Transfer Learning with Semi-Supervised Dataset Annotation for Birdcall Classification

Anthony Miyaguchi[1,*], Nathan Zhong[1], Murilo Gustineli[1] and Chris Hayduk[1]

[1]*Georgia Institute of Technology, North Ave NW, Atlanta, GA 30332*

Abstract

We present working notes on transfer learning with semi-supervised dataset annotation for the BirdCLEF 2023 competition, focused on identifying African bird species in recorded soundscapes. Our approach utilizes existing off-the-shelf models, BirdNET and MixIT, to address representation and labeling challenges in the competition. We explore the embedding space learned by BirdNET and propose a process to derive an annotated dataset for supervised learning. Our experiments involve various models and feature engineering approaches to maximize performance on the competition leaderboard. The results demonstrate the effectiveness of our approach in classifying bird species and highlight the potential of transfer learning and semi-supervised dataset annotation in similar tasks.

Keywords

Transfer Learning, Dataset Annotation, BirdNET, Bird-MixIT, CEUR-WS

1. Introduction

The BirdCLEF 2023 competition [1] focuses on classifying bird species in 10-minute-long soundscapes recorded in various parts of Africa as part of the LifeCLEF lab [2]. We need to label each 5-second interval in the test soundscape with the probability that each 264 target species is present. The training dataset comprises 16,941 tracks spanning 192 hours of audio tagged by the label of the target species present. While we have metadata about the species in the tracks, we do not have concrete labels on which audio sections contain calls. The dataset also poses significant challenges given the species distribution, with many having only one or two examples.

We focus our efforts on utilizing existing off-the-shelf models as the basis for our classification models. We hypothesize that we can reduce environmental noise and cross-talk using a sound-separation model and re-purpose knowledge in the embedding space learned by a domain-specific convolution neural network. These models can address some of the most significant impediments in the competition, including the need for labeled data through semi-supervision.

CLEF 2023: Conference and Labs of the Evaluation Forum, September 18–21, 2023, Thessaloniki, Greece

*Corresponding author.

acmiyaguchi@gatech.edu (A. Miyaguchi); nathanzhong@gatech.edu (N. Zhong); murilogustineli@gatech.edu (M. Gustineli); chayduk3@gatech.edu (C. Hayduk)

https://acmiyaguchi.me (A. Miyaguchi); https://linkedin.com/in/murilo-gustineli (M. Gustineli)

0000-0002-9165-8718 (A. Miyaguchi); 0009-0003-9818-496X (M. Gustineli)

2. Embedding Space and Transfer Learning

BirdNET [3] classifies 48kHz 3-second audio clips into 3337 classes using a convolutional neural network trained on scaled spectrograms computed by the short-time Fourier transform. The classes are primarily composed of bird species but include non-bird classes such as environmental noise or human voices. We obtain embedding tokens by taking the values at the second-to-last layer of the model, preceding a fully connected logit layer. The embedding maps the audio in the time domain into a vector space $\mathcal{R}^{320}$ that roughly preserves distances between points. We use embedding tokens as features in a supervised machine-learning model to take advantage of the compact representation of the audio data. We show a clear separation between the embedding tokens by running them through a dimensional reduction technique in figure 1. Clustering supports a hypothesis that we effectively utilize representation learned by BirdNET in new contexts.

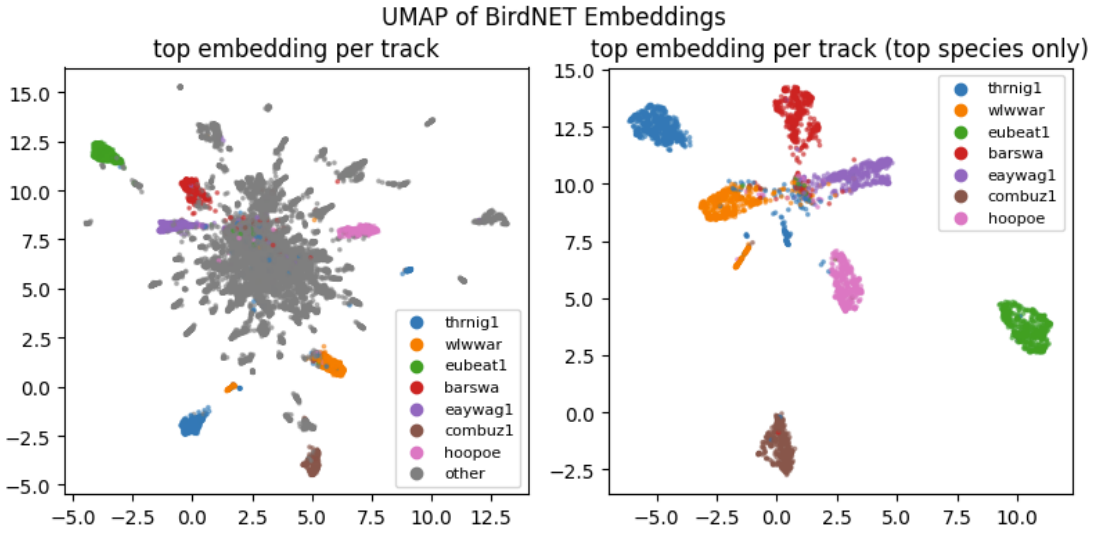

Figure 1: We demonstrate the clustering properties of the BirdNET embeddings by projecting them into $\mathcal{R}^2$ via UMAP [4] The projection preserves Euclidean distance in 2D space. We take the embedding token across each track with the most significant probability across the BirdNET prediction vector and assign it a positive label. The left plot shows clustering across the seven most common species in the training dataset. The right plot demonstrates a clear separation between the seven most common species.

