Zurich University of Applied Sciences

Department Life Sciences and Facility Management

Institute of Natural Resource Sciences

TERM PAPER 1

Accessibility of Ecoacoustics: A Deep Learning Approach to Insect Sound Classification

---

JULY 16, 2024

Author:
Julian Kraft[1]

Tutor:
Dr. Matthias Nyfeler[2]

Affiliations:
[1]Institute of Natural Resource Sciences
[2]Institute of Computational Life Sciences

Imprint

Project type:	Term Paper 1
Title:	Accessibility of Ecoacoustics: A Deep Learning Approach to Insect Sound Classification
Date:	July 16, 2024
Keywords:	ecoacoustics, bioacoustics, classification, deeplearning, machinelearning, signalprocessing, audioanalysis
Copyright:	Zurich University of Applied Sciences

Author:	Julian Kraft[1] (kraftjul@students.zhaw.ch)
Tutor:	Dr. Matthias Nyfeler[2] (nife@zhaw.ch)
Affiliations:	[1]Institute of Natural Resource Sciences [2]Institute of Computational Life Sciences

Abstract

Biodiversity is rapidly declining worldwide. The first step towards understanding the causes and taking measures to counteract this trend is the development of methods and tools to quantify and monitor it. The biodiversity of insects is highly relevant due to their crucial role in ecosystems. Insects contribute to processes such as pollination, decomposition, and serving as a food source for other species. While many methods exist to monitor insects, ecoacoustics provides a non-invasive paradigm to measure species via acoustic signals. Recently, data-driven approaches, particularly deep learning-based methods, have proven effective in detecting insect species from acoustic in situ field measurements. These advancements offer promising new ways to understand, monitor, and protect insect populations. This study aims to reproduce the results of a previous research on insect sound classification using deep learning and to assess the accessibility of this technology. Using the InsectSet32 dataset, which contains 335 recordings of 32 insect species, a convolutional neural network (CNN) with residual blocks (ResNet) was developed. The model was implemented and trained using the PyTorch framework and optimized via hyperparameter tuning. The best-performing model achieved an accuracy of 0.706 on the validation set and 0.649 on the test set, closely aligning with the original study's findings. This work demonstrates that, with appropriate knowledge and relatively affordable hardware, such as a regular gaming computer equipped with a graphics processing unit (GPU), non-experts can effectively utilize deep learning models for ecoacoustic applications. The study underscores the potential of combining ecoacoustics with artificial intelligence to advance biodiversity monitoring, providing a non-invasive and efficient method for large-scale environmental assessments.

Contents

Code, data, logs and LaTeX source are to be found on GitHub:
https://github.com/juliankraft/TermPaper1-Ecoacoustics

List of Figures

List of Tables

1. Introduction

The loss of biodiversity is one of the most urgent issues caused by climate and land use change (Cardinale et al., 2012). The term biodiversity is a contraction of biological diversity, which refers to the variety and variability of life forms on Earth. In recent years, a massive decline in biodiversity has been observed, which is mainly due to human activities. The loss of biodiversity is a major concern because it can have a significant impact on the ecosystem and the services it provides (Brondízio et al., 2019). In order to quantify biodiversity and monitor its changes, it is essential to have a reliable and efficient method for quantifying it. Many traditional methods for the quantification of biodiversity are time-consuming and expensive, and they are not suitable for large-scale monitoring. However, there are modern approaches which have the potential to monitor biodiversity fast and efficiently.

Bioacoustics, the study of sound production, dispersion and reception in animals, is a field of research that focusses on the side of an individual species, group or community and their interaction with each other or their environment. It has provided pivotal insights into the behavior and communication of animals. Ecoacoustics, a field of research very close to bioacoustics, has a slightly different focus. it is the study of the soundscape of an ecosystem – the sounds produced by all living organisms in an ecosystem. Ecoacoustics has a broad focus and is concerned with the composition of the soundscape, the temporal and spatial patterns of sound production, and the relationship between the soundscape and the environment. The study of biodiversity using bioacoustics is one of its promising applications, yet there are still major challenges ahead (Scarpelli et al., 2021). Among different approaches, the passive acoustic monitoring (PAM) is a promising and non-invasive approach to monitor biodiversity via sound recordings. The PAM paradigm is currently intensely studied and advanced, and recently also paired with modern artificial intelligence (AI) methods (Deng, 2023). Before there can be a holistic approach to the monitoring of biodiversity using ecoacoustics, some smaller steps have to be taken.

One of these steps towards the large-scale monitoting of biodiversity via PAM is the classification of the sounds of individual species or taxonomic groups. So far, the taxonomic coverage includes birds, cetaceans and other marine mammals, bats, terrestrial mammals, anurans, insects and fish (Stowell, 2022, p. 4). Birds are the most studied group of animals in the field of bioacoustics, with a commonly known and distributed application called the "BirdNET" (Kahl et al., 2021).

Recently, the applicability of a deep learning approach to classify insects was demonstrated based on an open-access dataset of a subset of insect voices (Faiss, 2022). The subset contains members of the two groups of insects: *Orthoptera* and *Cicadidae*. *Orthoptera* is an order of insects that includes grasshoppers, crickets, and katydids. (“Orthoptera”, 2008) *Cicadidae*, a members of the superfamily Cicadoidea Westwood are four-winged insects with sucking mouthparts that possess three ocelli and a rostrum that arises from the base of the head (Sanborn, 2008).

The focus of this study is to reproduce the results of this paper and, akin to their approach,

to create a model that can classify a subset of insects using their voices. More specifically, the goal is to demonstrate that this technology is accessible for everyone with the basic methodological knowledge and a regular gaming computer with a graphic processing unit (GPU). For this purpose, a model was constructed and trained using the same dataset as the original paper. A hyperparameter tuning was performed to find the best configuration for the model. The results of the hyperparameter tuning were evaluated and discussed. The best performing model was tested and evaluated for its performance and accuracy. The results were discussed and compared to the results of the original paper.

2. Methods

2.1. Dataset

In this study, a preexisting dataset was used – the InsectSet32 dataset (Faiss, 2022). The dataset includes 335 recordings of 32 insect species, totaling 57 minutes. About half of the recordings (147) feature nine *Orthoptera* species from a dataset originally compiled by Baudewijn Odé (unpublished). The remaining 188 recordings, from 23 *Cicadidae* species, were selected from the Global Cicada Sound Collection on Bioacoustica (Baker et al., 2015a), including recordings published in (Baker et al., 2015b; Popple, 2017). Speech annotations at the beginning of many recordings led to using the last ten seconds of audio. Files with strong noise or multiple species were removed. The number of files per species ranges from 4 to 22, with durations from 40 seconds to almost 9 minutes. All files, originally with a sampling rate of at least 44.1 kHz, were resampled to 44.1 kHz mono WAV for consistency. The dataset was already split into training, validation and test sets. There are two .csv files containing the labels and the filenames of the recordings. To demonstrate the difference and uniqueness of the insect voices, some examples are shown as Mel spectrograms in Figure 1. A detailed overview of the dataset and the data distribution is provided in Table 2.

