Final project report

Andrew Fritz, Leo Gordon, Nimay Goradia, Yen Nguyen

Kaggle notebook: https://www.kaggle.com/code/andrewf87/bigdata-submission

GitHub repo: https://github.com/andfritzhcms/BD2025.git

** Because the dataset is on kaggle this is the easiest way to see what it does - also it is prerun **

Introduction

Without being a biologist, identifying the exact species of birds chirping, insects buzzing, and frogs croaking in the middle of a rainforest is almost an impossible task. These ecosystems are extremely biodiverse, with hundreds of different species coexisting, many of which sound very similar to the human ear.

Traditionally, biologists and field researchers have to rely on manual audio analysis and expert knowledge to identify species, which can be time-consuming and impractical. This inspires the BirdCLEF competition, which itself arises from the need to automate the monitoring of endangered wildlife of a Colombian Natural Reserve (El Silencio).

The project is on classification of under-studied and endangered animal species based on audio datafiles. The goal is to be able to develop a network capable of working with the audio datafiles effectively and identifying these species using only passive recordings. This will aid the efforts of researchers and conservation practitioners to better understand the ecological effects of the restoration. Doing so will ultimately enable them to adjust their actions to best protect the endangered animals within the reserve.

Inspiration

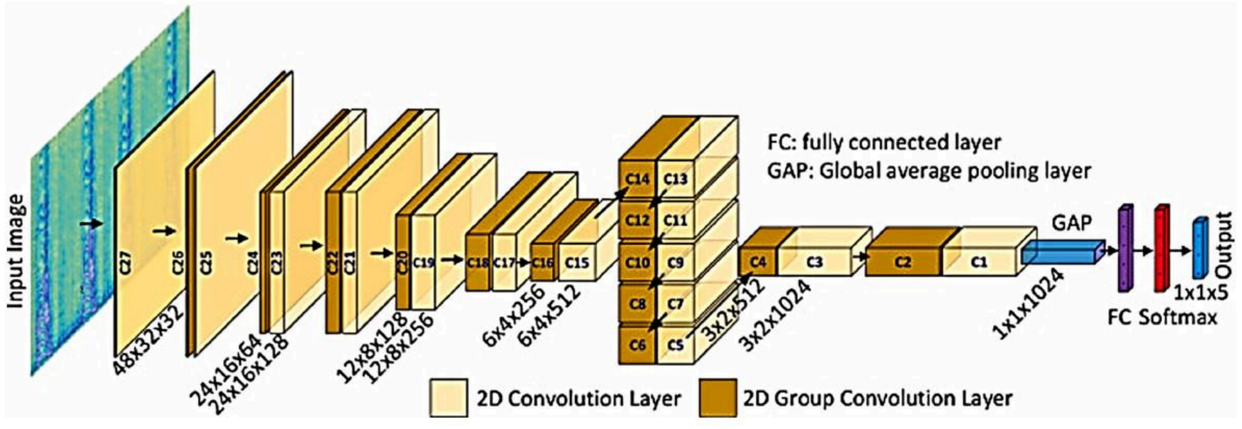
The BirdCLEF competition has been an annual competition for several years, and has helped several other conservation organizations.

Dataset

The dataset for the competition is made up of audio recordings from the El Silencio Natural Reserve in Colombia. They were collected based on passive acoustic monitoring and are spread out across 206 species of animal, in recordings of various lengths and audio levels. There are several important elements of the dataset making it more challenging to work with: First, because some species are rare and some are not, there is a large imbalance in the number of samples available for each species, with some having very few to work with. Second, the audio was recorded in different locations and with different devices, making it less standardized and thus more difficult for a model to create valid classifiers. Third, some species were recorded in the same audio file, making it hard to separate the two distinct sounds. The audio clips were all 32 kHz (although the code had measures to ensure this was the case) and was in .ogg format.

Methodology

The primary tool used was Google's yamnet audio event classifier model. Yamnet is a audio event classifier pretrained on Youtube videos which takes audio waveform as an input and makes independent predictions for each of 521 audio events from the AudioSet ontology. The model returns a 3-tuple, but the only element of the tuple used for this project is the embeddings, which is a float32 Tensor of shape ([number of frames], 1024) which contains the per-frame embeddings of each of 1024 classifiers. In the diagram below, this is the purple block.



