The exponent γ controls how harshly misclassifications should be punished (for $\gamma = 0$, the focal loss becomes the cross-entropy again).

$$FL(B) = -\frac{1}{m}\sum_{i=1}^{m}(1 - p(x_i)_{y_i})^{\gamma} log(p(x_i)_{y_i}) \tag{3.21}$$

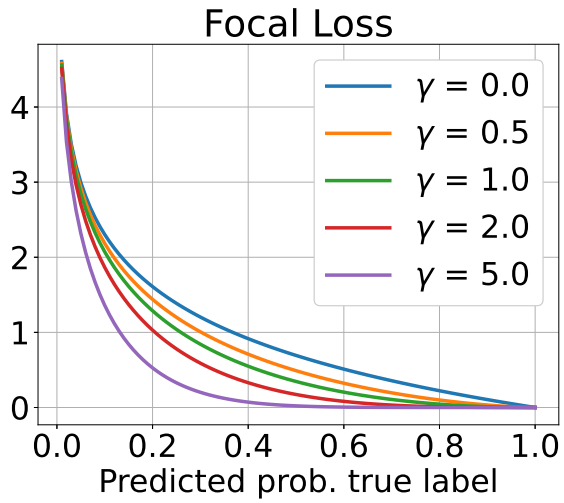


Figure 3.29. Focal loss curves for various γ parameters.

Given the problematic case of the bees being poorly classified (more on this in the following chapters), the idea of implementing a loss that is stricter with hard examples was worth looking into. Nonetheless, replacing the simple cross-entropy with the focal loss did not bring much improvement, neither in the overall accuracy (Figure 3.30) nor in the bees recognition accuracy. Hence, cross-entropy was chosen as the loss for the probabilistic outputs.

For the embedded output, a so-called *correlation* or *embedding loss (EL)* proposed by [1] was applied (Equation 3.22). The idea is to define the dot product between the predicted embedding $\psi(x_i)$ and the true class embedding $\phi(c_{y_i})$ as a similarity metric. Since both vectors lie in the positive octant of the unity hypersphere[22], their dot product will range between 0 and 1. This allows the difference *1 - dot product* to also move between the same boundaries and be used as a dissimilarity

[22] This was a constraint of the algorithm presented in Section 3.4.4.

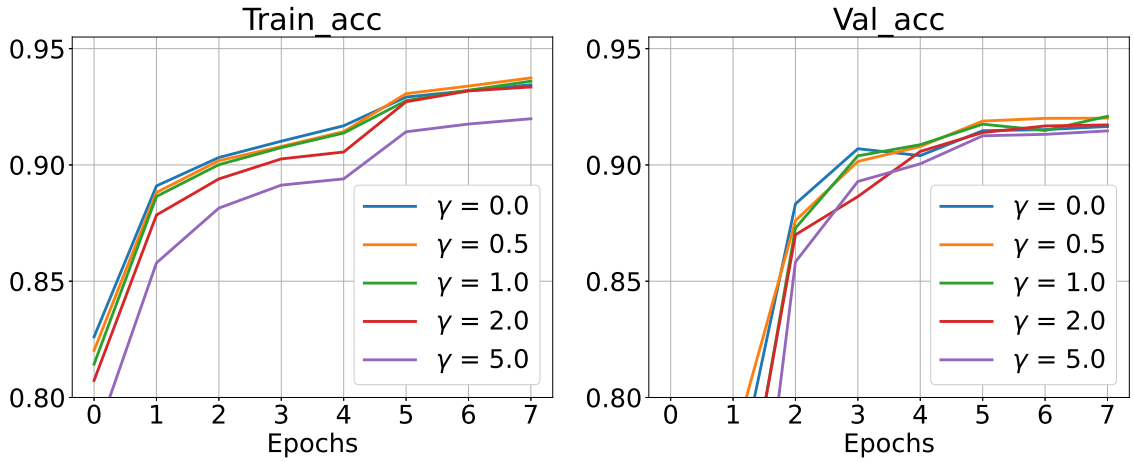


Figure 3.30. Training and validation accuracies for various γ parameters of the focal loss.

measure which needs to be minimized.

$$EL(B) = \frac{1}{m}\sum_{i=1}^{m}(1 - \psi(x_i)^T\phi(c_{y_i})) \tag{3.22}$$

Note that depending on what outputs the model produces, multiple losses may be used. The optimizer will aim at minimizing the sum of all those relevant losses[23].

Metrics

Again, different outputs require different metrics to track during and after training. For the probabilistic output, the genus/species accuracy scores were monitored on the whole training and validation sets, as well as on each class subset via confusion matrices. The embedded outputs were examined through the embedding similarity metric described above.

As an alternative to the confusion matrices, *ROC*-curves are also analyzed. In a simple binary classification scenario, the ROC curve plots the *recall* (*true positive rate*) against the *false positive rate* [11]. To upgrade it to multi-class problems, the *one-versus-all* strategy was applied, where the prediction rates of one label would be compared to the cumulated prediction rates of all the other labels (see Section 5.1).

[23]It is possible to weight the losses differently, but experiments have shown that a weighted loss sum is not necessarily better than a simple sum (where all weights are set to 1).

Implementation and Training

This section is dedicated to the actual training process. First and foremost, the design of the networks will be presented. A description of the hypertuning pipeline will follow, leading in the end to the particularities involved in applying Transfer Learning both on the extended Keogh dataset with all its 9 species and the Potatamis dataset.

4.1 Model Architectures

Standard classifiers from the literature can only reach a limited benchmark (Table 4.1). Hence, the ML problem this work addresses requires a more sophisticated approach. An important objective is to develop a flexible piece of software that can be adapted to changing aspects of the classification task. Therefore, the model components will be conceived as modules that can easily be swapped, connected or extended[1].

Classifier	**Accuracy (%)**
MLP	86.73
Linear SVC	75.15
Random Forest	86.90

Table 4.1. Accuracy scores on the Keogh validation set for some standard *sklearn* classifiers with default parameters.

Figures 6.5 to 6.9 from Appendix E depict the main general architectures used for the upcoming experiments in form of directed graphs. For space reasons, only the custom PSD classifiers are explicitly shown. The **CNN_Blocks** are represented by a custom **CNN1D** class (**CNN2D** for spectrograms) made up of 5 convolutional modules, each containing in order a *Convolution* layer, followed by *Batch Normalization*, *ReLU* and *Max Pooling*. Nevertheless, the feature extractor can also be

[1]This is made possible by the versatility of the *tf.keras.Model* API.

replaced by the CNN body of other well-known models such as MobileNet [15] or EfficientNet [42], as described in the next chapter[2]. It does not have to be a CNN at all and it can even miss completely.

Apart from the flexibility in choosing the feature extractor, the top layers responsible for the actual predictions may also vary to a high degree. What sets the models in this paper apart from standard architectures is really the embedding module (Figures 6.6, 6.7 and 6.9). The output of an embedding branch is not probabilistic but an approximation of the HCE computed as explained in Section 3.4.4. The claim of this thesis is that models that learn the hierarchical encoding along with the class labels are able to perform better at classifying insects.

4.2 Hyperparameter Tuning

When dealing with complex models - and especially in Deep Learning they can grow indefinitely complex - the number of hyperparameters can quickly become overwhelming. In an ideal setting with limitless resources, one would try to fully optimize all these values. But given the time and computation constraints for this work, some of the hyperparameters were either fixed straight from the beginning based on knowledge about the task (for instance the dropout rate, batch size, number of convolutional kernels) or partially optimized via experiments that would only allow one parameter to vary while keeping all the others constant, like in the case of the learning schedule and SMOTE strategy.

In this section, parameters that have a great(er) influence on the model performance, such as learning rate, regularizing term or architecture[3], are going to be optimized through more advanced techniques. The entire training pipeline is depicted in Figure 4.1.

For every input format, all 5 architectures are optimized on the clipped Keogh

[2]The reader may wonder why such well-known pre-trained models are not preferred here. Indeed, experiments done with networks like MobileNet ($\alpha = 0.25$), DenseNet121, ResNet50 and EfficientNetB0 proved that they can reach higher accuracy scores than their custom made counterparts (up to 2 percents more). However, there are other downsides. First of all, very large networks with millions of parameters do not come into question at all for the *KInsekt* project. Their inference times are by a factor of 10 longer than the custom classifiers, making them impractical to run on a Raspberry Pi. MobileNet would fulfill the compactness requirements, but further investigations revealed that it had problems with identifying relevant wingbeat features in spectrograms, which made the architecture unreliable (see Section 5.2).

[3]For the multi-output models the loss weights can also be optimized. Yet, experiments have shown that letting the hypertuning pipeline choose them would cause a drop in accuracy, as opposed to just setting them to their default value of 1.0. Arguably, searching for better loss weights is a requirement only if the losses have different scales, which is not the case here.

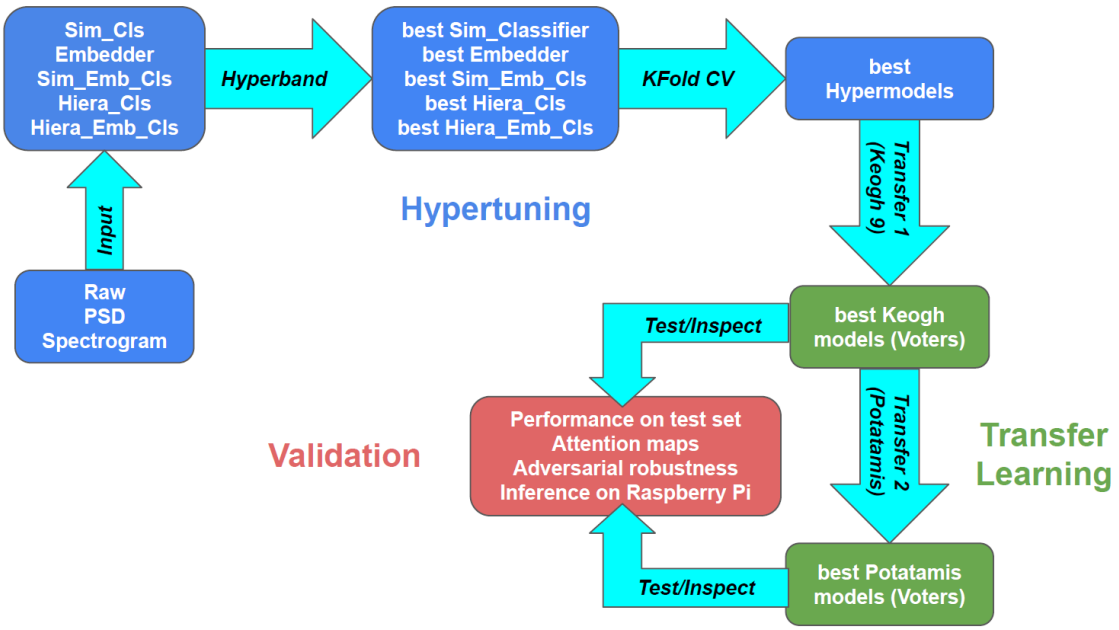


Figure 4.1. Training Pipeline

dataset (7 species) with respect to learning rate and regularization term via an algorithm called *Hyperband*, described more minutely in the next section. The best configurations are then passed on to retrain each model in *4-fold cross-validation*. Based on average genus accuracies, species accuracies and other metrics such as confusion matrices, the best 'hypermodels' are selected. These are the classifiers used to perform Transfer Learning on the extended Keogh dataset (9 species). Afterwards, judging by their generalization power, the best hypermodels will be promoted to *Voters* in the validation stage. That is, they will get to make predictions on the (until now) untouched test set, both individually and as a combined voting classifier. Additionally, they will be subject to further stress tests such as adversarial attacks, pixel flipping in attention maps and inference precision and speed on a Raspberry Pi. In parallel, in a second Transfer Learning phase, the same Voters will also be investigated on the Potatamis dataset.

Before proceeding to describe the optimization process, a few words about data split: for both datasets the same splitting ratio was applied: 60% training, 20% validation and 20% testing. The subsets are stratified, meaning that the sample ratios between classes are conserved. However, different splitting steps were implemented. Given that the Potatamis dataset is more than 15 times larger than Keogh, the splitting was done only once at the beginning, after which the wav-files from each subset were converted into *float*-amplitudes and stored as *pickle* files. Thus, on every new session, the Potatamis signals would already be available as split and

converted subsets. In contrast, the wav-files from the Keogh dataset are stored as such and on each new run, they are read and split again. To ensure reproducibility, a random seed is held constant each time the whole set is split.

4.2.1 Hyperband

In 2015, Jamieson and Talwalker [17] proposed an algorithm called *Successive Halving* that would formulate hyperoptimization as a *multi-armed bandit problem*. The goal is to find a set of configurations that minimizes the *regret* - here, the loss function - while constrained by a limited *budget* - here, computation time or number of training epochs. The idea was implemented as follows: all configurations would be trained for a limited amount of epochs (called the *exploration phase*), after which the worst-performing half of the models would be discarded. The rest would be trained further for another few epochs and again only the best half survived. In the end, there would only be one fully trained model (the so-called *exploitation phase*). The assumption is that there is enough information after only a few training iterations to be able to identify useless configurations. By discarding them as early as possible, precious time is saved up for the truly lucrative combinations. This is not the case with classical hypertuning techniques such as Grid Search or Random Search, where all configurations from the parameter space are considered till the end of training.

As an extension of *Successive Halving*, *Hyperband* was published in 2018 [25]. Unlike its predecessor, Hyperband repeatedly considers subsets of random configurations from the given parameter distribution, on which Successive Halving is applied. This step is named *bracket*. In each bracket, a different minimum number of training epochs (*minimal resources*) are allocated for every model. Thus, the algorithm aims at finding a trade-off between the number of combinations considered and minimal resources. To put it differently: is it better to train many configurations with few resources each or fewer configurations with more resources each?

There are two inputs required by Hyperband: number of maximal resources R, which dictates how many epochs a model can be trained at most, and an elimination factor η, which determines how many models are dropped every trial[4]. There are three guidelines in [25] regarding the best Hyperband setting for the task at hand:

1. Early Stopping should be integrated into the optimization process, which allows the user to set a maximal number of epochs slightly higher than the

[4] A trial is a step in Successive Halving.

expected convergence time. For the following experiments, $R = 30$ was chosen.

2. The elimination factor should be chosen such that 5 brackets are created. The number of brackets is given by $\lfloor \log_\eta(R) \rfloor + 1$, so η was set to 2. This means that in every trial half of the models are discarded.

3. Given the randomness of choosing configurations, there is a chance that optimal parameters are never considered. Therefore, one should repeat the whole process indefinitely to make the results more legitimate. In this paper, every Hyperband search was run three times.

The ideal configurations found by Hyperband can be read in the two columns **LR** and **Reg** of Table 4.2 in the following section.

4.2.2 K-fold Cross-Validation

After finding good values for the learning rates and regularization terms, the models proceed to the next hypertuning stage, namely the *k-fold Cross-Validation*. The objective is to have an idea of how much variance resides in the overall validation accuracy scores that are registered after one full training process. By retraining the network a few times on new train-validation splits, the accuracy scores will not be dependent on the data split anymore. Instead, they offer a confidence interval around the mean accuracy, revealing how that particular model would perform on average when trained on random new splits.