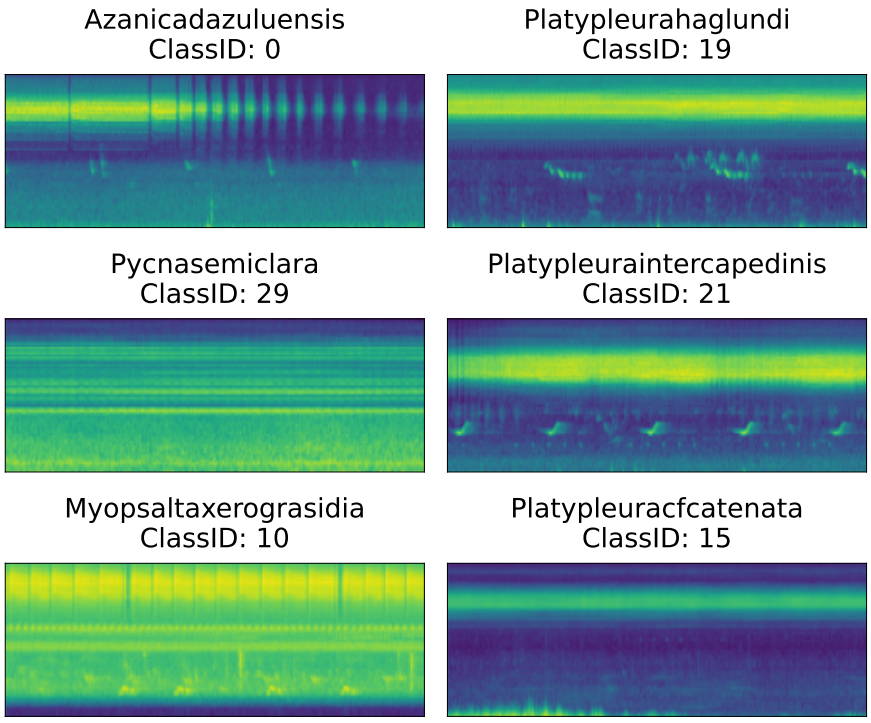

Figure 1: Some examples from the dataset displayed as Mel spectrograms.

2.2. Programming Language and Frameworks

To build and train the deep learning model, the programming language Python was used. The Frameworks PyTorch and Lightning are widely used and powerful tools for building deep learning models.

2.3. Deep Learning Model

The deep learning model used in this study is a convolutional neural network (CNN) with residual blocks (ResNet; He et al., 2016) illustrated in Figure 2. It consists of three main parts: An input layer, a number of stacked residual blocks and an fully connected output layer. The input layer is a convolutional layer with a kernel size of 1 and and output channels set to the base channels (bc) – for this experiment it was set to 8. Note that a convolution with a kernel size of 1 is equivalent to using a fully-connected feed-forward layer. After the convolutional layer, the input is normed with a batch normalization layer and then passed through a ReLU activation function. Batch normalization can make the training of neural networks more robust and faster, and the ReLU activation introduces non-linearity (Goodfellow et al., 2016). The output is than passed trough a number of residual blocks, where Residual connections facilitate the training of deep neural networks by enabling smoother flow of gradients during backpropagation, i.e., the backward pass performed to compute the model gradients (Goodfellow et al., 2016). The model is implemented dynamically, so the number of residual blocks can be set as a hyperparameter. Each residual block consists of two convolutional layers, each followed by a batch normalization layer and a ReLU activation function. The output channels are doubled with every residual block. The residual connection is implemented by passing the input trough a separated convolutional layer with a kernel size of 1 and the same number of output channels as the output of the convolutional layers in the residual block to match dimensions. This output is than added to the output of the transformation in the residual block. At the end of the residual block, the output is passed trough a max pooling layer to reduce the dimensions. The max pooling layer is implemented to alter the kernel size with every residual block. For every odd residual block the kernel size is set to n_max_pool – in this experiment it was set to 3 – reducing the dimensions by a factor of 3. For every even residual block the kernel size is set to 1, so the dimensions stay the same. The output of the residual blocks is passed through a dynamic global average pooling layer, and then passed through a fully-connected feed-forward layer. An additional convolutional layer with kernel size 1 to match the number of classes. The output was flattened and passed through a softmax activation function to get the probabilities for each class as a vector of length n_classes – in this experiment 32 – elements between 0 and 1 that sum up to 1.

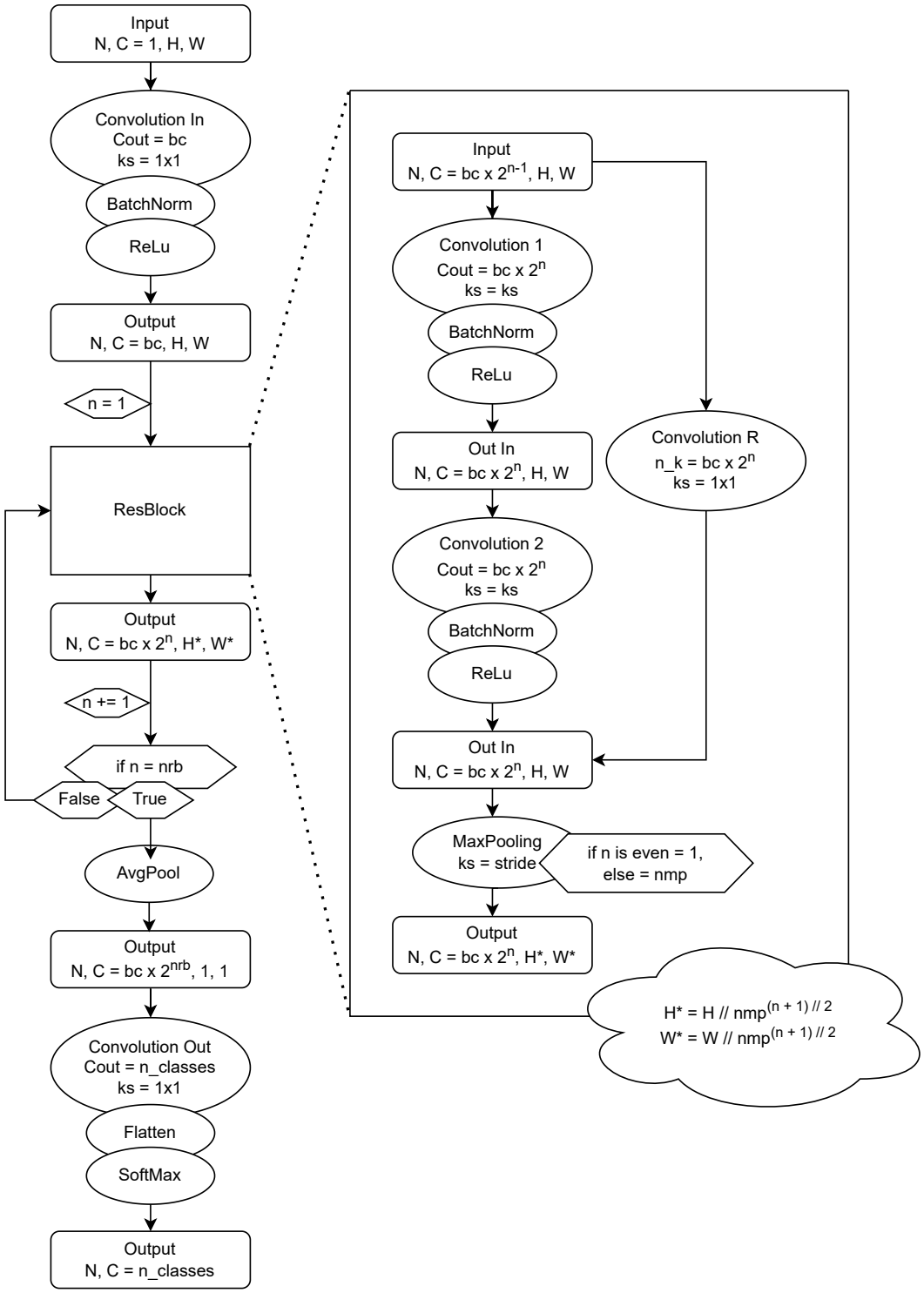

Figure 2: Flow chart illustrating the models architecture. N: elements in batch, C: channels, H: height, W: width, Cout: output channels, ks: kernel size, bc: base channels, nrb: number of residual blocks, n_classes: number of classes, nmp: parameter for max pooling

2.4. Data Processing

A custom Dataloader was implemented, to handle the data processing on the fly and provide the trainer with then data samples matching the chosen indices. To implement the dataloader, the PyTorch Dataset and DataLoader classes where used. For the data processing, the torchaudio and numpy libraries where used. There is two steps to the data processing: Sampling and Transformation.

