Duke
UNIVERSITY

Identifying Bird Species from Recordings Using Machine Learning

Christopher Hall, Michael Nauman, Siqi Liang, Rucha Patil

Table of Contents

A. Abstract

Biodiversity data is the foundation of conservation (Huang et al. 2021). With the development of conservation hardware, it is becoming more possible for researchers to use fully automated monitoring methods to collect raw biodiversity data from ecological communities (Marc et al. 2022). A few examples of these conservation hardware techniques include the use of infrared cameras to collect understory animal images, or hyperspectral cameras, which are able to process information across the electromagnetic spectrum. These cameras are able to identify different materials within an object, without causing any destruction. They can be mounted on drones to collect tree-crown spectral profiles. Finally, there are Audiomoths, which are acoustic devices that are able to collect animal calls, environmental sounds, or human activity. However, transferring these raw data into meaningful biodiversity data, i.e. specific species names, species abundance, and real-time locations…remains challenging. This semester, we focused on identifying bird species from bird sound recordings with machine learning. Our goal was to develop a bird sound identification software that can identify bird species from hour-long recordings, and the recordings can be from any place in the world.

We have presented two approaches for the problem. The first approach implements the BirdNET algorithm covered in Section 1 and the second approach covers the EfficientNetB0 in Section 2.

B. Section 1

I. Introduction

To the best of our knowledge, only two groups of people have successfully produced softwares relevant to our goal. A group of scientists from the University of Cornell developed a software named BirdNET (Kahl et al. 2021). BirdNET is an open-source software that can identify bird species from long recordings. BirdNET is an open-source software that can identify bird species from long recordings. The advantage of BirdNET is that it can process long recordings. The disadvantage of BirdNET is that its current platform is only able to identify around 3,000 of the world's most common birds, most of which are American. While this is an impressive metric, there are about 11,000 bird species worldwide (del Hoyo, 2021), essentially leaving 8,000 unaccounted for. It appears BirdNET's greatest hindrance is being restricted to recordings from certain areas of the world.

The second known bird sound identification software was developed by a group of electrical engineers from China. They developed a mobile application known as Bird ID Master. Bird ID Master is a software that can identify bird species from short recordings. Unlike BirdNET, Bird ID Master is able to identify more than 10,000 bird species, only leaving around 1,000 unaccounted for. This means that it has the capability of identifying birds all over the world and not just a centralized location, like America. On the flip side, Bird ID Master is only able to process recordings smaller than 240MB. Moreover, it is a phone application that requires a manual upload of each recording. The model of Bird ID Masters is based on open-source data (Xeno-Canto) and open-source algorithms (BirdNET). After exploring the structure of BirdNET on our computers and talking with the programmers of Bird ID Master, we believe we can achieve our goal by conducting the following steps:

1. Select a few species for exploration and obtain relevant open-source data as training data (from Xeno-Canto)
2. Produce a model based on open-source algorithms (from BirdNET) as well as the training data we obtained
3. Build the model into the BirdNET software
4. Test the customized BirdNET software with a long recording that contains the sounds of our target species
5. Based on findings, adjust the model and the customized software
6. If the steps above are successful, try more species.

II. Training and Evaluation

II.I. Obtaining Training Data

We selected five species to run the first round of exploration. Because the current model of BirdNET already can identify 3000 species, and we wanted to try some species the current BirdNET model cannot identify, we selected five leaf-warbler species from China. The five species we selected are: Smoky Warbler, Sulphur-bellied Warbler, Tickell's Leaf Warbler, Buff-throated

Warbler, and Sulphur-breasted Warbler. We obtain the relevant open-source data from Xeno-Canto with the following R codes. These codes not only allow us to patch download bird-sound recordings from Xeno-Canto but also store the recordings in separate files based on the species name.

~~~
```{r}#step1.install and library necessary packages
#install necessary package
install.packages("tuneR")
install.packages("seewave")
install.packages("NatureSounds")
install.packages("knitr")
install.packages("warbleR")
#library necessary package
library(tuneR)
library(seewave)
library(knitr)
library(NatureSounds)
library(warbleR)
library(stringr)
```
```{r}#step2. Provide species names, and patch download data into files
li = list()
# desired folder name
root_folder = "XenoCanto_Downloads"
li = append(li, 'Smoky Warbler')
li = append(li, 'Sulphur-bellied Warbler')
li = append(li, "Tickell's Leaf Warbler")
li = append(li, 'Buff-throated Warbler')
li = append(li, 'Sulphur-breasted Warbler')
# create root download folder, based on name above
rf = gsub(" ", "", paste("~/", root_folder)) #paste appends two strings, gsub removes
the spaces
dir.create(rf)
# for loop for all species
 # create sub directory
 # change to sub directory
 # download bird calls into sub directory
## If you dont want to download/only want to check the relevant metadata, turn
download = FALSE
for (p in li) {
 setwd(rf)
 name_ = gsub(" ", "_", p)
~~~

~~~~
 dir_name = gsub(" ", "", paste(rf,"/",name_))
 dir_name = gsub("'", "", dir_name)
 dir_name = tolower(dir_name)
 dir.create(dir_name)
 setwd(dir_name)
 species = p
 #species = str_replace_all(p, '_', ' ')
 temp <- query_xc(qword = species, download = TRUE)
}
#end of file
```
~~~~

II.II BirdNET – Survey and Evaluation

BirdNET is a machine learning workflow used to recognize and identify bird calls in audio clips. It is created, maintained, and developed by the K. Lisa Yang Center for Conservation Bioacoustics at the Cornell Lab of Ornithology. Not only does this tool have user friendly applications on platforms such as iOS, Android, and the web, it, crucially for our purposes, is open source, with all of its code available to the public at https://github.com/kahst/BirdNET-Analyzer.