Table 4.2 presents the results of the k-fold learning phase, for k = 4. The statistics in the columns **GenAcc** and **SpecAcc** are written in the form *mean +/- std. dev.*, while the last column only shows the mean bee classification accuracy (both for genus and species, if they are different). Highlighted in red are the best 'hyperclassifiers' qualified for the next round of Transfer Learning. Raw signals were dropped altogether as input format, as it became clear that the alternatives achieve higher performance irrespective of the model architecture. An example of how the k-fold statistics were computed can be seen in Figures 4.3 and 4.2.

PSD

Model	LR	Reg	GenAcc	SpecAcc	Bees mean acc
SimpleCls	1e-3	1e-4	0.9675 +/- 0.0025	0.9182 +/- 0.0029	0.6
Emb	1e-3	1e-4	0.9651 +/- 0.004	0.9114 +/- 0.0047	0.59
SimpleEmbCls	5e-3	1e-2	0.9682 +/- 0.0018	0.9174 +/- 0.0022	0.65
HieraCls	5e-4	1e-3	0.9665 +/- 0.0028	0.9178 +/- 0.0022	0.58/0.62
HieraEmbCls	5e-3	1e-3	0.9682 +/- 0.0052	0.9171 +/- 0.003	0.62/0.59

Raw

Model	LR	Reg	GenAcc	SpecAcc	Bees mean acc
SimpleCls	1e-4	5e-4	0.9412 +/- 0.0017	0.8603 +/- 0.0052	-
Emb	1e-4	1e-4	0.9412 +/- 0.0033	0.8459 +/- 0.0104	-
SimpleEmbCls	1e-3	5e-4	0.9503 +/- 0.006	0.889 +/- 0.0079	0.51
HieraCls	1e-4	1e-3	0.9413 +/- 0.004	0.8563 +/- 0.007	-
HieraEmbCls	1e-3	1e-4	0.9512 +/- 0.0054	0.8827 +/- 0.0052	0.28/0.37

Spectro

Model	LR	Reg	GenAcc	SpecAcc	Bees mean acc
SimpleCls	5e-4	1e-3	0.9614 +/- 0.0017	0.9107 +/- 0.0063	0.61
Emb	1e-3	1e-4	0.9621 +/- 0.0014	0.9097 +/- 0.0023	0.54
SimpleEmbCls	5e-3	5e-3	0.9602 +/- 0.0053	0.915 +/- 0.0086	0.62
HieraCls	5e-4	5e-4	0.9663 +/- 0.003	0.9205 +/- 0.0011	0.62/0.67
HieraEmbCls	5e-3	1e-3	0.9628 +/- 0.0032	0.9146 +/- 0.0019	0.56/0.61

Table 4.2. Hypertuning results on the Keogh dataset. Hyperband was used to find the optimal learning rate and regularization term. The accuracy values are summaries of the 4-fold cross-validation.

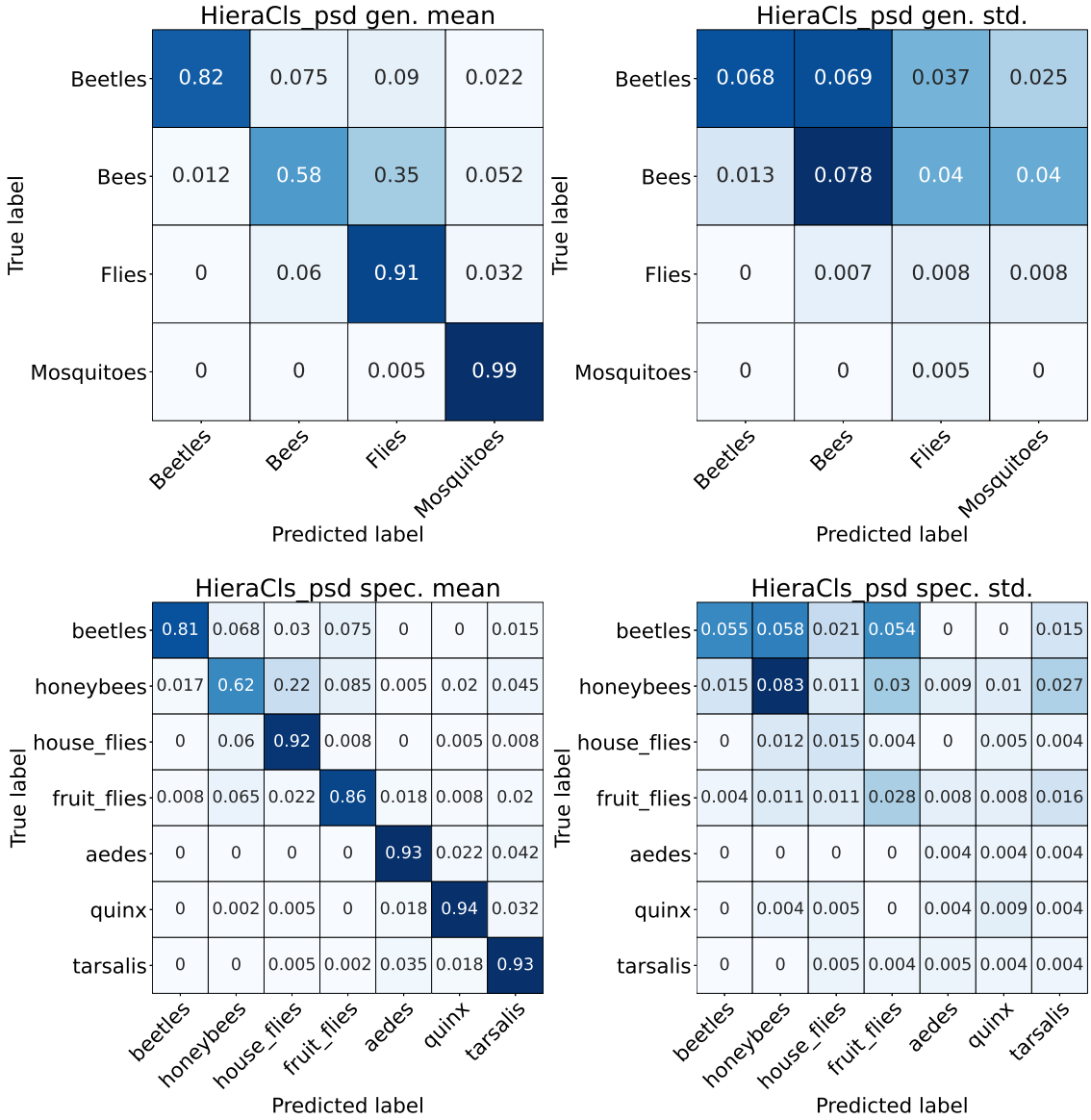


Figure 4.2. Mean confusion matrices (left) along with their corresponding standard deviations (right) for the Hierarchical Classifier (PSD) after 4-fold cross-validation on the Keogh dataset with 7 species.

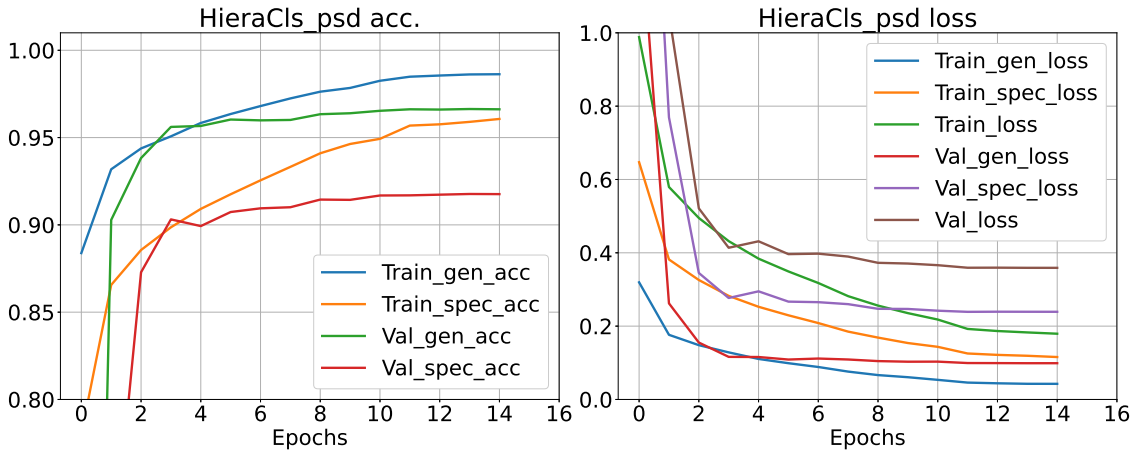


Figure 4.3. Mean training and validation curves for the Hierarchical Classifier (PSD). In the right plot, Train_loss is the sum of Train_spec_loss and Train_gen_loss. The same goes for Val_loss.

4.3 Transfer Learning

Now it is time to test how well the pre-trained models can generalize and be applied to other similar hierarchical classification tasks. Transfer Learning will be performed in two scenarios: on the extended Keogh dataset with all 9 species and the Potatamis dataset.

4.3.1 Keogh Dataset - 9 Species

A total of 10 models (5 per input format) have qualified out of the previous k-fold validation stage. They were trained one last time on a fixed train-validation split and their weights were stored to be reused now.

The Transfer Learning experiments consist of two steps:

1. **Frozen model training**: The weights of the convolutional blocks are reloaded into the current new network and set to non-trainable. Only the Dense layers are trained either until Early Stopping intervenes or the maximal number of epochs is reached.

2. **Unfrozen model training**: Now every weight is set to trainable again and training continues for a few more iterations with a much smaller learning rate[5].

[5]It was always the case that the learning schedule had already reduced the learning rate twice by a factor of 10 by the time the frozen training phase was completed. Hence, the unfrozen stage would just continue with that last value of the learning rate.

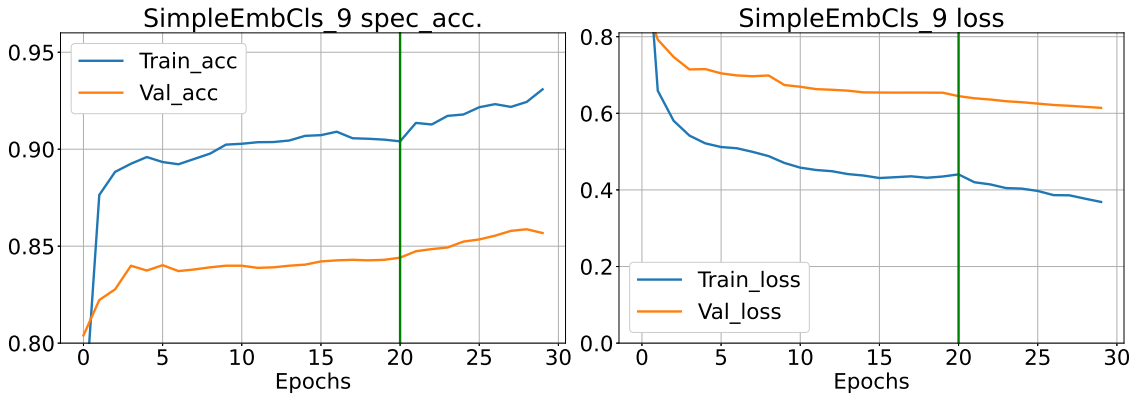


Figure 4.4. Simple Embedder Classifier accuracy and loss curves on spectrograms from all 9 Keogh species during frozen/unfrozen training. The green line marks the first completed epoch of unfrozen training. All samples from the training set have been used.

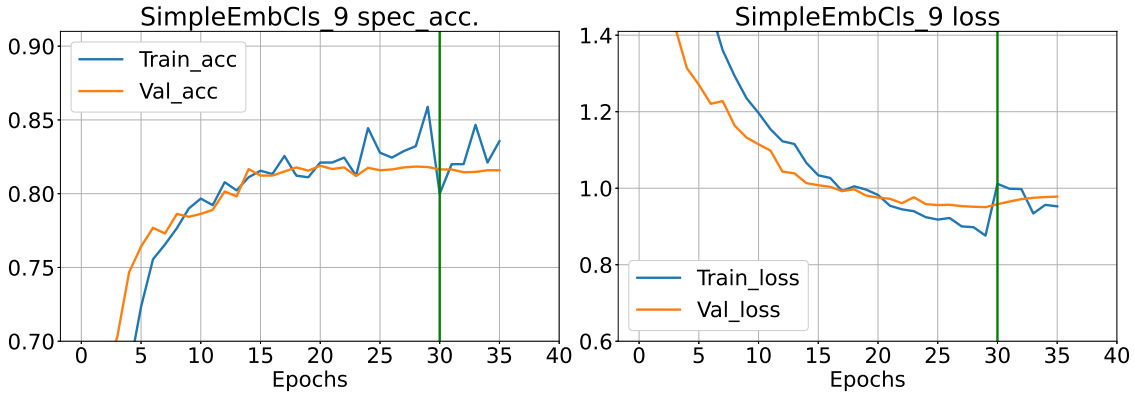


Figure 4.5. Simple Embedder Classifier accuracy and loss curves on spectrograms from all 9 Keogh species during frozen/unfrozen training. The green line marks the first completed epoch of unfrozen training. Only 100 random samples from each species were fed to the classifier.

Figures 4.4 and 4.5 give an example of how the training curves of a model would look like during frozen and unfrozen training. In the first case, the Simple Embedder Classifier was trained on the whole training set, while in the second only on subsets of 100 random samples from each species[6]. Notice that the difference in the validation accuracy only revolves around 5%, which is encouraging, as far as usability in future applications is concerned. As expected, performance scores are more erratic when drastic undersampling is applied. It also appears that the Early Stopper could have been set to be more 'impatient' since in both cases the algorithm had converged long before reaching the green mark. The left plot in Figure 4.4 even shows that the model is highly prone to overfitting if training does not end early enough.

The results of the Transfer Learning stage on the extended Keogh dataset are

[6]SMOTE was disabled here since the undersampling already took care of balancing the classes.

summarised in Table 4.3. This time the models marked in red are promoted to *Voters* and qualify for the next and final stage of the whole training pipeline - the Validation stage.

PSD

Model	GenAcc	SpecAcc	Bee acc.
SimpleCls	0.9671	0.8817	0.55
Emb	0.9646	0.859	0.52
SimpleEmbCls	0.9654	0.8748	0.56
HieraCls	0.9657	0.8822	0.49/0.55
HieraEmbCls	0.9671	0.8723	0.62/0.63

Spectro

Model	GenAcc	SpecAcc	Bee acc.
SimpleCls	0.961	0.8557	0.60
Emb	0.9657	0.8424	0.55
SimpleEmbCls	0.9613	0.8568	0.58
HieraCls	0.961	0.8435	0.49/0.59
HieraEmbCls	0.9643	0.8543	0.6/0.67

Table 4.3. Transfer Learning results (Keogh 9 species)

	PSD		Spectro	
Model	GenAcc	SpecAcc	GenAcc	SpecAcc
SimpleCls	0.91	0.8655	0.9155	0.8672
SimpleEmbCls	0.9006	0.8541	0.9174	0.8656
HieraCls	0.904	0.8619	0.9136	0.869
HieraEmbCls	0.9017	0.8539	0.9179	0.8619

Table 4.4. Transfer Learning results (Potatamis)

4.3.2 Potatamis Dataset

For the Potatamis dataset, the best architectures on the Keogh experiments (the Voters selected above) are tested directly. Apart from architecture and weights from the CNN body (for initialization), other hyperparameters (learning rate, for instance) have also been refactored. In contrast to the transition between the clipped and extended Keogh dataset, where the whole CNN was initially frozen, here only the first two convolutional blocks were set to non-trainable. Freezing more layers ended up in unsatisfactory accuracy values (only about 80% at most). The results of the Transfer Learning phase on the Potatamis set are summarised in Table 4.4.