2.4.1. Sampling

The audio files are of different lengths and the model was fed with consistent, potentially zero-padded samples for consistency. Since the smallest files are of a length of around 1 second and the longest file is around 160 seconds, a compromise had to be made. On the one hand, trimming the files to a length of 1 second would mean very little information being available for the model to learn from. On the other hand, if the files are sampled for a length of more than a second, the short files would need to be padded with zeros meaning the file starts with an empty part. This could lead to the model learning from the length of the empty part and not the actual audio signal, i.e., it would be prone to overfitting. To avoid this, the audio files where sampled to a random length between 1 and 10 seconds and then padded with zeros to the fixed length of 10 seconds. To implement this, a custom method was implemented as shown in Listing 1.

Listing 1: Python code for the sampling of the filies

```python
import numpy as np
import torch

def get_random_part_padded(self, waveform: Tensor, samplerate: int) -> Tensor:

    min_len_in_samples = int(self.min_len_in_seconds * samplerate)
    max_len_in_samples = int(self.max_len_in_seconds * samplerate)

    if self.min_len_in_seconds == -1:
        sample_start_index = -max_len_in_samples
        sample_end_index = None

    else:
        part_length = np.random.randint(min_len_in_samples, max_len_in_samples + 1)
        sample_length = waveform.shape[1]
        part_length = min(part_length, sample_length)
        sample_start_index = np.random.randint(0, sample_length - part_length + 1)
        sample_end_index = sample_start_index + part_length

    waveform_part = waveform[:, sample_start_index:sample_end_index]
    actual_part_length = waveform_part.shape[1]
    pad_length = max_len_in_samples - actual_part_length
    waveform_pad = torch.nn.functional.pad(waveform_part, pad=(pad_length, 0, 0, 0))

    return waveform_pad
```

2.4.2. Transformation

Audio signals are rich in information, but also contain spurious information. One way to condense the the information content of audio data is the transformation via spectrograms. A spectrogram is constructed using short time Fourier transformation (STFT). The STFT applies a mathematical transformation to short windows in the time domain, and transfers them from the time into the frequency domain. The transformed snippets are aligned along the time dimension, and thereby, we obtain a two-dimensional, compressed version of the original audio signal. As an extension of the plain spectrogram, the Mel spectrogram enhances certain frequencies, such that it reflects the frequencies which are perceived stronger by the the human ear, which is more sensitive to low frequencies. This transformation is commonly used in the field of ecoacoustics (Stowell, 2022, p. 7). A visualization of the two transformation – the plain spectrogram, and the Mel spectrogram – is provided in Figure 3 for a random sample with no padding. Both versions of the transformation where transformed into decibels and normalized before being passed into the model. To implement the two varieties of the transformation, the library torch and torchaudio where used. A custom transformation method was implemented, that can be used as a layer in the model as shown in Listing 2. Some of the parameters of the transformation where made configurable and others dependant on them where calculated. In the hyperparameter tuning phase, the only parameter that was tuned was the number of mel bins, which was set to either 64 to use the mel-spectrogram or -1 to use the standard spectrogram.

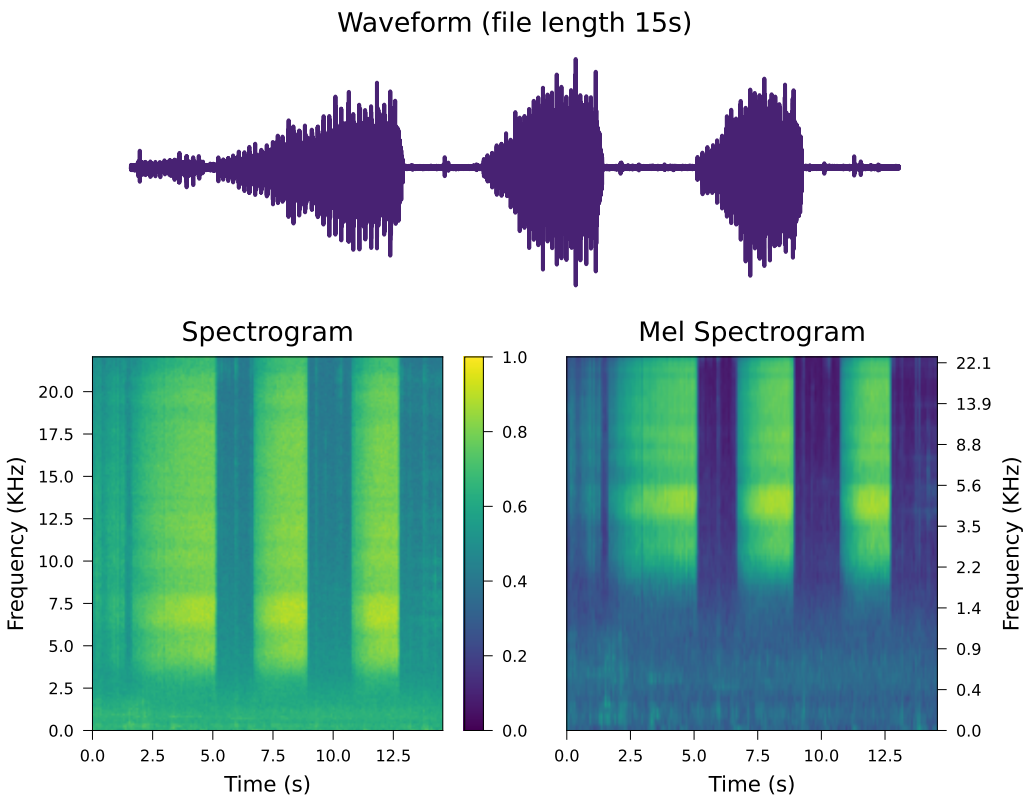

Figure 3: Visualization of the two transformations of the audio signal.

Listing 2: Python code for the transformation of the audio signal

```
import torch
import torchaudio

class NormalizeSpectrogram(torch.nn.Module):
    def forward(self, tensor):
    return (tensor - tensor.min()) / (tensor.max() - tensor.min())

normalize_transform = NormalizeSpectrogram()

if n_mels == -1:
    spectrogram = torchaudio.transforms.Spectrogram(
        n_fft=n_fft,
        hop_length=int(n_fft/2),
        win_length=n_fft)
else:
    spectrogram = torchaudio.transforms.MelSpectrogram(
        n_fft=n_fft,
        hop_length=int(n_fft/2),
        win_length=n_fft,
        n_mels=n_mels,
        f_max=self.sample_rate / 2)

db_transform = torchaudio.transforms.AmplitudeToDB(top_db=top_db)

self.transform = torch.nn.Sequential(
```

```
26      spectrogram,
27      db_transform,
28      normalize_transform)
```

2.5. Fitting the Model

The training was handled by the PyTorch Lightning framework. The logging was done with the TensorBoard logger and with an additional custom logger to get easier access to the data afterwards. The hyperparameter grid search was implemented with a custom Python script.