3. Semi-Supervised Dataset Annotation

We propose a process to derive an annotated dataset to fit data to traditional supervised learning algorithms. First, we chunk audio within the training examples so that no track lasts 3 minutes by recursively splitting the tracks until they are smaller than our threshold, padded to the nearest third second with additive white noise. Chunking the audio solves the problem of batch

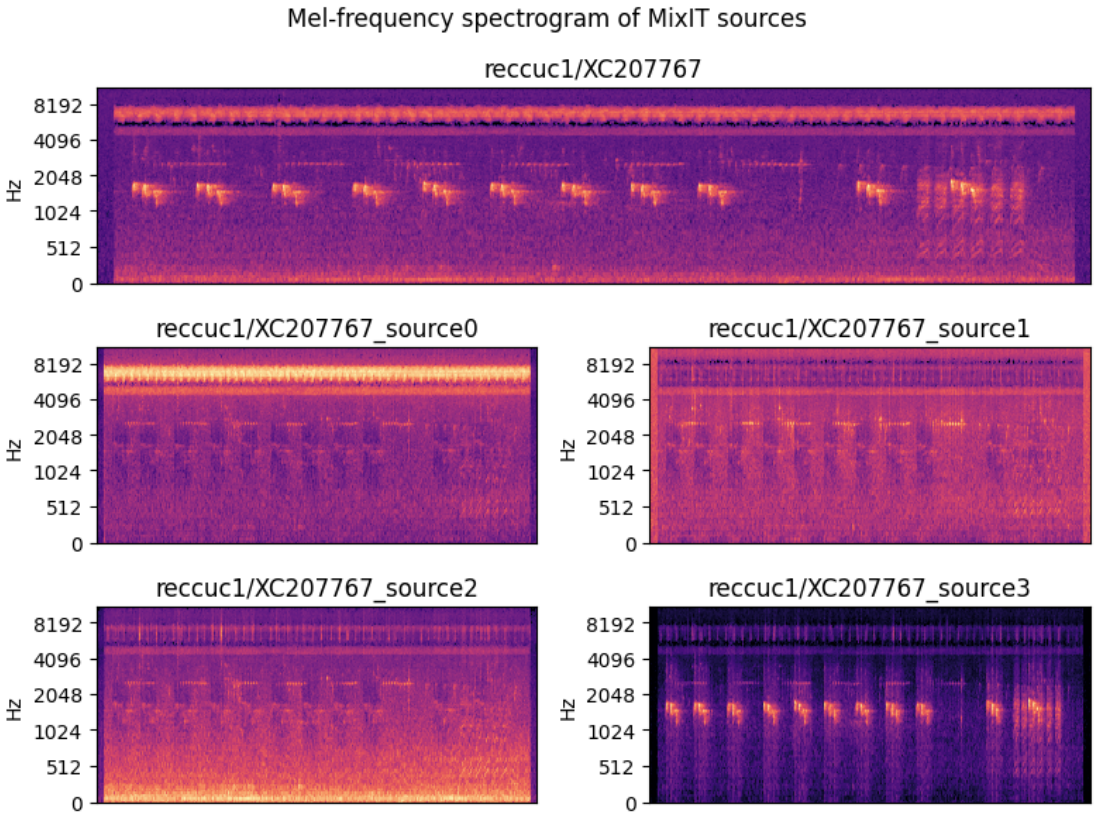

Figure 2: Bird-MixIT has improved the precision of downstream classifiers in experimental settings. We demonstrate separation across entities in the mel-spectrogram of track XC207767 containing a Red-chested Cuckoo. We observe separating three sound signatures into sources 0, 1, and 3. Source 2 is an amalgamation of the other sound signatures, an artifact of the model separating into four channels. An automated process should choose source 3 containing the species of interest.

processing skew introduced by several examples that are longer than 30 minutes. We assume the upper bound on the track length is sufficient to model temporal dependencies. We use Bird-MixIT [5] to isolate environmental noise and bird vocalizations. We then process each of the tracks using BirdNET to extract an embedding vector in $\mathbb{R}^{320}$ and a prediction logit vector in $\mathbb{R}^{3337}$ for every 3-second interval over a 1-second sliding window. Each interval is labeled with top prediction labels and the energy of the original track. We save the results from each track to disk and consolidate them into a parquet dataset. See table 1 for an example row of this process. We only have to pay for the expensive process of running TensorFlow models once by processing the audio training examples before fitting models.

