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

---

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 1, 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

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 was developed. The model was trained using the PyTorch framework and optimized through 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 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

All code, data, and the final report can be found on GitHub:

List of Figures

List of Tables

1. Introduction

1.1. Background

One of the global environmental issues is the loss of biodiversity (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 biodiversity. Traditional methods for quantifying biodiversity are time-consuming and expensive, and they are not suitable for large-scale monitoring. But what if there was a non invasive method that could be used to monitor biodiversity in a fast and efficient way?

Bioacoustics, the study of sound production, dispersion and reception in animals, is an interesting 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 amazing 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. The questions are typically a bit broader and about 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 even though there are still some issues to be fixed (Scarpelli et al., 2021). These passive acoustic monitoring (PAM) like the using of sound recordings to monitor biodiversity are currently widely researched and developed and even combined 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 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).

The focus of this study is to reproduce the results of the paper (Faiss, 2022) and to create a model that can classify a subset of insects using their voices. 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 goal is to proof that this technology is accessible for everyone with the knowledge and a regular gaming computer with a graphic processing unit (GPU). To achieve this, a model will be constructed and trained using the same dataset

as the original paper. A hyperparameter tuning will be performed to find the best configuration for the model. The results of the hyperparameter tuning will be evaluated and discussed. The best performing model will be tested and evaluated for its performance and accuracy. The results will be discussed and compared to the results of the original paper.

2. Methods

2.1. Dataset

In this study a preexisting dataset is 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 four to 22, with durations from 40 seconds to almost nine minutes. All files, originally 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.

2.2. Programming Language and Frameworks

To build and train the deep learning model, the programming language Python was used. The Frameworks PyTorch, Lightning are very popular 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 illustrated in Figure 2. It consists of three main parts: An input layer, a number of 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. After the convolutional layer, the input is normed with a batch normalization layer and then passed through a ReLU activation function. This output is than passed trough a number of residual blocks. 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 than passed trough a fully connected layer, consisting of a global average pooling layer to reduce the dimensions to the number of classes. An additional convolutional layer with kernel size 1 to match the number of classes. A flattening layer to get of the dimensions H and W already reduced to 1 and 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.

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 can only handle inputs of a fixed size. 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. To only sample 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 a basically empty part. This could lead to the model learning from the length of the empty part and not the actual audio signal. 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

```
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

The model is actually one, usually used for image classification and therefore expects two dimensional input while an audio file has only one dimension. To transform the samples into into a format, that the model can handle, a short time Fourier transformation (STFT) was applied. The Fourier transformation is a mathematical operation that transforms a function of time into a function of frequency. An other way to describe this is, that a series of spectrograms for short time slices of the audio signal are created and aligned in a 2D array - basically a visualization of the audio signal and therefore something a image classification model can handle. In addition a second version was implemented, where the STFT where transformed additionally. The two versions are visualized for a random sample with no padding in Figure 1. The frequency bins were transformed into mel bins, which are a more human like representation of the frequency content of the audio signal. This transformation is called a mel-spectrogram and is commonly used in the field of ecoacoustics (Stowell, 2022, p. 7). 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 try the mel-spectrogram or -1 to use the standard spectrogram.

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(
    spectrogram,
    db_transform,
    normalize_transform)
```

2.5. Fitting the Model

The training was completely 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. The model was trained for a maximum of 2000 epochs with a early stopping callback, that stopped the training if the validation loss did not improve for patience = 100 epochs. The model was trined 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 according to available data per class. Different learning rates where tried during the hyperparameter tuning referred to in subsubsection 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

To evaluate the models, the predictions already implemented at the end of the training. The output of the model was a vector of length 32 with probabilities for each class, they 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 chose the best model for further evaluation.

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 is the F1 score for each class. It 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. The results where than visualized in a bar chart combined with the available data per class - refer to Figure 5.

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 shown in Figure 7. The confusion matrix in Figure 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.

