Hochschule Bochum
Bochum University
of Applied Sciences
Campus
Velbert/Heiligenhaus

# Autonome Echtzeit-Vogelartenidentifikation anhand von Vogelgesang auf einem mobilen Computer unter Einsatz des BirdNET-Analyzers

von

**Sonia Passaler**

Matrikelnummer: 018 306 925

schriftliche Dokumentation im Studiengang
technische Informatik im Rahmen des Moduls
Software-Entwicklungsprojekt

**Eingereicht am:** 20. November 2024

**1. Prüfer:** Prof. Dr. rer.nat. Jörg Frochte
**2. Prüfer:** Benedikt Wildenhain

# Abstract

Ornithologen bzw. Vogelkundler nutzen das Monitoring von Vögeln für den Vogeltierschutz.

Monitoring ist die dauerhafte Überwachung von bestimmten Systemen.

Im Bereich der Biologie findet sie Anwendung bei Tier- und Pflanzenarten für die Erfassung der Bestände. Es ist durch solche Beobachtungen möglich, Pflanzen und Tiere zu erforschen. Die erfassten Monitoring-Daten helfen Biologen und Umweltschützern den Arten- und Lebensraum nachzuvollziehen und zu schützen .

Diese Arbeit stellt ein KI-gestütztes Tool vor, welches das Monitoring von Vögeln automatisiert unterstützt.

Über ein Mikrofon kann ein Computer mindestens 24 Stunden lang kontinuierlich die Umgebung aufnehmen. Die Software analysiert die Audioaufnahmen basierend auf einem Netzwerk, welches auf die Erkennung von Vogelarten anhand des Vogelgesangs trainiert ist. Anschließend selektiert die Software aus den Audioaufnahmen einzelne Abschnitte und isoliert daraus den erkannten Vogelgesang.

# Inhaltsverzeichnis

# 1 Einleitende Übersicht

Thema dieses Projekts ist die Entwicklung einer Software zur praktischen Anwendung in der Erkennung von Vogelarten anhand ihres Gesangs.

Das Projekt soll in der Lage sein, ununterbrochen 24 stundenlang Vogelstimmen in der freien Natur aufzunehmen und zu analysieren. Dieses Verfahren ermöglicht es Ornithologen (aktive Menschen im Bereich Vogelkunde) verschiedene Informationen im Bereich Ökologie und Taxonomie zu erlangen. Das Projekt ist ein Hilfsmittel für spezielle Untersuchungen. Diese Untersuchungen können das Monitoring

- von den jeweiligen verteilten Populationen
- des Vogelzugs
- der versprengten Arten

sein.

Das Programm soll folgende Kriterien erfüllen:

- Starten des Programms per Remote-Zugriff
- Aufnahme und Analyse laufen mindestens 24 Stunden ununterbrochen parallel
- Aufnahmen, die Vogelgesang beinhalten, für statistische Auswertungen wie Vogelzählung separat speichern und auswerten

Die vorliegende Arbeit ist folgendermaßen aufgebaut:

1. Abschnitt 2 stellt das Konzept dar
2. Abschnitt 3 skizziert und erklärt den theoretischen Versuchsaufbau
3. Abschnitt 5 erklärt die Umsetzung (Hardware/Software)
4. Abschnitt 7 demonstriert den durchgeführten Versuch
5. Abschnitt ... validiert die Ergebnisse aus den Versuchsdurchführungen
6. in Abschnitt 8 sind die Ergebnisse validiert und gibt einen Ausblick auf geschaffene Möglichkeiten durch dieses Projekt sowie mögliche weitere Entwicklungen des Projektes

# 2 Konzept

Wie in der Einleitung erwähnt soll ein mobiler Computer im Freien eine 24 stundenlange Audioaufnahme durchführen und parallel dazu die Aufnahmen mit einem KI-Netz analysieren. Möglichst zeitnah sind dann auch die Aufnahmen, in denen ein Vogel erkannt ist, herauszufiltern und die Nicht-Vogel-Audioabschnitte zu verwerfen.

Die Software läuft auf dem mobilen Computer, der durch eine Powerbank mit Strom versorgt ist. Beide Hardwarekomponenten sind in einer Box zum Schutz vor Umwelteinflüssen platziert. Die Box hat einen Kabelausgang der ein Mikrofonkabel vom Computer zum Mikrofon nach außen führt. Während die Box z.B. auf den Boden platziert ist, hängt das Mikrofon an einem Baum. Den genaue Versuchsaufbau veranschaulicht Abschnitt 3

Aufnahme, Analyse und Auswertung sollen möglichst parallel für Echtzeit ablaufen. Gleichzeitig ist aber durch geringe Rechenkosten der Stromverbauch und die Auslastung des Computers zu minimieren.

Eine weitere Anforderung an das Projekt ist Datenschutz. Nach StGB `gem. 201 StGB (Verletzung der Vertraulichkeit des Wortes)` ist das unerlaubte mitschneiden und speichern von Gesprächen dritter rechtswidrig. Da das Programm die Originalaudiodatein löscht und auch nur ausschnitte mit erkanntem Vogelgesang speichert, sind Menschen größtenteils bereits durch den Ansatz herausgefiltert. Audioabschnitte mit reinen Störgeräuschen wie z.B. Menschen sind somit ebenfalls herausgefiltert.

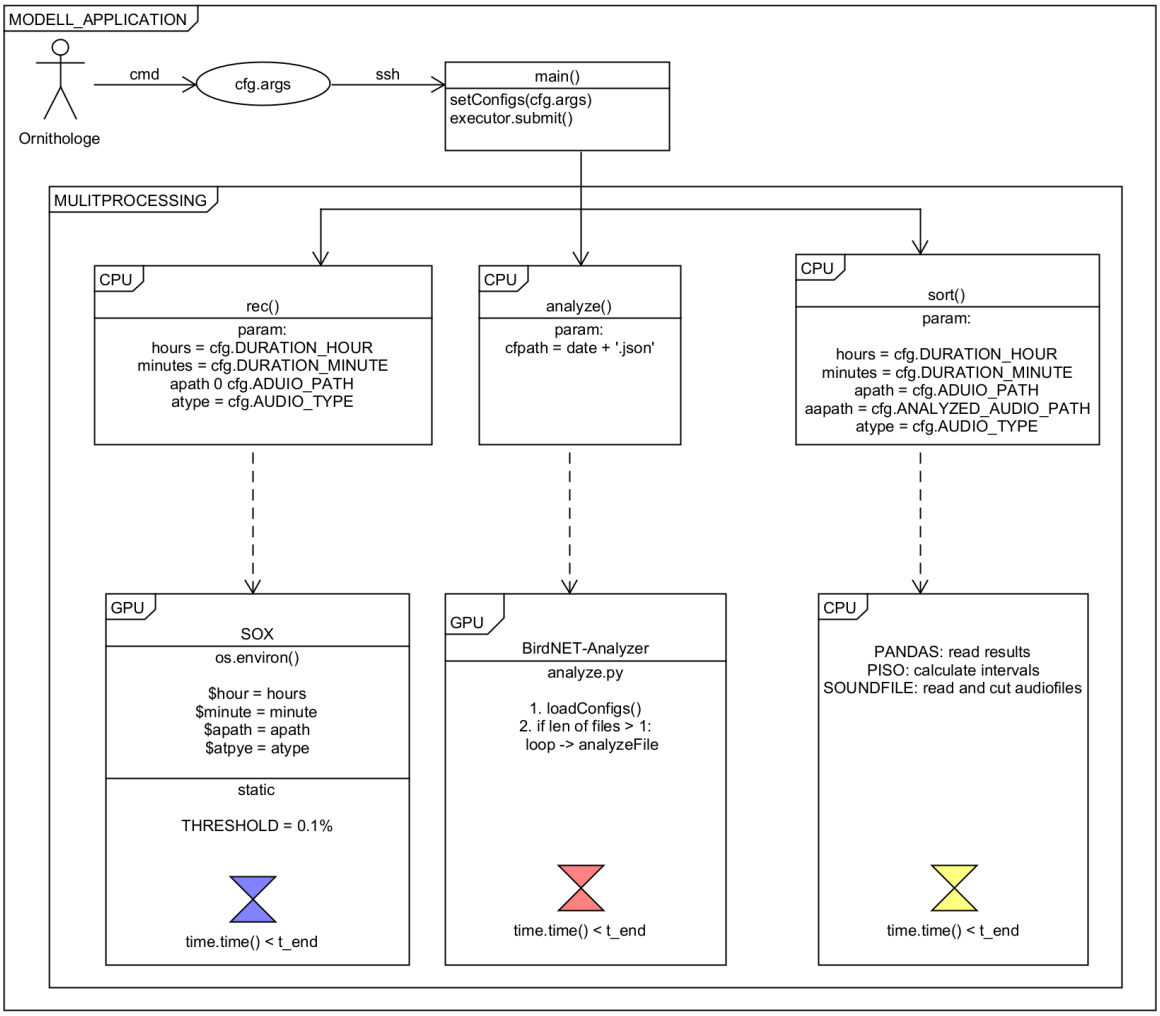


Abbildung 3.1: Modell der Applikation

# 3 Versuchsaufbau

Das Aktivitätsdiagramm in Abbildung 3.1 skizziert den Versuchsaufbau.