2.5.1. Training

The training was done on a single GPU. There was no limit to the maximum of training epochs but an early stopping callback, that stopped the training if the validation loss did not improve for a patience of 100 epochs and the best model was restored. The model was trained with a batch size of 10. The optimizer used was the AdamW optimizer of the PyTorch library with a weight decay of 0. The loss function used was the CrossEntropyLoss function of the PyTorch library with default parameters and class weights inversely proportional to the available data per class, in order to give more emphasis to classes with fewer samples. Different learning rates where evaluated during the hyperparameter tuning referred to in section 2.5.2. Only the best model was saved and used for the evaluation of the model. In order to simplify the evaluation, on completion of the training the whole dataset was predicted and saved to a csv file in the log folder.

2.5.2. Hyperparameter Tuning

For the hyperparameter tuning, a select number of hyperparameters where chosen to be tuned. Considerations like the experience of some early tests, the computational resources available and the time frame of the project where taken into account. The hyperparameters that where chosen for the grid search are shown in Table 1. For the grid search, models for all possible combinations of the hyperparameters – in this case $2 \times 3 \times 2 \times 3 = 36$ where trained. To implement the grid search a short Python script was written to create the system commands to start the training with the different hyperparameter combinations.

Table 1: Hyperparameters and values used for the grid search.

Hyperparameter	Description	Variations	Values
n_mels	transformation (-1 for regular STFT)	2	64, -1
n_res_blocks	number of res blocks	3	2, 3, 4
learning_rate	step size during optimization	2	0.001, 0.0001
kernel_size	dimension of the filter	3	3, 5, 7

2.6. Evaluation

The models were evaluated based on the predictions performed at the end of the training. The output of the model was a vector of length 32 with probabilities for each class, which where transformed into a predicted class ID by taking the index of the highest probability using the argmax function of the numpy library. For all the models they where stored in a csv file in their log folder. To evaluate the hyperparameter tuning, the predictions of the validation set where used, as they where not used during the training of the model except for the early stopping and the selection of the best model. To evaluate the best model, only the predictions of the test set where used.

2.6.1. Metrics and Tests

Accuracy: The accuracy is the ratio of correctly predicted observations to the total observations. It was calculated using the mean function of the numpy library:
`np.mean(y_true == y_pred)` The accuracy was the metric chosen to select the best model for further evaluation. Note that the accuracy can be misleading, as it does not take into account class imbalances, and it is possible to achieve a high accuracy overall while incorrectly classifying classes with fewer samples.

F1 Score: The F1 score is a metric that combines the precision and recall of a model. In this case the macro average was used, which calculates the F1 score for each class and then takes the mean. It was calculated using the f1_score function of the sklearn library:
`f1_score(y_true, y_pred, average='macro', sample_weight=None, zero_division='warn')`

F1 Score per Class: The F1 score per class was calculated using the f1_score function of the sklearn library with the average parameter set to None:
`f1_score(y_true, y_pred, average=None, sample_weight=None, zero_division='warn')`
It was calculated for each class and for each model from the hyperparameter tuning. The results where than aggregated to get the mean F1 score per class over all models.

Confusion Matrix: The result of the Predictions of the best model where used to create a confusion matrix. The confusion matrix is a table that is often used to analyze the performance of a classification model. Predictions and true labels are compared and

visualized in a matrix it was calculated using the confusion_matrix function of the sklearn library.

Pearson Test: The Pearson test was used to test the correlation between the model size and the accuracy or F1 score of the models. The Pearson test was calculated using the pearsonr function of the scipy library:

```
for i, metric in enumerate(['accuracy', 'f1']):
  pearson_corr, p_value = stats.pearsonr(ex.summary[metric], ex.summary['num_trainable_params'])
```

And the correlation between the train data count and the F1 score per class:

```
pearson_corr, p_value = stats.pearsonr(data['f1'], data['count'])
```

For both cases the null hypothesis (H0) posited that there is no correlation, with the significance level (alpha) set at 0.05.

3. Results

3.1. Hyperparameter Tuning

The detailed results of the Hyperparameter tuning are shown in Table 3. The ranking of the score according to the accuracy or the F1 score are similar but there are some differences. The best performing model had and accuracy of 0.706 with an F1 score of 0.571, while the best model according to the F1 (0.608) score had an accuracy of 0.667. It is quite difficult to find any obvious tendencies for the influence of the hyperparameters on the performance of the model. Figure 4 shows the distribution of the accuracy comparing different different hyperparameters grouped by the transformation used. There was a tendency visible for the MelSpectrogramm to work a bit better. A smaller learning rate seems to deliver more robust results in general, yet the difference was marginal. For the Mel spectrogram, the best performance was observed with 3 ResBlocks, while this hyperparameter had no impact with the standard spectrogram overall. Furthermore, there was a tendency for smaller kernel sizes to deliver more robust results, and the best performance was achieved with a kernel size of 5.

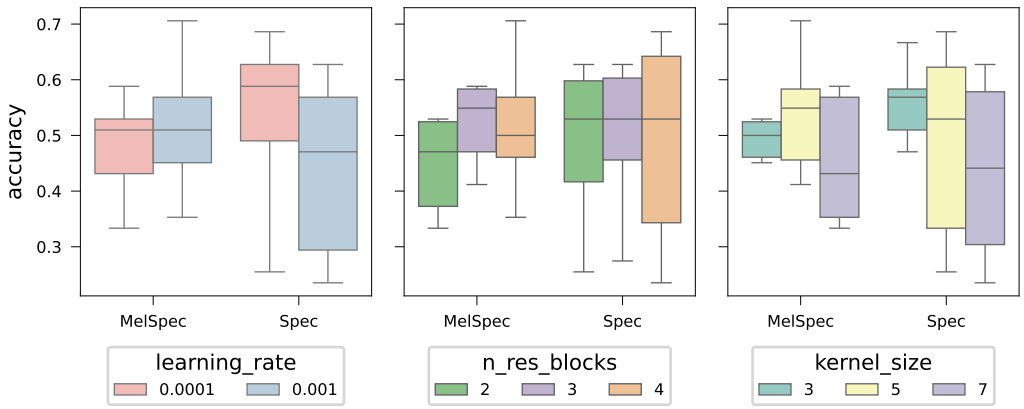

Figure 4: Accuracy of the models for different hyperparameter grouped by the transformation type.

In Figure 5, the models size measured by the number of trainable parameters is plotted against the accuracy and the F1 score. There was no obvious correlation visible between the model size and the performance of the model. This is also supported by the results of the Pearson Test. For both metrics, the p value was clearly above 0.05 (the chosen confidence interval), i.e., the null hypothesis – that there is no relationship between the performance and the number of model parameters – had to be rejected. The r values were both quite close to zero but negative, which indicates that there might be a slight –

yet not significant – tendency for a smaller model to perform better.