Validation

After completing the model selection cycle, it is time to test and validate the remaining classifiers[1]. Starting from this point on no more changes regarding architecture or hyperparameters are permitted. In this chapter, the final models will be exposed for the first time to the test sets of their corresponding datasets. The inference will be done on the Google CPU in Colab as well as on a Raspberry Pi. In addition to performance on the test set, some other so-called *stress tests* will also be applied, such as *adversarial attacks* and *pixel flipping* in attention maps, in an attempt to check whether the classifiers have successfully identified relevant features of the wingbeats and base their predictions on them.

5.1 Performance on Test Set

5.1.1 Google CPU

The classifiers are first evaluated on the test sets in Colab. The genus and species accuracies, as well as model sizes and inference latency[2], are gathered together in the tables in Appendix B (rows labeled 'Original'). Individual species accuracies are also compared to one another in Figures 5.1 and 5.2 for both datasets in form of bar plots, to avoid showing multiple confusion matrices. It appears that, in general, PSD models slightly outperform spectrogram models on the Keogh test set (the PSD bars rise higher than the spectrogram ones). For the Potatamis test set, the opposite is true.

[1]The models for the Keogh dataset have been retrained one last time on the combined training and validation sets. This extra step was not taken for the Potatamis dataset given its data abundance.

[2]For computing the latency per sample in Colab, the test set was not converted into a TensorFlow Dataset, in order to simulate a real-life scenario where signals would not come in as batches but as single instances.

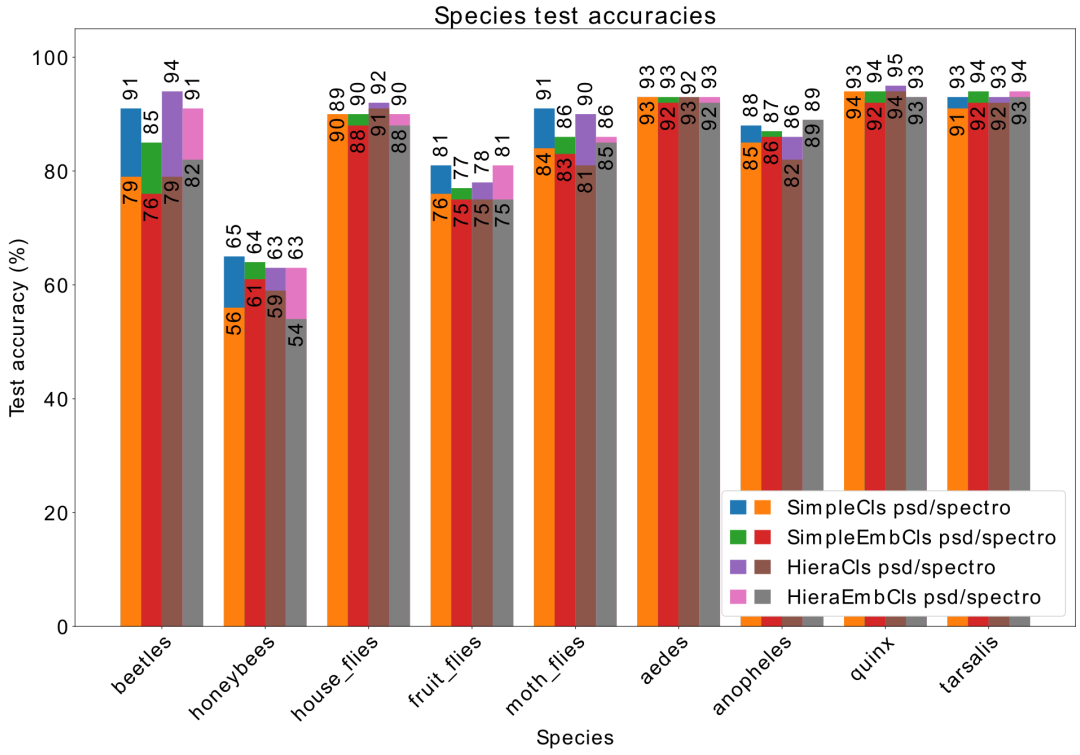


Figure 5.1. Bar plot with species accuracies on the Keogh test set. PSD models are plotted behind spectrogram models.

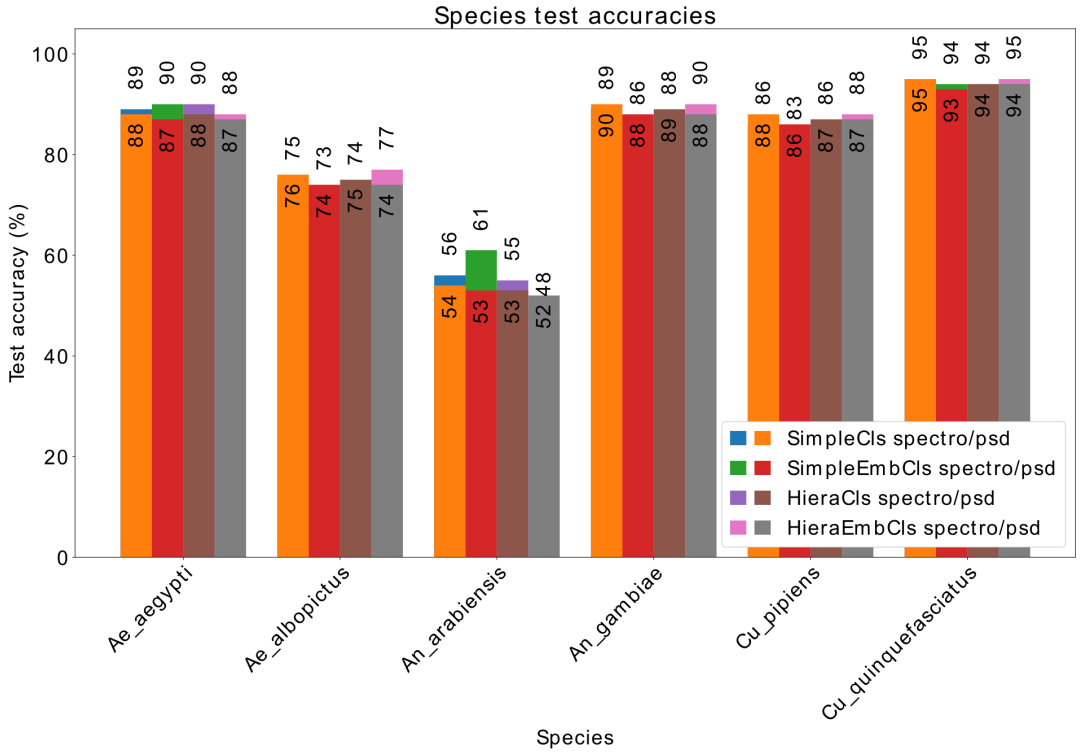


Figure 5.2. Bar plot with species accuracies on the Potatamis test set. Spectrogram models are plotted behind PSD models.

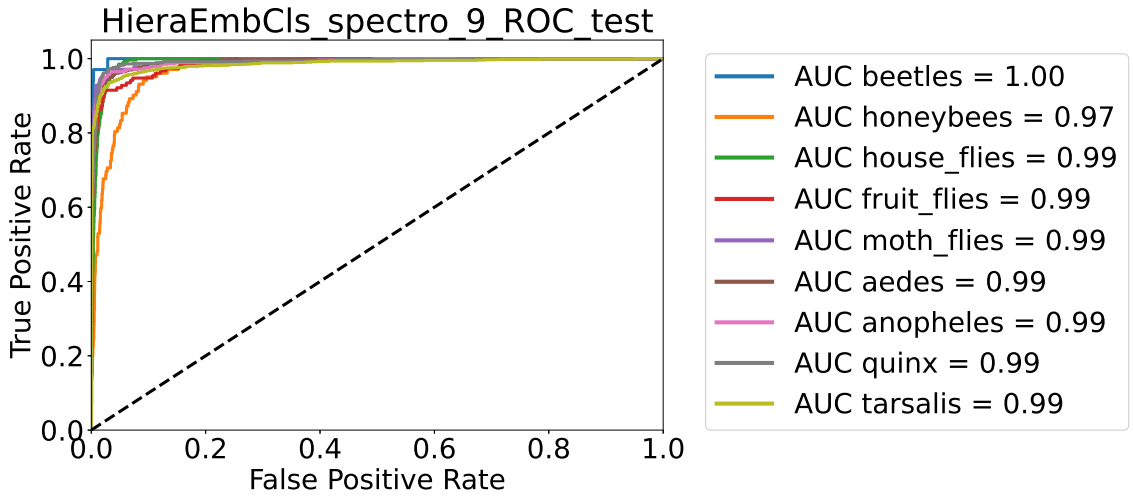


Figure 5.3. ROC Curves and AUC values after inference on the Keogh test set with a Hierarchical Embedder-Classifier.

Figure 5.3 shows an example of a ROC plot computed as described in Section 3.4.5. As expected, the honey bees have the lowest AUC value since they are the hardest category to recognize.

Investigating the worst predictions for all models revealed that different classifiers were having difficulties with different samples. This suggested that building a *voting ensemble*[3] out of the existing models, for instance, a general PSD and a general spectrogram classifier, would have a good chance of increasing the overall accuracy. And it turned out this was, indeed, the case. On the Keogh test set, the PSD Voting Classifier had an average species accuracy of 0.9193, while its spectrogram counterpart would reach 0.9096. When compared to the individual model accuracies in Appendix B, Voting won an additional percent on average for both PSD and spectrograms. The increase on the Potatamis test set was less remarkable for the PSD Voting - only 0.8662 - but significant for the spectrograms - 0.8809.

5.1.2 Raspberry Pi

The second stage of test set experiments took place on a Raspberry Pi 4 B (4GB RAM). When deploying neural networks on small devices, special formats are required that reduce the model size and speed up computations. TensorFlow offers the *TFLite* format which converts the pre-trained model into a so-called *interpreter*. From [43]:

[3]Soft Voting is meant here, where probability outputs from all Voters are averaged out. Hard Voting would not even make sense when having an even number of Voters.

"The TensorFlow Lite interpreter is designed to be lean and fast. The interpreter uses a static graph ordering and a custom (less-dynamic) memory allocator to ensure minimal load, initialization, and execution latency."

To further reduce the model size and accelerate computations, one can also *quantize* the weights [30]:

"Post-training quantization is a conversion technique that can reduce model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy."

For this they offer three methods:

1. **Dynamic range quantization**: stores the weights in *int8*-format and converts them to floating points only during inference.

2. **Full integer quantization**: weights are stored with 8-bit precision and are kept like this during inference as well.

3. **Float16 quantization**: keeps the weights as floating points but reduces their size to *float16*.

Apart from the quantized TFLite models, an unquantized version has also been saved as a baseline for the experiments on Raspberry. All results for both the Keogh and Potatamis dataset are summarised in the tables in Appendix B (every row except for 'Original' represents a TFLite model).

The first aspect worth mentioning is that the model size and inference time requirements have been successfully met. All networks were able to run efficiently on the Raspberry Pi, with little to no drops in accuracy (in the case of the Keogh test set, the accuracies would sometimes even slightly increase but this may be due to the different randomizations applied by the Colab environment and Raspberry's CPU). The gain in prediction speed is indisputable, the inference time often being cut down by a power of 10. And as far as model size is concerned, the quantized formats noticeably reduce the memory space needed to store the weights.

In terms of accuracy, none of the models seem to stand out on the Keogh dataset. In the case of genus, all models revolve around 96.5% for both PSD and spectrograms. In the case of species, they stay close to 91% for PSD and 90% for spectrograms. There is somewhat more variation in the numbers for the Potatamis dataset. Genus accuracies approximately range from 90 to 92% and species accuracies from 85.5 to 87%. All in all, it appears that the Simple Classifier outperforms the other

models on the Potatamis dataset by a slight margin, followed up by the Hierarchical Classifier on the PSD signals and the Hierarchical Embedder-Classifier on the spectrograms.

Nonetheless, test accuracies may not offer the whole picture of how a model is going to perform in real life. For this, supplementary stress tests will be applied, as described in the upcoming section.

5.2 XAI

The field of XAI (Explainable AI) is gaining more and more attention nowadays since there is a need to prevent intelligent systems from turning into what is called a 'black box' that nobody can understand anymore. A classical example comes from medical imagery, where doctors are not just interested in the correct diagnostic of a disease but also need to know if the machine based its decision on relevant disease indicators. Another example are systems designed to automatically sort candidates that applied for a job. The employer may want to ascertain that the hierarchy is built according to professional qualifications, not gender or citizenship.

Moreover, it is often the case with complex models that their performance on the test set is not that indicative of how well they will perform 'in the real world'. Section 3.4.3 already mentioned the problem with data shifts. Another related issue is *underspecificity* [8]. In Linear Algebra a system of equations is said to be underspecified when there are more unknowns than equations. The additional degrees of freedom disallow a unique solution and the system ends up having infinitely many solutions[4]. The same line of reasoning can be applied to overparameterized ML models. There are multiple configurations that achieve maximal performance on the validation metrics, but they remain, nevertheless, local solutions that do not guarantee similarly high performance on new data. Therefore, before deploying a model, one may want to check whether it is behaving as expected.

5.2.1 Attention Maps

In the case of the classifiers developed here, it is expected from them that they base their judgement on relevant wingbeat frequencies, not on noise. This can be checked by computing *attention maps*. These are methods that take a model and its

[4]For mathematical completeness: the matrix corresponding to the linear system is assumed to have full rank.

prediction as input and determine which features of the signal have contributed most to the decision. For this work, two methods have proven to be efficient[5]: *Saliency Maps* and *Guided Backpropagation*.

Saliency Maps

Also called *vanilla gradient method* or *gradient backpropagation*, Saliency Maps (SMs) are, indeed, the most straightforward idea one may think of for visualizing relevance in the input [36]. The assumption here is that the higher the gradients connected to a pixel are, the greater the influence that pixel has on the prediction. Therefore, given an image $x_0 \in \mathbb{R}^{m \times k}$ (remember that spectrograms are grayscale images without RGB channels) and a trained model f, the SM computes the derivative of the prediction $f(x_0)$ w.r.t. the input image (Equation 5.1). Typically, by prediction, a vector of n unnormalized[6] scores is implied. Hence, attention maps will only be computed for the probabilistic outputs of the models, although the method can easily be generalized for the embedded outputs as well.

$$SM(x_0, f) = \frac{\partial f}{\partial x}(x_0) \tag{5.1}$$

Notice that Equation 5.1 delivers an $m \times k \times n$ tensor containing one heatmap for every class label. In the experiments here, the focus will lie on the heatmaps w.r.t. the true class and the predicted class. Figure 6.1 in Appendix C shows some examples of such SMs for all custom spectrogram models plus MobileNet and EfficientNet. The maps have been normalized to [0, 1]. Irrelevant pixels are marked with dark blue; as the colors progress from blue to yellow and red, the relevance increases. Observe that while some networks seem to have correctly identified the characteristic frequencies from the yellow-hued regions in the spectrograms (Simple and Hierarchical Embedder-Classifiers in particular and partly EfficientNet), the others appear to be distracted by the noise in the upper regions of the spectrograms.