Der*Die Ornithologe*in gibt manuell den Befehl auf dem lokalen Rechner ein. Der lokale Rechner ist mit dem kabellosen Netzwerk des mobilen Computers verbunden und sendet über dieses Netzwerk den Befehl an den Computer. Somit ist dieser der SSH-Server, der für die Datenannahme per SSH-Tunnel und Datenverarbeitug per Software (s. Abschnitt 5) zuständig ist. Der mobile Computer ist an eine Powerstation verbunden (s. Abschnitt 3.2). Das Mikrofon ist per USB-Kabel an den Computer angeschlossen (s. Abschnitt 3.3).

## 3.1 Computer

Als mobiler Computer kommt der Nvidia Jetson in der Version Nano J1010 zum Einsatz.

Damit er sicher in der Box gelagert ist, sitzt er in einem Aluminiumgehäuse. Zudem ist eine Referenz-Trägerplatte von Seeed und ein passiver Aluminium-Kühlkörper eingebaut. Zur Speichererweiterung ist eine SD-Karte im Gerät hinzugefügt.

Um eine kabellose Übertragung der Daten vom Host zum Server zu ermöglichen, ist noch eine WiFi-Karte an den mobilen Computer angeschlossen (s. Abbildung 3.1).

Durch seine technischen Eigenschaften ist der Nano in seiner Performance für KI-Anwendungen geeignet. Es sei jedoch nochmal auf Abschnitt 3.1.1 hingewiesen.

Der Link zum Datenblatt befindet sich im Anhang **??** [6].

Alle technisch relevanten Daten sind auch auf der Seite von Nvidia gelistet [7].

Das Betriebssystem ist Linux mit Ubuntu 20.04.

### 3.1.1 Nutzung eines Hardwarebeschleunigers

Ursprünglich war geplant den Nvidia Jetson Nano aufgrund seiner GPU als Hardwarebeschleuniger und der damit einhergehenden Performanceverbessung für KI-Anwendungen zu nutzen. Jedoch hat sich im Laufe der Entwicklung dieses Projektes herausgestellt, dass Nvidia den Support, um Tensorflow auf der GPU laufen zu lassen, nicht (mehr) anbietet. Auch der Umweg durch Installationsanweisungen von anderen Quellen wie von [8] lohnen sich nicht, da laut diesen Quellen die GPU einen negativen Effekt auf die Rechengeschwindigkeit hat.

## 3.2 Stromversorgung

Für die Stromversorgung im Freien kommt eine Powerbank zum Einsatz. Die wichtigste Anforderung an die Powerbank ist die ausreichende Stromkapazität. Da der Jetson einen Leistungsverbrauch von 5 bis 10 Watt pro Stunde bei Auslastungen wie KI-Berechnungen oder High Perfomance Computing hat, kommt eine Powerbank mit 266.4 W h zum Einsatz. Durch einen praktischen Versuch ist verifiziert, dass die Stromkapazität der Powerbank ausreicht.

## 3.3 Mikrofon

Das Mikrofon hat die Rolle der Datenerfassung inne. Es zeichnet das umliegende Audiosignal auf und leitet es per USB-Kabel an den mobilen Computer weiter. Das hier gewählte Mikrofon Clippy EM272Z1 Mono [1] und der dazugewählte Windschutz [9] eine Empfehlung vom gekennzeichneten Amateurfunker [2]. Durch eigene Testversuche ist nach persönlichem Ermessen die Qualität des Mikrofons ausreichend.

### 3.3.1 GPS-Gerät

Zur Erfassung der Koordinaten ist ein GPS-Gerät angeschlossen. Diese Koordinaten verhelfen zu einer sicheren Vorhersage des Neuronalen Netzes. Zudem ermittelt es für den Computer die aktuelle Uhrzeit, da dieser keine Echtzeit-Uhr verbaut hat. Die Nutzung der Daten vom GPS in der Software sind in Abschnitt 6.1 erklärt.

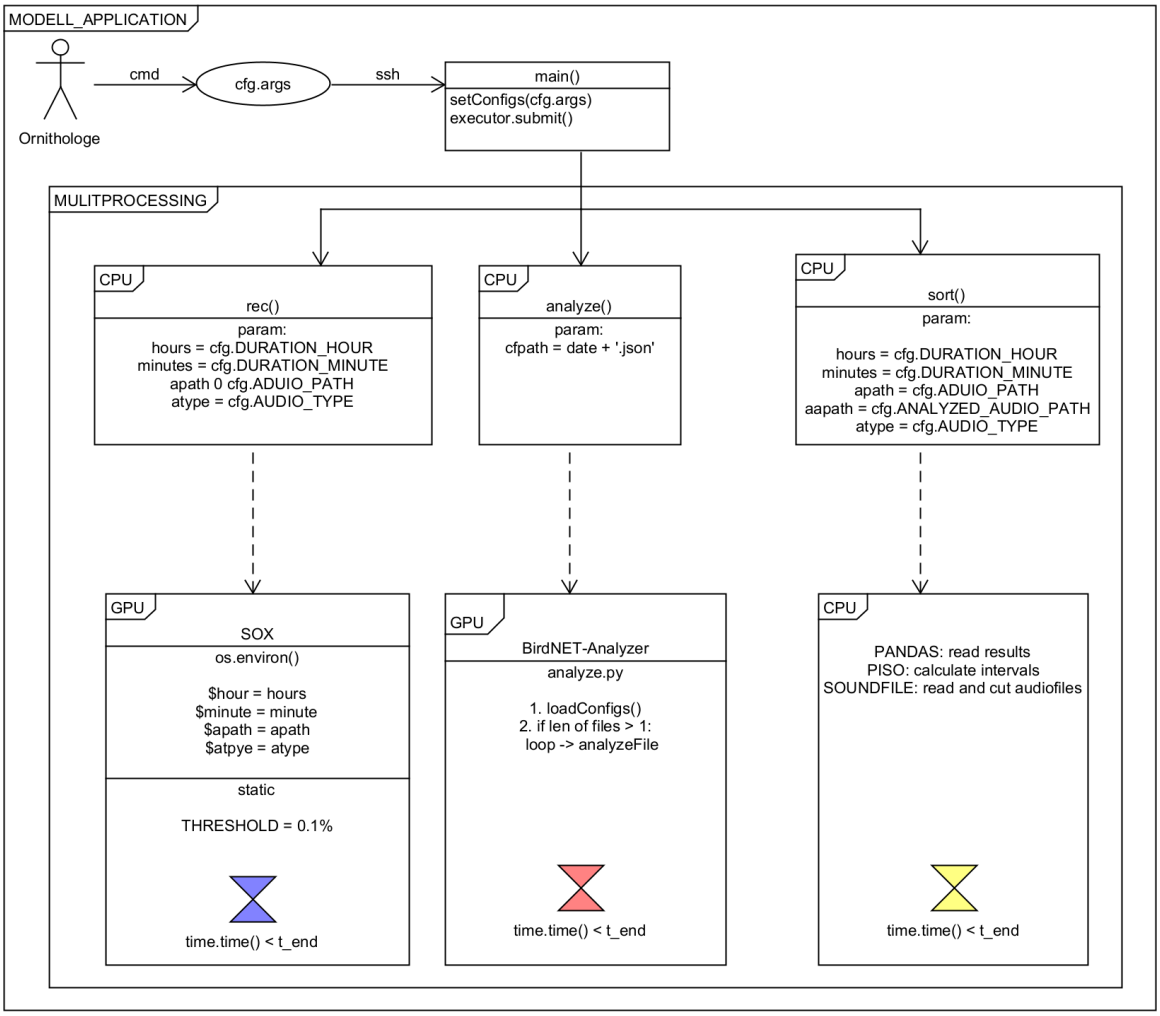


Abbildung 5.1: Modell der Applikation

# 4 test

testtesttesttesttesttesttesttesttest

# 5 Umsetzung

Dieser Abschnitt beschäftigt sich mit der softwarebasierten Umsetzung. Nach einer allgemeinen Übersicht über den Ablauf des Programms, gehen die Unterabschnitte auf die einzelnen Funktionen und die implementierten Bibliotheken ein.

## 5.1 Modell der Software

Der*Die Ornithologe*in ruft per Shell das Pythonskript `main.py` mit den optionalen Startargumenten (s. Abschnitt 6.1) auf. Beim Aufruf von main.py konfiguriert die main-Funktion das Programm mit den übergebenen Startargumenten. Danach ruft die Funktion über die Methoden der Pythonbibliothek *multiprocessing* (s. Abschnitt **??**) die Funktionen `rec()`, `analyze()` und `sort()` als parallel auszuführende Prozesse auf. Diese Funktionen bekommen die Parameter, die im Modell 5.1 in den jeweiligen Blö-

cken unter `param:` gelistet sind, übergeben. Die Ausführung der Funktionen ist zeitlich eingegrenzt. Diese wiederum rufen Unterprozesse auf.

`rec()` ruft SOX (s. Abschnitt 5.2.7) für die energiesparende kontinuierliche Audioaufzeichnung auf. `analyze()` stellt die Schnittstelle zum BirdNET-Analyzer her (s. Abschnitt5.5.2).