II.III BirdNET – Pros and Cons

While BirdNET is a powerful tool in many respects, there are some drawbacks. Its intentional focus on usability by both citizen scientists and researchers has a strong tie to functionality present within. Further, its open source nature allows for direct tinkering with the code and underlying systems, as well as implementation on various micro controllers, but this does require a degree of technical knowledge to do so. Finally, the scope of the species covered by BirdNET's model is large, but not all encompassing.

One of the main goals of BirdNET is to leverage citizen scientists in gathering data, a goal that they work towards in maintaining as high a level of usability as they can. Many of the usability features present within the code base stand to reflect this. First, there are many GUIs available to interface with the workflow as stated earlier. The multiplatform apps as well as the browser implementation allow users to submit audio data and get specific predictions out of it, a process which will be outlined momentarily. Additionally, when interfacing with the open source version of the workflow, there are many utility functions whose purpose is to ensure that any audio format is compatible with the process.

BirdNET's open source repository allows for further customization of its systems and allows for the tool to grow from outside contributions. This allows for easy integration into automation workflows as the BirdNET can be configured to analyze entire directories of audio data with just a few lines of code, a process which itself can be automated. Having access to the source code would also allow for this tool to be integrated onto microcontrollers such as an Arduino or Raspberry-Pi. The one drawback here, is that to access any of these features, a greater amount of technical knowledge is required, not being accessible to your average citizen scientist. Additionally, the documentation for the process is lacking to non-existent, making development with the code more difficult than it would be if there was a more clear and comprehensive manual.

Finally the coverage of BirdNET's model in terms of what bird species it can recognize is impressive, but not complete. It currently supports the identification of 3,337 species, with a large number of those being species present in North America. This is all well and good, however the tool becomes less useful in its default state when using it in regions whose bird species are not as well documented or aren't in the model as completely as other regions.

II.IV BirdNET – Installation and Example

With all these pros and cons in mind, let's take a deeper look at an example of the tool in action. The next few paragraphs aim to show the capabilities of the tool and to walk anyone through the particulars of how to use them.

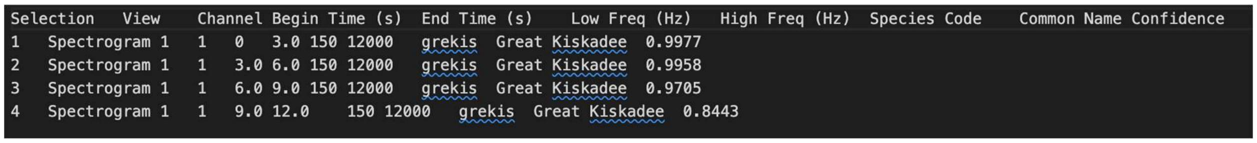
Follow the installation instructions found on BirdNET's Github: https://github.com/kahst/BirdNET-Analyzer. We experimented with various development environments in our dealings with the tool, and found success with Mac and Windows installations, however, our most consistent results were seen when we utilized the tool alongside Docker, whose installation guide can be found here: https://docs.docker.com/desktop/.

Once you have everything installed, you'll want to obtain the audio data that you want to analyze and have identified. As stated earlier, this data can be in many forms such as .wav, .mp3, etc. This data can either be recorded by you as the end user, or taken from a repository found from a source such as the internet. For our testing purposes we obtained data from Xeno Canto, a website dedicated to sharing wildlife sounds from all over the world. For our example, let's select the first recording of a Great Kiskadee: https://xeno-canto.org/species/Pitangus-sulphuratus. After it's downloaded, rename the file to gk.mp3 and place it in the examples folder within the BirdNET project.

Next, open up a command line interface and navigate to the root directory of the BirdNET project. Running the command:

```
python3 analyze.py --i example/
```

will run the analysis. If everything is set up correctly, a new file will appear within the example folder named gk.BirdNET.selection.table.txt. This file will contain the inferences made about the given audio clip. Each three second interval will have an associated inference where a bird species is identified by the BirdNET model, along with the algorithm's confidence in its inference as seen in the figure below.



```
Selection   View      Channel Begin Time (s)  End Time (s)    Low Freq (Hz)    High Freq (Hz)  Species Code     Common Name Confidence
1   Spectrogram 1   1   0   3.0 150 12000   grekis  Great Kiskadee  0.9977
2   Spectrogram 1   1   3.0 6.0 150 12000   grekis  Great Kiskadee  0.9958
3   Spectrogram 1   1   6.0 9.0 150 12000   grekis  Great Kiskadee  0.9705
4   Spectrogram 1   1   9.0 12.0     150 12000    grekis  Great Kiskadee  0.8443
```

Figure A.1 : Classification Confidence Interval

There are other arguments that can be used alongside specifying the input outlined below:

```
--i, Path to input file or folder. If this is a file, --o needs to be a file
      too.
--o, Path to output file or folder. If this is a file, --i needs to be a file
      too.
--lat, Recording location latitude. Set -1 to ignore.
--lon, Recording location longitude. Set -1 to ignore.
```

```
--week, Week of the year when the recording was made. Values in [1, 48] (4 weeks
      per month). Set -1 for year-round species list.
--slist, Path to species list file or folder. If folder is provided, species
      list needs to be named "species_list.txt". If lat and lon are provided,
      this list will be ignored.
--sensitivity, Detection sensitivity; Higher values result in higher
      sensitivity. Values in [0.5, 1.5]. Defaults to 1.0.
--min_conf, Minimum confidence threshold. Values in [0.01, 0.99]. Defaults to
      0.1.
--overlap, Overlap of prediction segments. Values in [0.0, 2.9]. Defaults to
      0.0.
--rtype, Specifies output format. Values in ['table', 'audacity', 'r', 'csv'].
      Defaults to 'table' (Raven selection table).
--threads, Number of CPU threads.
--batchsize, Number of samples to process at the same time. Defaults to 1.
--locale, Locale for translated species common names. Values in ['af', 'de',
      'it', ...] Defaults to 'en'.
--sf_thresh, Minimum species occurrence frequency threshold for location
      filter. Values in [0.01, 0.99]. Defaults to 0.03.
```