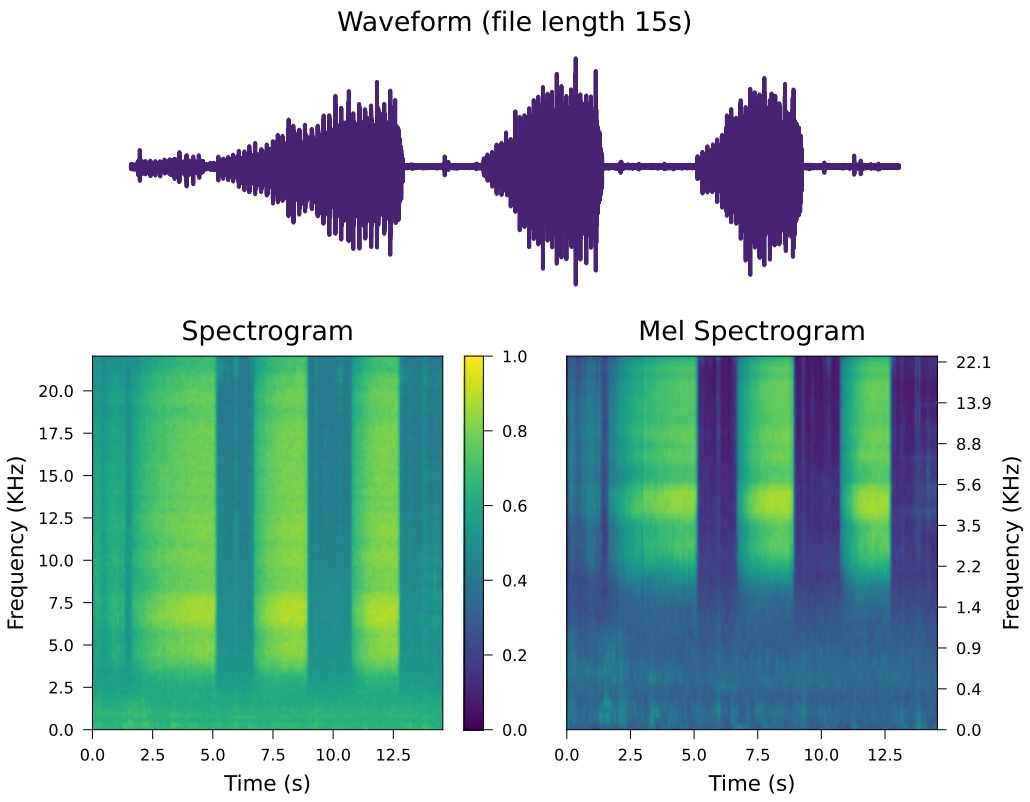

Figure 1: Visualization of the two transformations of the audio signal.