`sort()` nutzt zum Zuschneiden der Audiofiles die Bibliotheken Pandas (s. Abschnitt 5.2.2), Piso (s. Abschnitt 5.2.9) und Soundfile (s. Abschnitt 5.2.3).

## 5.2 Bibliotheken

### 5.2.1 Multiprocessing

Multiprocessing ist das zeitlich parallele Ausführen von Prozessen. Python bietet dafür die Bibliothek `concurrent.futures`. Über das Objekt `ProcessPoolExecutor` ist die Funktion `submit()` aufrufbar. Diese Funktion erwartet als Übergabeparameter den Prozess und die Parameter, die an den Prozess zu übergeben sind. Ihre Aufgabe ist die Aufstellung eines Future-Objekts, das die Ausführung des Prozesses darstellt [14].

### 5.2.2 Pandas

Pandas ist eine für Python bekannte Bibliothek. Ihre Funktionen sind die Analyse, Verarbeitung und Darstellung von Daten. Insbesondere bietet sie Funktionen und Datenstrukturen für Berechnungen mit numerischen Tabellen und Zeitreihen [5]. Pandas liest csv-Dateien mit `read_csv (filepath_or_buffer, sep=_NoDefault.no_default)` als DataFrame ein und verarbeitet diesen mit seinen eigenen zahlreichen Funktionen. Der Übergabeparameter `sep` erwartet zur Kennzeichnung einen String, mit dem die einzelnen Elemente in der csv-Datei, welche den zugehörigem Dateipfad `filepath_or_buffer` hat, separiert sind.

Series.unique() gibt eindeutige Werte einer panda-Series zurück als pd.Series[1].

Außerdem erstellt die Funktion pd.arrays.IntervalArray.from_arrays() aus zwei ( arrays) die linke und rechte Grenze eines Intervalls und gibt diese als panda-Datenstruktur `IntervalArray` zurück.

Pandas bietet zudem die Funktion `groupby()` an. Diese ordnet ein DataFrame mit Hilfe eines Mappers oder der Spalte einer panda-Series zu einer Gruppe zu.

Die Funktion `apply()` wendet eine Funktion entlang einer gewählten Achse eines DataFrames an. Das an die Funktion übergebene Objekt ist demnach eine panda-Series.

Eine weitere für dieses Projekt hilfreiches Funktion ist `pd.to_datetime()`, die eine Datumsangabe zu einem datetime-Objekt von Pandas umwandelt. Diese Datumsangabe kann z. B. vom Typ `datetime` sein. Diese Datenstruktur ist von der Biblitiothek `datetime` (s. Abschnitt 5.2.4).

Eine detailreichere Übersicht zu den genannten und weiteren Funktionen von Pandas stellt die Dokumentation von Pandas zur Verfügung [11].

[1] import pandas as pd

### 5.2.3 Soundfile

Soundfile schreibt mit write() und liest mit read() Audiosignale in eine und aus einer Audiodatei. Diese Bibliothek ist für die Soundverarbeitung innerhalb Pythons geeignet, denn sie wandelt Audiosignale in numpy-Arrays[2] um.

`Übergabeparameter:`

`file: Any` Angabe des Dateipfades

`data: Any` numpy-Array mit den Daten des Audiosignals

`samplerate: Any` Anzahl der Samples pro Periode

`start: int = 0` Startpunkt, ab dem das Audiosignal einzulesen ist. Angabe in Samples.

`stop: Any | None = None` Endpunkt, bis zu dem das Audiosignal einzulesen ist. Angabe in Samples.

Für `start` und `stop` gilt: Ein negativer Wert zählt die Samples startend vom Audiosignalende.

Write() bekommt die Parameter `file`, `data` und `samplerate` übergeben und liefert None zurück. read() bekommt die Parameter `file`, `start`, `stop` übergeben und gibt ein Tupel mit den Audiodaten als `ndarray` vom Typ `float64` und die Samplerate zurück.

### 5.2.4 datetime

Die Python-Bibliothek `datetime` ist zum Arbeiten mit Daten und Zeiten. Die Bibliothek stellt einige Objekte wie `datetime` oder `timedelta` zur Verfügung, die den Zugriff auf verschiedene Funktionen, Konstruktoren und Attribute ermöglicht. `datetime.timedelta` bekommt Werte für verschiedene Zeiteinheiten (die kleinste Zeiteinheit ist Millisekunden) und berechnet daraus eine Dauer um z. B. Zeitdifferenzen zwischen zwei `datetime`- oder `date`-Objekten zu berechnen.

```
class datetime.timedelta(days=0, seconds=0, microseconds=0,
   milliseconds=0, minutes=0, hours=0, weeks=0)
```

Zum Erstellen einer `datetime` oder einer `date` gibt es das Objekt datetime.datetime, welche Werte für Zeiteinheiten bis Mikrosekunden als kleinste Zeitheintheit bekommt.

```
class datetime.datetime(year, month, day, hour = 0, minute = 0,
   second= 0, microsecond = 0, tzinfo = None, *, fold = 0)
```

Zum Aufrufen der aktuellen Zeit ist datetime.now() verwendbar. Um die aktuelle Zeit in einer spezifischen Format als String zu speichern, ist die Funktion strftime() zu nutzen. Beispielsweise erstellt der Befehl

```
time.datetime.now().strftime("%Y%m%d")
```

[2] eine weitere Datenstruktur aus der Bibliothek numpy für Python

ein Datum mit der Angabe von Jahr, Monat und Tag. Die Angabe von %Y, %m und %d repräsentieren die unterschiedlichen Zeiteinheiten. Die Informationen über die unterschiedlichen Bezeichnungen sind der Tabelle aus deren Dokumentation zu entnehmen [12].

`strptime()` bewirkt das Gegenteil. Diese Funktion nimmt ein Datum in einer bestimmten Formatierung als ersten Übergabeparameter entgegen. Der zweite Übergabeparameter teilt der Funktion mit, in welchem Format das Datum gespeichert ist, damit die Funktion daraus die Werte der einzelnen Zeiteinheiten interpretieren und auslesen kann [12].

### 5.2.5 ALSA

ALSA ist eine unter Linux integrierte Soundsystemarchitektur. Sie besteht aus Linux-Kernelmodulen und betreibt Soundkarten. Mit den Funktionen von ALSA ist es möglich, Audio aufzunehmen und abzuspielen [10]. Mehr dazu in Abschnit /refsubsubsection:sox.

### 5.2.6 glob

Die Python-Bibltiothek glob bietet die Funktion glob(pathname, [...]).

Damit sind alle Pfadnamen, die einem vorgegebenen Muster entsprechen, auffindbar. Das Muster gestaltet sich nach den Vorgaben der Unix-Shell für Pfadangaben. Die Reihenfolge der gefundenen Pfade ist zufällig.

Platzhalter für variable (Pfad-)Angaben sind *****, **?**, **[]**. Letzteres ist Platzhalter für Zeichenbereiche [13].

### 5.2.7 Sox

Sox ist ein Audiotool zur Verarbeitung von Audios mit Filtern und Soundeffekten während der Konvertierung.

Mithilfe des Effekts *silence* verwirft Sox Audioabschnitte, in denen Stille herrscht, schon während der Konvertierung. Die Synopsis, der Befehle für die Audioaufzeichnung, ist wie folgt aufgebaut:

sox [global-options] [format-options] infile1 [[format-options] infile2] ... [format-options] outfile [effect [effect-options]] ...

rec [global-options] [format-options] outfile [effect [effect-options]] ...

Beide Befehlen zeichnen Audio auf.

Die für dieses Projekt relevanten Operationen sind:

**-t** Festlegung, mit welchem Audiogerät und Audioaufnahmesoftware Sox aufnimmt

**-r** Angabe der Samplerate, welche für die Anzahl an Samples pro Periode steht

**-b** Anzahl der Bits, mit denen ein Sample codiert ist

**-e** Audiodekodierungstyp. Mit der Angabe von **signed-integer** ist festgelegt, dass die PCM[3] -Daten als vorzeichenbehaftete Ganzzahlen zu speichern sind. In der Regel kombiniert man diese Angabe mit einer 16- oder 24-Bit-Codierungsgröße. Hierbei steht der Wert null für die minimale Signalleistung.

**silence** Effekt zum Trimmen von Stille während der Aufnahme. Die Synopse lautet:

```
above−periods [duration threshold[d|%] [below−periods duration threshold[d|%]]
```

Listing 1: Synopse für den Befehl von Stille während der Konvertierung

Die relevanten Operationen von **silence** sind:

**above-periods** Audio ist am Anfang der Audio zu trimmen. Der eingestellte Wert legt fest, wie oft am Anfang Stille zu trimmen ist. Bei einem Wert von null ist das Trimmen ausgeschaltet, bei eins passiert es nur einmal. Ist der Wert größer als eins, schneidet sox mehrere Stellen mit Stille ab Audioanfang raus.

**duration** Zeitangabe, wie lange Stille in einem Audiosegment kontinuierlich herrschen muss, damit Sox dieses entfernt.