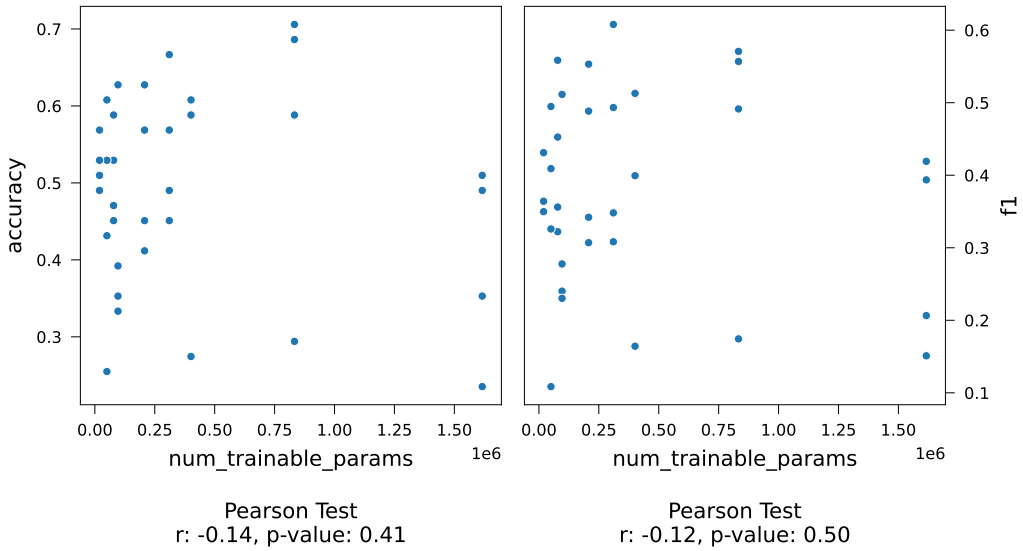

Figure 5: Model size compared to the accuracy and F1 Score of the models.

The F1 score per class is shown in Figure 6 and compared to the distribution of the available data. Class 11 has a very low F1 score compared to the available data while class 9 has a very high F1 Score compared to the available data. To further examine the relation between the F1 score and the distribution of the data in Figure 7, the F1 score is plotted against the distribution of the training data. There is no obvious correlation visible between the F1 score and the distribution of the data. This is also supported by the results of the Pearson Test. The p value is above 0.05, i.e., we accept the null hypothesis that there is no significant relationship. The r value is positive, and therefore, there might be a slight tendency for more training data to deliver a better F1 score.

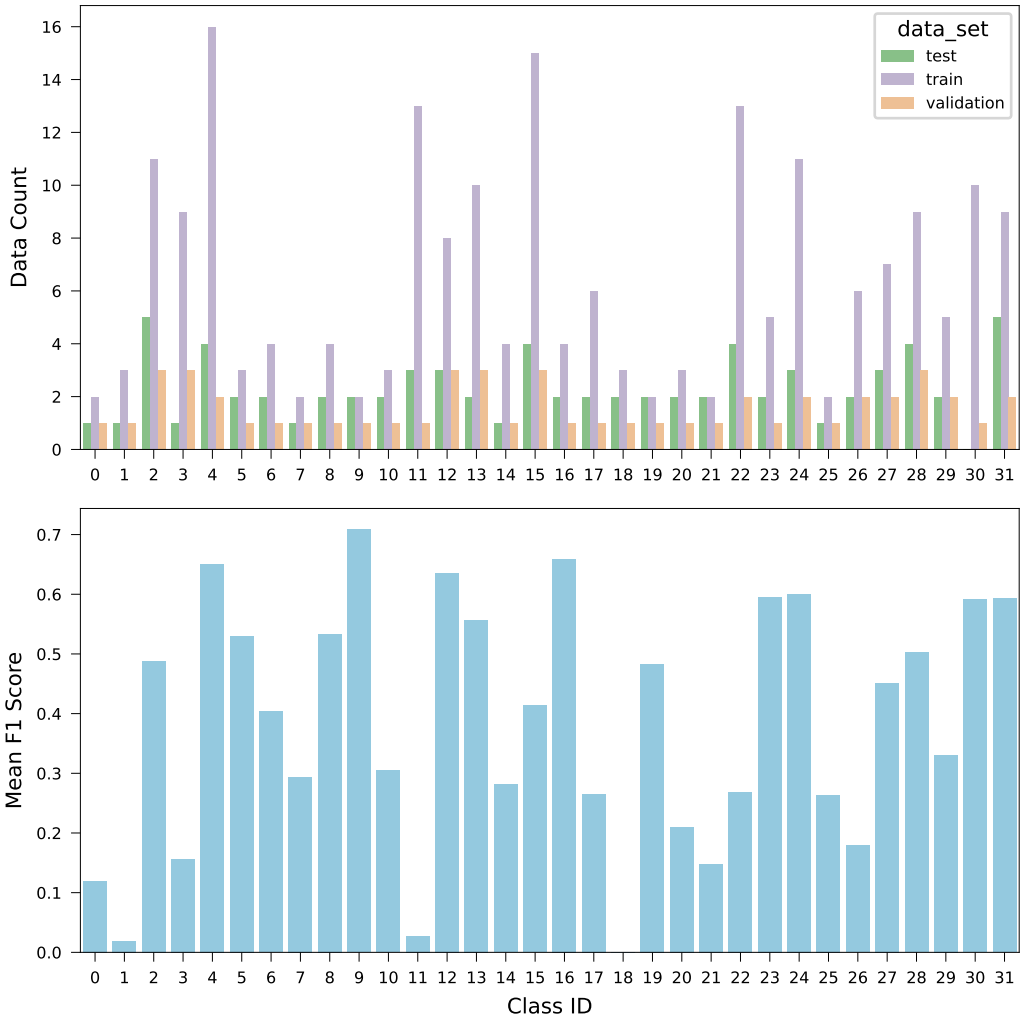

Figure 6: F1 Score per class as mean trough all models compared to data distribution.

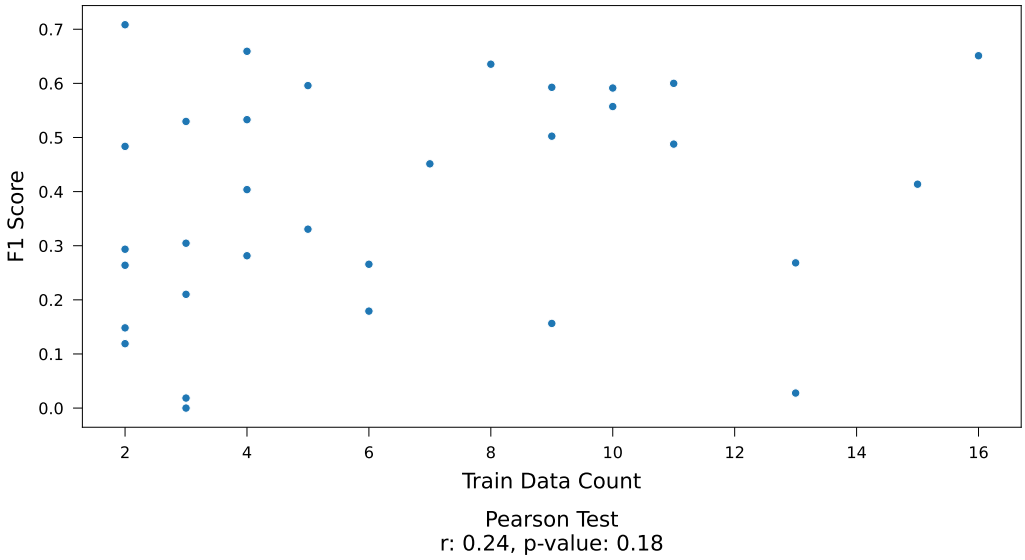

Figure 7: Available training data per classes compared to the F1 Score per classes in a scatter plot and the result of the Pearson correlation test.