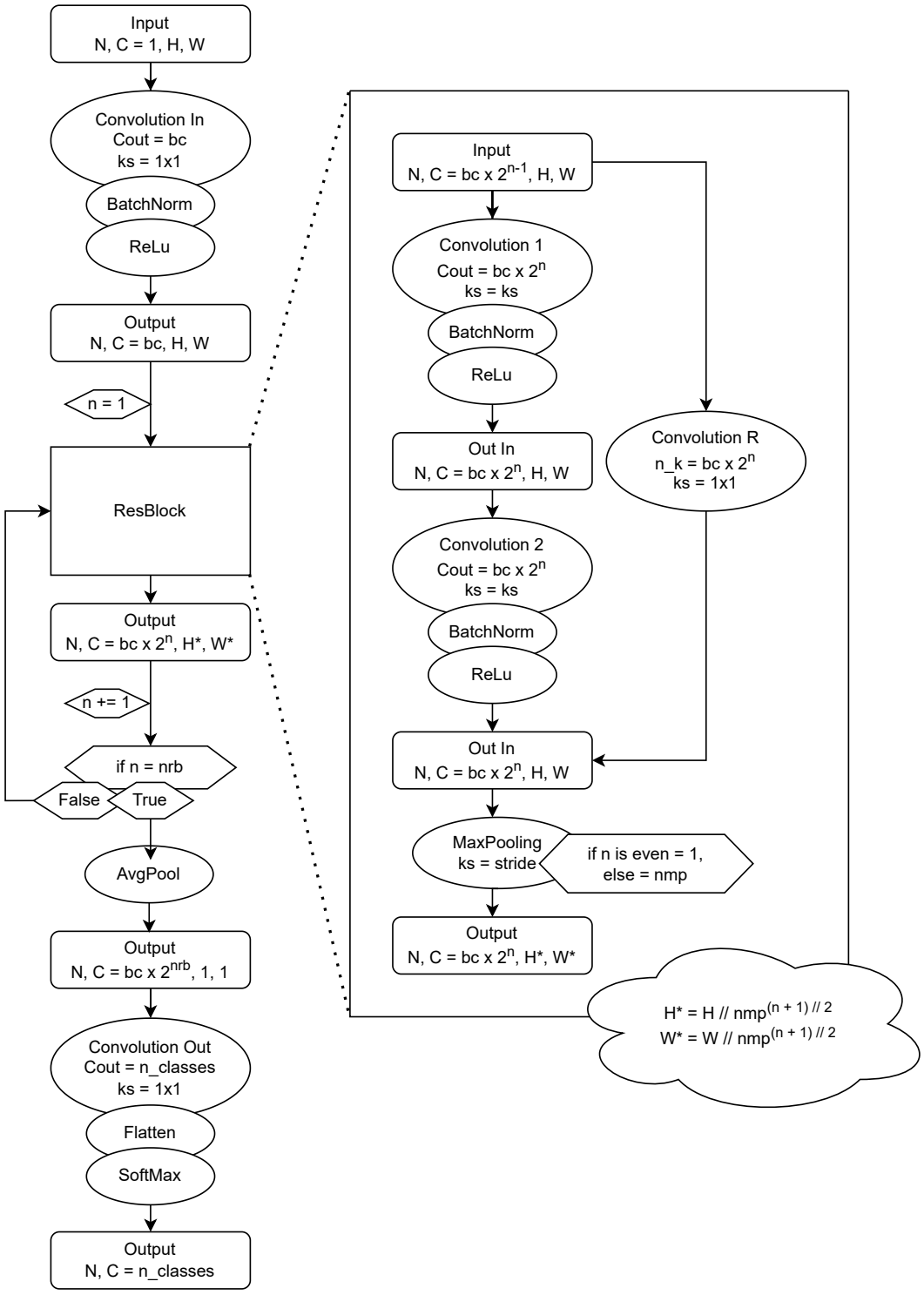

Figure 2: Flow chart illustrating the models architecture. N: elements in batch, C: channels, H: height, W: width, Cout: output channels, ks: kernel size, bc: base channels, nrb: number of residual blocks, n_classes: number of classes, nmp: parameter for max pooling

3. Results

3.1. Hyperparameter Tuning

The detailed results of the Hyperparameter tuning are shown in Table 2. The ranking of the score according to the accuracy or the F1 score are similar but there are some differences. As an example is the best performance according to the accuracy 0.706 with an F1 score of 0.571 while the best score according to the F1 score is 0.608 with an accuracy of 0.667 - so the best model divers in the two scores. It is quite difficult to find any obvious tendencies for the influence of the hyperparameters on the performance of the model. Figure 3 shows the distribution of the accuracy comparing different different hyperparameters grouped by the transformation used. There is a tendency visible for the MelSpectrogramm to work a bit better. A smaller learning rate seems to deliver results a bit less distributed but the difference is not very big. For the MelSpectrogram transformation the number of ResBlocks seems to have a bigger influence on the performance - two seems not to be enough. Furthermore there is a tendency for smaller kernel sizes to deliver less distributed results but a better performance is achieved with a kernel size of 5. In Figure 4 the models size measured by the number of trainable parameters is plotted against the accuracy and the F1 score. There is 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 is far above 0.05 leaving the null hypothesis that there is no significant correlation between the model size and the performance of the model. The r values both quite close to zero but negative indicate that there might be a slight tendency for a smaller model to perform better. The F1 score per class is shown in Figure 5 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 6 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 leaving the null hypothesis that there is no significant correlation. The r value is positive therefore there might be a slight tendency for more training data to deliver a better F1 score.