**threshold** Schwellwert für die Lautstärke. Ton unter diesem Wert ist als Stille definiert.

**below-periods** Aufnahme ist am Ende der Audio zu trimmen. Der eingestellte Wert legt fest, wie oft am Ende Stille zu trimmen ist. Die Einstellung des Wertes funktioniert genauso wie bei **above-periods** nur rückwärts beginnend bei Audioende.

**: newfile : restart** Dieser Befehl bewirkt einen Ketteneffekt durch den Operator **:** . Mit **newfile** erstellt Sox aus der durch den vorherigen Effekt verarbeiteten Audio eine neue Audiodatei. Jede neu erstellte Audiodatei bekommt am Dateinamensende eine eindeutige Nummer angehängt. Danach springt Sox mit **restart** zum ersten Effektkettenglied. Der Prozess wiederholt sich.

Beispielsweise kann ein Befehl für das Trimmen von Audio während der Aufzeichnung wie folgt geschrieben sein:

```
−t alsa hw:2,0 −r 48000 −b 16 −e signed−integer "song.wav" silence 1 0.50t 0.1% 1 2.0 0.1% :
```

Listing 2: Besipiel eines SOX-Aufrufs mit Trimmen von Stille während der Konvertierung

Dieser Befehlszeile nimmt als default-Gerät ein extern eingestecktes Mikrofon mit der Kartennummer 2 und der Gerätenummer 0, gekennzeichnet durch hw:2,0, und als Soundsystem **alsa**. Die von **alsa** erkannten Geräte mit den jeweiligen Nummerierungen von Karte und Gerät sind unter dem Befehl arecord −l zu finden.

```
card 2: Device [USB PnP Sound Device], device 0: USB Audio [USB Audio]
```

### 5.2.8 gps

Für Python gibt es die Bibilothek gps

- Einlesen der Daten, die das GPS Gerät ermittelt und an den Computer sendet - Mit den Funktionen latitude, Longitude und Uhrzeit auslesen - funktion zum Prüfen, ob das Gerät erkannt wurde oder so

[3] Pulse Code Modulation

Die Einbindung der Bibliothek und Datenauslesung des GPS-Geräts findet in Abschnitt ... statt.

### 5.2.9 Piso

Piso unterstützt die Berechnung von Intervallklassen in Pandas für Mengenoperationen wie Vereinigungs- oder Schnittmenge.

Die Funktion piso.register_accessors() ermöglicht Piso den Zugriff auf pd.arrays.IntervalArray.

union Die Zusammenführung der Intervalle passiert durch die Funktion union, die bekommt ... übergeben und verbindet Intervalle mit gemeinsamen Schnittstellen.

## 5.3 Initialisierung

Das Modell 5.1 zeigt, dass die Initialisierung von main.py in drei zusammenfassende Schritte unterteilt ist:

1. Start des Programms über einen Shell-Befehl
2. Einlesen der Startargumente und konfigurieren
3. Start der drei parallelen Prozesse mit dem `ProcessPoolExecutor()` aus der //concurrent.futures-Bibliothek

Der Start des Programms über die Shell ist am Ende diese Kapitels in Abschnitt 6.2.1 erklärt.

Schritt 2 passiert drei Unterschritte.

Mit

parser = argparse.ArgumentParser(description="Start␣audio␣recording␣session")

kann main.py auf die Startargumente zugreifen. Danach ist festgelegt, wie mit diesen Startargumenten umzugehen ist. Ein Startargument ist `region`.

```
er.add_argument(
    "--region", default="COUNTRIES/world/", help="Path to folder where
        audiofiles, analyzed audiofiles and results are saved."


REGION = args.region{}
```

Hier ist festgelegt, dass der Default-Wert von `region` ein Dateipfad ist. Dieser Wert wird als Konfiguration in config.py gespeichert.

Wenn das Startargument `--config` mit dem Wert True übergeben ist, versucht das Programm die Koordinaten und die Zeit vom GPS-Gerät für die Postfilterung zu Nutzen. Was die Postfilterung ist, erklärt Abschnitt 6.1.1. Da das Gerät aber nicht sofort und überall Koordinaten findet, wartet die Funktion `getCoordinates()` nach diesen solange, bis der*die Anwender*in diesen Vorgang abbricht oder das GPS-Gerät die Koordinaten gefunden hat. Bricht der*die Anwedner*in das Programm vorher ab, fragt jenes ihn*sie, ob es die Audiosession dennoch fortgeführt werden soll oder abbrechen

soll. Hierfür ist der*die Anwender*in zur Eingabe aufgefordert. *y* steht für *yes* und *n* für *no*. Hat der*die Anwender*in die Startargumente `--lat`, `--lon` und `--week` übergeben, nutzt das Programm diese, ansonsten nutzt es die Default-Werte, die alle minus eins sind und somit nutzt das Modell keine Koordinaten für die Vorhersage. Sollte das GPS-Gerät die Zeit nicht finden, wird die Offline-Zeit genutzt. Bei Diskrepanz mit der Echtzeituhr ergibt sich ein Zeitfehler, der bei der Auswertung nach der Audiosession zu beachten ist.

`def getCoordinates()`

Zum Speichern der Config-Datei ist in main.py die Funktion `saveConfigsAsJSON( configs:str, cfpath:str)` definiert. Diese erwartet für den Parameter `configs` das Dictionary mit den Konfigurationen, aufrufbar über die Funktion `getConfigs()` von `config.py`, und den Dateinamen der JSON-Datei, in den sie das Dictionary schreiben soll.

```
def saveConfigsAsJSON(configs: Dict, cfile_path: str):
    configs_json = json.dumps(configs)
    with open(cfile_path, 'w') as cfp:
        cfp.write(configs_json)

configs = cfg.getConfig()
cfname = datetime.now().strftime("%Y%m%d") + '.json' #
    configs_filename
cfpath = os.path.join('configs_files', cfname)
saveConfigsAsJSON(configs, cfpath)
```

Schritt drei ist der Aufruf aller drei Parallelprozesse.

```
  with concurrent.futures.ProcessPoolExecutor() as executor:
    p2 = executor.submit(analyze, cfpath)
    p1 = executor.submit(rec, cfg.DURATION_HOURS, cfg.
        DURATION_MINUTES, cfg.AUDIO_PATH, cfg.AUDIO_TYPE)
    p3 = executor.submit(slice_audio, hours = cfg.DURATION_HOURS,
        minutes = cfg.DURATION_MINUTES, apath = cfg.AUDIO_PATH, aapath
         = cfg.ANALYZED_bei seinemAUDIO_PATH, atype = cfg.AUDIO_TYPE)
```

## 5.4 Aufnahme

### 5.4.1 Programmfunktionen und -parameter sowie Sox-Aufruf

Für die Aufnahme ruft main.py die Funktion `rec()` auf.

Die Werte der übergebenen Parameter `audio_path` und `atype` initialisieren die Umgebungsvariablen `audio_path` und `audio_type`. Diese nutzt Sox (s. Abschnitt 5.2.7) bei seinem Aufruf.

```
sox -t alsa hw:2,0 -r 48000 "$audio_path$(date␣+"%YY%mM%dD%Hh%Mm%Ss%3Nms_").$au
```

```
sox -t alsa hw:2,0 -r 48000 "$audio_path$(date +"%YY%mM%dD%Hh%Mm%Ss%3
    Nms_").$audio_type" silence 1 0.50t 0.1% 1 2.0 0.1% : newfile :
    restart
```

Der Datumsstempel ist im Dateinamen festgehalten, um später wie in Abschnitt 5.5.2 erwähnt, einen Datumsstempel in der Ergebnistabelle hinzuzufügen.

Umgeben ist der Aufruf der Funktion mit dem Zeitbegrenzer.

```
t_end = time.time() + hours * 3600 + minutes * 60
while time.time() < t_end:
    # rufe sox auf
```

### 5.4.2 Mikrofonrauschen und Schwellwertanpassung

Der gewählte `threshold` soll den Schwellwert für Stille festlegen. Da analoge Audio immer mit einem Rauschen durch das Mikrofon unterlegt ist, ist der Schwellwert mit einer bestimmten Toleranz zu wählen. Diese kann der*die Anwender*in selbst einstellen, da jedes Mikrofon durch seine Eigenschaften unterschiedlich starkes Rauschen verursacht. Für das hier gewählte Mikrofon ist durch Testversuche der `threshold` von `0,062` gewählt, das der Rauschlautstärke des Mikrofons näherungsweise entspricht. Schwellwert soll also die Lautstärke des Rauschens selbst sein. Zur Bestimmung dieses Wertes ist das Rauschen aufzunehmen und daraus ein Mittelwert zu ermitteln. Es bedarf aber noch einige Versuche diesen Wert anzupassen bis das Ergebnis zufriedenstellend ist. Da auch das Rauschen kein konstanter Wert ist, sollte eine ausreichende Toleranz über die durchschnittliche Rauschlautstärke gewählt sein.

## 5.5 Analyse

### 5.5.1 Vorstellung des BirdNet-Analyzers

Für die Erkennung von Vogelgesang nutzt die Software das Modell vom BirdNET-Analyzer [4] sowie weitere Programme und Funktionen von dessen API. Die erste Gitversion heißt BirdNET, ist aber veraltet. Die neue Version ist unter dem Namen BirdNET-Analyzer herausgegeben. Die Entwickler geben an, diese Version weiterhin zu updaten.