Before working with any model, we first had to clean and standardize the data. The data came in 32 kHz files, but yamnet requires 22 kHz files to run correctly. Thus, within the code which read the audio files to a dataframe, we included a function ensure_sample_rate which reconfigured each file to the correct frequency. To read each audio file, we used os.walk to navigate the folder hierarchy, extract the labels for the species of the using os.path.basename (because the species corresponded to the name of the folder it was in), and store the path to each audio file for later reading.

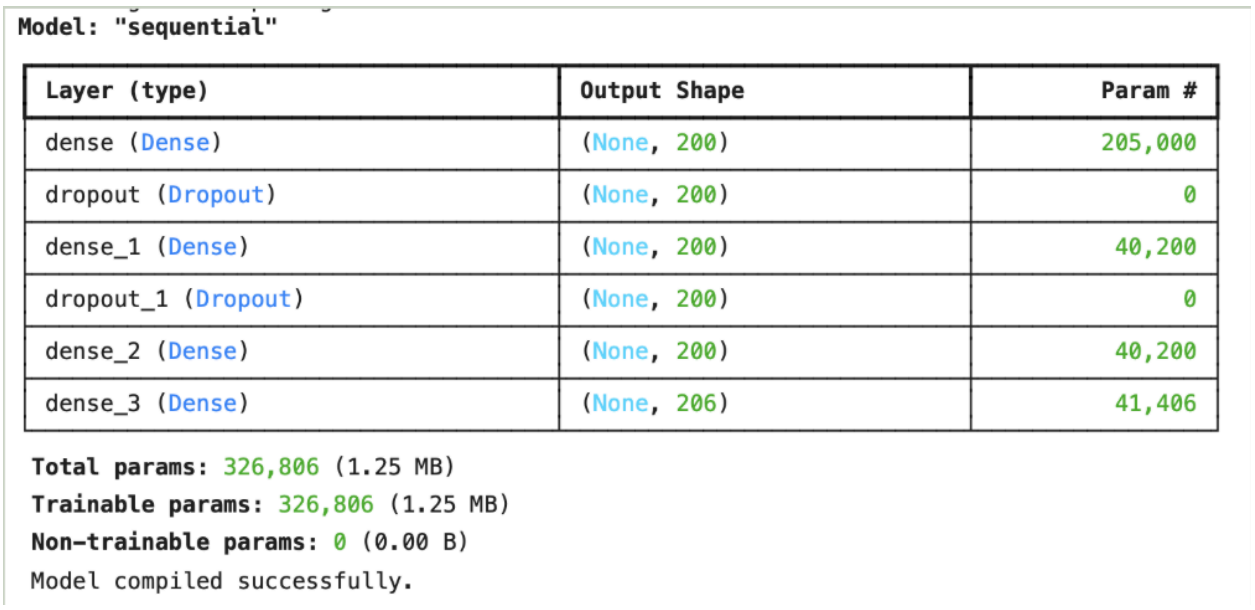
Next, due to restrictions on RAM (and the fact that more samples did not improve model accuracy), we selected two audio samples from each species to analyze. This was done primarily for RAM reasons (because the notebook would time out using yamnet embeddings) but also because some species were underrepresented, meaning that this size ensured each species had equal representation in the overall dataset, requiring the model to treat each species as equal and not learn more from any given one.

Next, we used parallel computing via concurrent.futures.ProcessPoolExecutor to efficiently read the 400+ minutes worth of audio files in under 4 minutes. We also implemented error handling within it to ensure that any given audio file not working wouldn't destroy the dataset.

Next, because the data was of varying lengths, it had to be standardized such that a model would be able to work with the data. This involved cutting or padding the audio files to make sure they were all the same length, which we deemed to be 1 second after some trial and error. For each audio clip, if it was less than one second, we padded the audio sequence with zeroes to make sure it reached the target length, and if it was more than one second, we split it into chunks of one-second length and any remainder was padded and added as an additional chunk. This was done using np.pad, and the results were stored in a NumPy array for fast computation. The array final_audio_array had a shape of (360928, 1000), representing the 360,928 1-second chunks and all 1000 ms of their length.