3.2. Performance of the best Model

The best performing configuration for the model was found to be the one with the following hyperparameters:

- **n_mels:** 64
- **n_res_blocks:** 4
- **learning_rate:** 0.001
- **kernel_size:** 5

On the test set, the model achieved an accuracy of 0.649 and an F1 score of 0.546. The confusion matrix for the predictions generated by this model is shown in Figure 8. Since for this approach the data was sliced randomly on the fly, for the evaluation of the model, the test data is even more limited because every file was only predicted once instead of multiple times for every slice. This is why the confusion matrix is not as clear as it could be. The concentration of results in the diagonal of the confusion matrix indicates that the model is performing well. Class 22 was predicted the most frequent with more than 50% of the predictions being wrong.

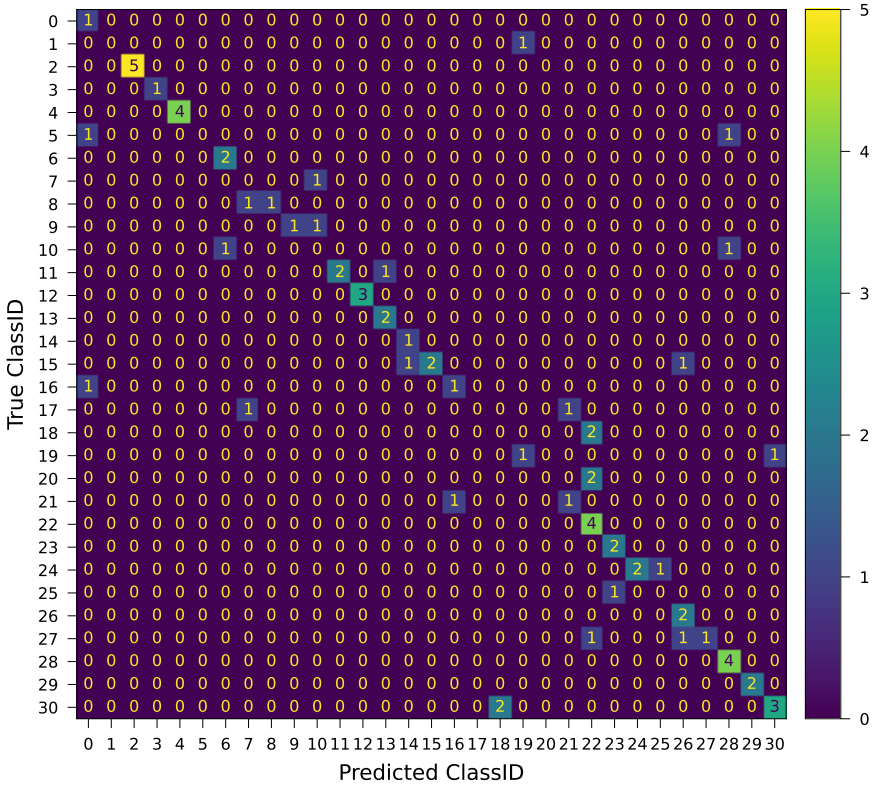

Figure 8: Confusion matrix for the predictions of the test set using the best model. The over all accuracy on the test set was 0.649.

Next, we take a closer look at the training curves of the best-performing model. The validation accuracy and loss at the end of every epoch is shown in Figure 9. The graphs show a typical training process. The validation accuracy is increasing while the validation loss is decreasing. Towards the end, both graphs are flattening out which is a sign that the model was not learning anymore when the training was stopped. This indicates that the patience of 100 for the early stopping was an appropriate choice. Since the validation loss and accuracy are very noisy, it was necessary to use a large patience value.

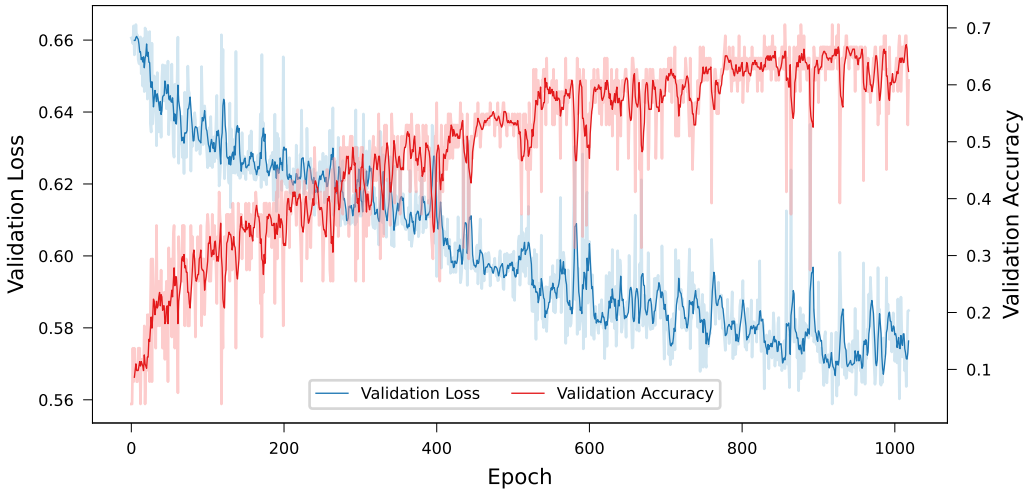

Figure 9: The validation loss and accuracy of the best model. Light version is not smoothed and dark version is smoothed with a window size of 5.

4. Discussion

4.1. Hyperparameter Tuning

The detailed results of the hyperparameter tuning are shown in Table 3. It is quite difficult to find any obvious tendencies for the influence of the hyperparameters on the performance of the model. There is no direct correlation between the model size and the performance of the model. This might be explicable by the fact that the available data is not sufficient to train larger models This could be helped by extending the dataset with additional recordings or by implementing some sort of data augmentation (Stowell, 2022, p. 3). Some ideas being to add random levels of noise to the recordings or to randomly add slight shifts in time or frequency. An other approach could be to mix data from different classes to create new samples. This would have two main advantages. First, the combination possibilities are nearly endless and therefore the dataset could be extended massively. Second, the model would be trained to to detect multiple classes in one sample wich could be useful in a real world scenario where multiple species are present at the same time. The data augmentation could be implemented in the dataloader to be done on the fly.

An other aspect to look into is the selection of the hyperparameters. Due to limited resources, the hyperparameter tuning was done with a limited number of configurations. There is still a list of potential hyperparameters that could be tested. As an example, the base channels after the input layer, or different parameters for the transformation such as the hop length, window size or the number of mel bins could be further investigated. As the model is, the kernel size stays constant for all layers – the possibilities there are endless. Furthermore, other deep learning architectures could be explored. It is possible that – given the low amount of training samples – a smaller and more efficient model architecture could perform better than the ResNet used here.