Also fascinating is that the different architectures of the feature extractors draw different shapes in the SMs: dashes for EfficientNet, grid pattern for MobileNet and dots for the custom models.

[5] Although 'efficient' is still a fuzzy term when it comes to evaluating attention methods, as will be pointed out in the upcoming paragraphs.

[6] The Softmax layer is removed.

Guided Backpropagation

Guided Backpropagation (GBP) works according to the same principle of gradient backward pass but the novelty consists in applying a so-called *Guided* or *backward ReLU* [40]. This extra layer maps any negative gradients to 0 during the backward pass, just like a normal ReLU would do during the forward pass. The motivation for this is that positive gradients (responsible for the selected class probability) tend to cancel out with the negative gradients that contribute rather to the prediction of the other classes. Beware that this does not mean that the GBP maps will not have any negative pixels. They will because Guided ReLU is not applied between the input layer and the first convolution, where weights may as well be negative. Figure 6.2 from Appendix C shows examples of GBP maps for the same inputs and models as in Figure 6.1. The maps have been normalized to [0, 1]. Irrelevant pixels are marked with black; as the colors progress from black to purple and yellow, the relevance increases. Again, both Embedder-Classifiers appear to have paid attention to the right regions.

The 1D Case

Just as a short extrapolation: although initially designed for two-dimensional image classifiers, attention maps can easily be applied to one-dimensional signals as well. Figure 5.4 gives an example.

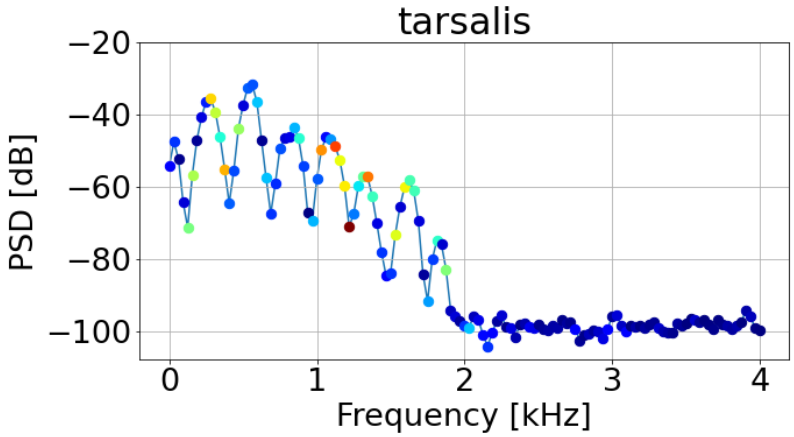


Figure 5.4. SM method applied on a tarsalis PSD signal classified correctly by a Simple Embedder-Classifier. The visualization of the SM is made possible through the colored scatter plot (same normalization as in the 2D-SMs, so dark blue means 0 and red is 1). Notice the bright colors assigned to frequencies of up to 2 kHz, which correspond to the true flight event (body movement and wingbeats). Above 2 kHz, the whole region is marked as irrelevant by the SM, meaning that the classifier did not pay attention to noise.

5.2.2 Monte Carlo Dropout

Slightly different parameter settings can lead to models that perform equally well in supervised contexts but are very unpredictable when let loose in the real world. It is, therefore, a good idea to create ensembles of similar networks that share the same architecture but differ in a few aspects such as batch size used for training or random seed and let them run multiple predictions on the same data. The effect is that the performance scores will be more representative for the chosen model design, giving an idea about how much variability there is in that classifier's predictions.

In the context of XAI, attention maps computed on such ensembles will better highlight the truly relevant input features because they will have prevailed across multiple predictions. In [4], an ensemble of models was implemented for this purpose via the Monte Carlo Dropout (MCD) discussed in Section 3.4.1. Along with Pixel Flipping - presented in the next section - this method offers a good stress test to evaluate classification performance beyond test accuracies.

Figure 5.5 proves how unsure some models actually are when seeing hard examples such as the honey bees from the Keogh dataset. Three models - Simple Embedder-Classifier, MobileNet and EfficientNet - had their Dropout layer replaced by an MCD layer with the same dropout rate of 0.5. They were each run 500 times on the spectrogram shown on the left side. Next to it, the resulting softmax probabilities of the true class are plotted as histograms. Notice the prediction variability in each model. The MobileNet even predicts the wrong label more often than the right one.

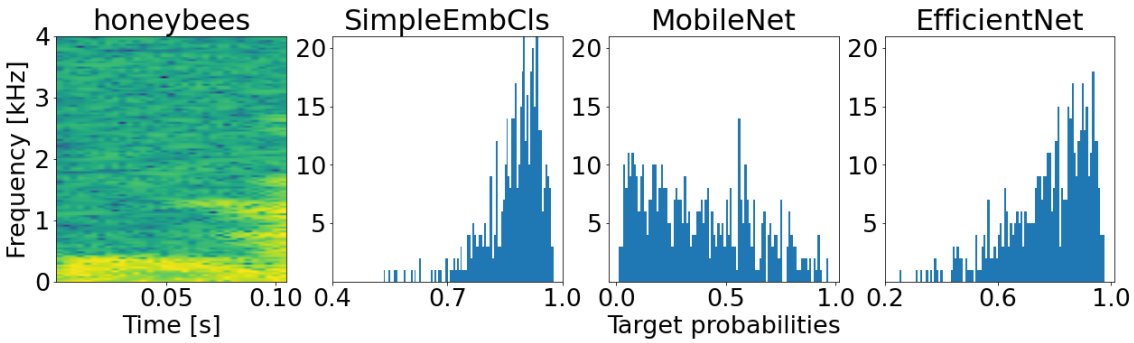


Figure 5.5. Distribution of 500 softmax probabilities predicted by a Simple Embedder-Classifier, MobileNet and EfficientNet with MCD activated for the spectrogram shown on the left side.

For each of these models, an SM was computed after each prediction w.r.t. the true class. The result is a total of 500 SMs per classifier. To gain a better un-

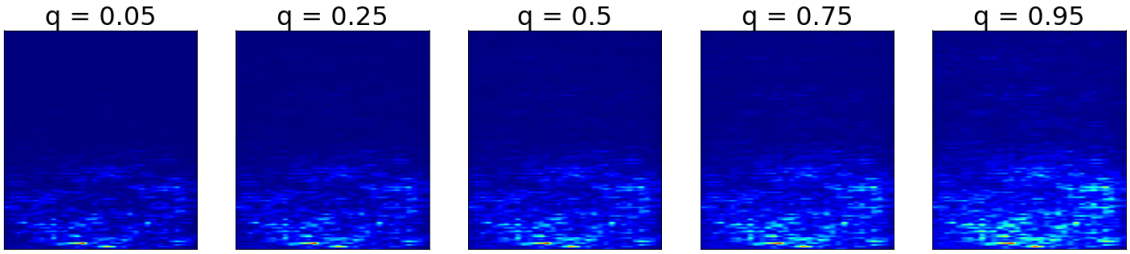


Figure 5.6. Quantiles computed over the 500 SMs for the bee in Figure 5.5 with the Simple Embedder-Classifier.

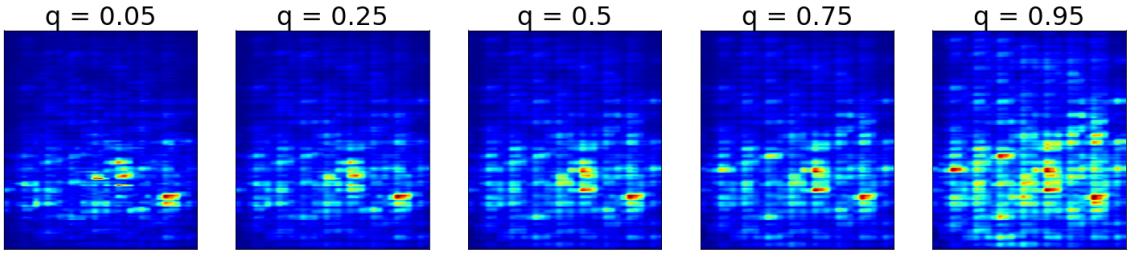


Figure 5.7. Quantiles computed over the 500 SMs for the bee in Figure 5.5 with the MobileNet.

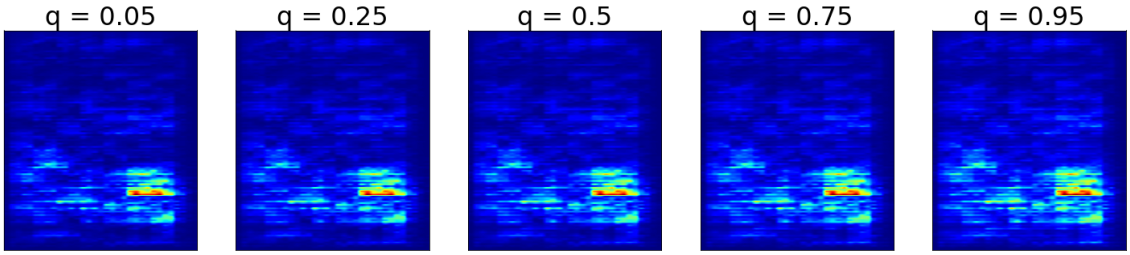


Figure 5.8. Quantiles computed over the 500 SMs for the bee in Figure 5.5 with the EfficientNet.

derstanding of how pixel relevance evolves throughout all 500 predictions, quantile statistics are computed for every pixel. Figures 5.6 to 5.8 depict 5 quantile SMs for each of the tested models. The greater the quantile, the more pixels are marked as relevant. So models at $q = 0.05$ are very strict while those at $q = 0.95$ are more permissive. In the end, the pixels that are marked as relevant throughout all the quantiles are what the model thinks of as important features.

When comparing the networks, it seems that only the Simple Embedder-Classifier succeeded in finding the truly relevant frequencies in the lower half of the spectrogram. The MobileNet and the EfficientNet appear to have spotted the half-cut frequency wave at around 1.5 kHz (which may be a marker for some bees in the Keogh dataset but suggests a Clever-Hans Effect bias), yet ignore most of the region below 1 kHz. This may also explain why their prediction distributions in Figure

5.5 have a higher standard deviation than the Simple Embedder-Classifier's: they did not base their judgement on solid evidence.

5.2.3 Pixel Flipping

Until now, the observations made by XAI are somewhat contradictory to the test set accuracies measured in the previous section. The top-performing models there such as Simple Classifier and MobileNet now seem to miss the important features in the spectrograms. Meanwhile, classifiers based on embeddings concentrate their attention on the true wingbeat specific frequencies. A few questions arise from this:

1. Are these explainable methods trustworthy, that is, are they really revealing what the model deems as relevant? How can this be checked?

2. What does it mean for an XAI method to be efficient? For instance, can SMs be compared to GBP maps?

Unfortunately, the array of tools to investigate such questions is currently not that wide. Good metrics for attention maps are still missing and there is an increasing need in ML for finding ones. The only available instrument is Pixel Flipping (PF) [4]. The idea is to iteratively replace pixels in an image in the relevance order proposed by an attention map and check how this changes the prediction score. Hypothetically, if the attention map did, indeed, highlight relevant pixels, then the prediction score should drop. The swifter the drop, the more relevant the pixels are. If not, then there might be a problem with the XAI method. For the models here, this means the following: if the maps are to be trusted, then well-performing networks on the test set such as MobileNet and EfficientNet base their judgment on non-generalizable features (noise or cropped signal edges).

Pixel Flipping is described in Algorithm 2, taken from [4] but readapted: here, the selected pixels are replaced by a random value drawn from a normal distribution of pixel values calculated on the Keogh training + validation set (Figure 5.9). Replacing the pixels with 0's or the mean value of the image did not work well, arguably because the new high-contrast pixels would lend the image a watermark (again the Clever Hans Effect mentioned in Section 3.1.3). The sequence of prediction scores is the normalized probabilities for the chosen class.

Figure 5.10 gives an example of what a spectrogram would look like after undergoing PF. The flipping will not only be done according to the relevance in the

Algorithm 2 *Pixel Flipping*

Input: Input image $x \in \mathbf{R}^{m \times k}$, relevance heatmap $h \in \mathbf{R}^{m \times k}$, trained model f and multivariate normal distribution $X \sim \mathcal{N}_{m \cdot k}(\mu, \Sigma)$

Output: Sequence of prediction scores s

$s \leftarrow [\ ]$
for p in $argsort(-R)$ **do**
    Select $\hat{x} \sim \mathcal{N}(\mu_p, \Sigma^2_{pp})$
    $x_p \leftarrow \hat{x}$
    $s.append(f(x))$
end for

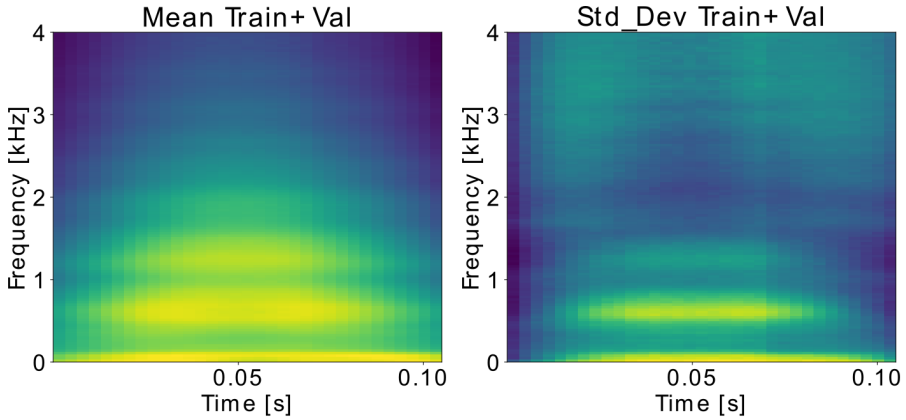


Figure 5.9. Left: mean pixel values over spectrograms in the Keogh training + validation set; Right: Standard deviations of every pixel value over the same set. Because the Keogh dataset is predominantly made up of mosquito signals, the mean spectrogram also resembles a typical one for mosquitoes.

SMs and GBP maps but also randomly. This offers a good baseline that the attention maps are expected to surpass. 25% of the pixels are already enough to outline a trend. Each PF iteration will replace 0.5% more pixels. You see that the sorting resulted from the attention maps will concentrate the PF in the bottom half of the spectrogram, where the characteristic frequencies are located, while the random PF will just erratically move throughout the whole image, even in the upper half predominated by noise. The results of applying all three PF strategies on the aedes spectrogram are outlined in Figure 5.11. Notice that the random flipping curve varies very little throughout flipping, which was to be expected because most randomly selected pixels are not deemed as relevant. On the other hand, the ones based on attention maps decrease fairly rapidly. This is a sign that the attention maps identified, indeed, relevant pixels that contributed to the prediction and these pixels, fortunately, coincide with the frequencies of interest.

