Chirps and Choices: Deep Learning Models for Bird Call Classification

By: Eloho Okoloko

Introduction

For this project, I explored how deep learning can be used to classify bird calls by analyzing their audio spectrograms. My work focused on two main tasks: first, a binary classification model to distinguish between the species “daejun” and “spotow”; and second, a multiclass classification model to identify any one of 12 bird species commonly found in the Seattle area. I built, trained, and evaluated custom convolutional neural networks (CNNs), paying special attention to how the data was preprocessed, how the models performed, and how interpretable the results were.

Theoretical Background

To build these models, I leaned on the core ideas behind deep learning, especially convolutional neural networks (CNNs), which are particularly good at finding patterns in data that has a spatial structure, like images or spectrograms.

At a high level, a CNN processes data through a series of layers. It starts with an input layer, which takes in the spectrogram (in my case, a 2D image showing how frequencies change over time). Then, it passes that input through convolutional layers, where filters slide across the spectrogram and pick up on local patterns, like changes in frequency or repeating rhythms in a bird’s call. After each convolution, there’s an activation function (I used ReLU) that introduces non-linearity so the model can learn more complex features. I also added pooling layers to reduce the size of the data while keeping the important parts. Dropout helped prevent overfitting by randomly turning off neurons during training. Toward the end of the network, I used global average pooling and fully connected (dense) layers, which pulled together everything the model had learned and made the final prediction.

For the binary classification task, I used a sigmoid function to output a probability between 0 and 1. For the multiclass task, I used softmax, which gives a probability for each of the 12 classes. I trained the models using binary cross entropy and I used the Adam optimizer because it adjusts the learning rate automatically and is generally reliable for deep learning tasks. I also added early stopping so that training would stop if the validation loss stopped improving, this helped avoid overfitting and saved time. To evaluate my models, I used standard metrics like accuracy, precision, recall, and F1-score, and I looked at confusion matrices to see where the model was getting confused. All of this gave me a solid framework for building models that could learn from audio patterns in bird calls and generalize to new examples.

Methodology

About the Data

The dataset I used for this project came from the Bird Call competition, which was originally based on recordings from Xeno-Canto, a global, crowd-sourced archive of bird sounds. The full archive is massive, around 36 GB of audio, but for this project, a curated subset was provided. It included only high-quality recordings from 12 different bird species. These were then converted into Mel spectrograms and stored in an HDF5 file.

Choosing the Species

For the binary classification task, I intentionally selected “daejun” and “spotow” as my two target species. I first looked through the dataset to make sure they had similar numbers of samples, because imbalanced data can throw off model performance. I also wanted to pick two birds with strong calls. I did this by visually inspecting the spectrograms of all 12 species and found the best two. For the multiclass classification, I didn’t filter anything out, I went with all 12 species.

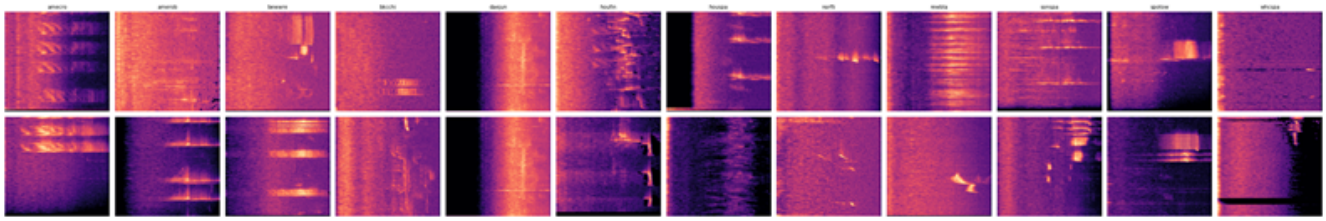

Figure 1: Spectrogram of all bird species

Preparing the Data

The audio clips had already been converted to Mel spectrograms, which basically turn sound into a visual representation of how the frequency content of the signal changes over time. But even though they were formatted consistently, the spectrograms still varied in length, especially along the time axis.