Wie BirdNET grob funktioniert, beschreiben die folgenden Zeilen. Für genauere Information über BirdNET, wie es zu installieren und anzuwenden ist, ist auf das Git-Repository [3] verwiesen.

Über verschiedene Terminalbefehle sind verschiedene Programme und Funktionen vom BirdNET-Analyzer aufrufbar. Zur Analyse von Audiodateien ist das Pythonskript analyze.py mit den gewünschten Einstellungen übergeben als Startargumente aufzurufen.

Informationen zu allen Startargumenten ist in der ReadMe im Git von BirdNET zu finden.

Nach der Analyse der Audiodatei(n) gibt BirdNET die Default- oder vom Anwendenden gewünschte Ergebnistabelle zurück. Je nach Ergebnistabelle können bei jeder erkannten Vogelart in einem bestimmten Zeitabschnitt in der Audio verschiedene andere Informationen hinterlegt sein, die zur Weiterverarbeitung nutzbar sind.

### 5.5.2 Änderungen in BirdNET

Das Schaubild 5.1 zeigt, dass die Funktion `analyze()` aus `main.py` die Schnittstelle zum BirdNET-Analyzer ist.

Bei Aufruf von `analyze()`, ruft diese wiederum per Shell-Befehl das Programm analyze.py von BirdNET auf. Statt wie vorher in Abschnitt 5.5.1 gezeigt, die Startargumente an `analyze.py` zu übergeben, bekommt main.py diese Startargumente und lädt sie als JSON-Datei in den Ordner *configs-files*. Diese Änderungen hat den Vorteil, dass die Einstellungen wiederverwendbar sind. Auf die genaueren Umsetzung geht der Abschnitt 6.1 ein.

`analyze()` bekommt den Dateinamen der JSON-Datei übergeben. Mittels des Befehls `os.system()` ruft sie `analyze.py` auf und übergibt als Startargument diesen Dateinamen.

```
_name__ == "__main__":
# Freeze support for executable
freeze_support()

parser = argparse.ArgumentParser(description="Analyze audio files with
    BirdNET")

parser.add_argument(
    "--configs",
    type =str,
    default=None,
    help="Path to configs. Defaults to None. If set, --configs is ignored.
        ",
)

args = parser.parse_args()
cfile_path: str = ""
print(type(cfile_path))
configs_json = ""
if args.configs is not None :
    cfile_path = str(args.configs)
    with open(cfile_path, 'r', encoding='utf-8') as cfp:
        configs_json = json.load(cfp)

    cfg.setConfig(configs_json)
```

`analyze.py` kann mit der Angabe des Dateinamens auf die gewünschte Konfigurationsdatei zurückgreifen.

Da es sich bei diesem Vorgehen um eine mehrstündige Aufnahme handelt, sind für eine besseren Auswertung der Ergebnisse zu den Ergebnistabellen csv, table und R zwei Datumsstempel im UTC-Format beigefügt.

### 5.5.3 analyze.py

Beim den Aufruf von analyze.py sind

### 5.5.4 Effizienzsteigerung

Eine Anforderung des Projekts ist, dass der mobile Computer mit der Stromversorgung der Powerbank für mindestens 24 Stunden auskommt. Zudem soll die Analyse möglichst in Echtzeit und zeiteffizient sein. Um diese Anforderungen zu optimieren, gibt es Tests zu unterschiedlichen Einstellungen und Szenarien, die diese Eigenschaften beeinflussen können. Dazu gehört zum einen die Batchsize. Diese hat eine Effekt auf die Performance des Modells sowie auf die Trainingszeit. Der größte Vorteil einer hohen Batchsize ist, dass Hardwarebeschleuniger wie GPU damit bessere Performance erbringen können sowie einen positiven Effekt auf die Zeit haben.

# 6 Erste Versuchsdurchführungen

Zwar ist der Jetson mit seiner CPU relativ

Zwar hat der Jetson Nano mit seiner GPU eine hoher Leistungsfähigkeit, dennoch sollen die hier vorgenommenen Test die Effizienz steigern. Dafür ist zum einen die optimale Batchsize bestimmt worden, die mmmm vereint.

default-Wert von Batchsize

sensitivity und overlap haben Auswirkungen auf die Rechenleistung

-> Test und deren Ergebnisse

Auch `overlap` hat einen Einfluss auf die Zeitperformance des Modells. Da das Programm eine Audio immer in Drei-Sekunden-Segmente schneidet, analysiert das Netz auch immer nur diese drei Sekunden. Das hat zum Nachteil, dass eventuell das Programm mitten im Vogelgesang die Aufnahme trennt. Das kann wiederrum negative Effekte auf die Vorhersagbarkeit haben. Mit Overlap ist beeinflussbar, wie das Programm die Audio zuschneidet. Zwar kann das Netz immer nur 3 Sekunden auf einmal analysieren, aber diese 3-Sekunden-Intervall verschiebt sich auf der Audio immer nur um den gegebenen Overlap-Wert. Als Beispiel, ein `overlap` von 0 bewirkt, dass das Programm alle drei Sekunden einen Schnitt macht. Mit einem `overlap` von 1 startet das Modell die Analyse auch bei zwei Sekunden und analysiert dann in drei Sekundenschritten die Audio.

## 6.1 Startargumente und Konfigurationen

Wie schon in Abschnitt 5.5.2 erwähnt, bekommt nicht mehr analyze.py von BirdNET die Startargumente, sondern das Hauptprogramm main.py. Diese Startargumente sind in der Konfigurationsdatei `config.py` von BirdNET hinterlegt.

Neben dieser Änderung gibt es weitere Änderungen bei den Startargumenten selbst. Das Programm nimmt einige zusätzliche Startargumente an, andere sind aus der Liste entfernt worden. Diese Änderungen sind in diesem Abschnitt erläutert.

Die hinzugefügten Startargumente sind `--hour`, `--minute`, `--region`, `--atype`, `--coordinates`, `--trim`, `--threshold`, `--rtpye2`, `--delete`

`--hour` gibt die Stunden an, die aufzuzeichnen sind. Default-Wert ist null.

`--minute` gibt die Minuten an, die aufzuzeichnen sind. Default-Wert ist 20.

`--atype` legt das gewünschte Audioformat fest. Dazu sei angemerkt, dass für jedes Audioformat ein sogenannter *handler* zu installieren ist [**?** ].

`--region` gibt an, in welchem Ordner Audioaufnahmen, Ergebnisse und die daraus zugeschnittenen Audios zu speichern sind. Dieser Ornder hat dabei immer zwei Unterordner. In den Unterornder
`audiofiles` speichert `rec()` die aufgenommenen Audios und nach der Analyse verschiebt `analyze()` die Audios in den Unterordner `analyzed_audiofiles`, wobei jede Audiodatei nochmal in einen eigenen Ordner zusammen mit den dazugehörigen Ergebnissen und den zugeschnittenen Audios gespeichert ist. Der Default-Wert von `--region` ist `COUNTRIES/world`. Dieser Pfad befindet sich auch schon im Projektverzeichnis.

Der Variablenname `region` soll den*die Anwender*in dazu motivieren, den Ordner nach dem Standortrt zu benennen, wo er*sie die Aufnahme durchgeführt hat. Optimalerweise sollte der Ordner dem Ordner `COUNTIRES` untergeordnet sein. Damit entwickelt sich eine strukturierte Übersicht über alle Audiosessions, die an verschiedenen Standorten entstanden sind.

`--coordinates` ist eine boolesche Variable, die bei dem Wert True dem Programm mitteilt, dass es für die Analyse die Koordinaten vom GPS-Gerät nutzen soll.

`--threshold` hat einen Default-Wert von 0.062. Dieser definiert den Schwellenwert für den Silence-Effekt wie in Abschnitt 5.2.7 erklärt. Da dieser Wert aber auf das hier genutzte Mikrofon abgestimmt ist, steht dem*der Anwender*in offen, diesen Wert für sein*ihr Mikrofon oder die eigenen Präferenzen anzupassen. So ist es auch möglich mit einem größeren Schwellenwert den Aufnahmeradius einzuschränken und mit einem kleineren Radius zu erweitern.

`--trim` legt fest, was die maximale Audiolänge einer einzelnen Aufnahme sein darf, wenn der Silence-Effekt nicht vorhehr selbst die Audio zuschneidet. Damit ist bei einem zu kleinen Threshold für Silence verhinderbar, dass eine Audiodatei zu lang wird oder gar so lang wie die gesamte Audiosession ist.

`--delete` ist Default auf False, was heißt, dass das Programm die Originalaufnahmen nach der Auswertung nicht löscht. Durch setzen auf True ist das Gegenteil der Fall. Dieses Startargument erfüllt die Anforderungen für die Einhaltung des Gesetztes welches die Vertraulichkeit des Wortes bewahrt.

Entfernte Startargumente sind `--i` und `--o`. `--i` gab an, in welchem Ordner sich die zu analysierende Audiodatei befindet. `--o` gab an, in welchen Ordner die Ergebnisse der analysierten Audio zu speichern sind.