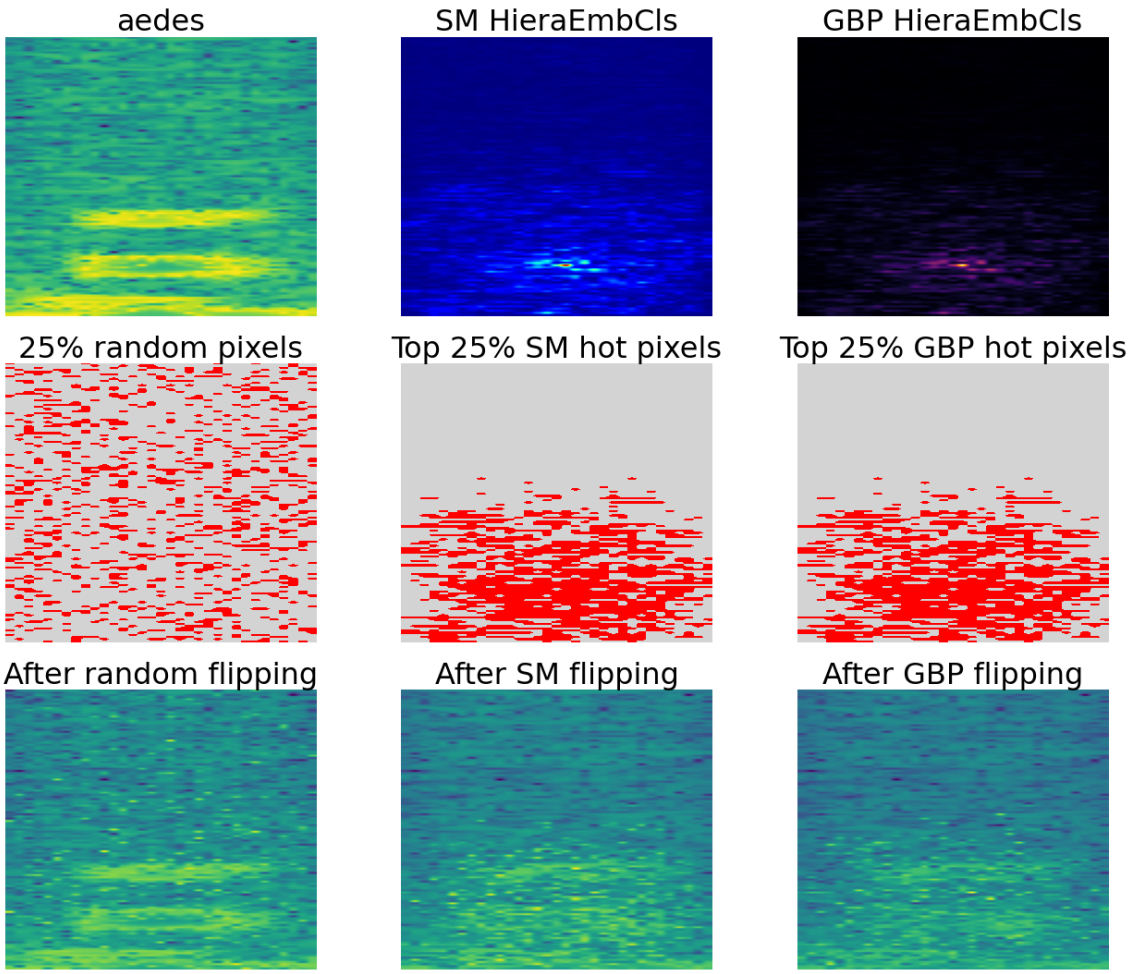


Figure 5.10. Visualization of the 3 PF strategies: random and according to an attention map (SM or GBP). *Top row*: spectrogram of an aedes mosquito from the Keogh test set along with its SM and GBP map, respectively. The maps were computed from the prediction made by the Hierarchical Embedder-Classifier. Incidentally, the prediction was correct. *Middle row*: the 25% of the pixels (marked in red) from the spectrogram above selected to be flipped iteratively in each PF strategy. By flipping a pixel, its value is replaced by a random value drawn from the normal distribution with mean and standard deviation computed over all images in the training + validation Keogh sets (Figure 5.9). *Bottom row*: resulting spectrograms after all selected pixels have been flipped.

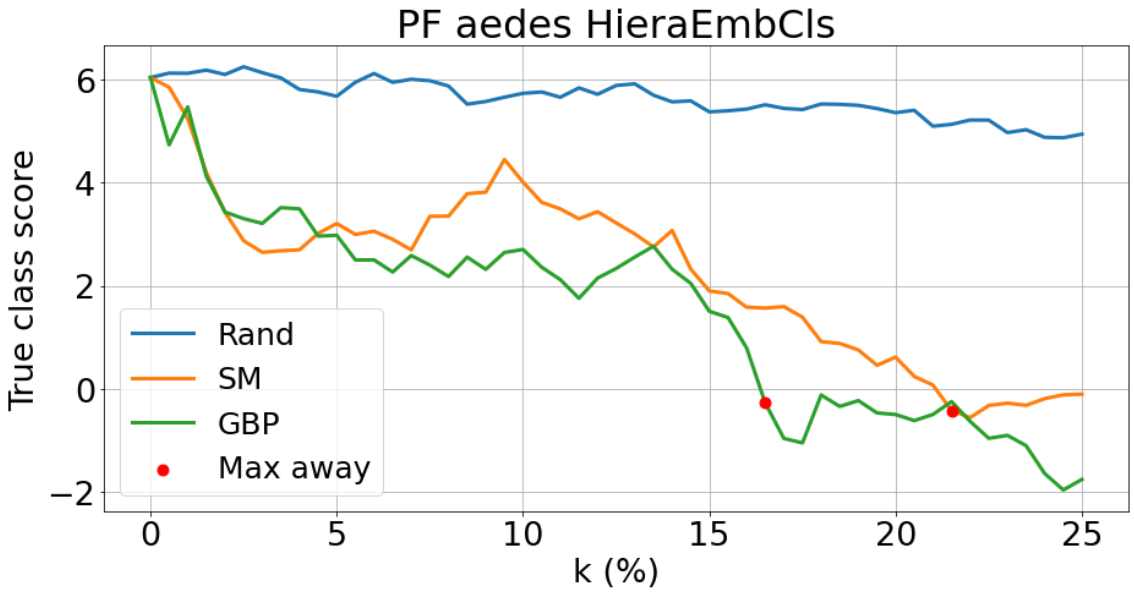


Figure 5.11. Evolution of the true prediction scores (non-normalized outputs before the Softmax layer) output by the Hierarchical Embedder-Classifier after each PF iteration. Each new iteration would replace 0.5% of the pixels until the limit of 25% is reached. The input spectrogram is the same as in Figure 5.10. The red dots on the green and orange curves mark the last iteration where the true label coincides with the predicted label (maximal output score).

To gather more evidence in favor of the attention maps, this whole process has also been applied in junction with the MC-Dropout on species subsets. Figures 6.3 and 6.4 in Appendix D show some representative results and draw a parallel between the Simple Embedder-Classifier and the MobileNet. For each sample, the models have made 100 MCD predictions. For each prediction, the corresponding SM and GBP map were computed. Out of the 100 maps each, three quantile maps were derived, as described in the previous section. In each iteration step, 0.5% more hot pixels are flipped as sorted by the quantile maps and the new prediction scores (unnormalized true class probabilities) are computed and averaged out. This is how the colored curves are created. For the dotted black curve, random flipping was used. The first thing to note is that in every case the PF curves w.r.t. quantile maps decrease with the number of flipped pixels. On the one hand, this indicates that the maps succeeded in highlighting the relevant pixels for both models. The descent is steeper for the Simple Embedder than for the MobileNet, which may suggest that the features found by the former are more characteristic for the species. On the other hand, if one were to trust the validity of the maps, this would mean that the MobileNet is, indeed, biased to modeling noise and other signal preprocessing artifacts rather than the true wingbeat patterns.

5.3 Adversarial Robustness

Last but not least, the models[7] will be tested for their adversarial robustness, which is a measure of how well they perform when their inputs are perturbed by injecting noise (also called 'adversarial attacks') [34] [33]. The perturbed images look so similar to the original ones that the human eye cannot tell the difference (Figure 5.12). It is, therefore, surprising that classifiers can sometimes get fooled by these noisy versions and end up misclassifying them.

The measure for adversarial robustness is the *robust accuracy*. This is nothing more than the accuracy on a test set that has been subject to attacks. Figure 5.12 reveals how the 2D-classifiers behave on the Keogh and Potatamis test sets after gradually increasing the amount of noise in the attacks (epsilon is a measure for how much the spectrograms should be modified). In both cases, the custom Embedder-Classifiers perform better than their counterparts. The MobileNet is even surpassed by a large margin in the Keogh test set. This only strengthens the observations made in the previous section where the Simple and Hierarchical Embedder-Classifiers proved to identify the right wingbeat frequencies, as opposed to MobileNet. Also, notice that the robust accuracies decrease much faster on the Potatamis test set. The reason for this may be the fine structures present in the spectrograms there, which can easily be blurred or destroyed with little noise. In contrast, the spectrograms from the Keogh dataset have more coarse structures, so adding a small amount of noise does not make that much of a difference.

[7] Only spectrogram models, but the method can be extended to PSD as well, just like the attention maps.

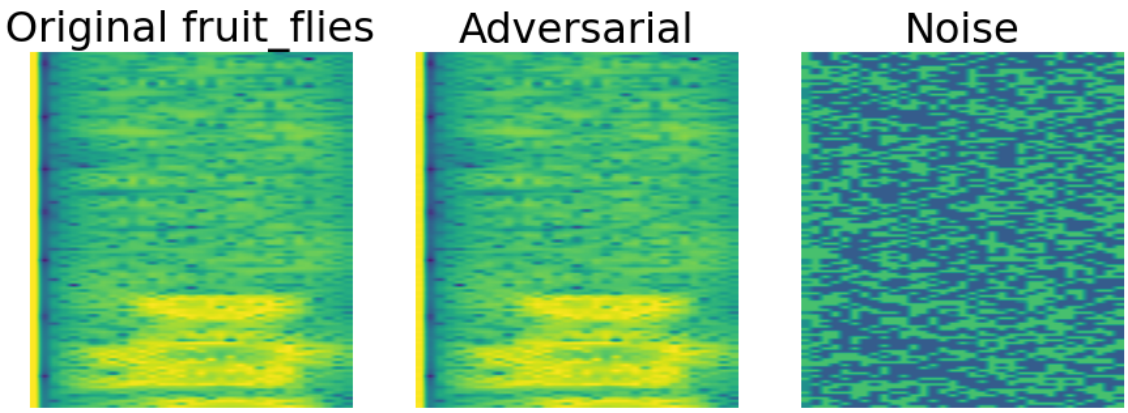
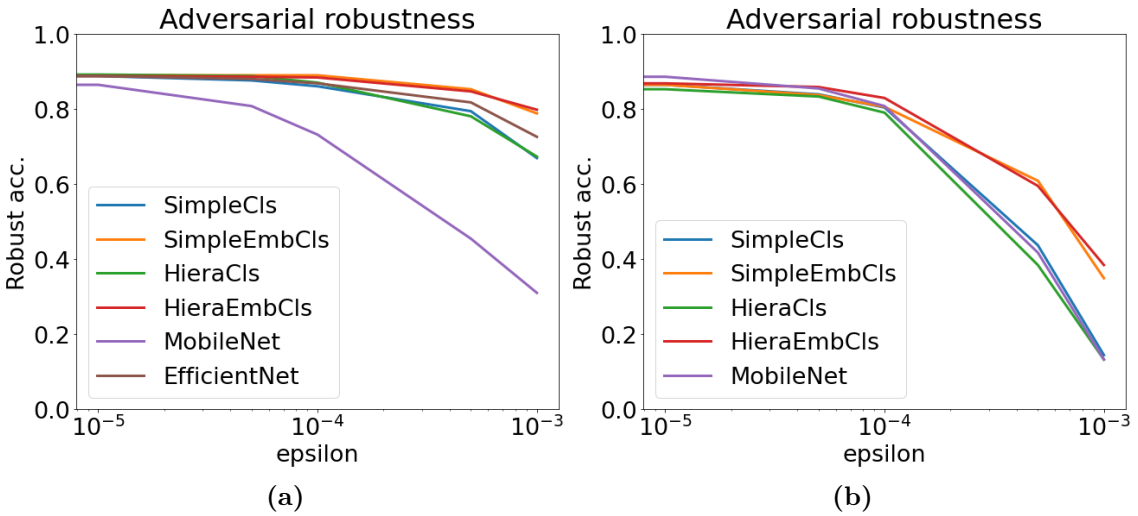


Figure 5.12. Example of a spectrogram after undergoing an adversarial attack. The image in the middle is the result of adding the noise on the right to the original image on the left. The noise is only made up of two pixel values: around $\pm 4 \cdot 10^{-3}$. The naked human eye cannot spot any difference between the original and the modified image, yet the Simple Classifier and the MobileNet failed to classify it correctly after adding the noise, although they had assigned it the right label before.



(a) **(b)**

Figure 5.13. Adversarial robustness for spectrogram models and different values of epsilon on 512 spectrograms from the Keogh **(a)** and Potatamis **(b)** test sets. The higher the epsilon value, the more noise is injected into the spectrograms.

Summary and Conclusions

The topic of this thesis was hierarchically classifying insect species based on optoacoustic signals derived from their wingbeats. The problem at hand was fairly intricate and posed a number of challenges even to state-of-the-art ML approaches.

As you could see, the quality of the wingbeat recording plays an important role. For example, it turned out that spectrograms are not necessarily better than PSD vectors when the original time signals are too short (like in the Keogh dataset). Secondly, overfitting is almost unavoidable with deep neural networks, so special care needed to be taken with adding efficient regularization techniques. Thirdly, class imbalance becomes a serious issue, especially when the data is not very abundant. In this respect, oversampling done with SMOTE proved to offer a certain degree of support. Fourthly, stress tests such as Pixel Flipping in attention maps generated by MC-Dropout and adversarial attacks indicated that established pre-trained networks from the literature may fail at consistently modeling characteristic wingbeat patterns, although they achieved the highest prediction scores on the test sets. On the other hand, custom architectures that integrate taxonomic information in the form of hierarchical embeddings proved to be more successful at identifying these features, with the second great advantage of running efficiently on small devices.

The software created here offers a template for further developments. Indefinite extensions and reformulations of the classification task are possible through the flexibility of the ML models, allowing, for instance, more species or taxonomic levels to be added (as exemplified in the Transfer Learning experiments). Other input formats can also be easily integrated, such as wavelets, MFCC or camera photos. The optimization pipeline makes further model fine-tuning straightforward as well.

Insects in Germany and worldwide are in dire need of help and their survival depends on swift measures taken to protect them. It seems that Deep Learning can offer good solutions for more detailed and automatic monitoring of insect abundance. Hopefully, this leads to more public awareness and proper actions to be enforced.

