Contents

1 Introduction **1**

1.1 Background . . . 1

1.2 Insects of Interest . . . 1

1.3 Dataset . . . 1

2 Materials and Methods **3**

2.1 Programming Language and Frameworks . . . 3

2.2 Deep Learning Model . . . 3

2.3 Data Processing . . . 3

2.3.1 Sampeling . . . 3

2.3.2 Transformation . . . 3

2.3.3 Training . . . 5

2.3.4 Evaluation . . . 5

2.4 Hyperparameter Tuning . . . 5

3 Results **6**

4 Discussion **7**

1 Introduction

1.1 Background

The question about biodiversity and its importance has been a topic of interest for many years. 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 measuring biodiversity. Traditional methods for measuring 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? Ecoacoustics might deliver a promising solution to this problem even tough there remain some issues to be fixed (Scarpelli et al., 2021). 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).

The focus of this study is to reproduce the results of the paper (Faiss, 2022) and to create a model that can classify the insect sounds with a high accuracy. Furthermore the 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. The goal is to prof that this technology could be accessible for everyone with the knowledge and a regular gaming computer with a graphic processing unit (GPU).

1.2 Insects of Interest

There are countless species of insects in the world, and many of them produce sounds for various reasons. In this study, we are interested in the sounds produced by 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)

1.3 Dataset

The dataset used in this study is the InsectSet32 dataset (Faiss, 2022). Containing this description:

This dataset contains recordings of 32 sound producing insect species with a total 335 files and a length of 57 minutes. The dataset was compiled for training neural networks to automatically identify insect species while comparing adaptive, waveform-based frontends to conventional mel-spectrogram methods for audio feature extraction. This work will be submitted for publication in the future and the dataset can be used to replicate the results or for similar research. Roughly half of the recordings (147) are of nine species belonging to the order Orthoptera. These recordings stem from a dataset that was originally compiled by Baudewijn Odé (unpublished). The remaining recordings (188) are of 23 species in the family Cicadidae. These recordings were selected from the Global Cicada Sound Collection hosted on Bioacoustica (Baker et al., 2015a), including recordings published in (Baker et al., 2015b; Popple, 2017). Many recordings from this collection included speech annotations in the beginning of the recordings, therefore the last ten seconds of audio were extracted and used in this dataset. All files were manually inspected and files with strong noise interference or with sounds of multiple species were removed. Between species, the number of files ranges from four to 22 files and the length from 40 seconds to almost nine minutes of audio material for a single species. The files range in length from less than one second to several minutes. All original files were available with sample rates of at least 44.1 kHz or higher but were resampled to 44.1 kHz mono WAV files for consistency.

The files are split into training, validation and test sets. And there are two .csv files containing the labels and the filenames of the recordings.

2 Materials and Methods

2.1 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.

2.2 Deep Learning Model

2.3 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. There is two steps to the data processing: Sampling and Transformation.

2.3.1 Sampeling

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.

2.3.2 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.

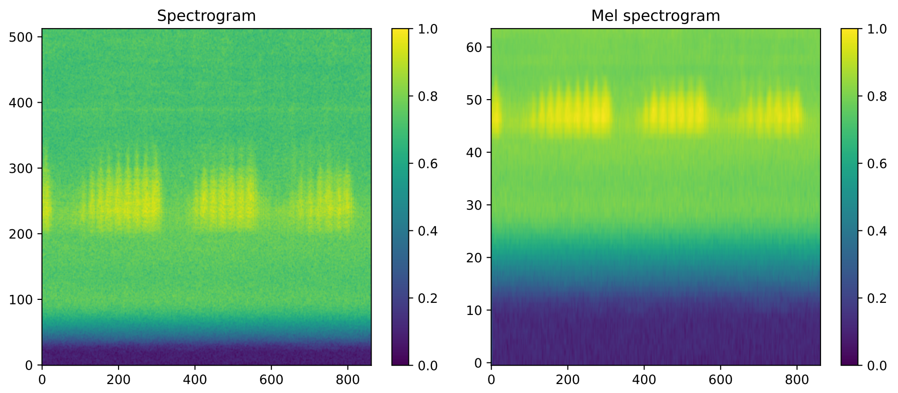

Figure 1: Visualization of the two transformations of the audio signal.

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 1. 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 1: Python code for the transformation of the audio signal

```python

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:
    spectogram = torchaudio.transforms.Spectrogram(
        n_fft=n_fft,
        hop_length=int(n_fft/2),
        win_length=n_fft)
else:
    spectogram = 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(
    spectogram,
    db_transform,
    normalize_transform)


```

2.3.3 Training

2.3.4 Evaluation

2.4 Hyperparameter Tuning

3 Results

4 Discussion