Next, for each species, we selected up to 1000 random chunks from each species to study, essentially cutting the dataset in half to size (166725, 1000). This was done to again standardize the model for each species, and also to not overload the allotted RAM. If a species had less than 1000 chunks, every chunk was kept.

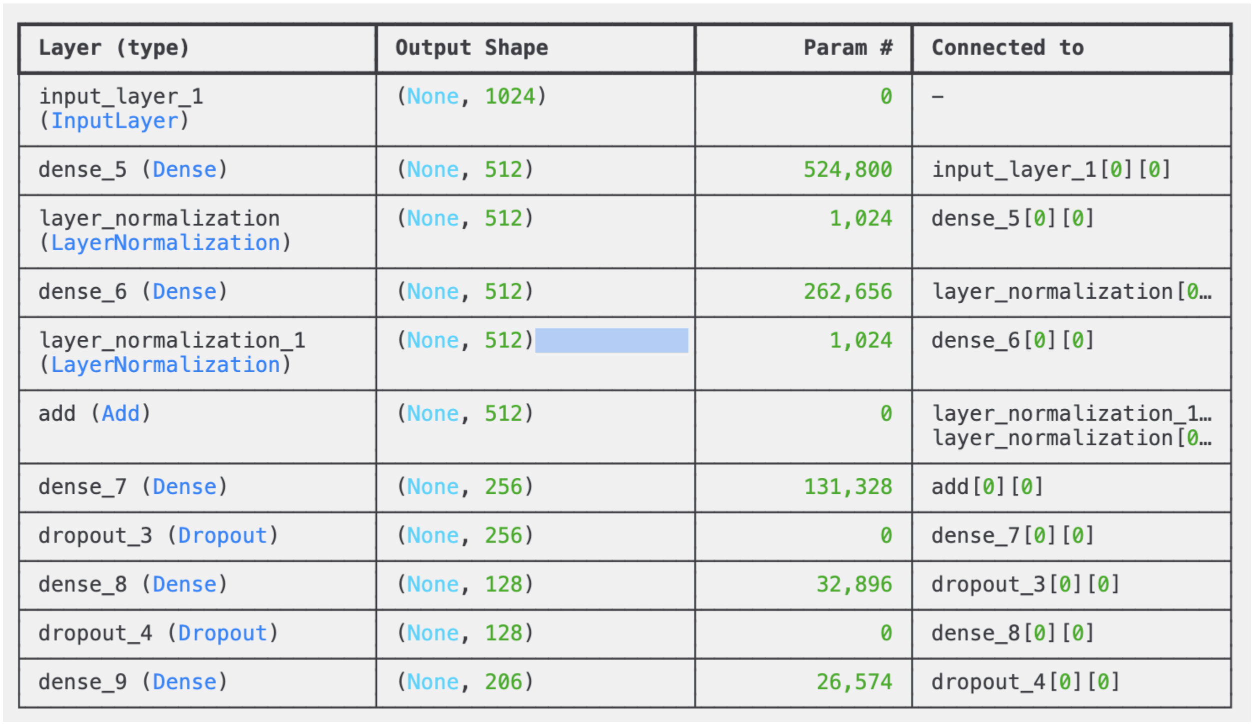
Next, the yamnet model was loaded in, and the embeddings were generated. These embeddings were then split into training and testing sets, with the labels being one-hot encoded for simplicity. The model was a CNN with the input being the 1024-dimensional array and the output being one of the 206 species as an identifier. It contained 3 dense 200-node layers, with a 0.1 dropout layer in between each to prevent overfitting. The activation function for each of the inner layers was reLu and the output activation function was a softmax.



```
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ dense (Dense)                        │ (None, 200)                 │         205,000 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout (Dropout)                    │ (None, 200)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_1 (Dense)                      │ (None, 200)                 │          40,200 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_1 (Dropout)                  │ (None, 200)                 │               0 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                      │ (None, 200)                 │          40,200 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_3 (Dense)                      │ (None, 206)                 │          41,406 │
└──────────────────────────────────────┴─────────────────────────────┴─────────────────┘
 Total params: 326,806 (1.25 MB)
 Trainable params: 326,806 (1.25 MB)
 Non-trainable params: 0 (0.00 B)
 Model compiled successfully.
```

The model was then trained and was set to stop when the validation loss did not improve after 3 consecutive epochs.