We had four versions of the embedding dataset (`emb`). In `emb_v1`, we encountered missing entries and Docker permission problems. We fixed this in `emb_v2` but incorrectly mapped input to the wrong model layer. Our first usable dataset was `emb_v3`, which fixed previous issues and addressed the skew of long tracks by recursively chunking them. `emb_v4` includes all possible

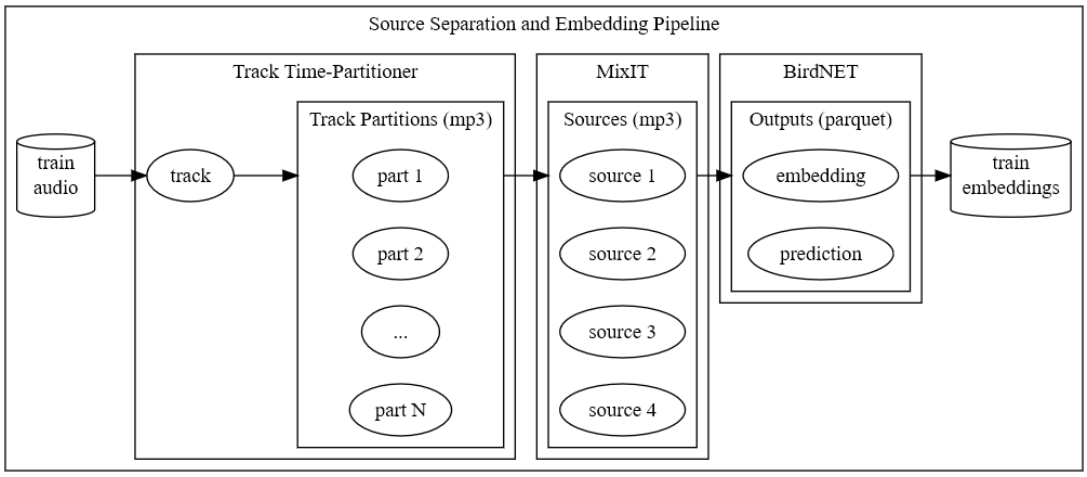

Figure 3: We use Luigi to coordinate a processing pipeline spanning days on an n2-standard-16 compute instance. We prevent processing skew across workers by recursively training audio. The audio is then source separated and embedded, resulting in a parquet file per audio chunk. We consolidate the parquet files into the final dataset.

3-second intervals at a 1-second resolution. With the embedding and prediction vectors in a dataset, we apply a series of heuristics and feature engineering for our final models.

Table 1
Example output row from the embedding dataset (`emb_v4`). The `emb_3` dataset is about 20 GB, and the `emb_v4` dataset is about 60 GB.

column name	value
species	grecor
track_stem	XC629875_part003
track_type	source0
track_name	grecor/XC629875_part003_source0.mp3
embedding	[0.6731137633323669, 1.1389738321304321, 0.6284520626068115, ...
prediction_vec	[-8.725186347961426, -7.3204827308654785, -9.82101821899414, ...
predictions	[{0, 3026, Tetrastes bonasia_Hazel Grouse, hazgro1, 0.02235649898648262}, {1,...
start_time	75
energy	0.01598571054637432

4. Implementation and Workflow

We split our workflow into training and inference. Our training pipeline runs on the Google Cloud Platform (GCP), while inference runs in a Kaggle notebook optimized for offline usage.

We implement a shared Python package on GitHub. [1] The package defines environment

[1]Implementation at github.com/dsgt-birdclef/birdclef-2023

dependencies to run the training and inference workflows. It contains helper PySpark code [6], wrappers around BirdNET, and utilities for manipulating audio samples into matrices representing sliding windows. We also define a workflow package containing Luigi [7] scripts that implement dataset processing and annotation. We build Docker images for BirdNet and MixIT and integrate modified versions of the canonical inference script into our data pipeline as per figure 3.

The inference pipeline is composed of three notebooks. The package sync notebook downloads the shared Python package with all dependencies into a local directory. The model sync notebook similarly downloads serialized models and weights from object storage. The final inference notebook runs offline after attaching the package and model sync notebooks as data sources. We read the soundscapes, split them into chunks, obtained BirdNET embeddings, and computed predictions submitted to the competition. See the appendix for the source code.

5. Experiments

We run experiments on the embedding dataset to maximize our performance on the public leaderboard. The feature engineering process of embedding tokens and prediction logits is part of the model-fitting process. We first focused on a baseline with minimal modifications to the embedding dataset and then worked toward overcoming the idiosyncrasies of the training dataset by more complex feature engineering. The input interval of BirdNET and the output interval of the competition do not match, so we had to consider this difference. The former expects 3-second intervals, while the latter expects 5-second intervals. Our two main approaches are to aggregate the output of models that represent 3-second intervals and to aggregate input to represent 5-second intervals.

While we use simple measures such as macro-precision and accuracy to assess models against the derived dataset in hyperparameter searches, we note that they did not reflect performance against the public leaderboard. The results of our derived datasets and models are summarized in table 3 and table 4, respectively.