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

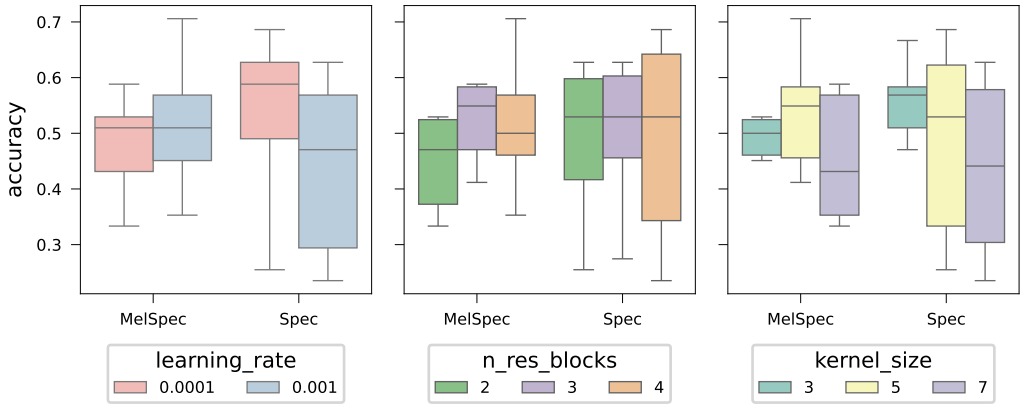

Figure 3: Accuracy of the models for different hyperparameter grouped by the transformation type.

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 7. 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.

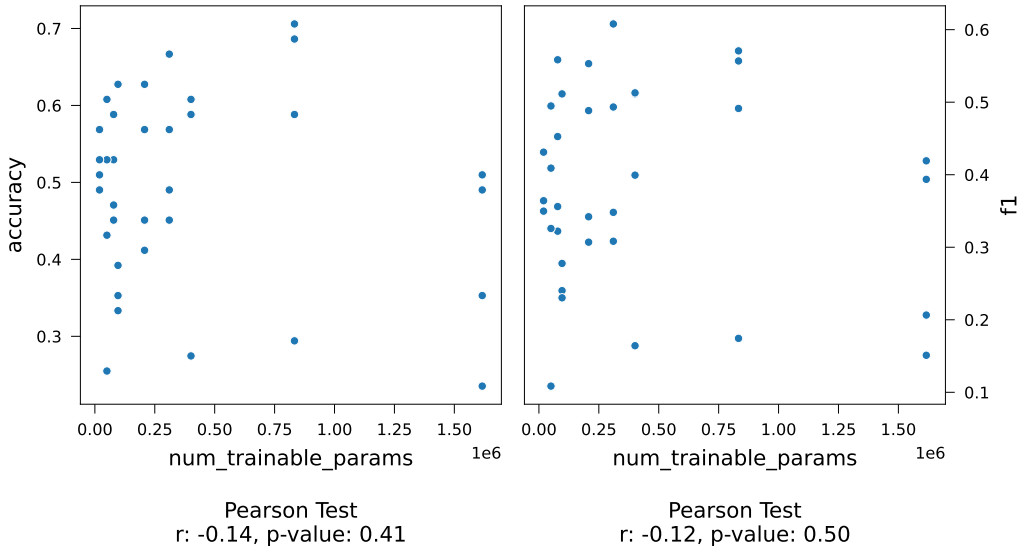

Figure 4: Model size compared to the accuracy and F1 Score of the models.