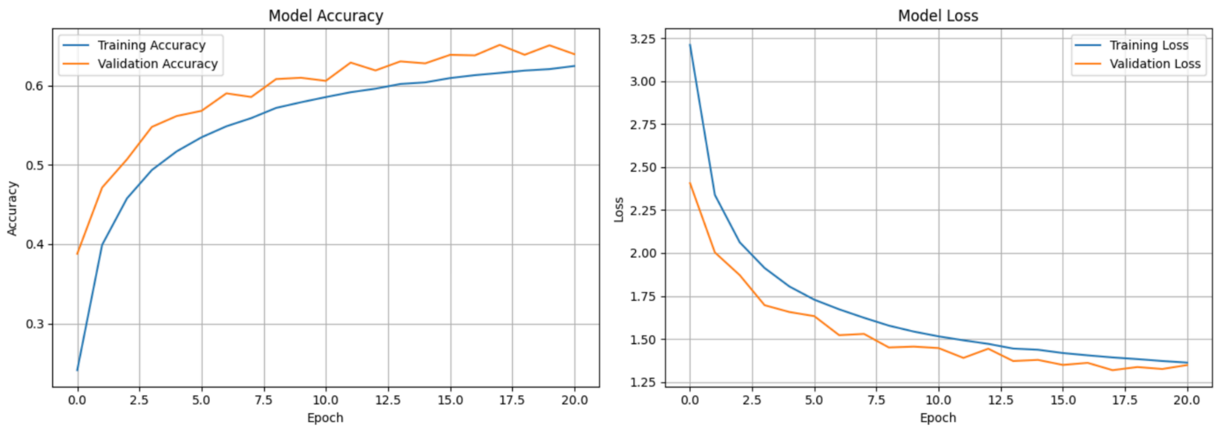
We also included in the code another more complicated model which was set to stop when the accuracy did not improve after 5 consecutive epochs. This model had 2 512-node dense layers with normalization layers in between each, and then 3 256-node layers with 0.3 and 0.2 dropout layers in between them respectively. Despite the increased complexity, model performance did not improve by much, and the increased runtime leads me to solely discuss results in terms of the first model.



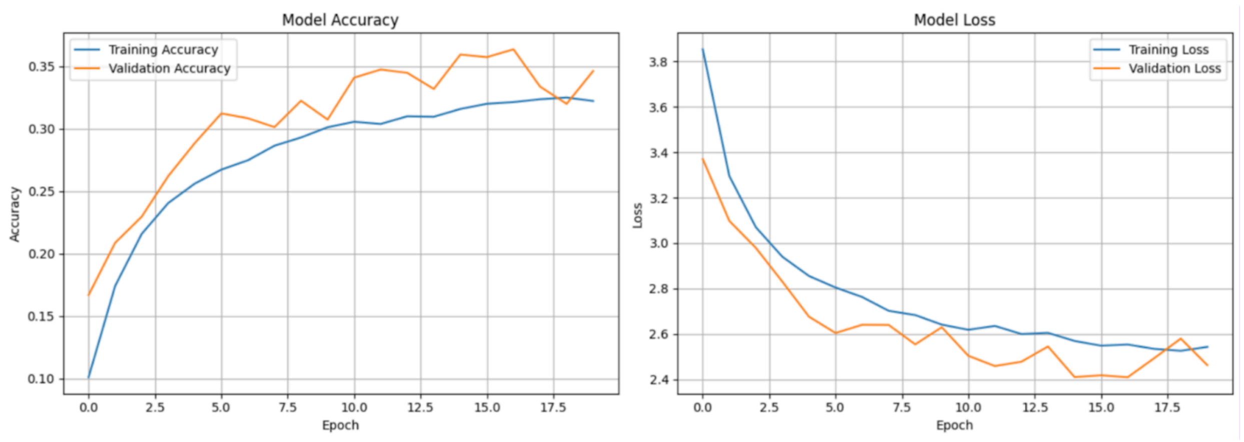
Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 1024)	0	-
dense_5 (Dense)	(None, 512)	524,800	input_layer_1[0][0]
layer_normalization (LayerNormalization)	(None, 512)	1,024	dense_5[0][0]
dense_6 (Dense)	(None, 512)	262,656	layer_normalization[0…
layer_normalization_1 (LayerNormalization)	(None, 512)	1,024	dense_6[0][0]
add (Add)	(None, 512)	0	layer_normalization_1… layer_normalization[0…
dense_7 (Dense)	(None, 256)	131,328	add[0][0]
dropout_3 (Dropout)	(None, 256)	0	dense_7[0][0]
dense_8 (Dense)	(None, 128)	32,896	dropout_3[0][0]
dropout_4 (Dropout)	(None, 128)	0	dense_8[0][0]
dense_9 (Dense)	(None, 206)	26,574	dropout_4[0][0]