Table 2
Example output row from the `post` v6 dataset on the left and v7 on the right.

column name	value
track_stem	XC116777
track_type	source1
start_time	0
primary_label	ratcis1
metadata_species	[ratcis1]
probability	0.348693
embedding	[0.64499, 0.45046, 0.36006, ...
next_embedding	[0.86384, 0.81126, 0.26452, ...
track_embedding	[0.67923, 0.53318, 0.36459, ...

column name	value
track_stem	XC213642
track_type	original
start_time	55
species	afmdov1
embedding	[1.76885, 0.83265, 0.49710, ...
prediction_vec	[-14.44950, -12.09912, -15.69485, ...

Table 3
An overview of changes in the post-processed dataset (`post`).

Dataset	Source	Description
post v1	emb v3	Simplified dataset where embedding tokens come from the channel with the highest number of positive classifications against the baseline model. Multi-label generated by averaging pairs and triplets of embedding tokens together.
post v2	emb v4	Tokens are now the average of the first and third tokens of each 5-second interval. We generate current, next, and track embeddings using only source-separated tracks. Tokens are multi-labeled by the primary and secondary species associated with the track.
post v3	post v2	Augments above but the top-20 tokens in each species are averaged against random no-call tokens to simplify train-test splits.
post v4	emb v4	Same methodology as post v2 and v3. Multi-label is generated by confident baseline predictions filtered by plausible primary and secondary metadata labels.
post v5	emb v4	Same as post v4, but it fixes a modulo bug in previous averaged-token datasets.
post v6	emb v4	Drops notion of multi-label prediction. It uses the original track and the best source-separated track to increase the number of training examples.
post v7	emb v4	Adds logic to assign the primary label and no-call labels. It uses concatenation instead of interpolation for a 5-second interval token and includes the prediction logits for the current interval.

Table 4
A summary of experiments and their associated datasets.

Model	Dataset	Description	Public Score	Private Score
Logistic Regression	emb v3	Baseline	0.78541	0.68369
MLP	emb v3	Baseline	0.74014	0.62283
XGBoost	post v1	Multi-label one-vs-rest strategy. Interpolated token pairs and triplets.	0.79068	0.68181
XGBoost	post v1	Same as above, but weighted samples and native multi-label training.	0.78829	0.68053
XGBoost	post v5	Current interpolated-token.	0.7692	0.65937
XGBoost	post v5	Current and track interpolated-token.	0.7489	0.63059
XGBoost	post v3	Current, next, and track interpolated-token.	0.75049	0.63877
XGBoost	post v3	Current and next interpolated-token.	0.76484	0.65346
XGBoost	post v7	Current concatenated token.	0.75997	0.64414
XGBoost	post v4	Ensemble of best logistic regression and boost model.	0.75091	0.64242
Complement Naive Bayes	post v7	Current prediction logit softmax vector.	0.71093	0.59652

5.1. Baseline Model

The baseline model uses embedding tokens taken from the isolated track source with the highest energy, assuming that the loudest voice in the track is associated with the primary label. We label the tokens according to the primary label if the max probability of the prediction vector exceeds a threshold, e.g., 0.5; otherwise, we label the token as "no-call". We fit the data to logistic regression, multi-layer perceptron (MLP), support vector machine (SVM), and gradient-boosted decision tree (GBDT) classifiers. We use scikit-learn [8] for the first three of these models and the scikit-learn compatible interface against XGBoost [9] for the latter. When applicable, we perform a hyper-parameter search using sci-kit-optimize [10], which performs sequential optimization using Bayesian methods.

We hypothesize that classes cluster together in the low-dimensional embedding learned by BirdNET and that linear models can effectively learn to discriminate between new classes. Our baseline logistic regression classifier trained on the embedding tokens reaches a public/private score of 0.78/0.68, notably better than the starter Kaggle notebook using the Google Research Bird Vocalization Classifier with a score of 0.72/0.61. We find that SVM and GBDT are comparable to the logistic regression and that MLP models require significant tuning to reach good performance.

GBDT via XGBoost is our preferred model because it trains quickly with a GPU with relatively high predictive performance. While logistic regression has fewer parameters and performs just as well, training can be slow as the number of examples increases. In table 5, logistic regression takes 12x as long to train as XGBoost with GPU-based histogram binning.

Table 5
A comparison between fit and predict the time for various models fit on a GCP n1-standard-4 compute instance with a Telsa T4 GPU. We fit the post-v7 dataset, which has 255,372 rows.

Model	GPU	Fit time	Predict time
Logistic Regression (Newton-Cholesky)	No	59 min 17 s	1.5 s
SVC	No	90 min +	-
MLP	No	4 min 14 s	3.5 s
XGBoost (hist)	No	48 min 20 s	14.4 s
XGBoost (gpu_hist)	Yes	5 min	15.3 s
ComplementNB	No	4.2 s	1.5 s

5.1.1. Baseline Binary No-call Model

We explore and analyze the performance of a binary classifier to further our understanding of the embedding space. While we do not use this model directly in the competition, it helps quantify the quality of our automated labeling process.