Additionally, BirdNET has the capability to extract feature embeddings rather than making inferences about classifications. These feature embeddings would be useful in cases where not a lot of labeled data is available for processes such as clustering, a process that will be further outlined in our future work section. To extract these feature embeddings, one would run a command very similar to the above command, like so:

```
      python3 embeddings.py --i example/
```

Finally, you can use BirdNET to extract audio clips to sanity check the predictions that it has made. Below are the flags for use with the command as well as an example of what the command might look like.

```
--audio, Path to folder containing audio files.
--results, Path to folder containing result files.
--o, Output folder path for extracted segments.
```

```
--min_conf, Minimum confidence threshold. Values in [0.01, 0.99]. Defaults to
      0.1.
--max_segments, Number of randomly extracted segments per species.
--seg_length, Length of extracted segments in seconds. Defaults to 3.0.
--threads, Number of CPU threads.
```

```
      python3 segments.py --audio example/ --results example/ --min_conf 0.8
```

The above command would take the results present in the example directory, along with the corresponding audio data in the same directory, and return audio snippets for inferences of confidences that were at least 0.8.

II.V BirdNET – Customization

The model that BirdNET uses to make inferences about birds is impressive but not all encompassing, and this is why we set out to learn how to use the framework of BirdNET's infrastructure and utilities in conjunction with our own custom ML algorithm. The standard BirdNET model can be found as a .tflite file within the checkpoints folder in the BirdNET project. This model was made to be able to identify 3,337 different species of birds, however, for the purposes of the XPRIZE competition, we wanted two things that the current model is not providing. First, we wanted to make sure that we had the capability to identify all bird species known to be present in an area, not just the ones present in the current BirdNET model. Second and further, if we train a model with a focus on birds present in the region, the custom model will do a better job of identifying them compared to the BirdNET model that has a less specialized focus.

With this goal in mind, we set out to see how we could create our own .tflite model for use in the BirdNET framework. After some searching we found a tool called TensorFlow Model Maker.

III. TensorFlow

TensorFlow was chosen for development and testing of a new model for classification, using data contributed by online databases. It suits our needs of both training, and testing. So what exactly is TensorFlow?

TensorFlow is a widely popularized open-source family of tools and libraries for machine learning and artificial intelligence. Under the TensorFlow umbrella exists an ecosystem of tools built for both beginners and experts when it comes to machine learning. Among these are tools such as libraries for training new models, libraries for using existing models and making inference, and libraries for converting models as well. Each of the models can be converted to be run more effectively for your platform, whether it is high powered GPUs or low power cell phones or other edge devices.

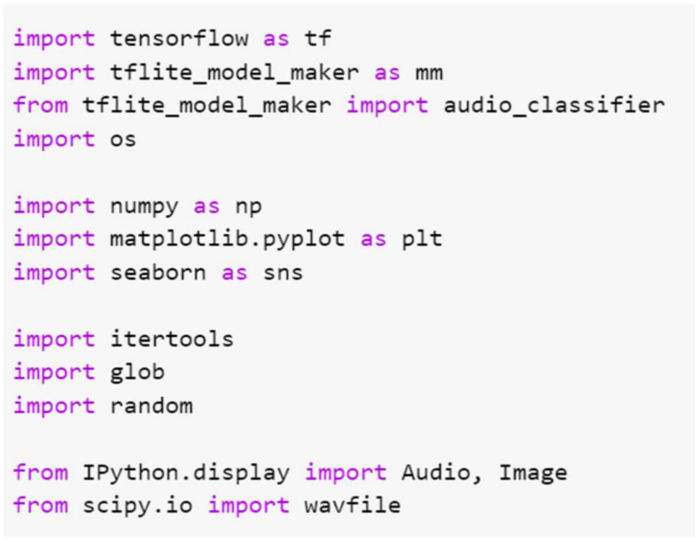
Our selection of TensorFlow is two-fold: First, BirdNET employs a TensorFlow pretrained model (using their training algorithm). Secondly, its ease of use allows us to create a model for comparative results, albeit much fewer species. As a matter of fact, BirdNET uses a TensorFlow lite model, so it was our goal to train a separate model and use it for comparison.

The data that we aim to use is a set of long snippets of “clean” data. That is, for each species, only one species bird call may be found in the entire audio track for training. Throughout the duration of the audio track, there may be several calls from one bird, or calls from another bird and many pauses, but all the calls must be of that bird species for our data to be clean.

An important precursor for our data to work with TensorFlow Model Maker for training is detection. For our case, what we need to input into the training algorithm are the snippets of the bird calls, or single bird calls. In order to do this, we need to detect the start of the bird calls, as well as define the expected duration of the calls. We utilize YAMNet in the TensorFlow library to do this, where we can define parameters such as expected duration, and overlap of samples. This will allow us to parse the set of input files, and output a set of distinct events that capture the individual bird sounds found at different times of all our files.

III.I Training a TensorFlow model using TensorFlow’s Model Maker

First, we setup a system by installing the necessary requirements for TensorFlow training using Model Maker. The installation of libportaudio2 and tflite-model-maker are installed, along with Python3. Following that we can setup the Python environment with the required libraries. These include:



```
import tensorflow as tf
import tflite_model_maker as mm
from tflite_model_maker import audio_classifier
import os

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import itertools
import glob
import random

from IPython.display import Audio, Image
from scipy.io import wavfile
```

Figure A.2 : TensorFlow Model Maker Requirements

Next in the process, we need to identify the species and data that we want to train and classify for our purposes. For our demonstration, 5 species were selected. These species data were pulled from XenoCanto using the web-scraping technique and steps mentioned earlier. Next, the species are separated into their respective species folders, and then again inside, we separate them out into the 'train' and 'test' portion. It is important to treat these two as separate, to get a legitimate observation when analyzing the performance of the training. Please try to stick between a 90-10 split or 80-20 split for train-test splits. We use audio_classifierDataLoader.from_folder() for the loading of the data.