Appendix A

Computing the HCEs for the Keogh dataset

Given that the tree in Figure 3.26 has three levels including the root, its total height will be 2 and there are only three possible distances between its leaves:

1. The distance between two different species of the same genus will always be $\frac{1}{2}$ (i.e. $d(v_6, v_8) = d(v_3, v_4) = \frac{1}{2}$)
2. The distance between two species of two different genera will always be 1 (i.e. $d(v_1, v_2) = d(v_4, v_9) = \frac{2}{2} = 1$)
3. The distance from any species to itself will always be 0 (i.e. $d(v_8, v_8) = \frac{0}{2} = 0$)

Let $e_i \in \mathbb{R}^9$ be the unit vector with all entries 0, except at position i where it has a 1. A hierarchical embedding of the 9 classes would be computed as follows:

1. $\varphi(v_1) = e_1$ (**beetle**)
2. $\varphi(v_2) = e_2$ (**honey bee**)
3. $\varphi(v_3) = e_3$ (**house fly**)

3.1. $$\left.\begin{array}{l} \varphi(v_4)^T \cdot \varphi(v_1) \overset{!}{=} s(v_4, v_1) = 1 - 1 = 0 \\ \varphi(v_4)^T \cdot \varphi(v_2) \overset{!}{=} s(v_4, v_2) = 1 - 1 = 0 \\ \varphi(v_4)^T \cdot \varphi(v_3) \overset{!}{=} s(v_4, v_3) = 1 - \frac{1}{2} = \frac{1}{2} \end{array}\right\} \iff \left\{\begin{array}{l} \varphi(v_4)_1 \cdot 1 = 0 \\ \varphi(v_4)_2 \cdot 1 = 0 \\ \\ \varphi(v_4)_3 \cdot 1 = \frac{1}{2} \end{array}\right\} \implies$$

$$\implies \hat{\varphi}(v_4) = \left(0, 0, \frac{1}{2}\right)^T$$

$$\implies \varphi(v_4)_4 = \sqrt{1 - ||\hat{\varphi}(v_4)||^2} = \sqrt{1 - \frac{1}{4}} = \frac{\sqrt{3}}{2}$$

$$\implies \varphi(v_4) = \left(0, 0, \frac{1}{2}, \frac{\sqrt{3}}{2}, 0, ..., 0\right)^T \text{ (**fruit fly**)}$$

3.2.
$$\left.\begin{array}{l}\varphi(v_5)^T \cdot \varphi(v_1) \stackrel{!}{=} s(v_5, v_1) = 1 - 1 = 0 \\ \varphi(v_5)^T \cdot \varphi(v_2) \stackrel{!}{=} s(v_5, v_2) = 1 - 1 = 0 \\ \varphi(v_5)^T \cdot \varphi(v_3) \stackrel{!}{=} s(v_5, v_3) = 1 - \frac{1}{2} = \frac{1}{2} \\ \varphi(v_5)^T \cdot \varphi(v_4) \stackrel{!}{=} s(v_5, v_4) = 1 - \frac{1}{2} = \frac{1}{2}\end{array}\right\} \iff$$

$$\iff \left\{\begin{array}{l}\varphi(v_5)_1 \cdot 1 = 0 \\ \varphi(v_5)_2 \cdot 1 = 0 \\ \varphi(v_5)_3 \cdot 1 = \frac{1}{2} \\ \varphi(v_5)_3 \cdot \varphi(v_4)_3 + \varphi(v_5)_4 \cdot \varphi(v_4)_4 = \frac{1}{2} \cdot \frac{1}{2} + \varphi(v_5)_4 \cdot \frac{\sqrt{3}}{2} = \frac{1}{2}\end{array}\right\} \implies$$

$$\implies \hat{\varphi}(v_5) = \left(0, 0, \frac{1}{2}, \frac{\sqrt{3}}{6}\right)^T$$

$$\implies \varphi(v_5)_5 = \sqrt{1 - ||\hat{\varphi}(v_5)||^2} = \sqrt{1 - (\frac{1}{4} + \frac{1}{12})} = \sqrt{1 - \frac{1}{3}} = \frac{\sqrt{6}}{3}$$

$$\implies \varphi(v_5) = \left(0, 0, \frac{1}{2}, \frac{\sqrt{3}}{6}, \frac{\sqrt{6}}{3}, 0, ..., 0\right)^T \textbf{ (moth fly)}$$

4. $\varphi(v_6) = e_6$ (**aedes**)

4.1. $\varphi(v_7) = \left(0, ..., 0, \frac{1}{2}, \frac{\sqrt{3}}{2}, 0, 0\right)^T$ (**anopheles**, analogous to 3.1)

4.2. $\varphi(v_8) = \left(0, ..., 0, \frac{1}{2}, \frac{\sqrt{3}}{6}, \frac{\sqrt{6}}{3}, 0\right)^T$ (**quinx**, analogous to 3.2)

4.3. Since it is clear that the first 5 coordinates are going to be 0, this step only focuses on the dot products with respect to the mosquito species:

$$\left.\begin{array}{l}\varphi(v_9)^T \cdot \varphi(v_6) \stackrel{!}{=} s(v_9, v_6) = \frac{1}{2} \\ \varphi(v_9)^T \cdot \varphi(v_7) \stackrel{!}{=} s(v_9, v_7) = \frac{1}{2} \\ \varphi(v_9)^T \cdot \varphi(v_8) \stackrel{!}{=} s(v_9, v_8) = \frac{1}{2}\end{array}\right\} \iff$$

$$\iff \left\{\begin{array}{l}\varphi(v_9)_6 \cdot \varphi(v_6)_6 = \frac{1}{2} \\ \varphi(v_9)_6 \cdot \varphi(v_7)_6 + \varphi(v_9)_7 \cdot \varphi(v_7)_7 = \frac{1}{2} \\ \varphi(v_9)_6 \cdot \varphi(v_8)_6 + \varphi(v_9)_7 \cdot \varphi(v_8)_7 + \varphi(v_9)_8 \cdot \varphi(v_8)_8 = \frac{1}{2}\end{array}\right\} \implies$$

$$\implies \left\{ \begin{array}{l} \varphi(v_9)_6 \cdot 1 = \frac{1}{2} \Rightarrow \varphi(v_9)_6 = \frac{1}{2} \\ \frac{1}{2} \cdot \frac{1}{2} + \varphi(v_9)_7 \cdot \frac{\sqrt{3}}{2} = \frac{1}{2} \Rightarrow \varphi(v_9)_7 = \frac{\sqrt{3}}{6} \\ \frac{1}{2} \cdot \frac{1}{2} + \frac{\sqrt{3}}{6} \cdot \frac{\sqrt{3}}{6} + \varphi(v_9)_8 \cdot \frac{\sqrt{6}}{3} = \frac{1}{2} \Rightarrow \varphi(v_9)_8 = \frac{\sqrt{6}}{12} \end{array} \right\} \implies$$

$$\implies \hat{\varphi}(v_9) = \left(0, ..., 0, \frac{1}{2}, \frac{\sqrt{3}}{6}, \frac{\sqrt{6}}{12}\right)^T \in \mathbb{R}^8$$

$$\implies \varphi(v_9)_9 = \sqrt{1 - ||\hat{\varphi}(v_9)||^2} = \sqrt{1 - (\frac{1}{4} + \frac{1}{12} + \frac{1}{24})} = \sqrt{1 - \frac{3}{8}} = \frac{\sqrt{10}}{4}$$

$$\implies \varphi(v_5) = \left(0, ..., 0, \frac{1}{2}, \frac{\sqrt{3}}{6}, \frac{\sqrt{6}}{12}, \frac{\sqrt{10}}{4}\right)^T \text{ (\textbf{tarsalis})}$$

Appendix B

Keogh Test Set

PSD

Model	Format	GenAcc	SpecAcc	Latency (s/samp)	Size (KB)
SimpleCls	***Original***	0.969	0.9085	0.0763	597
	Unquantized	0.9655	0.9105	0.0036	570
	Dynamic	0.9662	0.9096	0.0033	171
	Full Integer	0.9665	0.9108	0.0031	179
	Float16	0.9665	0.9105	0.0036	303
SimpleEmbCls	***Original***	0.966	0.9082	0.0797	604
	Unquantized	0.9646	0.9133	0.0038	609
	Dynamic	0.9653	0.9146	0.0034	183
	Full Integer	0.9650	0.9136	0.0030	191
	Float16	0.9646	0.9133	0.0037	325
HieraCls	***Original***	0.9704	0.9096	0.0806	617
	Unquantized	0.9681	0.9143	0.0037	586
	Dynamic	0.9690	0.9143	0.0034	175
	Full Integer	0.9671	0.9136	0.0031	184
	Float16	0.9681	0.9143	0.0037	312
HieraEmbCls	***Original***	0.9704	0.9096	0.0828	611
	Unquantized	0.9684	0.9136	0.0037	646
	Dynamic	0.9687	0.9130	0.0034	193
	Full Integer	0.9618	0.9043	0.0031	202
	Float16	0.9684	0.9136	0.0037	345

Table 6.1. Accuracy scores of the 4 PSD models on the Keogh test set along with the inference times and sizes. Each model has an original version tested on the Google CPU - *Original* row - and 4 *tflite* versions, one without weight quantization and three quantized.

Spectro

Model	Format	GenAcc	SpecAcc	Latency (s/samp)	Size (KB)
SimpleCls	***Original***	0.9624	0.8964	0.0872	1591
	Unquantized	0.9609	0.9027	0.0237	1552
	Dynamic	0.9606	0.9014	0.0220	408
	Full Integer	0.9603	0.8996	0.0202	414
	Float16	0.9609	0.9024	0.0246	785
SimpleEmbCls	***Original***	0.9638	0.8919	0.0900	1598
	Unquantized	0.9621	0.8980	0.0246	1563
	Dynamic	0.9628	0.8986	0.0223	412
	Full Integer	0.9618	0.8949	0.0201	418
	Float16	0.9625	0.8983	0.0237	792
HieraCls	***Original***	0.9668	0.8939	0.0899	1599
	Unquantized	0.9656	0.9036	0.0245	1556
	Dynamic	0.9656	0.9036	0.0218	409
	Full Integer	0.9656	0.9027	0.0200	416
	Float16	0.9659	0.9033	0.0246	787
HieraEmbCls	***Original***	0.9655	0.8997	0.0936	1607
	Unquantized	0.9631	0.9033	0.0236	1573
	Dynamic	0.9631	0.9036	0.0220	416
	Full Integer	0.9631	0.9055	0.0199	422
	Float16	0.9625	0.9039	0.0237	798

Table 6.2. Accuracy scores of the 4 spectrogram models on the Keogh test set along with the inference times and sizes. Each model has an original version tested on the Google CPU - *Original* row - and 4 *tflite* versions, one without weight quantization and three quantized.

Potatamis Test Set

PSD

Model	Format	GenAcc	SpecAcc	Latency (s/samp)	Size (KB)
SimpleCls	***Original***	0.9133	0.8694	0.0934	584
	Unquantized	0.9133	0.8694	0.0033	558
	Dynamic	0.9134	0.8693	0.0033	168
	Full Integer	0.9115	0.8672	0.0033	176
	Float16	0.9132	0.8693	0.0033	297
SimpleEmbCls	***Original***	0.9008	0.8546	0.0935	593
	Unquantized	0.9008	0.8546	0.0033	584
	Dynamic	0.9006	0.8539	0.0033	176
	Full Integer	0.8997	0.8532	0.0033	185
	Float16	0.9008	0.8546	0.0033	313
HieraCls	***Original***	0.9065	0.8647	0.0915	600
	Unquantized	0.9065	0.8647	0.0033	570
	Dynamic	0.9060	0.8650	0.0033	171
	Full Integer	0.9053	0.8643	0.0033	180
	Float16	0.9064	0.8649	0.0033	304
HieraEmbCls	***Original***	0.9029	0.8556	0.0938	600
	Unquantized	0.9029	0.8556	0.0034	609
	Dynamic	0.9032	0.8556	0.0034	183
	Full Integer	0.9018	0.8542	0.0033	192
	Float16	0.9029	0.8556	0.0034	326

Table 6.3. Accuracy scores of the 4 PSD models on the Potatamis test set along with the inference times and sizes. Each model has an original version tested on the Google CPU - *Original* row - and 4 *tflite* versions, one without weight quantization and three quantized.

Spectro

Model	Format	GenAcc	SpecAcc	Latency (s/samp)	Size (KB)
SimpleCls	***Original***	0.9170	0.8699	0.1230	1588
	Unquantized	0.9172	0.8701	0.0396	1549
	Dynamic	0.9168	0.8695	0.0368	407
	Full Integer	0.9165	0.8676	0.0411	413
	Float16	0.9170	0.8697	0.0395	783
SimpleEmbCls	***Original***	0.9145	0.8619	0.1081	1595
	Unquantized	0.9145	0.8619	0.0393	1557
	Dynamic	0.9141	0.8616	0.0363	410
	Full Integer	0.9087	0.8548	0.0407	417
	Float16	0.9146	0.8620	0.0392	789
HieraCls	***Original***	0.9140	0.8669	0.1100	1595
	Unquantized	0.9140	0.8669	0.0392	1552
	Dynamic	0.9139	0.8666	0.0369	410
	Full Integer	0.9131	0.8646	0.0404	415
	Float16	0.9141	0.8667	0.0382	785
HieraEmbCls	***Original***	0.9213	0.8640	0.1101	1602
	Unquantized	0.9210	0.8640	0.0384	1564
	Dynamic	0.9211	0.8637	0.0364	413
	Full Integer	0.9207	0.8641	0.0400	420
	Float16	0.9212	0.8638	0.0393	794

Table 6.4. Accuracy scores of the 4 spectrogram models on the Potatamis test set along with the inference times and sizes. Each model has an original version tested on the Google CPU - *Original* row - and 4 *tflite* versions, one without weight quantization and three quantized.

Appendix C

Attention Maps

One spectrogram from four species in the Keogh test set has been selected. All four custom 2D models along with MobileNet and EfficientNet have made predictions on the given spectrograms and for each prediction the SM (Figure 6.1) and the GBP map (Figure 6.2) were computed. The predicted label is written above the maps. For these illustrations, only examples correctly classified by all models are shown. The same normalization between 0 and 1 was applied for both maps; the different color bars are only meant to visually distinguish between the two attention methods.