We construct a binary dataset with positive and negative embedding samples. Positive signifies the presence of a birdcall in the sample, while negative denotes its absence. The distribution of positive and negative samples within the dataset is well-balanced, with 49.7% being positive and 50.3% being negative. About half of the available training data is empty across original and sound-separated tracks. A logistic regression classifier achieved an accuracy

of 0.88 on this binary dataset.

We create a second binary dataset from a subset of the first using the top three most common species, with 49.8% of samples being positive and 50.2% being negative. A logistic regression classifier is trained and predicted 1.0 accuracy on the smaller dataset. We hypothesize that this behavior is due to the large number of species in the dataset. The distribution of some species is highly skewed, as shown in Figure 4.

Furthermore, we classified audio embeddings of the `freefield1010` background soundscape dataset [11] using the logistic regression classifier. The classifier predictions are probability scores indicating the presence or absence of birdcalls in each sample. It predicted 77.3% of the samples as having no birdcalls (no-call) and 22.7% as having birdcalls (call). However, if the classifier's predictions were perfect, the no-call percentage should have been 1.0, as the entire dataset consisted only of background noise.

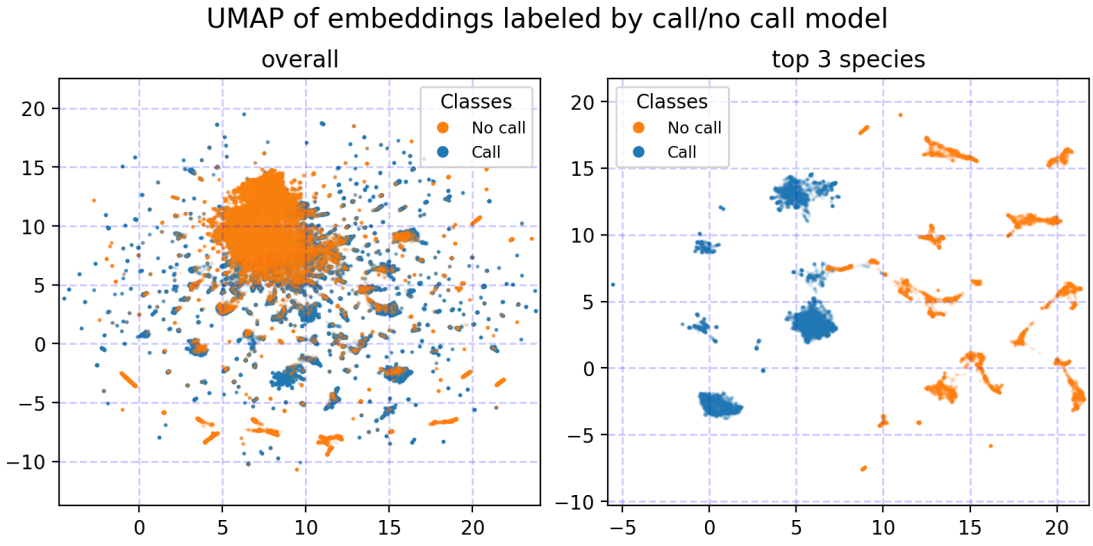

Figure 4: We demonstrate the clustering properties of the binary no-call embeddings by projecting them into $\mathcal{R}^2$ via UMAP. The clustering compares the binary dataset with the overall species versus the top-3 species.

5.2. Interpolated Embedding Models

We build upon the baseline embedding model by interpolating embedding tokens to synthesize new examples and labels. We assume that positive examples from each class tend to cluster together in the high-dimensional embedding space. We also assume that embedding space takes on a Euclidean geometry or admits some approximation. By interpolating examples, we hypothesize that the resulting coordinate lies between the clusters and, therefore, closer to the decision boundary of the sources.

We first use interpolation to address the alignment problem between the 3-second interval of BirdNET and the 5-second interval of the competition by generating a new feature. In addition,

we also construct a method that generates pairs and triplets of tokens sampled evenly across classes. These interpolated tokens are assigned multiple labels used in a multi-label classifier.

We experiment with interpolation to add contextual information to each example. For each token in a track from our dataset v_t, we generate the token that follows it directly in time v_{t+1} and the token that represents the entire track $\sum_{i=0}^{n} v_i$. We generate features by concatenating ($\oplus$) each of these tokens and evaluate performance relative to each other using the same set of labels.

$$\hat{y} \sim M_1(v_t) \tag{1}$$

$$\hat{y} \sim M_2(v_t \oplus v_{t+1}) \tag{2}$$

$$\hat{y} \sim M_3(v_t \oplus \sum_{i=0}^{n} v_i) \tag{3}$$

$$\hat{y} \sim M_4(v_t \oplus v_{t+1} \oplus \sum_{i=0}^{n} v_i) \tag{4}$$

Interpolation may provide some value, in particular around multi-labeling. We found that our model trained on a dataset does not perform worse than our baseline model while avoiding issues related to having a small number of training examples for the class. Most models trained using a form of interpolated tokens resulted in lower leaderboard scores. However, these models did not include interpolated pair and triplet tokens.