Zu den hinzugefügten Konfigurationen gehören AUDIO_PATH und ANALYZED_AUDIO_PATH
AUDIO_PATH speichert die Pfadangaben *audiofiles/* und ANALYZED_AUDIO_PATH *analyzed_audiofiles/*. Beide Variablen beinhalten also den Pfadnamen des Unterordners von `--region`.

`-rtype` bestimmt den Typ der primären Ergebnistabelle. Zu jedem analysierten Au-

diosegment erstellt das Programm eine Tabelle, die standardmäßig als CSV formatiert ist. Optional speichert das Programm die Tabelle stattdessen als Plain-Text-Tabelle.

`-rtype2` ist ein optionales Startargument, das es ermöglicht, zusätzlich zur CSV- oder Plain-Text-Tabelle eine weitere Tabelle für unterschiedliche Auswertungsarten zu erzeugen. Bei Angabe von `r`, `kaleidoscope` oder `audacity` speichert das Programm die formatierte Tabelle in den Formaten *.r*, *.csv* oder *.txt*, die entsprechend von den Programmen R, Kaleidoscope oder Audacity gelesen werden können.

`--configs` ermöglicht die Wiederverwendung von alten Konfigurationen. Das Programm speichert bei jedem Start die dazu festgelegten Konfigurationen im Ordner `configs_files`. Der Dateiname ist das aktuelle Datum. Mit der Angabe des Dateinamens der Konfigurationsdatei, welche dem Muster `YYYYMMDD.json`[4] entspricht, sind alle Konfigurationen von der gewählten Konfigurationsdatei für den neuen Start festgelegt. Trotz Angabe von weitere Startargumenten, ignoriert das Programm jene.

### 6.1.1 Species List

Ein Ziel dieses Projektes ist die Differenzierung zwischen Vogel und Nicht-Vogel. Dabei sind alle Audioabschnitte, in denen das Netz ein Vogel erkannt hat, gesondert abzuspeichern und alle Nicht-Vogel zu ignorieren oder sogar zu verwerfen.

Die Liste aller Vogelarten basiert auf der eBird checklist frequency [**?** ]. Dies ist ein Vogelglossar mit zahlreichen Vogelarten. Zu jeder Vogelart ist mit Angabe von Latitude, Longitude und Kalenderwoche die Wahrscheinlichkeit dessen Auftretens in bestimmten Breiten- und Längengraden zu einer bestimmten Zeit im Jahr hinterlegt. Das neuronale Netz ist mit diesen Daten trainiert. Da das Netz aber auch erkennen sollte, wann es sich bei bestimmten Geräuschen um Nicht-Vogelarten handelt, sind in der Liste mit den Klassenvorhersagen auch Nicht-Vogel wie Elektrowerkzeuge (Power tools) oder menschliche Stimmen (Human vocal). Dies dient beim Training des Netzes dazu, verstärkt den Unterschied zwischen Vogelstimmen und Nicht-Vogelstimmen zu erkennen.

Da also in der Ergebnistabelle auch Nicht-Vogelarten stehen können, die das Programm nach der Analyse zum Zuschneiden der Audios nutzt, sollte das Programm die Nicht-Vogelarten herausfiltern.

Umgesetzt ist das mit einem Postfilter. BirdNET hat ein Startargument `--slist`. Dieses Argument erwartet einen Pfad zu einer Textdatei, die zur Filterung der Ergebnisse nur die Vogelarten enthält, die am Ende tatsächlich in der Ergebnisstabelle stehen dürfen. Für dieses Projekt ist die Liste mit den Labels (in der Version 2.4), welche im Ordner `labels/V2.4` vorliegt, überarbeitet. Alle Nicht-Vogel sind aus der Liste manuell entfernt und diese Liste ist im Ordner [breaklines=true]multiply_species_lists/filtered_non_birds mit dem Dateinamen `filtered_non_birds.txt` gespeichert. Der Pfad zu dieser Liste ist als default-Wert vom Argument `--slist`. Diese Datei enthält alle Labels auf Englisch. Somit stehen, ohne die Erstellung einer eigenen Speziesliste mit der gewünschten Sprache, die Vogelarten in der Ergebnistabelle auf Englisch. Durch das Erstellen einer eigenen Speziesliste mit einem Standortfilter erstellt das Programm `species.py` von BirdNET eine gefilterte Liste.

Eine benutzedefinierte Erstellung einer Speciesliste ermöglicht das Programm ebenfalls. Zur Erstellung einer Speziesliste bietet BirdNET das Python-Programm `species.py`

[4] Y=Jahr (Year), M=Monat (Month), D=Tag (Day)

. Unter Angabe von Longitude, Latitude und Woche generiert das Programm einen Standortfilter. Dieser Standortfilter ist durch den Wert `sf_thresh` beeinflusst. Dieser Wert ist die Schwelle für die Mindesthäufigkeit einer Vogelart, die am gegebenen Standort vorkommen muss. Vogelarten, dessen Vorkommensswahrscheinlichkeit größer als der Schwellwert sind, stehen dann in der Speziesliste. Die Wahrscheinlichkeiten mit den Daten aus Longitude, Latitude und Woche sind im Vogelglossar hinterlegt. Durch das Erstellen von einer nach standortgefilterten Speziesliste sind auch die Nicht-Vogelarten herausgefiltert, denn diese sind im Vogelglossar nicht hinterlegt.

## 6.2 Weiterverarbeitung

Nach der Analyse kommt das *Slicen*. Die Ergebnistabelle besagt, in welchen Zeitintervallen das Netz einen Vogel erkannt hat. `read_csv()` aus der Pandas-Bibliothek liest die Tabelle und gibt sie als DataFrame zurück.

Standardmäßig ist die Tabelle als CSV für die Weiterverarbeitung gewählt. Diese enthält die Start- und Endzeiten von jedem Drei-Sekunden-Segment, welcher mit einer Vogelart klassifiziert ist.

Die Funktion `slice()` führt dabei sechs Arbeitsschritte aus.

1. neue Ordner laden
2. aus den Ordnern Tabelle und Audio laden
3. Tabelle nach Vogelart gruppieren
4. Zeitintervalle zu jeder Vogelart erstellen
5. Vereinigungsmenge der Zeitintervalle zu den einzelnen Vogeln berechnen
6. Audiofiles einlesen, daraus das gegebene Zeitintervall herausschneiden, Datei entsprechend benennen und im selben Ordner wieder ablegen.

In Schritt 1 passieren folgende UNterschritte, wenn der Timer abgelaufen ist

1. **while** die Audiosession noch läuft, führe den nächsten Schritt aus
2. **if** die vom Anwendenden gesetzte Zeit ist vorbei
3. warte nochmal einige Sekunden, falls BirdNET noch Audiofiles analysiert.
4. **if** keine Audios mehr im Ornder *audiofiles* sind, gehe zum nächsten Schritt
5. **if** keine analysierten Audios noch zu verarbeiten, beende die while-Schleife aus Schritt 1

Dieses Verfahren soll sicherstellen, dass nach Ablauf des Timers auch alle noch zu analysierenden Audios fertig verarbeitet sind.

Das Laden der während der Audiossesion neu generierten Ordner (s. Schritt 1) realisiert sich durch die Berechnung der Differenz zwischen den alten und neuen Ordner.

```
analyzed_audio_folders = glob.glob(cfg.ANALYZED_AUDIO_PATH + '/*')
new_folders = set(analyzed_audio_folders).difference(set(
    old_folders))
print(f"new_folders: {new_folders}")
```

Aus den geladenen Ordnern ruft das Programm den Pfad der Ergebnistabelle und der Audiodatei mit der `glob()`-Funktion (s. Abschnitt 5.2.6) auf. Mit dem Pfad zur Tabelle kann read_csv() daraus ein DataFrame erstellen (s. Abschnitt 5.2.2). Danach gruppiert `groupby()` das DataFrame nach den erkannten Vogelarten und `apply()` sortiert zu jeder Vogelart das entsprechende Intervall, welches die Funktion `from_arrays()` zum Datenobjekt `IntervalArray` aus den Einträgen in den Elementen der Tabelle konstruiert hat. Die Funktion `cut_audios` bekommt einen erkannten Vogel, die dazugehörigen Zeitintervalle und den Pfad zu Audio.

```
try:
    for this_folder in new_folders:
        table_path = glob.glob(this_folder + "/*.csv")[0]
        table = pd.read_csv(table_path, sep=',')
        audio_path = glob.glob(this_folder + "/*.wav")[0]
        print(f"Audio_Path: {audio_path}")
        rec_birds = table['Common name'][:].unique()
        time_intervals = (table.groupby('Common name').apply(
            lambda table : pd.arrays.IntervalArray.from_arrays(
                table['Start (s)'],
                table['End (s)'],
                closed = 'right'
            )))
        for bird in rec_birds:
            cut_audio(bird, time_intervals[bird], audio_path)
```

Listing 3: Einlesen der Ergebnistabelle und des Audiodateinamens

In dieser Funktion findet das eigentliche *slicen* statt. Soundfile liest die Audiodatei ein, dabei immer nur das Segment im aktuellen Intervall. Zur Berechnung der Grenzen dieses Audiosegments sei kurz erklärt, dass ein digitales Audiossignal immer aus mehreren Samples (Proben) besteht.