Results

After tuning the first model, the best results achievable were around 65% accuracy with a validation loss of around 1.3.

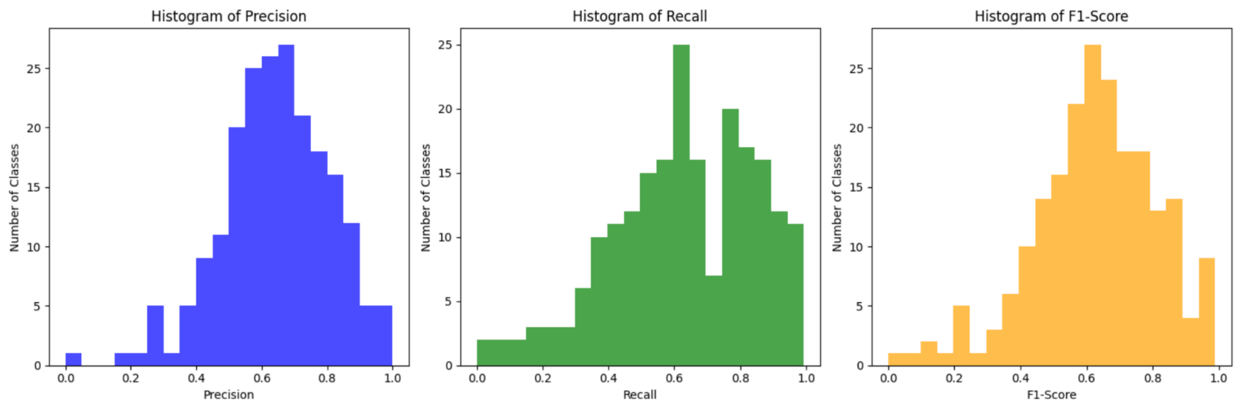


The second model took around six times longer to run, but achieved 70% accuracy and had similar validation loss.



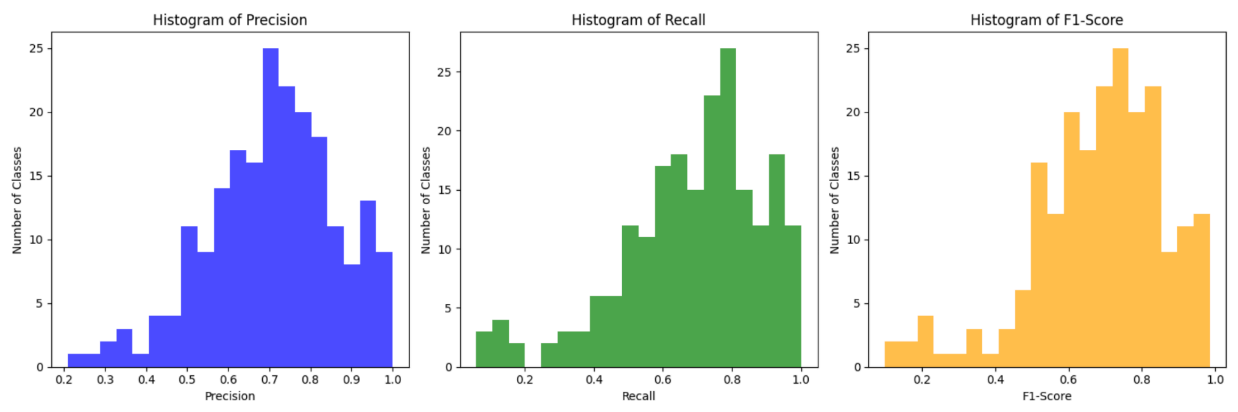
Due to the large number of species, this seems to be a good result, especially considering the dataset being as difficult to manage and learn from as it was and the computational resources being as limited as they were. There was some overfitting observed despite the dropout, noted by the light plateau of validation loss, but ultimately it still performed well on the test data which is the important element.