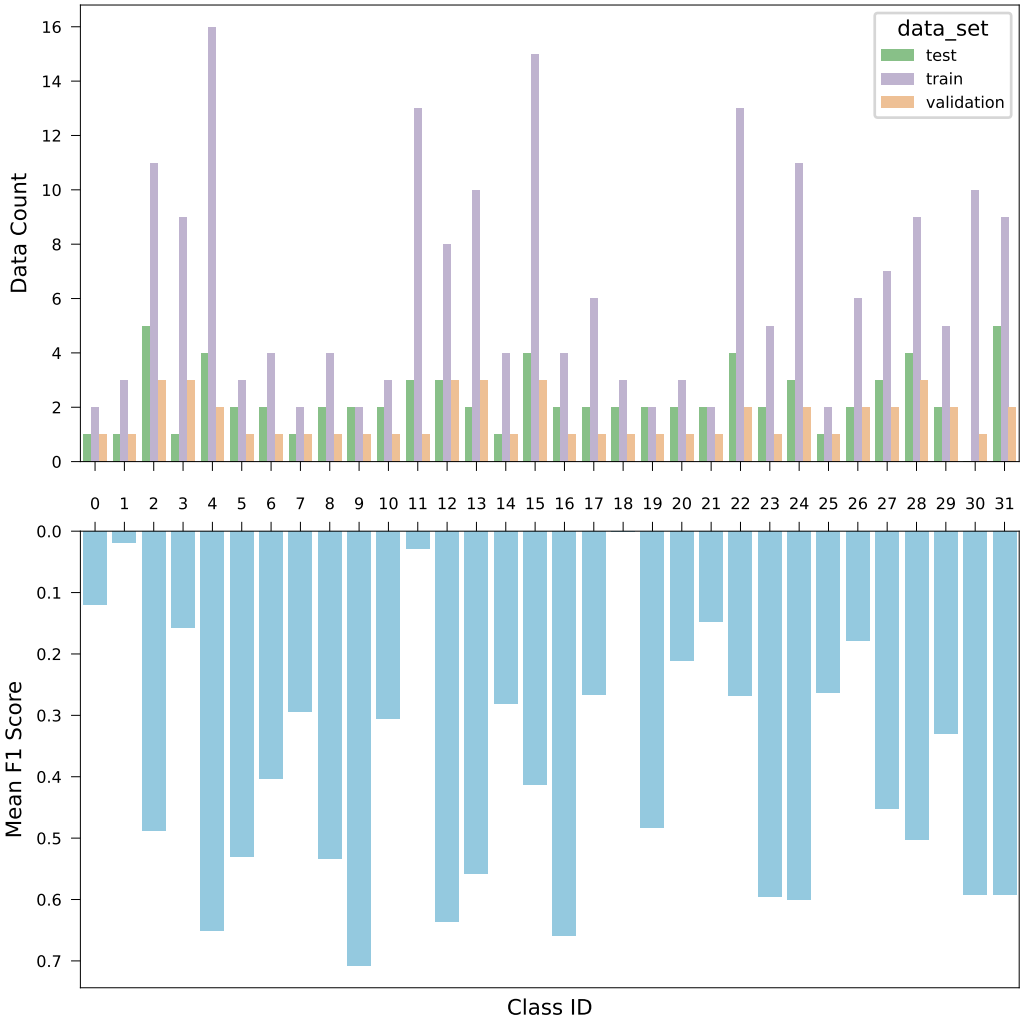

Figure 5: F1 Score per class as mean trough all models compared to data distribution.