Die Umwandlung eines analogen in ein digitales Signals realisiert sich durch Quantisierung und Abstatung, wodurch auch immer Informationen verloren gehen.

Da hier grundsätzlich das Netz nur Audiosignale mit 48 000 Samples pro Sekunde (oder auch Hertz) analysiert, sind die Audiosignale auch in 48 000 Hz aufgenommen. Zur besseren Übersicht der Ordnerstruktur und Dateinamen bekommen die zugeschnittenen Audios eine Kennung mit der darin erkannten Vogelart sowie einen Datums- und Zeitstempel.

```
def cut_audio(bird, interval_array, audio_path):

for interval in interval_array:
    samplerate = 48000
    begin = int(interval.left * samplerate)
    end = int(interval.right *samplerate)
    data, samplerate = sf.read(file=audio_path, start = begin, stop=end)
    sf.write(os.path.join(os.path.dirname(audio_path), f"rec_{bird}_{begin}_{end}.wav"), da
```

Weiter geht es mit dem Code aus 6.2.

```
if table_path.rsplit(".", 1)[-1].lower() in ['csv']:
```

```
                old_folders.append(this_folder)

                print(f"Append this_folder -> {this_folder}")
        except Exception as ex:
            time.sleep(3)
            print("Warte auf Ergebnisse der Analyse")

    return ("Done sorting")
```

Wenn im Pfad zur Ergebnistabelle auch tatsächlich eine Ergebnistabelle vorhanden war, zählt dieser Ordner zu den *alten Ordnern*.

### 6.2.1 Start von BirdNET per Remote-Shell

test

# 7 Versuchsdurchführung

die Versuchsdurchführungen fanden an drei Orten statt auf dem Balkon in Unterilp in Heiligenhaus Dachterrasse vom Stundentenwohnheim des Campus Velbert/Heiligenhaus sowie die Dachterrasse vom Campus selbst

insgesamt gibt es 82? h Daten, die das Programm in den Versuchsdurchführung aufgenommen und analysiert hat

Im folgenden ist beschrieben wie die einzelnen Versuche durchgeführt wurden und welche Ergebnisse dabei herauskommen,. Diese sind im Anschluss jeweils validiert. Am Ende gibt es ein Ranking, welche Top fünf Vögel bei einer bestimmten Wahrscheinlichkeit in der Klassifikation (wie umschreibt man confidence?) vorhergesagt wurden.

## 7.1 Realer Aufbau

Bilder vom Versuchsaufbau mit der Hardware -< nur den Inhalte der Horchbox erklären

mobiler Computer liegt in der Horchbux GPS-Tracker und Mikrofon per USB-Kabel an die USB-Port angeschlossen Die Kabel sind durch die abgedichteten Löcher des Box durchgeführt, um die Box zu verschließen, aber die kabel nach draußen führen zu können.

Für diesen hier dargstellten Aufgbau sei noch folgende zu wissen. Mikrofon ist an einem verlängerten USB-Kabel (wie lang?) angeschlossen. Zusätzlich liegt zwischen Verlängerungskabel und Mikrofonklingenanschluss ein Adapter (Klingenanschluss zu USB). Dieses sollte einen möglichst weiten Abstand zu dem hier genutzen Nividia Jetson (Nochmal bibiligraphy auf den Jetson referenzieren?) haben, da es sonst zu Interferenzstörungen kommt.

Zum Start ist der Host-Computer mit dem mobilen Computer per Hotspot vom mobilen Computer verbunden. Die Anwendung bekommt das den cmd-Befehl mit den entsprechenden Starargumenten, wordurch die Audiosession gestartet wird.

(Empfehlung aussprechen es über eine tmux shell zu starten?)

Bei jedem Test unterscheiden sich einige Startargumente zu den vorherigen.

Vorbereitende Tests:

testen, wie gut es menschen erkennt testen, wie gut das Mikrofon aufnehmen kann

### 7.1.1 Menschenfilter - Test

mit einer Wahrscheinlichkeit unter 0.3 Kirchenglocke als Mensch erkannt. Das ist ein Problem, denn bei einem overlap größer 0, wo dann die Zeitintervalle der Vögel und der Kirchenglocke gleichzeitig zu hören sind, würde das Programm diese Vögel rauswerfen, obwohl das nicht nötig ist.

-> Beispiel in Wohnheim_Dachterrasse_100524 um 7 Uhr

Abbildung 7.1: Bild vom Versuchsaufbau und der Umgebung

## 7.2 Testversuch 1: Wohnheim Dachterrasse

Der erste Testversuch ist in mehreren Testversuche aufgeteilt, um die Analysezeit des Netzes im Multiprocessing zu testen, wenn verschiedene Werte für das Startargumente `-overlap` zu testen. Alle drei Teile des Testversuchs fanden auf der Dachterrasse des Wohnheims statt. Hier der Aufbau:

Insgesamt hat dieser ...(26h?) gedauert. Die einzelnen Testversuche haben ...(12?) Stunden, vier Stunden und zehn Stunden gedauert.

### 7.2.1 Testversuch 1.1

Einstellungen:

Auflistung oder Text oder Bild?

Eckdaten:

Datum: 10. 05. 24 Dauer: 12 h Ordner: COUNTRIES/Deutschland/Heiligenhaus/Wohnheim_Dachterrasse_100524 Koordinaten vom GPS: ja

Da das GPS-Gerät nicht den richtigen Tag bestimmen konnte, liegt das Datumsstempel für die Dateinamen um zwei Tag zurück. Jedoch stimmen die Uhrzeiten, weswegen es für die Validierung der Testergebnisse keine wirklichen Auswirkungen hat.

Abbildung 7.2: Mikrofon in einem Wasserflaschenkopf zum Schutz vor eventuellem Regen

Abbildung 7.3: weiteres Bild vom Versuchsaufbau mit Horchbox und GPS-Tracker

Wohnheim_Dachterasse_100524_02 > Analyze_Time.txt > data

```
1   Filename,Analyze Time
2   2024Y05M08D20h58m59s365999ms.wav,134.001533
3   2024Y05M08D20h48m56s454000ms.wav,131.872364
4   2024Y05M08D21h19m00s289999ms.wav,118.154796
5   2024Y05M08D21h08m59s433999ms.wav,120.254398
6   2024Y05M08D21h29m00s349999ms.wav,114.294378
7   2024Y05M08D21h39m02s969998ms.wav,117.25121
8   2024Y05M08D21h59m04s933998ms.wav,117.778885
9   2024Y05M08D21h49m03s033998ms.wav,118.242182
10  2024Y05M08D22h09m04s997998ms.wav,116.466146
11  2024Y05M08D22h19m06s189997ms.wav,118.743902
12  2024Y05M08D22h29m06s249997ms.wav,119.110392
13  2024Y05M08D22h39m08s301997ms.wav,121.818926
14  2024Y05M08D22h59m09s637997ms.wav,120.00444
15  2024Y05M08D22h49m08s369997ms.wav,127.076995
16  2024Y05M08D23h19m11s889996ms.wav,120.63682
17  2024Y05M08D23h09m09s697996ms.wav,120.206169
18  2024Y05M08D23h29m11s945996ms.wav,121.984407
19  2024Y05M08D23h39m12s753996ms.wav,117.819149
20  2024Y05M08D23h59m12s885995ms.wav,113.97146
21  2024Y05M08D23h49m12s825995ms.wav,113.170885
22  2024Y05M09D00h19m13s013995ms.wav,110.101825
```

Abbildung 7.4: Analysezeit des testversuch Nr. 1.1 / Wohnheim Dachterrasse mit overlap von 0

### 7.2.2 Ergebnisse der Analysezeit

Abbildung .. zeigt einen Ausschnitt der Tabelle mit der berechneten Analysezeit mit der der Jetson

Die Analysezeit ist hier sogar länger als die Dauer der Audioaufnahme selbst. Insgesamt hat die Session ... Sekunden gedaurt.