As an important precursor to classification, detection libraries are also leveraged. As mentioned before, YAMNet within TensorFlow can utilize this capability and specify the expected length of the waveform. For our use cases, this is set conservatively at 3s. We use audio_classifier.YamNetSpec() to configure this.

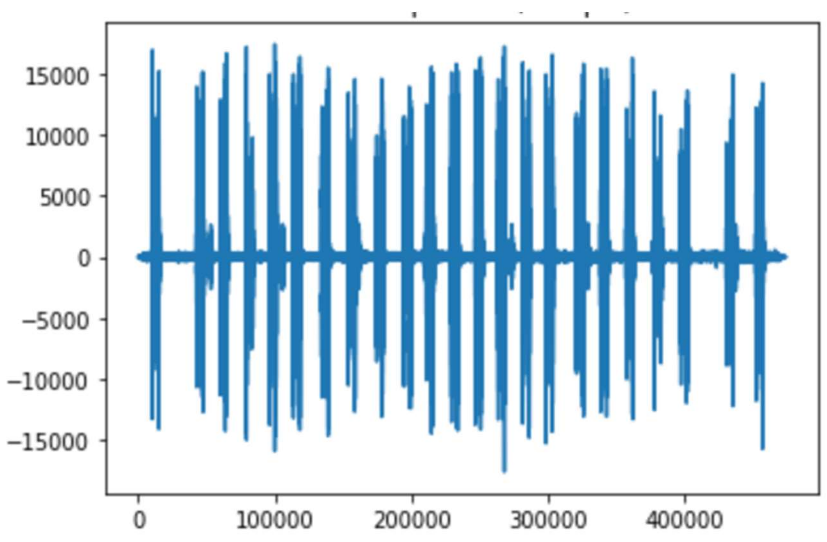


Figure A.3 : Raw Data Inputs – pre-Detection of YAMNNet

Above is an example audio plot of one dataset, where the duration of bird calls stands in contrast to the direct pauses between.

Next, we load the data into our Python environment, using the test and train separation as mentioned before. We then split the train data again for our validation set, this time we use only 20% of the training data for validation. The final step in our TensorFlow training process is to create and train the model. We wanted the number of epochs to be low to reduce training time, and used 100 epochs and a batch size small as well at 128, to reduce the number of samples per training step. We use audio_classifier.create() to train.

After this, we run model.evaluate() on the test data to test the data and determine the accuracy of the model.

After we obtained the model, our goal was to use the TensorFlow model in addition to the BirdNET tflite model. So after the model was created, it was exported as a .tflite using model.export() command.

BirdNET contains its own model, so the first thing we tried was to deploy our model for testing and use the testing framework to analyze the performance of our individually trained algorithm. What we found was that the model dimensions were mismatching, and thus, could not be used in the BirdNET tester. Rather, the model had to be used separately. The BirdNET .tflite model can be located from the BirdNET github, in the Checkpoints folder.

III.II Future Work

We want to expand our model for many more species for future use cases. This was a demonstration that learning is capable even for those students without Machine Learning backgrounds.

One area of interest is to try and adapt the shape of the model to be able to work well with the BirdNET framework. There is a reshape() functionality, but there was substantial effort to use this but so no success. We believe this was either a shape issue, or it could have been a stereo/mono issue.

The next logical area for expansion is to train on more than 5 bird species.

C. Section 2

I. Introduction

Bird Call Identification has multiple challenges which include

1. Ambient noise, especially when collecting data from urban recordings (e.g. city noises, cars)

2. The challenge of classifying many labels simultaneously when multiple bird species are singing (as described earlier)

3. Birds of the same species living in various regions or nations may have different calls.

4. The data may be greatly unbalanced since one species is more popular than another, there are many distinct species, and recordings of different species may vary in length and quality (volume, cleanliness)

With the prior implemented models [1,2,3] we found that CNN based approach often yielded a better output hence we chose this method. The model chosen is EfficientNetB0.

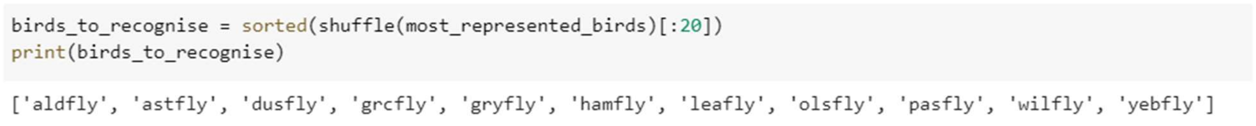
EfficientNet is a convolutional neural network design and scaling technique that uses a compound coefficient to consistently scale all depth, breadth, and resolution dimensions. The EfficientNet scaling method evenly scales network breadth, depth, and resolution using a set of preset scaling coefficients, in contrast to standard practice, which scales these variables arbitrarily. For instance, to employ times more computing power, we may simply raise the network depth, width, and picture size, where and are constant coefficients discovered by a tiny grid search on the initial small model. Network width, depth, and resolution are all uniformly scaled by EfficientNet using a compound coefficient.

The rationale behind the compound scaling method is that larger input images require more layers in order to expand the network's receptive field and more channels in order to capture more fine-grained patterns on the larger picture. In addition to squeeze-and-excitation blocks, the foundational EfficientNet-B0 network is built upon the MobileNetV2 inverted bottleneck residual blocks. On the CIFAR-100 (91.7%), Flowers (98.8%), and 3 other transfer learning datasets, EfficientNets likewise transfer well and achieve state-of-the-art accuracy while using orders of magnitude fewer parameters.