5.3. Concatenated Embedding Model

Another class of models that handles the time-interval discrepancy involves concatenating the embedding tokens. We must train a classifier that accepts input in $\mathcal{R}^{2x320}$. This model performs worse than the interpolated model. The increased dimensional of the underlying feature also increases the model fit time, which leads us to skip augmented feature sets.

5.4. Ensemble Embedding Model

Our last embedding model uses the best models from our baseline and interpolated embedding experiments. We train an XGBoost model trained on the outputs from the best classifiers. This model is likely affected by the quality of the training dataset and the differences in embedding token semantics.

5.5. Probability Logit Model

Our final experiment uses the outputs of the final logit layer in the BirdNET model to determine a class' presence directly. We generate a probability vector by taking a softmax of the logit layer. We ran into out-of-memory issues on our GPUs fitting XGBoost models using the probability vector in $\mathcal{R}^{3337}$ and found the scikit-learn logistic regression implementation needed to be faster. We fit the data using a Complement Naive Bayes classifier instead of a GBDT or logistic regression classifier. Naive Bayes assumption works well in this problem, where each feature is

treated independently toward the classification goal. It is also swift because it simply computes counts over features. We use the Complement Naive Bayes model to address the heavy skew in the class distribution but also find comparable performance across this family of classifiers. The predictive performance is far worse than the baseline, with a public/private score of 0.71/0.59.

6. Discussion

6.1. Semi-Supervised Annotation Quality

The data quality is an aspect of our training dataset that we would like to explore more deeply because the data quality affects the model's quality. We have built a training workflow that enables flexibility in labeling timestamps across the entire training audio to 1-second granularity. While we succeeded in our baseline transfer learning experiment, having a human-labeled test dataset independent of the competition leaderboard would be helpful. We can use ground-truth annotations to test how well an automated labeling process does at assigning primary, secondary, and no-call labels.

It would also be worth exploring human annotation in the sound separation process. While we have listened to several examples to judge the quality of sound separation, we need a method to quantify quality. In the case of sorting by the highest energy source, we may confuse a source with a significant amount of noise for the true birdcall just by the nature of higher entropy in the source. In the case of sorting by the highest number of matching classifications from an existing model, we may need to produce a classifier that can distinguish between noise and birdcall for rare classes. In this case, we continue to propagate uncertainty into the resulting labels, leading to poor performance, mainly when the number of examples is small.

We could introduce a metric to measure the resulting separation quality rigorously. We could create many positive birdcall clips and have a human determine which channel mainly contains the primary species. These clips let us see how well our automated channel selection process does at choosing the right channel based on human-generated labels. However, this metric would not allow us to determine the separation quality in degenerate cases where a single separated source contains multiple bird vocalizations.

6.2. Audio Source Separation

The MixIT source separation model plays a significant role in the embedding pipeline used for our experiments. One of the most significant benefits is noise suppression. We also relied on sound-separated channels to generate training examples and semi-supervised labels. Running an ablation study by evaluating the labeling process without access to the sound separation model would have been helpful.

We wanted to explore the 8-source model, which could have different performance characteristics than the 4-source model. However, we decided to ignore this model because the training examples generally have few distinct vocalizations, and creating more source channels increases the necessary disk space for the intermediate files.

We are also interested in the effect of the sound separation model during inference. However, we observed that the sound separation stage in training was a significant fraction of the

computation budget. There are also issues with different sample rates. BirdNET expects audio sampled at 48kHz, while MixIT expects audio at 32KHz. We did not attempt to integrate the model into this project because it would have put us over the submission time limit and required significant engineering effort to run inside the competition environment.

6.3. Embedding Space and Transfer Learning

Our experiments with interpolated embeddings in section 5.2 had mixed results with respect to the baseline embedding model. Including embedding context from other time intervals had substantially lower performance than our baseline. The new labeling process may have overshadowed potential positive effects. On the other hand, we saw a slight increase in our model performance when using the mean of pairs and triplets during model fitting. Further research is necessary to determine whether it is valuable to manipulate embeddings similarly to mix up [12] that interpolates between training examples in the time domain to increase and augment training data.

We want to experiment with the embeddings from another model, such as the Google Research Bird Vocalization Classifier. This classifier has seen more of the species in this competition than BirdNET. It would be helpful to see how these two embeddings compare, and we could try this out as another feature.

Sequential models could be helpful in the competition by capturing dynamics and imbuing contextual information between embedding tokens. The simplest model would be an auto-regressive linear model using an embedding from a single timestep to predict the next timestep optimized by a squared error loss. We made initial forays into attention-based sequence-to-sequence models to address the output time-interval issue, but we needed more time to complete our experimentation. Future work might explore data-driven methods like HAVOC [13] to analyze the dynamics of birdcall audio and their embeddings.