### 7.2.3 Testversuch 1.2

Eckdaten: Datum: 11. 05. 24 Dauer: 4h Koordinaten: True Dauer pro Audio: 10 Minuten Overlap: 0 Mindestwahrscheinlichkeit: 0.3

Tabelle 7.1: Testversuch 1.2

| | |
|---|---|
| Datum | 11.05.24 |
| Ordner | COUNTRIES/Deutschland/Heiligenhaus/ Wohnheim_Dachterrasse_110524 |
| Dauer | 4 h |
| Koordinaten | –lon: 51.327644614<br>–lat: 6.96835278<br>–coordinates True<br>Koordinaten wurden gefunden |
| Dauer pro Audio | 10:00 |
| Overlap | 0 (default) |
| Mindestwahrscheinlichkeit | 0.1 (default) |

Tabelle 7.2: Testversuch 1.2

| | |
|---|---|
| Datum | 11.05.24 |
| Ordner | COUNTRIES/Deutschland/Heiligenhaus/ Wohnheim_Dachterrasse_110524 |
| Dauer | 4,0 Stunden |
| Koordinaten | –lon: 51.327644614<br>–lat: 6.96835278<br>–coordinates True<br>Koordinaten wurden gefunden |
| Dauer pro Audio | 10:00 |
| Overlap | 0 (default) |
| Mindestwahrscheinlichkeit | 0.1 (default) |

Tabelle 7.3: Testversuch 1.2

| | |
|---|---|
| Datum | 11.05.24 |
| Ordner | COUNTRIES/Deutschland/Heiligenhaus/ Wohnheim_Dachterrasse_110524 |
| Dauer | 4,0 Stunden |
| Koordinaten | –lon: 51.327644614<br>–lat: 6.96835278<br>–coordinates True<br>Koordinaten wurden gefunden |
| Dauer pro Audio | 10:00 |
| Overlap | 0 (default) |
| Mindestwahrscheinlichkeit | 0.1 (default) |

Wohnheim_Dachterasse_110524_abends > Analyze_Time.txt > data

```
1   Filename,Analyze Time
2   2024Y05M08D17h46m27s487999ms.wav,170.917351
3   2024Y05M08D17h36m27s400000ms.wav,170.48718
4   2024Y05M08D18h06m28s911999ms.wav,159.294441
5   2024Y05M08D17h56m28s843999ms.wav,159.674018
6   2024Y05M08D18h16m28s991999ms.wav,159.807732
7   2024Y05M08D18h26m29s051999ms.wav,160.305924
8   2024Y05M08D18h46m29s175998ms.wav,159.435397
9   2024Y05M08D18h36m29s119998ms.wav,159.300225
10  2024Y05M08D18h56m29s243998ms.wav,160.033014
11  2024Y05M08D19h06m29s307998ms.wav,159.663362
12  2024Y05M08D19h16m29s371997ms.wav,159.410806
13  2024Y05M08D19h26m29s435997ms.wav,158.9835
14  2024Y05M08D19h46m29s563997ms.wav,159.140517
15  2024Y05M08D19h36m29s495997ms.wav,159.677642
16  2024Y05M08D20h06m29s695996ms.wav,159.804724
17  2024Y05M08D19h56m29s631997ms.wav,159.258959
18  2024Y05M08D20h16m29s755996ms.wav,159.62537
19  2024Y05M08D20h26m29s819996ms.wav,158.907214
```

Abbildung 7.5: Enter Caption

Tabelle 7.4: 11.∼05.∼2024 08:30 Uhr - 8:40 Uhr

| Zeitintervall in Sek | Zeitintervall in mm:ss | BirdNET Vogelart | Confidence | Merlin ID |
|---|---|---|---|---|
| 63.0 | 01:03 | Tree Pipit | 0.6763 | - |
| 447.0 | 07:27 | Common Swift | 0.9929 | - |
| 465.0 | 07:45 | Common Swift | 0.7170 | European Robi |
| 474.0 | 07:54 | Common Chiffchaff | 0.7823 | Common Chiffc |
| 498.0 | 08:18 | Common Chiffchaff | 0.6213 | Common Chiffc |
| 507.0 | 08:27 | Common Chiffchaff | 0.5170 | Common Chiffc |
| 525.0 | 08:45 | Common Chiffchaff | 0.8636 | Common Chiffc |
| 537.0 | 08:57 | Common Chiffchaff | 0.8070 | Common Chiffc |

### 7.2.4 Ergebnisse der Analysezeit

### 7.2.5 Validierung

## 7.3 Testversuch 2 - Balkon in Unterilp

Abdeckung zum Schutz vor Wärmeeinstrahlung Mikrofon hängt über Geländer, um möglichst ohne Schallwände drumherum die Umgebung klar aufnehmen zu können

### 7.3.1 Ergebnisse der Analysezeit

## 7.4 Start der Software

-> Vorbereitung z.B. species-list anlegen -> Verbindung mit dem Jetson aufbauen -> Befehl an den Jetson schicken

## 7.5

vierundzwanzig Stunden später:

Was sind die Ergebnisse?

Abbildung 7.6: Ablageort der Box: Tisch auf dem Balkon


Abbildung 7.7: Plane als Sonnenschutz über die Box gelegt

Abbildung 7.8: Mikrofon hängt über das Balkongeländer, ist mit Tesafilm am Kabel fixiert

Abbildung 7.9: weiteres Bild vom Mirkofon aus einem anderen Blickwinkel

Balkon_Unterilp_120524 > Analyze_Time.txt > data

```
1   Filename,Analyze Time
2   2024Y05M09D11h55m38s427974ms.wav,617.353634
3   2024Y05M09D12h05m38s507974ms.wav,621.399627
4   2024Y05M09D12h25m38s623974ms.wav,614.873042
5   2024Y05M09D12h15m38s571974ms.wav,611.504772
6   2024Y05M09D12h45m38s763973ms.wav,612.52198
7   2024Y05M09D12h35m38s711974ms.wav,613.936031
8   2024Y05M09D12h55m38s851973ms.wav,614.550326
9   2024Y05M09D13h05m38s903973ms.wav,614.094758
10  2024Y05M09D13h25m39s043972ms.wav,613.641509
11  2024Y05M09D13h15m38s991973ms.wav,613.604359
12  2024Y05M09D13h45m39s179972ms.wav,614.73195
13  2024Y05M09D13h35m39s119972ms.wav,617.724255
```

Abbildung 7.10: Analysezeit des Testversuch Nr. 2 / Balkon Unterilp mit einem overlap von 2.5

# 8 Zusammenfassung und Ausblick

Die Zusammenfassung und der Ausblick schließen den Hauptteil der Ausarbeitung ab. Im Gegensatz zur einleitenden Übersicht sollte eine Zusammenfassung losgelöst von der tatsächlichen Gliederung der Arbeit verfasst sein und sich ganz auf die Zusammenfassung der Ergebnisse bzw. der im Rahmen der Dokumentation geschriebenen Inhalte beziehen.

Nach der Zusammenfasung erfolgt ein Ausblick. Dieser umfasst zum einen Themen, die im Rahmen der Beschränkungen der vorliegenden Arbeit nicht abschließend untersucht werden konnten, zum anderen die Anregung, neue Themen zu untersuchen, die sich im Verlauf der Bearbeitung als interessant herausgestellt haben.

Abstract: diese Projekt kann als Biodiversitätserfassungsinstrument dienen und ist ein Hilfsmittel für naturschutz, Schutz der Artenvielfalt und Sammlung von Informationen über das Weltgeschehen.

Biodiversitätserfassungsinstrument

Bedeutung des Projektes: Naturschutz, Schutz der Artenvielfalt, Informationen über das Weltgeschehen sammeln

# Literatur- und Quellenverzeichnis

[1] Clippy. EM272Z1 Mono Microphone. Mono-Mikrofon. Händler: Veldshop.nl.

[2] Frank DD4WH. Suggestions for usb microphone systems. https://github.com/mcguirepr89/BirdNET-Pi/discussions/39.

[3] Stefan Kahl. Birdnet-analyzer. https://github.com/kahst/BirdNET-Analyzer, letzter Zugriff am 22.03.24.

[4] Stefan Kahl, Connor M Wood, Maximilian Eibl, and Holger Klinck. Birdnet: A deep learning solution for avian diversity monitoring. *Ecological Informatics*, 61: 101236, 2021.

[5] Bernd Klein. Python-kurs - einführng in pandas. https://www.python-kurs.eu/pandas.php, letzter Zugriff am 22.03.24.

[6] nvidia, . https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/jetson-nano-developer-kit/, letzter Zugriff am 16.03.24.

[7] nvidia, . https://files.seeedstudio.com/wiki/reComputer/reComputer-J1010-datasheet.pdf, letzter Zugriff am 16.03.24.

[8] Q-engineering. Install tensorflow lite 2.4.0 on jetson nano. https://qengineering.eu/install-tensorflow-2-lite-on-jetson-nano.html, letzter Zugriff am 22.03.24.

[9] Immersive Soundscapes. Comica cvm-mf1 windshield. Händler: Veldshop.nl.

[10] ALSA Development Team. Alsa, . https://wiki.ubuntuusers.de/ALSA/ letzter Zugriff am 15. März 2024.

[11] Pandas Development Team. pandas documentation, . http://pandas.pydata.org/docs/ Stand Feb 23, 2024 Version: 2.2.1.

[12] Python Development Team. datetime — basic date and time types, . https://docs.python.org/3/library/datetime.html letzter Zugriff am 15. März 2024.

[13] Python Development Team. glob — unix style pathname pattern expansion, . https://docs.python.org/3/library/glob.html#module-glob letzter Zugriff am 20. März 2024.

[14] Python Development Team. concurrent.futures — launching parallel tasks, . https://docs.python.org/3/library/concurrent.futures.html letzter Zugriff am 19. März 2024.

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit selbständig verfasst und keinen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Die Regelungen der geltenden Prüfungsordnung zu Versäumnis, Rücktritt, Täuschung und Ordnungsverstoß habe ich zur Kenntnis genommen.

Diese Arbeit hat in gleicher oder ähnlicher Form keiner Prüfungsbehörde vorgelegen.

Heiligenhaus, den _____ _____

Unterschrift