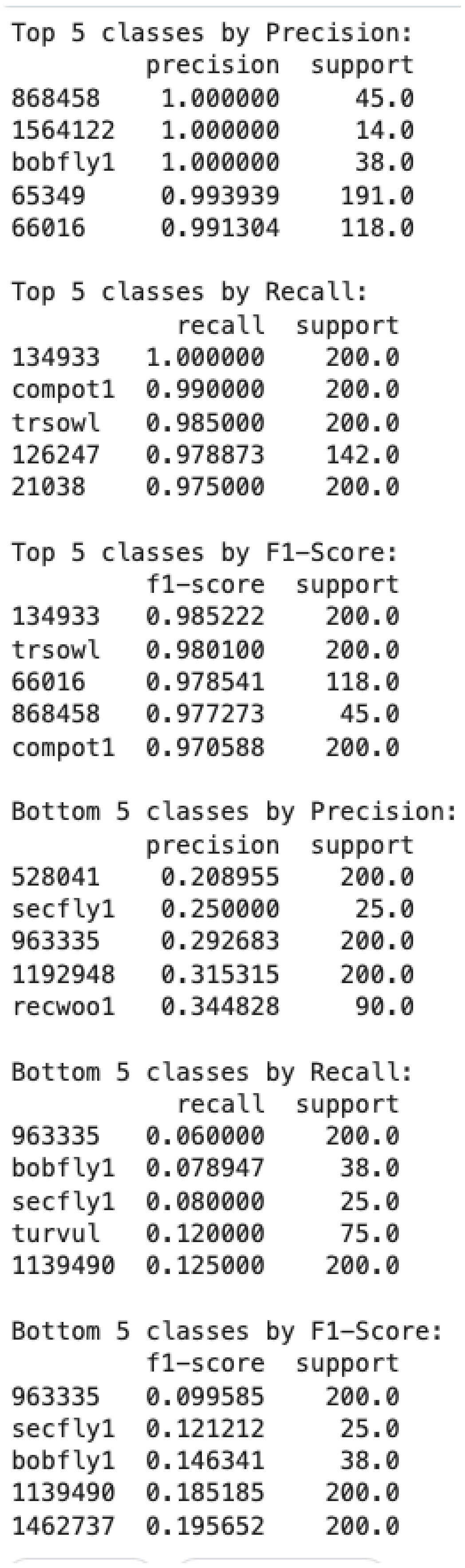
The first model attained a 65% precision and a 64% recall, along with a 64% f1 score. Below is the histogram of the distribution of each of these statistics by species, along with a list of the top and bottom classes for each.



```
Top 5 classes by Precision:
         precision  support
868458    1.000000     45.0
1564122   1.000000     14.0
66016     0.982759    118.0
134933    0.979798    200.0
roahaw    0.969543    200.0

Top 5 classes by Recall:
           recall  support
126247   0.992958    142.0
trsowl   0.990000    200.0
gretin1  0.980000    200.0
868458   0.977778     45.0
compot1  0.975000    200.0

Top 5 classes by F1-Score:
        f1-score  support
868458  0.988764     45.0
134933  0.974874    200.0
66016   0.974359    118.0
trsowl  0.968215    200.0
roahaw  0.962217    200.0

Bottom 5 classes by Precision:
         precision  support
548639    0.000000     18.0
963335    0.198630    200.0
1192948   0.221774    200.0
secfly1   0.250000     25.0
1462737   0.268293    200.0

Bottom 5 classes by Recall:
           recall  support
548639   0.000000     18.0
528041   0.030000    200.0
bobfly1  0.078947     38.0
secfly1  0.080000     25.0
bugtan   0.141304     92.0

Bottom 5 classes by F1-Score:
         f1-score  support
548639   0.000000     18.0
528041   0.054299    200.0
secfly1  0.121212     25.0
bobfly1  0.133333     38.0
963335   0.167630    200.0
```