4.2. Comparison of Results with Original Study

In the original paper (Faiss & Stowell, 2023), different datasets and models were evaluated. The model was run five times with different seeds and the accuracy was averaged to achieve an ensemble of predictions, which can be more robust. The model using a MelSpectrogram frontend on the InsectSet32 dataset achieved an mean accuracy of 0.6 with a range of 0.57 to 0.65 on the validation set and an accuracy of 0.62 with a range of 0.57 to 0.67 on the test set. The best performing configuration for the model in this study achieves an accuracy of 0.706 on the validation set and an accuracy of 0.649. For the test set, this is well in the range of the original paper. Why the model in this study performs better on the validation set compared to the difference between the validation and test set in the original paper is not clear. The accuracy achieved by the model with the other frontend from the original paper is unreached by the model in this study. Consequently, there is still room for improvement especially in the data processing and, as already mentioned, data augmentation.

4.3. Conclusion

Species classification with deep learning can support the non-invasive monitoring of biodiversity. However, the advanced methodological approaches and computation demands needed for such endeavors can be an obstacle. In this study, I reproduced a prior study on classifying insects based on an open-access dataset. I wanted to investigate whether the results can be replicated with limited technical knowhow, computational resources, and under temporal constraints. The model was trained on a regular gaming computer with a GPU by a student with none to little experience in the field of deep learning – notable with quite some demand for support (refer to Acknowledgments 5). Still, it is save to say that this technology has become broadly accessible. The model performance was comparable to the original study in terms of accuracy. The model was able to classify the sounds of a limited subset of insects with an accuracy of 0.649. I identified some shortcomings of the approach: The training dataset is rather small, and additional samples could potentially improve the results further. Similarly, data augmentation could enhance the classification skill. I encourage the investigation of additional deep learning model architectures and an an exhaustive sampling of the hyperparameter space to further improve model performance.

5. Acknowledgment and Declaration

5.1. Acknowledgment

I thank Dr. Matthias Nyfeler for his support and the many interesting discussions we had. I would like to express my gratitude to my brother, Dr. Basil Kraft, for his continuous support with his expertise in the field of machine learning.

5.2. Declaration of AI Usage

GitHub Copilot was used to assist writing the code and text for this project.

ChatGPT was used to assist researching as well as writing the code and text for this project.

Zürcher Hochschule
für Angewandte Wissenschaften

N-FO-Formular Statement of Authorship for

Student Work

Life Sciences und Facility Management

Education Unit

Statement of Authorship for Student Work at the School of Life Sciences and Facility Management

By submitting the enclosed

☐ Project
☐ Literature review
☐ Course work
☒ Minor paper
☐ Bachelor's thesis
☐ Master's thesis (tick as appropriate)

the student affirms independent completion of the(ir) work without outside help.

The undersigned student declares that all printed and electronic sources used in the text and the bibliography are correctly indicated, i.e., that the work does not contain any plagiarism (no parts that have been taken in whole or in part from another's or his/her own text or from another's or his/her own work without clear identification and without stating the source).

In the event of misconduct of any kind, Paragraph 39 and Paragraph 40 of the General Academic Regulations for Bachelor's and Master's degree programmes at the Zurich University of Applied Sciences (dated 29 January 2008) and the provisions of the Disciplinary Measures of the University Regulations shall apply.

Location, date: Student signature:

Neuhausen, 2024-7-16 Julian Kraft

Note on submitting the Statement of Authorship:

Direct submission of the work: This Statement of Authorship is to be inserted in the appendix of the ZHAW version of all work with original signatures and date (copies will not be accepted).

Submission of the work via Complesis: The Statement of Authorship should be made directly in Complesis by clicking as directed and should not be inserted in the appendix of the work.

Erlassverantwortliche/-r	LeiterIn Stabsbereich Bildung	Ablageort	2.05.00 Lehre Studium
Beschlussinstanz	LeiterIn Stab	Publikationsort	Public
Genehmigungsinstanz			

Version	Beschluss	Beschlussinstanz	Inkrafttreten	Beschreibung Änderung
1.0.0	15.03.2022	LeiterIn Stab	15.03.2022	Originalversion
1.1.0	15.04.2024	Leiter:in Stab	01.5.2024	Adding clarification on self-plagiarism

References

Baker, E., Price, B., Rycroft, S., Hill, J., & Smith, V. S. (2015a). BioAcoustica: A free and open repository and analysis platform for bioacoustics. *Database*, *2015*, bav054. https://doi.org/10.1093/database/bav054

Baker, E., Price, B., Rycroft, S., & Villet, M. (2015b). Global Cicada Sound Collection I: Recordings from South Africa and Malawi by B. W. Price & M. H. Villet and harvesting of BioAcoustica data by GBIF. *Biodiversity Data Journal*, *3*, e5792. https://doi.org/10.3897/BDJ.3.e5792

Brondízio, E. S., Settele, J., Díaz, S., & Ngo, H. T. (Eds.). (2019). *The global assessment report of the intergovernmental science-policy platform on biodiversity and ecosystem services*. Intergovernmental Science-Policy Platform on Biodiversity and Ecosystem Services (IPBES).
OCLC: 1336011247.

Cardinale, B. J., Duffy, J. E., Gonzalez, A., Hooper, D. U., Perrings, C., Venail, P., Narwani, A., Mace, G. M., Tilman, D., Wardle, D. A., Kinzig, A. P., Daily, G. C., Loreau, M., Grace, J. B., Larigauderie, A., Srivastava, D. S., & Naeem, S. (2012). Biodiversity loss and its impact on humanity. *Nature*, *486*(7401), 59–67. https://doi.org/10.1038/nature11148

Deng, I. (2023). Harnessing the Power of Sound and AI to track Global Biodiversity Framework (GBF) Targets.

Faiss, M. (2022, September 12). *InsectSet32: Dataset for automatic acoustic identification of insects (Orthoptera and Cicadidae)* (Version 0.1). Zenodo. https://doi.org/10.5281/zenodo.7072196

Faiss, M., & Stowell, D. (2023). Adaptive representations of sound for automatic insect recognition (R. Martinez-Garcia, Ed.). *PLOS Computational Biology*, *19*(10), e1011541. https://doi.org/10.1371/journal.pcbi.1011541

Goodfellow, I., Benigo, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Kahl, S., Wood, C., Eibl, M., & Klinck, H. (2021). BirdNET: A deep learning solution for avian diversity monitoring. *Ecological Informatics*, *61*, 101236. https://doi.org/10.1016/j.ecoinf.2021.101236

Orthoptera. (2008). In J. L. Capinera (Ed.), *Encyclopedia of Entomology* (pp. 2695–2695). Springer Netherlands. https://doi.org/10.1007/978-1-4020-6359-6_1892

Popple, L. W. (2017). A revision of the Myopsalta crucifera (Ashton) species group (Hemiptera: Cicadidae: Cicadettini) with 14 new species from mainland Australia. *Zootaxa*. https://doi.org/10.11646/zootaxa.4340.1

Sanborn, A. (2008). Cicadas (Hemiptera: Cicadoidea). In J. L. Capinera (Ed.), *Encyclopedia of Entomology* (pp. 874–877). Springer Netherlands. https://doi.org/10.1007/978-1-4020-6359-6_666

Scarpelli, M. D. A., Liquet, B., Tucker, D., Fuller, S., & Roe, P. (2021). Multi-Index Ecoacoustics Analysis for Terrestrial Soundscapes: A New Semi-Automated Approach

Using Time-Series Motif Discovery and Random Forest Classification. *Frontiers in Ecology and Evolution*, *9*, 738537. https://doi.org/10.3389/fevo.2021.738537

Stowell, D. (2022). Computational bioacoustics with deep learning: A review and roadmap. *PeerJ*, *10*, e13152. https://doi.org/10.7717/peerj.13152

A. Appendix: Tables

Table 2: Available data for each class and data set. The file length is given in seconds. The file count is the number of files available for each class and data set. The file lengths are sorted in ascending order.