II. About the Data

The Data chosen was that of 11 Flycatcher species from the Cornell image bird database[5] because of the availability of the appropriate annotations already. Custom recordings can also be used to train the data provided, the annotations and the metadata are it the correct format. A short code can be written to get the required csv file.

The flycatcher species are as in the Image below



```
birds_to_recognise = sorted(shuffle(most_represented_birds)[:20])
print(birds_to_recognise)
```

```
['aldfly', 'astfly', 'dusfly', 'grcfly', 'gryfly', 'hamfly', 'leafly', 'olsfly', 'pasfly', 'wilfly', 'yebfly']
```

Figure B.1 : Birds Species

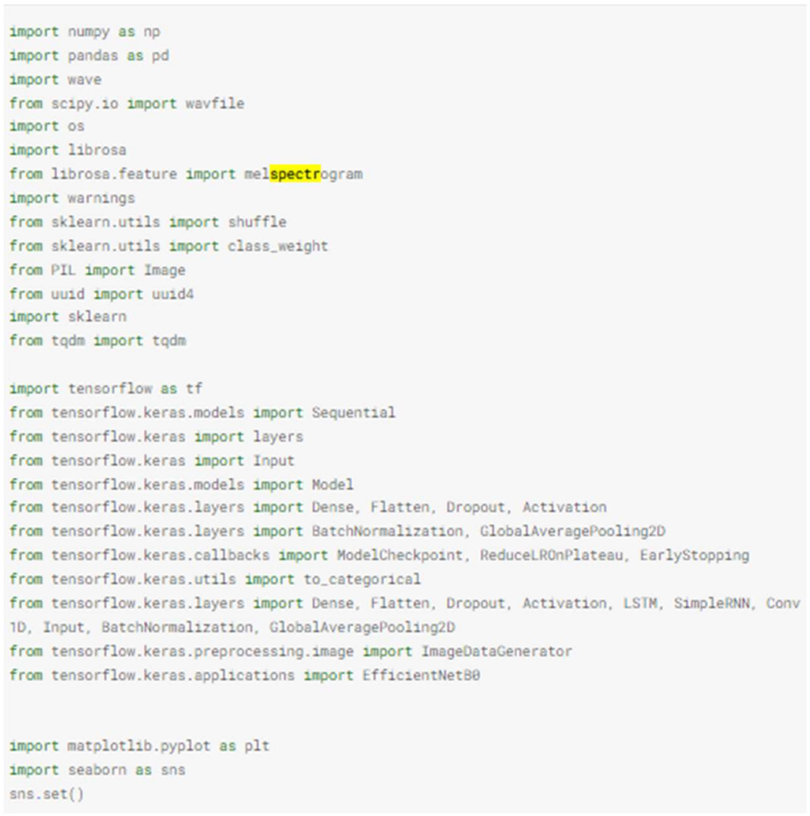
These short codes stand for Alder Flycatcher, Ash-Throated Flycatcher, Dusky Flycatcher, Grey-Chested Flycatcher, Gray Flycatcher, Hammonds Flycatcher, Least Flycatcher, Olive-Sided Flycatcher, Pacific-Slope Flycatcher, Willow Flycatcher and Yellow-Bellied Flycatcher. The Data can be downloaded from [5]. We have about 100 recordings of each bird and a total of 1003 recordings.

III. Method [4]

III.I Complete the required installations and import

In order to be able to code on a local device we need the following packages

Librosa(Audio processing), Scipy(data analysis), opencv-python(Image processing), tqdm, sklearn(machine learning models), tensorflow and keras(machine learning frameworks), pandas(data analysis), numpy(mathematical operations), wave(audio processing), pillow(image manipulation). The installation can be done by simply running the "pip install package_name" in the terminal. Python and pip need to be preinstalled as well. This step can be eliminated if running on cloud-based platform like Google-Colab. Hence we import the packages in the code as in figure B.2.

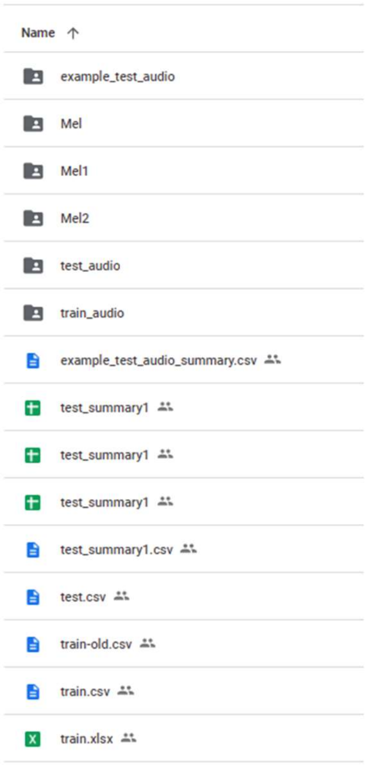


```
import numpy as np
import pandas as pd
import wave
from scipy.io import wavfile
import os
import librosa
from librosa.feature import melspectrogram
import warnings
from sklearn.utils import shuffle
from sklearn.utils import class_weight
from PIL import Image
from uuid import uuid4
import sklearn
from tqdm import tqdm

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras import Input
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout, Activation
from tensorflow.keras.layers import BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.callbacks import ModelCheckpoint, ReduceLROnPlateau, EarlyStopping
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Flatten, Dropout, Activation, LSTM, SimpleRNN, Conv
1D, Input, BatchNormalization, GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import EfficientNetB0


import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Figure B.2 : Packages