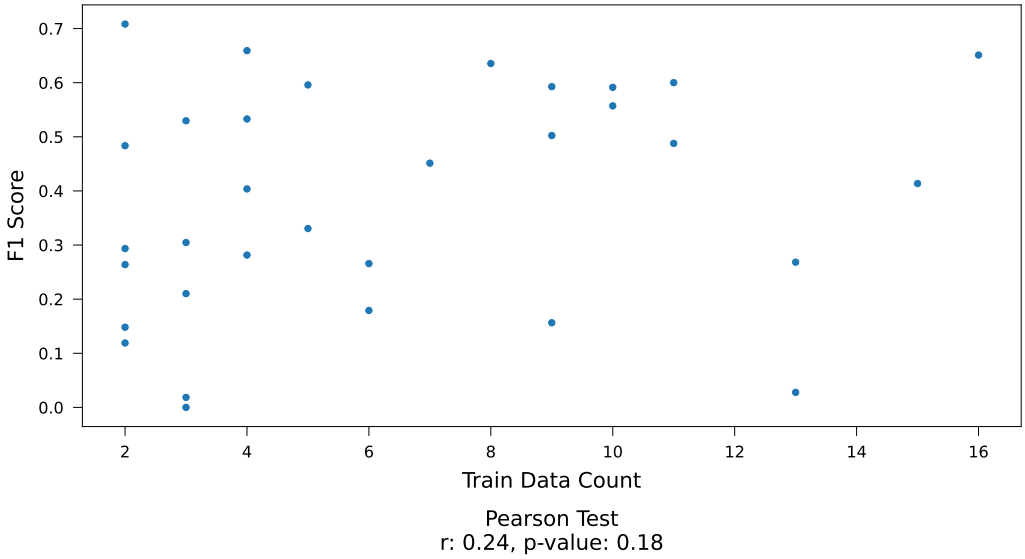

Figure 6: Available training data per classes compared to the F1 Score per classes in a scatter plot and the result of the Pearson correlation test.

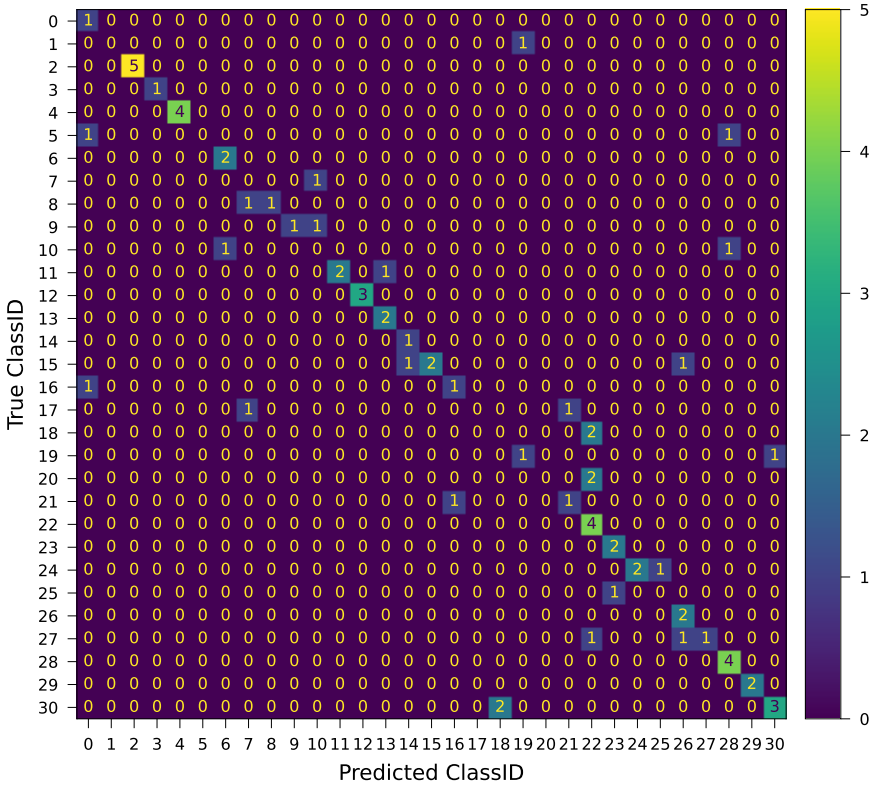

Figure 7: Confusion matrix for the predictions of the test set using the best model. The over all accuracy on the test set was 0.649.

4. Discussion

4.1. Hyperparameter Tuning

The detailed results of the Hyperparameter tuning are shown in Table 2. 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. 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 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. In order to implement this, the output layer of the model must be changed - specifically the softmax activation function must be replaced by a sigmoid activation function. The data augmentation could be implemented in the dataloader to be done on the fly.

An other thing to look into is the selection of the hyperparameters. Dew 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 in layer or different parameters for the transformation such as the hop length, window size or the number of mel bins. As the model is, the kernel size stays the same for all layers - the possibilities there are endless.

4.2. Performance of the best Model

Using the best performing configuration it is worth to take a closer look to the training process of the model. The validation accuracy and loss at the end of every epoch is shown in Figure 8. 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. So 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 patience that high. This might be a sign, that the data processing or the model architecture could be improved.