Species	Class ID	Data Set	File Length Sum	File Count	File Lengths
Azanicadazuluensis	0	train	20	2	[10, 10]
		validation	10	1	[10]
		test	10	1	[10]
Brevisianabrevis	1	train	30	3	[10, 10, 10]
		validation	10	1	[10]
		test	10	1	[10]
Chorthippusbiguttulus	2	train	126	11	[4, 7, 9, 9, 10, 11, 12, 14, 15, 17, 19]
		validation	32	3	[6, 12, 14]
		test	47	5	[5, 7, 11, 12, 13]
Chorthippusbrunneus	3	train	90	9	[4, 6, 6, 6, 8, 9, 15, 15, 21]
		validation	35	3	[7, 14, 14]
		test	11	1	[11]
Grylluscampestris	4	train	176	16	[4, 4, 6, 7, 7, 8, 8, 9, 10, 11, 11, 11, 13, 14, 25, 29]
		validation	16	2	[7, 8]
		test	28	4	[4, 7, 7, 8]
Kikihiamuta	5	train	30	3	[10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Myopsaltaleona	6	train	40	4	[10, 10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Myopsaltalongicauda	7	train	20	2	[10, 10]
		validation	10	1	[10]
		test	10	1	[10]
Myopsaltamackinlayi	8	train	38	4	[8, 10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Myopsaltamelanobasis	9	train	20	2	[10, 10]
		validation	10	1	[10]
		test	13	2	[3, 10]
Myopsaltaxerograsidia	10	train	30	3	[10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Nemobiussylvestris	11	train	341	13	[2, 4, 5, 6, 6, 8, 9, 10, 25, 31, 31, 39, 167]
		validation	13	1	[13]
		test	144	3	[6, 40, 98]
Oecanthuspellucens	12	train	145	8	[14, 15, 15, 16, 18, 18, 19, 29]
		validation	48	3	[8, 18, 22]
		test	76	3	[22, 24, 29]
Pholidopteragriseoaptera	13	train	71	10	[3, 4, 4, 4, 6, 7, 7, 9, 10, 17]
		validation	35	3	[5, 7, 22]
		test	8	2	[4, 5]
Platypleuracapensis	14	train	40	4	[10, 10, 10, 10]
		validation	10	1	[10]
		test	10	1	[10]
Platypleuracfcatenata	15	train	144	15	[4, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
		validation	30	3	[10, 10, 10]
		test	40	4	[10, 10, 10, 10]
Platypleurachalybaea	16	train	36	4	[6, 10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Platypleuradeusta	17	train	53	6	[3, 10, 10, 10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Platypleuradivisa	18	train	30	3	[10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Platypleurahaglundi	19	train	20	2	[10, 10]
		validation	10	1	[10]

Table 2: Available data for each class and data set (continued)

Species	Class ID	Data Set	File Length Sum	File Count	File Lengths
		test	20	2	[10, 10]
Platypleurahirtipennis	20	train	24	3	[4, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Platypleuraintercapedinis	21	train	20	2	[10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Platypleuraplumosa	22	train	130	13	[10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10]
		validation	20	2	[10, 10]
		test	40	4	[10, 10, 10, 10]
Platypleurasp04	23	train	50	5	[10, 10, 10, 10, 10]
		validation	10	1	[10]
		test	20	2	[10, 10]
Platypleurasp10	24	train	95	11	[1, 3, 10, 10, 10, 10, 10, 10, 10, 10, 10]
		validation	20	2	[10, 10]
		test	30	3	[10, 10, 10]
Platypleurasp11cfhirtipennis	25	train	20	2	[10, 10]
		validation	10	1	[10]
		test	10	1	[10]
Platypleurasp12cfhirtipennis	26	train	60	6	[10, 10, 10, 10, 10, 10]
		validation	20	2	[10, 10]
		test	20	2	[10, 10]
Platypleurasp13	27	train	70	7	[10, 10, 10, 10, 10, 10, 10]
		validation	20	2	[10, 10]
		test	30	3	[10, 10, 10]
Pseudochorthippusparallelus	28	train	63	9	[3, 4, 4, 5, 5, 5, 6, 12, 21]
		validation	15	3	[3, 5, 7]
		test	30	4	[4, 5, 6, 15]
Pycnasemiclara	29	train	50	5	[10, 10, 10, 10, 10]
		validation	20	2	[10, 10]
		test	20	2	[10, 10]
Roeselianaroeselii	30	train	43	10	[1, 2, 2, 3, 3, 4, 4, 4, 8, 11]
		validation	4	1	[4]
Tettigoniaviridissima	31	train	51	9	[3, 4, 4, 5, 6, 7, 7, 7, 9]
		validation	13	2	[6, 7]
		test	30	5	[4, 5, 5, 5, 10]

Table 3: Results of the hyperparameter tuning by descending accuracy.

n_mels	res blocks	learning rate	kernel size	parameters	epochs	accuracy	F1
64	4	0.001	5	832,912	1018	0.706	0.571
-1	4	0.0001	5	832,912	672	0.686	0.557
-1	4	0.0001	3	310,544	891	0.667	0.608
-1	3	0.001	5	207,376	394	0.627	0.553
-1	2	0.0001	7	96,528	1198	0.627	0.511
-1	2	0.001	5	50,256	640	0.608	0.495
-1	3	0.0001	7	401,104	760	0.608	0.512
-1	3	0.0001	3	78,224	897	0.588	0.558
64	4	0.0001	5	832,912	371	0.588	0.491
64	3	0.0001	7	401,104	536	0.588	0.513
64	3	0.001	7	401,104	350	0.588	0.399
64	3	0.001	5	207,376	505	0.569	0.488
-1	2	0.001	3	19,408	469	0.569	0.431
-1	4	0.001	3	310,544	562	0.569	0.493
64	3	0.0001	3	78,224	1115	0.529	0.453
64	2	0.001	5	50,256	661	0.529	0.409
64	2	0.0001	3	19,408	1354	0.529	0.365
64	4	0.0001	7	1,616,464	273	0.510	0.419
64	2	0.001	3	19,408	455	0.510	0.364
-1	4	0.0001	7	1,616,464	296	0.490	0.394
-1	2	0.0001	3	19,408	1138	0.490	0.350
64	4	0.001	3	310,544	408	0.490	0.348
-1	3	0.001	3	78,224	456	0.471	0.356
-1	3	0.0001	5	207,376	531	0.451	0.342
64	4	0.0001	3	310,544	330	0.451	0.308
64	3	0.001	3	78,224	474	0.451	0.322
64	2	0.0001	5	50,256	892	0.431	0.326
64	3	0.0001	5	207,376	395	0.412	0.307
-1	2	0.001	7	96,528	375	0.392	0.240
64	4	0.001	7	1,616,464	403	0.353	0.207
64	2	0.001	7	96,528	229	0.353	0.230
64	2	0.0001	7	96,528	545	0.333	0.278
-1	4	0.001	5	832,912	290	0.294	0.174
-1	3	0.001	7	401,104	268	0.275	0.164
-1	2	0.0001	5	50,256	432	0.255	0.109
-1	4	0.001	7	1,616,464	145	0.235	0.151