```
Name ↑
example_test_audio
Mel
Mel1
Mel2
test_audio
train_audio
example_test_audio_summary.csv
test_summary1
test_summary1
test_summary1
test_summary1.csv
test.csv
train-old.csv
train.csv
train.xlsx
```

Figure B.3 : File Organization

III.II Data Preprocessing

Then we proceed to import the Data. The Data is uploaded on google drive and then the drive is mounted in the code. The organization of the files is as in the figure B.3. "train_audio" Consists of subfolders labelled with name of the bird, which consist of the recordings. We also need a csv file with the metadata of the recordings. It needs to be in a specified format. Refer to the drive files attached above.

For the sake of simplicity of the model, “bad” recordings are not taken into consideration. The dataset that was downloaded had a quality column. The data is filtered to preserve only the good quality recordings. It is implemented in code found in Colab. However, the model will still work will the unfiltered images. The only drawback being the additional data and time consumed. We might need to introduce an additional step of filtering the noise out, which can be done by using a bilateral filter after spectrograms are generated.

III.II.I Generating Spectrograms

CNNs are made to extract unique features from images. Hence to apply it to audios, we need to convert them to images. We do so by using spectrograms. Spectrograms also can make it easier to identify features.

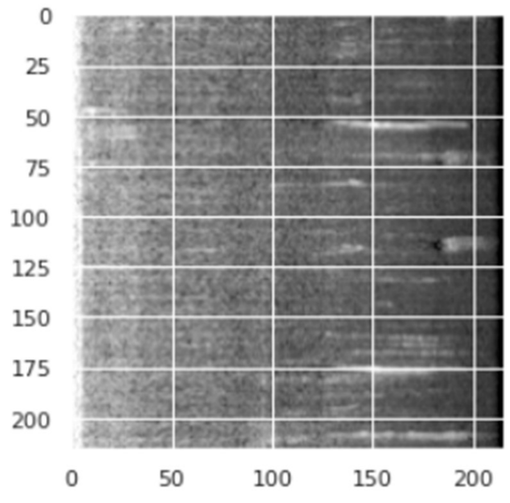
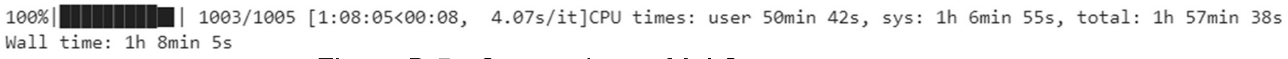


Figure B.4 : Sample Spectrogram

Each sound we hear is made up of several different sound frequencies occurring simultaneously. This is what gives the audio its "depth" quality. The secret to a spectrogram is to display both of those frequencies in one plot as opposed to the waveform's sole display of amplitude. The Mel scale is an audio scale of sound pitches that, to listeners, appear to be equally spaced apart. That concept has something to do with how people hear. When we combine those two concepts, we have a modified spectrogram (Mel-frequency spectrum) that only plots the most significant portions while ignoring the noises that people cannot hear. The longer the audio file used to build the spectrogram, the more information you may extract from it, but the risk of overfitting your model also increases. Five second audio clips may not be able to capture the necessary information if your data contains a lot of noise or silence.

A function to generate spectrograms from the train folder is written, hence manual conversion is not required. Implementation in Figure B.5. The figure B.4 shows a sample spectrogram generated.

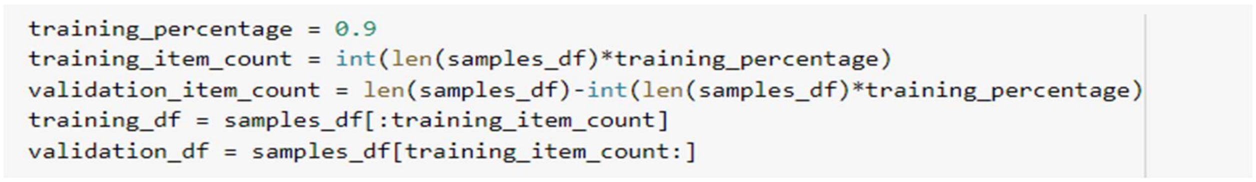

```
100%|██████████| 1003/1005 [1:08:05<00:08,  4.07s/it]CPU times: user 50min 42s, sys: 1h 6min 55s, total: 1h 57min 38s
Wall time: 1h 8min 5s
```
Figure B.5 : Conversion to Mel Spectrograms

Post that, the image generator was used to rescale the images to normalize the image data. In order to normalize the data, we have rescaled the image out to 1. Image pixel values range from 0-255 for the RGB (red, green, blue) channels. We rescaled them to range from 0 to 1. The reasons being

1. To normalize the loss: There are photos with high and low pixel ranges. The model, weights, and learning rate are all shared by all the photos. The aggregate of them will contribute to the back propagation update. High range images typically produce larger loss, whilst low range images typically produce weaker loss. The loss will be distributed more evenly if all the photos are scaled to the same range [0,1].
2. Using a typical learning rate: If two works do the scaling preprocessing over an image data collection, we can directly refer to their learning rates when referencing learning rates from those works. In this case, lower pixel range images will require a bigger learning rate whereas higher pixel range images will result in higher loss.

III.III Creating the Model

The Data was then split in 90% - training and 10%-validation as in Figure B6.

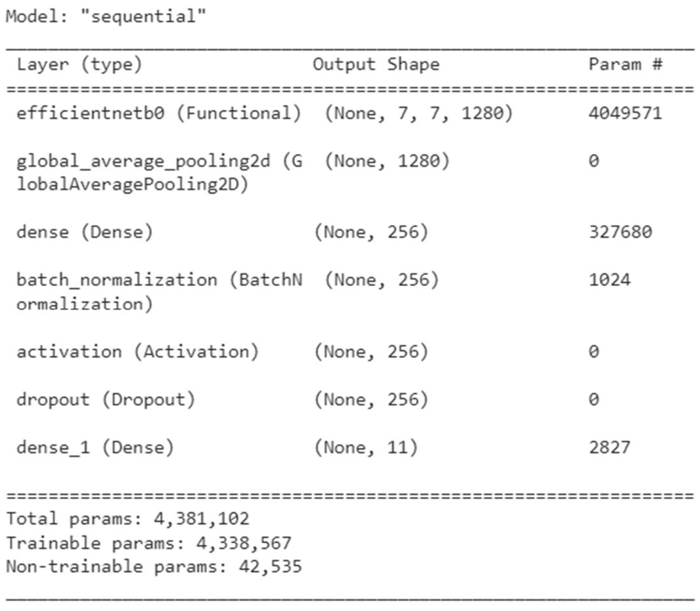