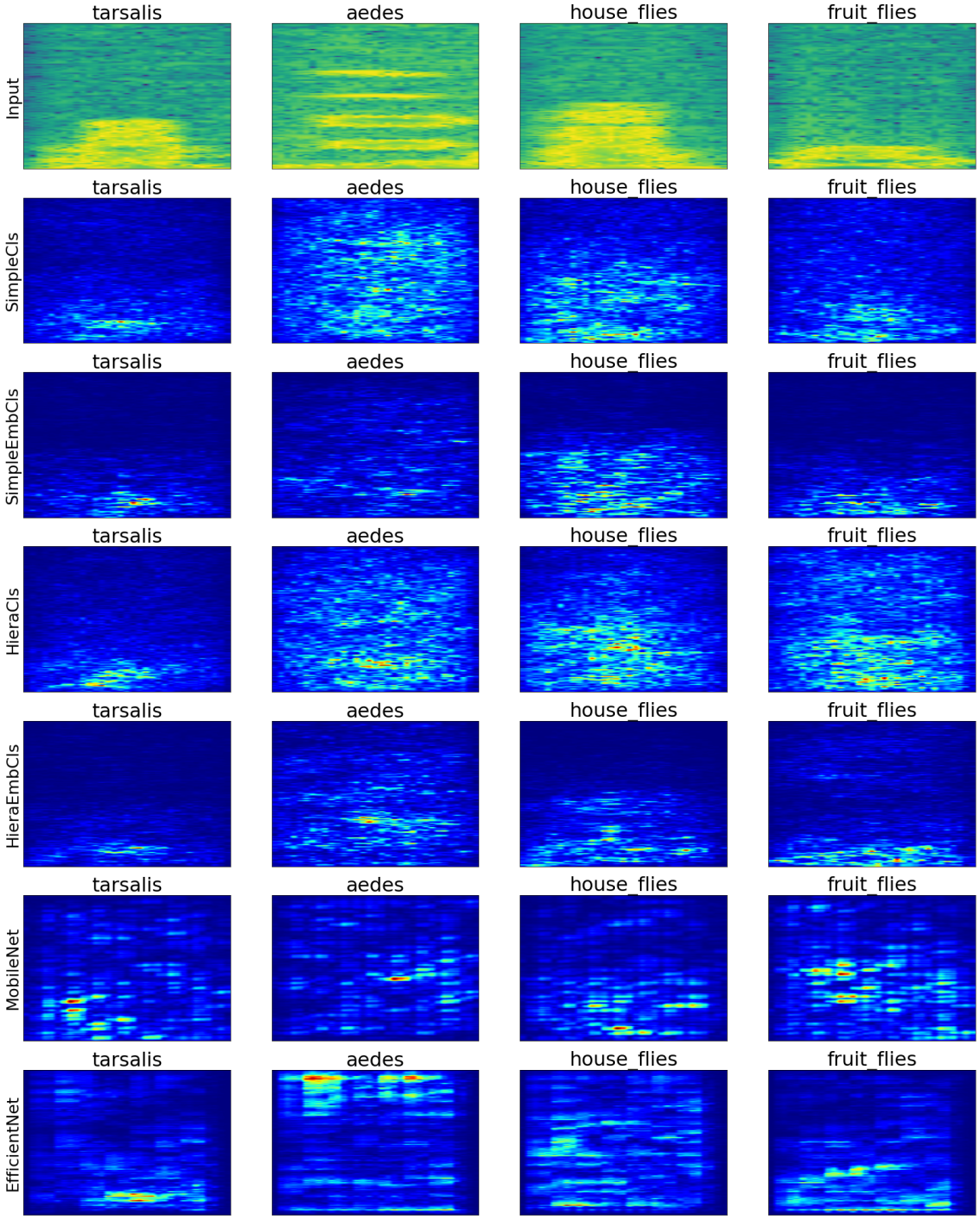


Figure 6.1. SMs for all spectrogram models computed w.r.t. 4 insect species from the Keogh test set. The maps are normalised between 0 (dark blue) and 1 (red). In the first row, the true labels are written above the samples, while in the subsequent rows the classes predicted by the corresponding models are specified.

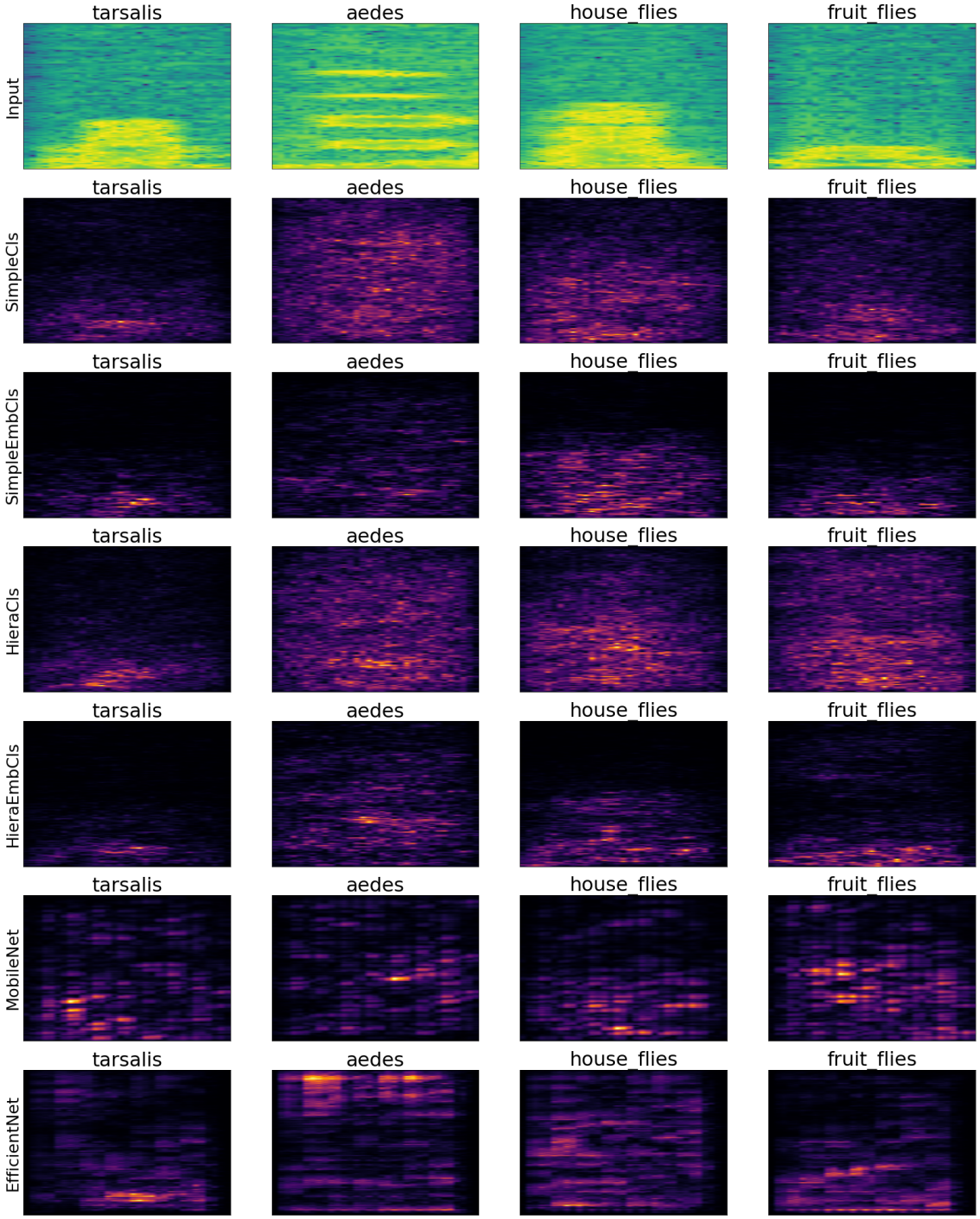


Figure 6.2. GBP maps for all spectrogram models computed w.r.t. 4 insect species from the Keogh test set. The maps are normalised between 0 (black) and 1 (yellow). In the first row, the true labels are written above the samples, while in the subsequent rows the classes predicted by the corresponding models are specified.

Appendix D

Pixel Flipping w.r.t. MC-Dropout quantile-maps

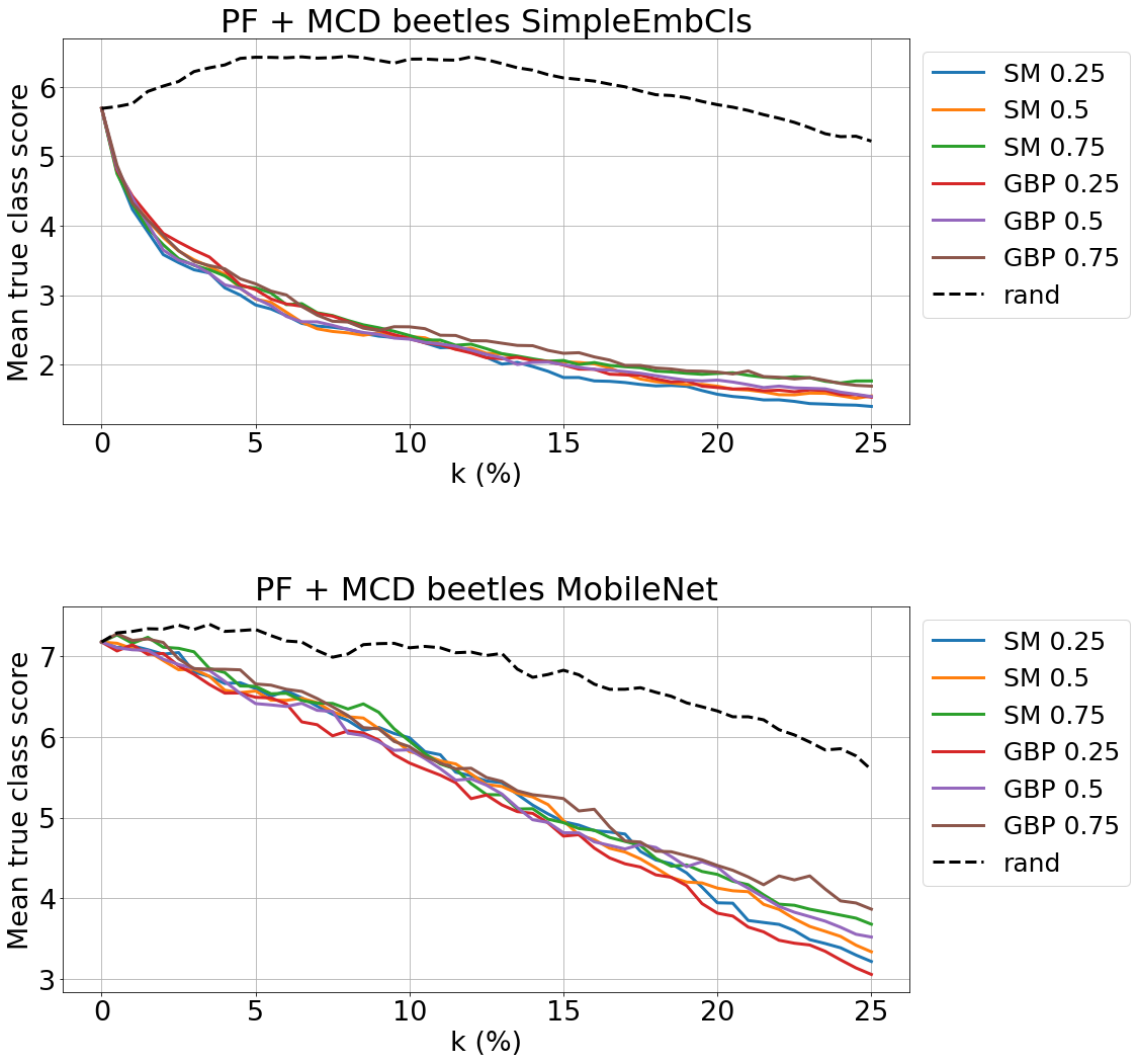


Figure 6.3. Comparison between Simple Embedder-Classifier and MobileNet on the beetle spectrograms from the Keogh test set. For every sample, 100 MCD predictions have been made. For each prediction, an SM and a GBP map were computed, from which 3 quantile maps were derived. At every iteration, 0.5% more hot pixels marked in the quantile maps are flipped and the new prediction scores are calculated (unnormalized true class outputs before Softmax). The dotted curves stand for random flipping.

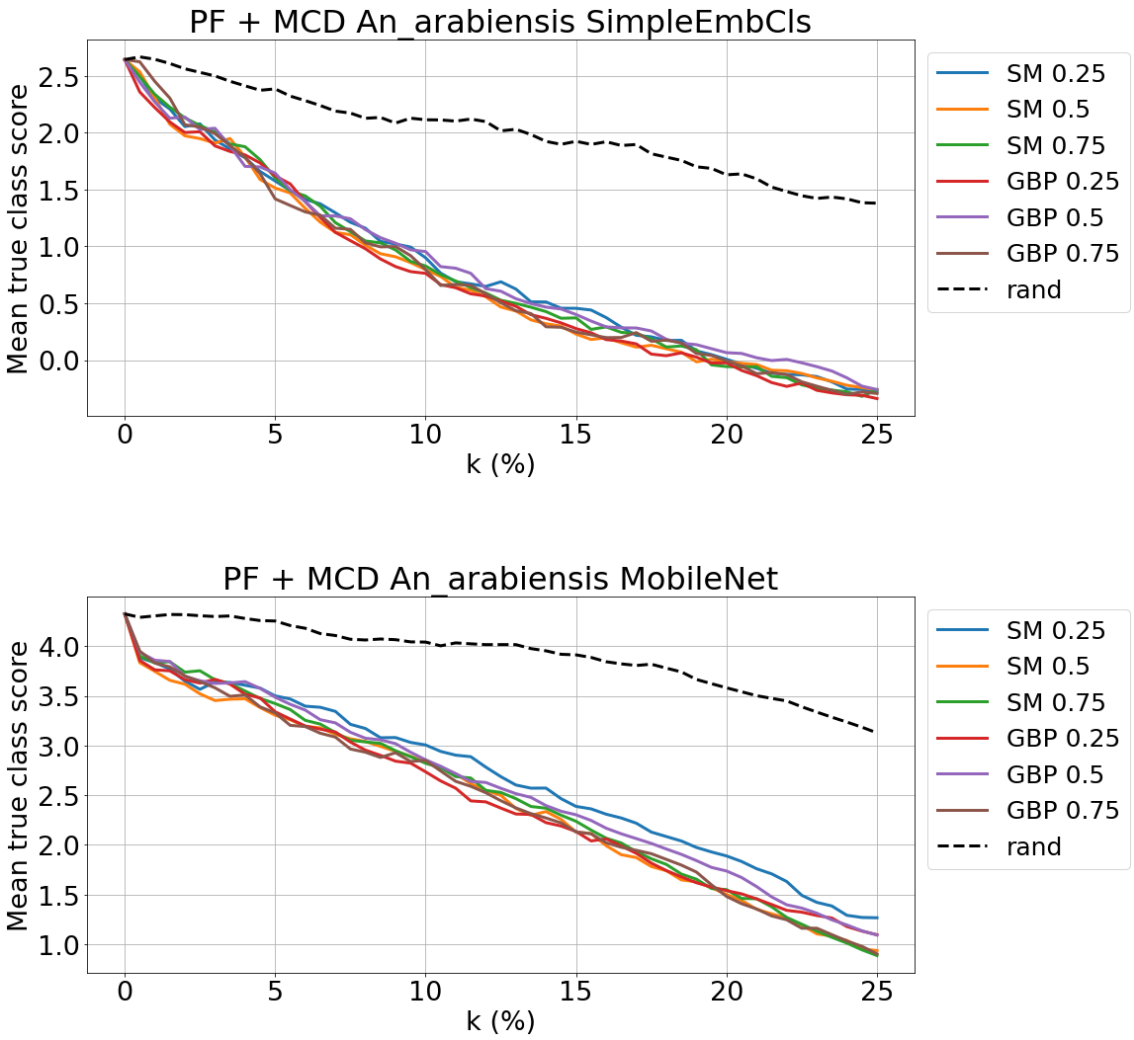


Figure 6.4. Comparison between Simple Embedder-Classifier and MobileNet on 50 arabiensis spectrograms from the Potatamis test set. For every sample, 100 MCD predictions have been made. For each prediction, an SM and a GBP map were computed, from which 3 quantile maps were derived. At every iteration, 0.5% more hot pixels marked in the quantile maps are flipped and the new prediction scores are calculated (unnormalized true class outputs before Softmax). The dotted curves stand for random flipping.