In the original paper (Faiss & Stowell, 2023) the model was used on different datasets and with different models. Furthermore the model was run five times with different seeds and the accuracy was averaged. 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

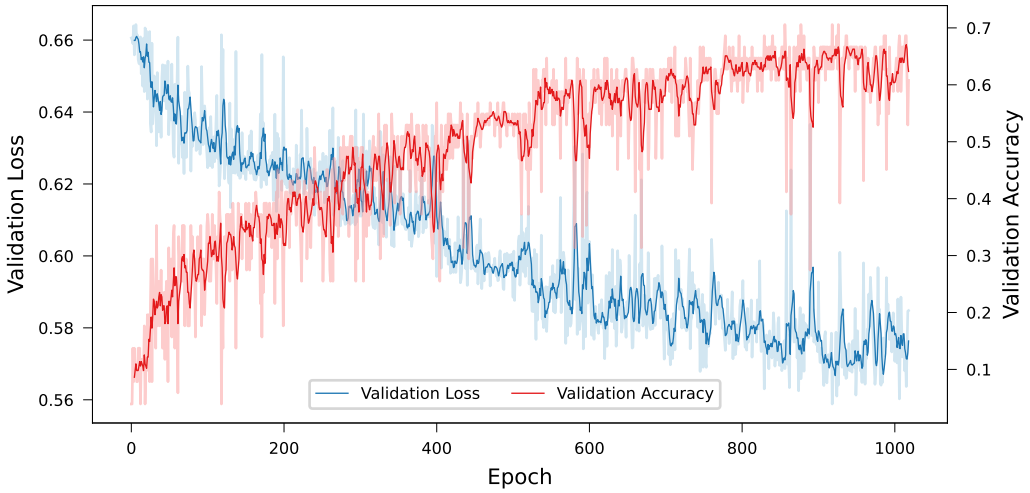

Figure 8: The validation loss and accuracy of the best model. Light version is not smoothed and dark version is smoothed with a window size of 5.

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. So there is still room for improvement especially in the data processing.

4.3. Sum Up

To sum up, the model was successfully trained and evaluated. The results are comparable to the results of the original paper. The model is able to classify the sounds of a limited subset of insects with an accuracy of 0.649. This was all done on a regular gaming computer with a GPU by a student with none to little experience in the field of deep learning - notbabel with quite some demand for support. Still it is save to say that this technology has become accessible to everyone with the knowledge and some quite affordable hardware.

5. Acknowledgment and Declaration

5.1. Acknowledgment

I would like to express my gratitude to my brother, Dr. Basil Kraft for his continuous support and expertize in the field of machine learning. Without his help, this project would not have been possible.

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, 2024-6-27 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

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

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.

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

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

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

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

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

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

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

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

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

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

A. Appendix: Tables

Table 2: 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

Table 3: ClassID legend.

ClassID	Species	ClassID	Species
0	Azanicadazuluensis	16	Platypleurachalybaea
1	Brevisianabrevis	17	Platypleuradeusta
2	Chorthippusbiguttulus	18	Platypleuradivisa
3	Chorthippusbrunneus	19	Platypleurahaglundi
4	Grylluscampestris	20	Platypleurahirtipennis
5	Kikihiamuta	21	Platypleuraintercapedinis
6	Myopsaltaleona	22	Platypleuraplumosa
7	Myopsaltalongicauda	23	Platypleurasp04
8	Myopsaltamackinlayi	24	Platypleurasp10
9	Myopsaltamelanobasis	25	Platypleurasp11cfhirtipennis
10	Myopsaltaxerograsidia	26	Platypleurasp12cfhirtipennis
11	Nemobiussylvestris	27	Platypleurasp13
12	Oecanthuspellucens	28	Pseudochorthippusparallelus
13	Pholidopteragriseoaptera	29	Pycnasemiclara
14	Platypleuracapensis	30	Roeselianaroeselii
15	Platypleuracfcatenata	31	Tettigoniaviridissima