```
training_percentage = 0.9
training_item_count = int(len(samples_df)*training_percentage)
validation_item_count = len(samples_df)-int(len(samples_df)*training_percentage)
training_df = samples_df[:training_item_count]
validation_df = samples_df[training_item_count:]
```

Figure B.6 : Test Train Split

Then we define the model with the following parameters as in Figure B.7

Efficient net was generated using the parameters training_batch_size = 32 ,
validation_batch_size = 32, target_size = (216,216).



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb0 (Functional)  (None, 7, 7, 1280)       4049571

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 256)               327680

 batch_normalization (BatchN  (None, 256)              1024
 ormalization)

 activation (Activation)     (None, 256)               0

 dropout (Dropout)           (None, 256)               0

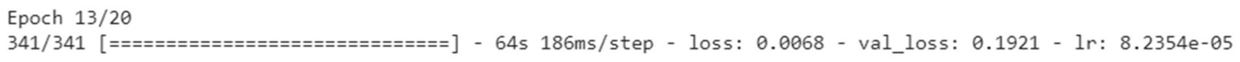
 dense_1 (Dense)             (None, 11)                2827

=================================================================
Total params: 4,381,102
Trainable params: 4,338,567
Non-trainable params: 42,535
_________________________________________________________________
```

Figure B.7 : Model Parameters

III.IV Fit and Train

Post that we fit and train the model. We trained the model initially for 5 epochs then proceeded to 15 and then 20 (total 40). The best model is provided in drive. the details of the best model are as in figure B.8



```
Epoch 13/20
341/341 [==============================] - 64s 186ms/step - loss: 0.0068 - val_loss: 0.1921 - lr: 8.2354e-05
```

Figure B.8: Best model

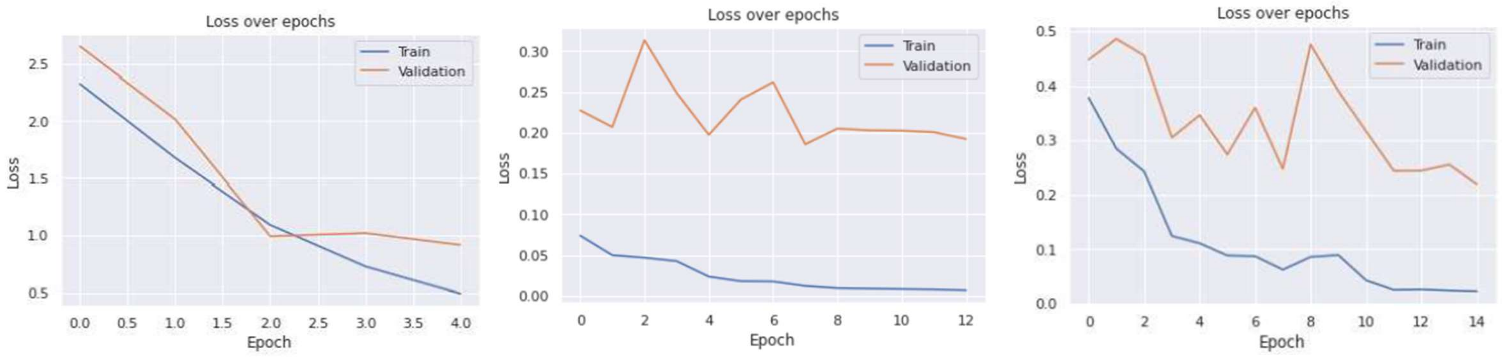
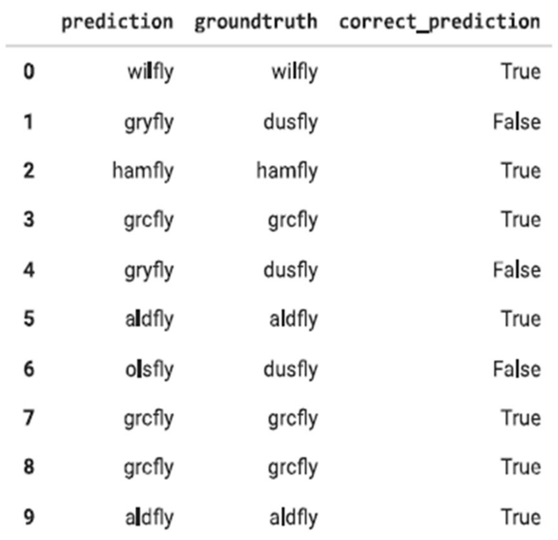


Figure B.9: Training loss and Validation loss across 5, 15 and 20 epochs (on same model trained again)