Appendix E

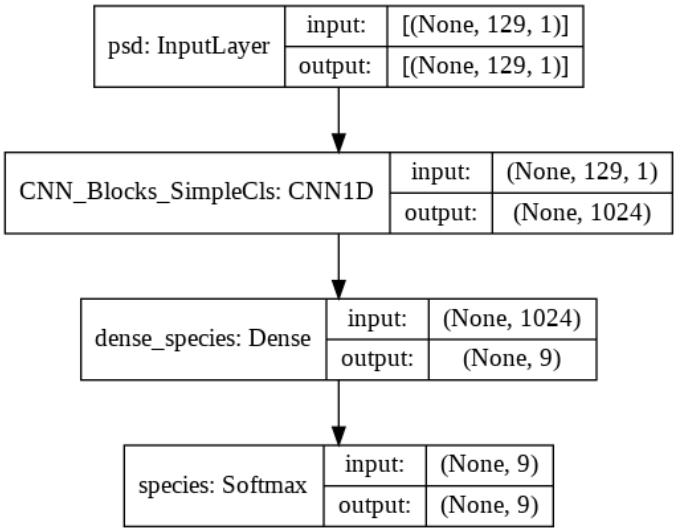


Figure 6.5. Simple Classifier: only predicts one taxonomic level; in this case, the species. Superior levels such as genus can be inferred from the predicted species.

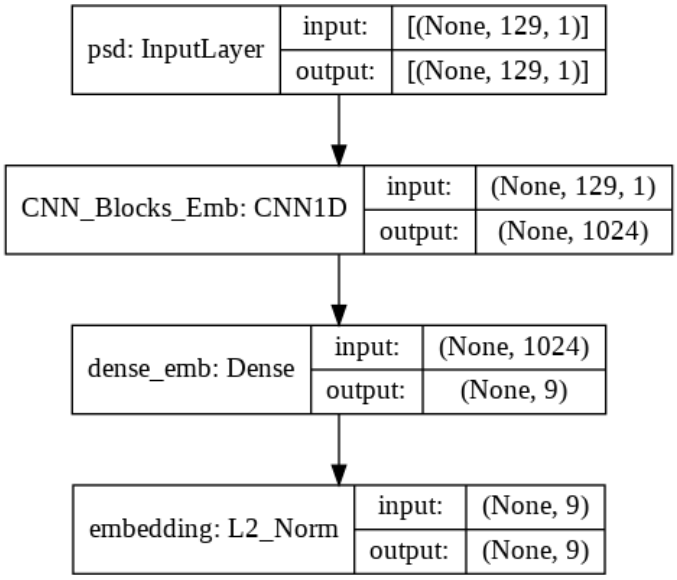


Figure 6.6. Embedder: predicts the HCE for a specific taxonomic level. The true class label can be inferred from the predicted embedding through nearest neighbor search.

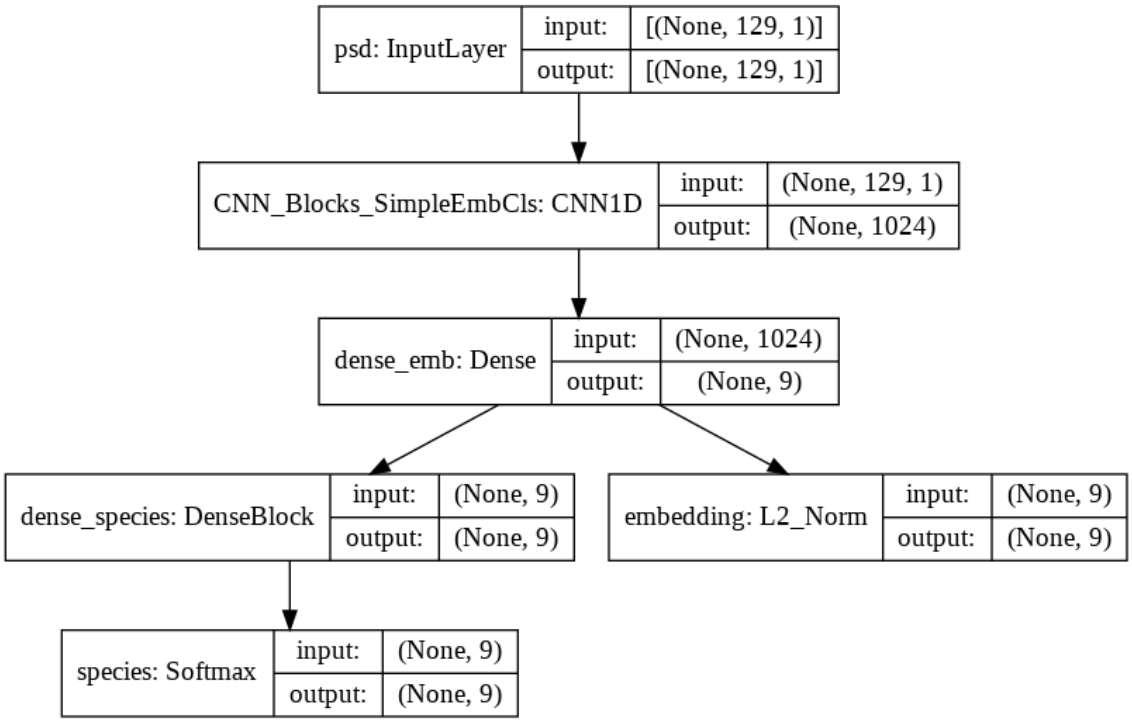


Figure 6.7. Simple Embedder-Classifier: predicts both the HCE and the class probability for a taxonomic level.

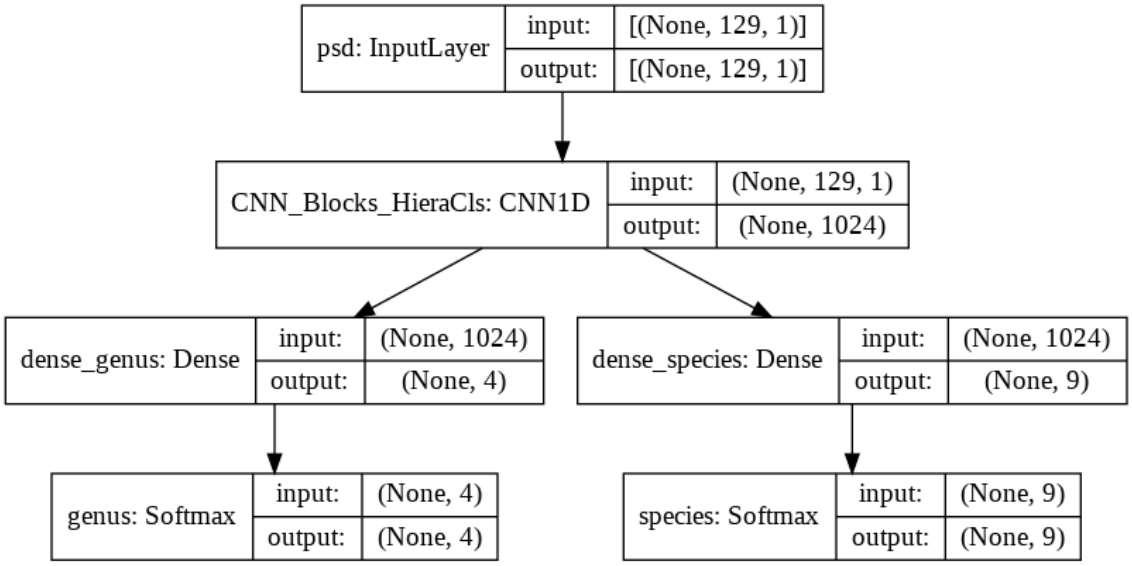


Figure 6.8. Hierarchical Classifier: predicts probabilities for more than just one taxonomic level. The branches can be extended to as many levels as desired.

Figure 6.9. Hierarchical Embedder-Classifier: predicts class probabilities for multiple taxonomic levels and also the HCE for one of those levels.

Table of Symbols

ML	Machine Learning
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
$STFT$	Short Time Fourier Transform
PSD	Power Spectral Density
$SMOTE$	Synthetic Minority Oversampling Technique
MSE	Mean Squared Error
$EMSGE$	Expected Mean Squared Generalization Error
HCE	Hierarchy-based Class Embedding
$t-SNE$	t-Distributed Stochastic Neighbor Embedding
$w.r.t.$	with respect to
$MFCC$	Mel Frequency Cepstrum Coefficients
SM	Saliency Map
GBP	Guided Backpropagation
MCD	Monte-Carlo Dropout
PF	Pixel Flipping

Bibliography

[1] B. Barz and J. Denzler. "Hierarchy-based Image Embeddings for Semantic Image Retrieval". In: *IEEE Winter Conference on Applications of Computer Vision (WACV)* (2019).

[2] *Beim Insektensommer zählen wir, was zählt.* 2020. URL: https://www.nabu.de/tiere-und-pflanzen/aktionen-und-projekte/insektensommer/index.html?ref=nav.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science + Business Media, 2006. ISBN: 978-0387-31073-2.

[4] K. Bykov et al. "How Much Can I Trust You? - Quantifying Uncertainties in Explaining Neural Networks". In: (2020). URL: https://arxiv.org/abs/2006.09000.

[5] *Cepstrum Analysis*. URL: https://de.mathworks.com/help/signal/ug/cepstrum-analysis.html.

[6] N. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research 16* (2002).

[7] K. Coakley and P. Hale. "Alignment of Noisy Signals". In: *IEEE Transactions on Instrumentation and Measurement* (2001). URL: https://www.researchgate.net/publication/3089887_Alignment_of_noisy_signals.

[8] A. D'Amour, K. Heller, and D. Moldovan. "Underspecification Presents Challenges for Credibility in Modern Machine Learning". In: (2020). URL: https://arxiv.org/abs/2011.03395.

[9] *Diversität von Insekten in Naturschutz-Arealen (DINA) Verbundforschungsvorhaben zum Insektenschwund*. URL: https://www.dina-insektenforschung.de/.

[10] Y. Gal and Z. Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: (2016). URL: https://arxiv.org/abs/1506.02142.

[11] A. Geron. *Hands-on Machine Learning with Scikit-Learn, Keras and Tensorflow, 2nd Edition*. O'Reilly, 2019. ISBN: 9781492032649.

[12] C. A. Hallmann et al. *More than 75 percent decline over 27 years in total flying insect biomass in protected areas*. 2017. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0185809.

[13] K. Hao. *AI has cracked a key mathematical puzzle for understanding our world*. 2020. URL: https://www.technologyreview.com/2020/10/30/1011435/ai-fourier-neural-network-cracks-navier-stokes-and-partial-differential-equations/.

[14] K. Hao. *sklearn.manifold.TSNE*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html.

[15] A. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: (2017). URL: https://arxiv.org/abs/1704.04861.

[16] G. James et al. *An Introduction to Statistical Learning with Applications in R*. Springer New York Heidelberg, 2017. ISBN: 978-1461471370.

[17] K. Jamieson and A. Talwalkar. "Non-stochastic best arm identification and hyperparameter optimization". In: *International Conference on Artificial Intelligence and Statistics (AISTATS)* (2015).

[18] L. R. Jospin et al. "Hands-on Bayesian Neural Networks - a Tutorial for Deep Learning Users". In: (2020). URL: https://arxiv.org/abs/2007.06823.

[19] N. Kehtarnavaz and N. Kim. *Digital Signal Processing System-Level Design Using LabVIEW*. Newnes, 2005. ISBN: 0-7506-7914-X.

[20] E. Keogh et al. "Applying Machine Learning and Audio Analysis Techniques to Insect Recognition in Intelligent Traps". In: *12th International Conference on Machine Learning and Applications (ICMLA)* (2013).

[21] E. Keogh et al. "Towards Automatic Classification on Flying Insects using Inexpensive Sensors". In: *Journal of Insect Behavior* (2014).

[22] *KI-Leuchttürme für Umwelt, Klima, Natur und Ressourcen*. URL: https://www.z-u-g.org/aufgaben/ki-leuchttuerme/projektuebersicht-fl2/kinsekt/.

[23] B. Krawczyk. "Learning from imbalanced data: open challenges and future directions". In: (2016). URL: https://link.springer.com/article/10.1007/s13748-016-0094-0.

[24] A. Krizhevsky, I. Sutskever, and G.E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *ILSVRC-2012* (2012).

[25] L. Li et al. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research* (2018). URL: https://arxiv.org/abs/1603.06560.

[26] T. Lin et al. "Focal Loss for Dense Object Detection". In: (2018). URL: https://arxiv.org/abs/1708.02002.

[27] L. van der Maaten. "Learning a Parametric Embedding by Preserving Local Structure". In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics* (2009).

[28] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN: 0521865719.

[29] M. P. Norton and D. G. Karczub. *Fundamentals of Noise and Vibration Analysis for Engineers*. Cambridge University Press, 2003. ISBN: 0-521-49913-5.

[30] *Post-training quantization*. URL: https://www.tensorflow.org/lite/performance/post_training_quantization.

[31] I. Potatamis, M. Geismar, and E. Fanioudakis. "Mosquito wingbeat analysis and classification using deep learning". In: *26th European Signal Processing Conference* (2018).

[32] I. Potatamis and I. Rigakis. "Large Aperture Optoelectronic Devices to Record and Time-stamp Insects' Wingbeats". In: (2016). URL: https://www.semanticscholar.org/paper/Large-Aperture-Optoelectronic-Devices-to-Record-and-Potamitis-Rigakis/9958d3acd58488f8abe478d0fb568cf817313c90.

[33] Jonas Rauber, Wieland Brendel, and Matthias Bethge. "Foolbox: A Python toolbox to benchmark the robustness of machine learning models". In: *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*. 2017. URL: http://arxiv.org/abs/1707.04131.

[34] Jonas Rauber et al. "Foolbox Native: Fast adversarial attacks to benchmark the robustness of machine learning models in PyTorch, TensorFlow, and JAX". In: *Journal of Open Source Software* 5.53 (2020), p. 2607. DOI: 10.21105/joss.02607. URL: https://doi.org/10.21105/joss.02607.

[35] J. Redmon and A. Farhadi. "YOLO9000: Better, Faster, Stronger". In: (2016). URL: https://arxiv.org/abs/1612.08242.

[36] K. Simonyan, A. Vedaldi, and A. Zisserman. "Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps". In: (2014). URL: https://arxiv.org/abs/1312.6034.

[37] L. N. Smith. "Cyclical Learning Rates for Training Neural Networks". In: (2017). URL: https://arxiv.org/abs/1506.01186.

[38] K. Sohn, X. Yan, and H. Lee. "Learning Structured Output Representation using Deep Conditional Generative Models". In: (2015). URL: https://papers.nips.cc/paper/2015/hash/8d55a249e6baa5c06772297520da2051-Abstract.html.

[39] O. Solomon. "PSD Computations Using Welch's Method". In: *Sandia Report* (1991).

[40] J. T. Springenberg et al. "Striving for Simplicity: The All Convolutional Net". In: (2015). URL: https://arxiv.org/abs/1412.6806.

[41] N. Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[42] M. Tan and Q. V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: (2020). URL: https://arxiv.org/abs/1905.11946.

[43] *TensorFlow Lite inference*. URL: https://www.tensorflow.org/lite/guide/inference.

[44] R. Wittig and M. Niekisch. *Biodiversität: Grundlagen, Gefährdung, Schutz*. Springer Berlin Heidelberg, 2014. ISBN: 9783642546945.

[45] C. R. Woese, O. Kandler, and M. L. Wheelis. "Towards a natural system of organisms: proposal for the domains Archaea, Bacteria, and Eucarya." In: *Proceedings of the National Academy of Sciences* 87.12 (1990), pp. 4576–4579. ISSN: 0027-8424. DOI: 10.1073/pnas.87.12.4576. eprint: https://www.pnas.org/content/87/12/4576.full.pdf. URL: https://www.pnas.org/content/87/12/4576.