The second model attained a 71% precision and a 70% recall, along with a 71% f1 score. Below is a histogram of the distribution of each of these statistics by species, along with a list of the top and bottom classes for each.





```
Top 5 classes by Precision:
         precision  support
868458    1.000000     45.0
1564122   1.000000     14.0
bobfly1   1.000000     38.0
65349     0.993939    191.0
66016     0.991304    118.0

Top 5 classes by Recall:
           recall  support
134933   1.000000    200.0
compot1  0.990000    200.0
trsowl   0.985000    200.0
126247   0.978873    142.0
21038    0.975000    200.0

Top 5 classes by F1-Score:
         f1-score  support
134933   0.985222    200.0
trsowl   0.980100    200.0
66016    0.978541    118.0
868458   0.977273     45.0
compot1  0.970588    200.0

Bottom 5 classes by Precision:
         precision  support
528041    0.208955    200.0
secfly1   0.250000     25.0
963335    0.292683    200.0
1192948   0.315315    200.0
recwoo1   0.344828     90.0

Bottom 5 classes by Recall:
           recall  support
963335   0.060000    200.0
bobfly1  0.078947     38.0
secfly1  0.080000     25.0
turvul   0.120000     75.0
1139490  0.125000    200.0

Bottom 5 classes by F1-Score:
         f1-score  support
963335   0.099585    200.0
secfly1  0.121212     25.0
bobfly1  0.146341     38.0
1139490  0.185185    200.0
1462737  0.195652    200.0
```

Model 1 and Model 2 exhibit some similarities and differences in performance. Both models show high precision for classes like *868458* and *1564122*, with Model 1 having more diversity in its top 5 precision classes, including *roahaw*, while Model 2 includes *bobfly1*, which achieves perfect precision but struggles with recall. Model 2 outperforms Model 1 in recall, with classes like *134933* achieving 1.0 recall, while Model 1's top classes have slightly lower recall values, such as *trsowl* at 0.99. When it comes to F1-score, Model 1 tends to have slightly higher values overall, with *868458* achieving a 0.988 score compared to Model 2's *134933* at 0.985. In terms of bottom performance, Model 1 shows a complete lack of precision for class *548639* (0.0), while Model 2's worst precision class, *528041*, shows a score above 0.2. Both models struggle with certain classes, with Model 1 displaying very low recall for *548639* and *528041*, while Model 2 has lower recall for *963335* and *bobfly1*, which has relatively high precision. Overall, for the extremes of each statistic, Model 2 excels in recall, while Model 1 performs better in precision and F1-score, with both models facing challenges with classes of smaller support.

Conclusive Remarks

Our project showed that training a neural network to classify birds based on short audio clips, even in biodiverse environments, is possible. Usingyamnet embeddings and a simple neural network architecture, we achieved a test accuracy of around 65% with 206 classes, which is promising given the complexity of the task.

We faced challenges related to imbalanced data and small sample sizes for certain species, but regularization techniques such as dropout and early stopping helped improve the model's generalization. Our results suggest that with more data and stronger models, this approach could become even more effective and serve as a valuable tool for biodiversity research.

Personally, this was the first major machine learning task we embarked on, and thus we were presented with the full array of challenges when working with data in regard to preparation for modeling. We had to learn how to manage audio files and how to convert them into something that could be analyzed, figure out how to best filter the data to make the model run as well as possible, and overcome the limitations of RAM and CPU in order to make the model usable.

In terms of future research, the most promising way would be to see if yamnet is the best way to approach this type of problem or if there are other and more specific pretrained models already out there that can be used. If we had more knowledge of data manipulation or any way to speed up the preprocessing, we feel the model would be much better and as such that is another area where we feel a lot of improvement can be made. However, for a first try, this was a great project and we are glad we were given the opportunity to try it out.