III.V Analyzing the Model

The Training and validation curves are as in Figure B.9

Figure B.10 Shows the sample output of the validation data

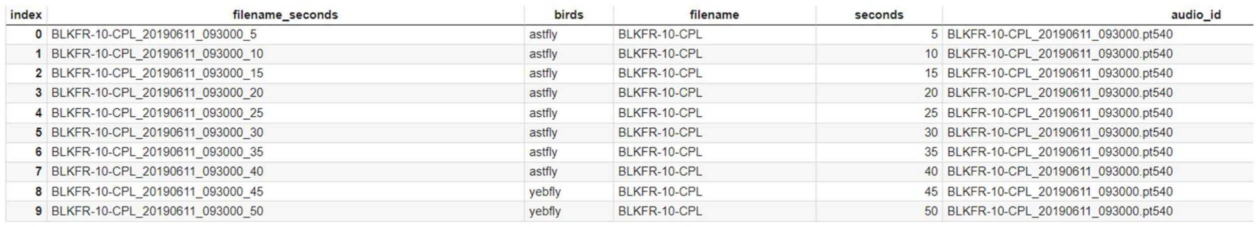


	prediction	groundtruth	correct_prediction
0	wilfly	wilfly	True
1	gryfly	dusfly	False
2	hamfly	hamfly	True
3	grcfly	grcfly	True
4	gryfly	dusfly	False
5	aldfly	aldfly	True
6	olsfly	dusfly	False
7	grcfly	grcfly	True
8	grcfly	grcfly	True
9	aldfly	aldfly	True

Figure B.10: Sample Validation results

III.VI Predicting on the unknown Data

Running it on sample audio results in a file like in figure B.11
If the recording contains a birdcall not from the 11 chosen bird species, it will try to fit the existing species to the same resulting in wrong outcome. Hence these predictions were not correct. The unknown data also needs to be annotated in the standard format.



index	filename_seconds	birds	filename	seconds	audio_id
0	BLKFR-10-CPL_20190611_093000_5	astfly	BLKFR-10-CPL	5	BLKFR-10-CPL_20190611_093000.pt540
1	BLKFR-10-CPL_20190611_093000_10	astfly	BLKFR-10-CPL	10	BLKFR-10-CPL_20190611_093000.pt540
2	BLKFR-10-CPL_20190611_093000_15	astfly	BLKFR-10-CPL	15	BLKFR-10-CPL_20190611_093000.pt540
3	BLKFR-10-CPL_20190611_093000_20	astfly	BLKFR-10-CPL	20	BLKFR-10-CPL_20190611_093000.pt540
4	BLKFR-10-CPL_20190611_093000_25	astfly	BLKFR-10-CPL	25	BLKFR-10-CPL_20190611_093000.pt540
5	BLKFR-10-CPL_20190611_093000_30	astfly	BLKFR-10-CPL	30	BLKFR-10-CPL_20190611_093000.pt540
6	BLKFR-10-CPL_20190611_093000_35	astfly	BLKFR-10-CPL	35	BLKFR-10-CPL_20190611_093000.pt540
7	BLKFR-10-CPL_20190611_093000_40	astfly	BLKFR-10-CPL	40	BLKFR-10-CPL_20190611_093000.pt540
8	BLKFR-10-CPL_20190611_093000_45	yebfly	BLKFR-10-CPL	45	BLKFR-10-CPL_20190611_093000.pt540
9	BLKFR-10-CPL_20190611_093000_50	yebfly	BLKFR-10-CPL	50	BLKFR-10-CPL_20190611_093000.pt540

Figure B.11 : Unknown Data Results

IV. Conclusions

Overall, the model has produced good results. The model has high validation accuracy and low loss and hence can be used to identify birds from 11 different types of flycatchers. The model can also be trained for other custom bird species using the procedure above.

In the future, adding more species to the training dataset and utilizing different kind of noise filters will help to optimize the performance of the model better.

A clustering-based approach also might prove to be beneficial for the identification of unknown species as the major limitation of this model currently is that it does not return "unidentified" species and makes a prediction out of the known ones.

D. Comparison and Concluding Remarks

Utilizing existing libraries and frameworks for training and classification can offer many advantages – speed of development, ease of access to less-experienced developers in machine learning, and often times open-source. BirdNET offered a framework for evaluation and testing of trained models, yet we found that the models had to match in number of layers, and such meta-data file formats. As a result, our comparison from deploying our trained model fell short of our initial expectations – we funneled ourselves into BirdNET framework in that regard.

Moving forward, utilizing more flexible learning frameworks, such as discussed in Section II, offer many advantages such as customization in evaluation, as well as density of the models. This method is also more easily expandable as the desired subset of classification species grows. The drawback being that the model can be applied to only the known species as it cannot predict an "unidentified". A clustering-based approach can be used to approach this problem or an ensemble of multiple algorithms working together can be created.

E. References

Section 1

[1] Huang, R. M., Medina, W., Brooks, T. M., Butchart, S. H., Fitzpatrick, J. W., Hermes, C., ... & Pimm, S. L. (2021). Batch-produced, GIS-informed range maps for birds based on provenanced, crowd-sourced data inform conservation assessments. *PloS one*, *16*(11), e0259299.

[2] Besson, M., Alison, J., Bjerge, K., Gorochowski, T. E., Høye, T. T., Jucker, T., ... & Clements, C. F. (2022). Towards the fully automated monitoring of ecological communities. *Ecology Letters*.

[3] Kahl, S., Wood, C. M., Eibl, M., & Klinck, H. (2021). BirdNET: A deep learning solution for avian diversity monitoring. *Ecological Informatics*, *61*, 101236.

[4] del Hoyo, J. (2021). All the Birds of the World.

Section 2

[1] https://www.imageclef.org/BirdCLEF2019

[2] https://dcase.community/challenge2019/index

[3] https://towardsdatascience.com/sound-based-bird-classification-965d0ecacb2b

[4] https://www.kaggle.com/code/frlemarchand/bird-song-classification-using-an-efficientnet/notebook#Data-preparation

[5] https://www.kaggle.com/competitions/birdsong-recognition/data

F. Appendix

Section 2 : Code and Data for training , along with weights

[1]Code:https://colab.research.google.com/drive/1qmQTRla__mOLXqOu6g7XZxYULpHtbHCF?usp=sharing

[2]Drive:https://drive.google.com/drive/folders/1fqFGyPb2UO0ob-bJOfmU7tAA7MLQcSm-?usp=sharing