We would also like to explore the relationship between the sound-separated tracks and the embedding. The separation model is constrained so that the sum of the sources results in the original track. It would be interesting to verify a relationship between embeddings of various tracks by fitting a predictive model that takes tokens from each source to predict the embedding of the original track. This line of thought does not directly help with model performance on the final task, but it does help understand the nature of the classifier embedding space.

7. Conclusion

In summary, our approach leveraged the embedding space learned by BirdNET to address the representation and labeling challenges in the competition. We developed a pipeline that included sound separation with MixIT, extraction of embedding tokens using BirdNET, and the creation of annotated datasets. Our results showcased the competitive performance of our logistic regression baseline model as an effective method on unseen species and the comparative performance of various feature engineering work. Our approach demonstrated the potential of transfer and semi-supervised learning for bird species classification in soundscapes.

Acknowledgments

Thanks to the Data Science at Georgia Tech (DS@GT) club for hosting our Kaggle competition team. Thanks to DS@GT leadership for publicizing recruitment, particularly Krishi Manek as the Director of Projects. Thanks to Erin Middlemas and Grant Williams for their support and engagement as initial team members.

References

[1] S. Kahl, T. Denton, H. Klinck, H. Reers, F. Cherutich, H. Glotin, H. Goëau, W.-P. Vellinga, R. Planqué, A. Joly, Overview of BirdCLEF 2023: Automated bird species identification in eastern africa, Working Notes of CLEF 2023 - Conference and Labs of the Evaluation Forum (2023).

[2] A. Joly, C. Botella, L. Picek, S. Kahl, H. Goëau, B. Deneu, D. Marcos, J. Estopinan, C. Leblanc, T. Larcher, R. Chamidullin, M. Šulc, M. Hrúz, M. Servajean, H. Glotin, R. Planqué, W.-P. Vellinga, H. Klinck, T. Denton, I. Eggel, P. Bonnet, H. Müller, Overview of LifeCLEF 2023: evaluation of AI models for the identification and prediction of birds, plants, snakes and fungi, in: International Conference of the Cross-Language Evaluation Forum for European Languages, Springer, 2023.

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

[4] L. McInnes, J. Healy, J. Melville, Umap: Uniform manifold approximation and projection for dimension reduction, 2020. arXiv:1802.03426.

[5] T. Denton, S. Wisdom, J. R. Hershey, Improving bird classification with unsupervised sound separation, in: ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2022, pp. 636–640.

[6] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al., Spark sql: Relational data processing in spark, in: Proceedings of the 2015 ACM SIGMOD international conference on management of data, 2015, pp. 1383–1394.

[7] Luigi 2.8.13 documentation, https://luigi.readthedocs.io/en/stable/, 2023. Accessed: 2023-06-07.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[9] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, ACM, New York, NY, USA, 2016, pp. 785–794. URL: http://doi.acm.org/10.1145/2939672.2939785. doi:10.1145/2939672.2939785.

[10] T. Head, MechCoder, G. Louppe, I. Shcherbatyi, fcharras, Z. Vinícius, cmmalone, C. Schröder, nel215, N. Campos, T. Young, S. Cereda, T. Fan, rene rex, K. K. Shi, J. Schwabedal, carlosdanielcsantos, Hvass-Labs, M. Pak, SoManyUsernamesTaken, F. Callaway, L. Es-

tève, L. Besson, M. Cherti, K. Pfannschmidt, F. Linzberger, C. Cauet, A. Gut, A. Mueller, A. Fabisch, scikit-optimize/scikit-optimize: v0.5.2, 2018. URL: https://doi.org/10.5281/zenodo.1207017. doi:10.5281/zenodo.1207017.

[11] D. Stowell, M. D. Plumbley, An open dataset for research on audio field recording archives: freefield1010, arXiv preprint arXiv:1309.5275 (2013).

[12] H. Zhang, M. Cisse, Y. N. Dauphin, D. Lopez-Paz, mixup: Beyond empirical risk minimization, 2018. arXiv:1710.09412.

[13] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, J. N. Kutz, Chaos as an intermittently forced linear system, Nature communications 8 (2017) 19.

A. Kaggle Inference Sync Notebooks

A.1. birdnet-transfer-learning-package-sync

```
# %% [code]
# Clone the repo and all of the submodules; this way, we can include the extras
! if [[ -d birdclef-2023 ]]; then rm -rf birdclef-2023; fi
! git clone --recurse-submodules https://github.com/dsgt-birdclef/birdclef-2023.git
! pip download -d pip-packages --prefer-binary ./birdclef-2023 tensorflow==2.11.0
```

A.2. birdnet-transfer-learning-model-sync

```
from google.cloud import storage
from pathlib import Path

def download(client, bucket, prefix, relative):
    for blob in client.list_blobs(bucket, prefix=prefix):
        name = Path(blob.name).relative_to(relative)
        name.parent.mkdir(parents=True, exist_ok=True)
        blob.download_to_filename(name)
        print(f"downloaded {name}")

client = storage.Client(project="birdclef-2023")
download(client, "birdclef-2023", "data/models/birdnet-analyzer-pruned", "data")
download(client, "birdclef-2023", "data/raw/sound_separation", "data")
download(client, "birdclef-2023", "data/models/baseline", "data")
download(client, "birdclef-2023", "data/models/ensembles", "data")
```

A.3. birdnet-transfer-learning-inference

```
# %% [code]
# move the package into a temp directory so it's editable
! mkdir -p /kaggle/temp/birdclef-2023
! rsync -r \
    /kaggle/input/birdnet-transfer-learning-package-sync/birdclef-2023/ \
    /kaggle/temp/birdclef-2023/
! pip install \
    --no-index \
    --find-links=/kaggle/input/birdnet-transfer-learning-package-sync/pip-packages \
    /kaggle/temp/birdclef-2023

# %% [markdown]
# ## main submission code

# %% [code]
import pickle
import time
from functools, import partial
from pathlib import Path

import librosa
import numpy as np
import tensorflow as tf
```

```
import tqdm
from pyspark.sql import functions as F

from birdclef import birdie
from birdclef.utils import get_spark
from birdclef.data.utils import slice_seconds


repo_path = Path(
    "/kaggle/input/birdnet-transfer-learning-model-sync"
    "/models/birdnet-analyzer-pruned"
)
birdnet_model = birdnet.load_model_from_repo(repo_path)
embedding_func = birdnet.embedding_func(birdnet_model)
prediction_func = birdnet.prediction_func(birdnet_model)

model_prefix = "/kaggle/input/birdnet-transfer-learning-model-sync/models"
model_name = "acm-model-concat-v2"
model_path = Path(f"{model_prefix}/baseline_v2/{model_name}.pkl")
clf = pickle.loads(model_path.read_bytes())

# re-encode the classes properly for the inference script on xgboost
encoder_name = f"{model_name}_mlb"
le_path = Path(f"{model_prefix}/baseline_v2/{encoder_name}.pkl")
le =  pickle.loads(le_path.read_bytes())
clf.classes_ = le.classes_


# %% [code]
def prepare_embedding(X, use_next=True, use_global=True):
    # (120*2, sr*3)
    # the current tokens
    X_current = X.reshape(-1, 2, X.shape[-1]).mean(axis=1)
    X_res = X_current

    # the next token
    if use_next:
        X_next = np.roll(X_current, shift=1, axis=0)
        X_next[0] = X_current[0]
        X_res = np.concatenate([X_res, X_next], axis=1)

    # the global or track tokens. we average over every 2-minute interval
    if use_global:
        X_global  = X_current.reshape(-1, 12*2, X_current.shape[-1]).mean(axis=1)
        X_global = np.repeat(X_global, 12*2, axis=0)
        X_res = np.concatenate([X_res, X_global], axis=1)

    return X_res

def prepare_embedding_concat(X, **kwargs):
    return X.reshape(-1, X.shape[1]*2)

def run_inference(path, embedding_func, prediction_func, clf, sr=48000, **kwargs):
    y, sr = librosa.load(path.as_posix(), sr=sr, mono=True)
```

```
    X = slice_seconds(y, sr, seconds=3, step=1)
    # drop every 4th/5th index, so we're not processing more than we need to
    # ,First pad the resulting slices by 2
    X = np.pad(X, ((0, 2), (0, 0)))
    # then reshape it
    X = X.reshape(-1, 5, X.shape[-1])
    # Now drop the last 2 seconds of each 5-second frame
    X = X[:, [0,2], :].reshape(-1, X.shape[-1])
    assert X.shape == (120*2, sr*3), X.shape

    emb = embedding_func(X)[0]
    prob = clf.predict_proba(prepare_embedding_concat(emb, **kwargs))
    assert prob.shape == (120, len(clf.classes_)), (prob.shape, len(clf.classes_))
    rows = []
    for ts, probs in zip(range(0, 600, 5), prob):
        row = dict(
            row_id=f"{path.stem}_{ts+5}",
            **dict(zip(clf.classes_, np.around(probs, 6).tolist()))
        )
        rows.append(row)
    return rows


test_path = Path("/kaggle/input/birdclef-2023/test_soundscapes")
rows = []
timings = []
for path in tqdm.tqdm(test_path.glob("*.ogg")):
    start = time.time()
    rows += run_inference(
        path, embedding_func, prediction_func, clf, use_next=True, use_global=True
    )
    timings.append(time.time() - start)

avg_time_sec = np.mean(timings)
est_time_min = avg_time_sec*200/60
print(
    f"took {round(avg_time_sec,2)} seconds per loop, "
    "estimated {round(est_time_min,2)} minutes"
)

# %% [code]
# normalize the output schema with the sample
spark = get_spark()
sample_submission_df = spark.read.csv(
    "/kaggle/input/birdclef-2023/sample_submission.csv",
    header=True,
    inferSchema=True
)
rows_df = spark.createDataFrame(rows)
submission_df = rows_df.select(sample_submission_df.columns).toPandas()
submission_df.to_csv("submission.csv", header=True, index=False)
```