I realized the model wasn’t learning because the spectrograms weren’t properly normalized. Some looked completely blank, especially quieter recordings, and the model was stuck at around 50% accuracy. I fixed this by normalizing each spectrogram individually using its own min and max values, which revealed the important frequency patterns.

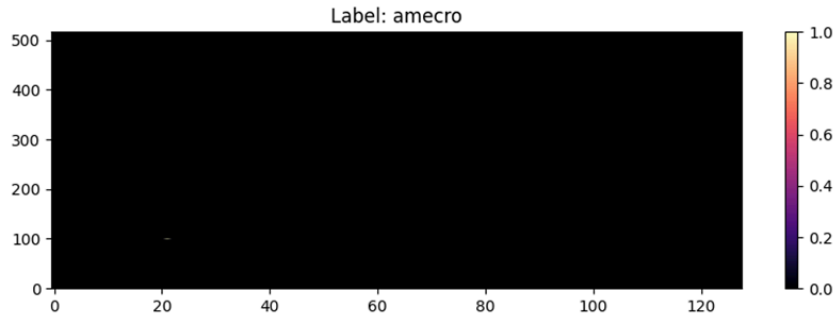

Figure 2: Blank Spectrogram due to bad normalization

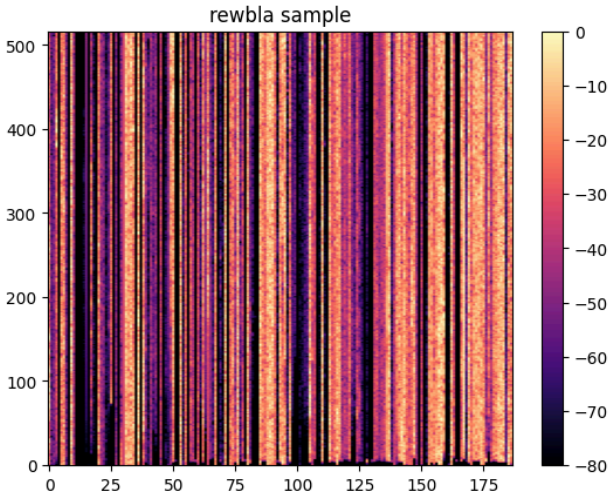

Figure 3: Properly normalized spectrogram

Results

Binary Classification

At first, the binary model wasn't learning at all. Accuracy hovered around 50%, and the loss barely changed. It felt like the model was just guessing which, as good as a coin flip. That's when I discovered the normalization issue. Once I adjusted the preprocessing pipeline to normalize each spectrogram individually, things finally started to move in the right direction. After that change, the training accuracy began to improve steadily, and validation accuracy followed. The model finally had something meaningful to learn from. I used early stopping to avoid overfitting, and by the end of training, I had a model that performed well not just on the training and validation sets, but also on the test data.

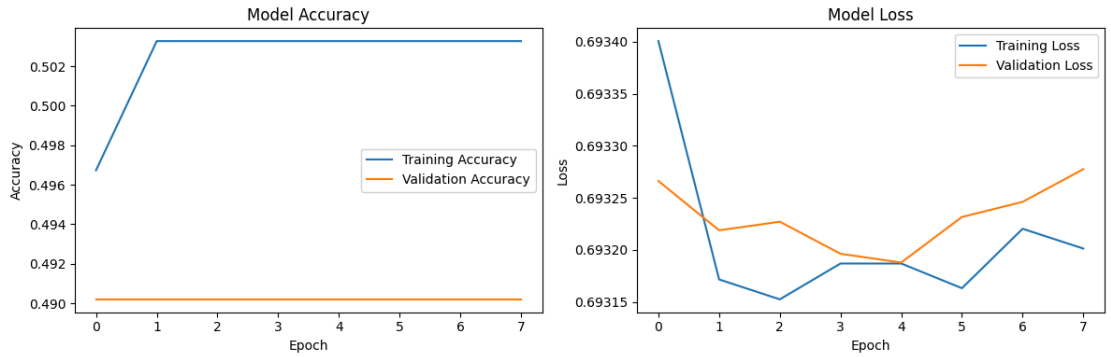

Figure 4: Binary classification training accuracy/loss - Iteration 1

Figure 5: Binary classification training accuracy/loss - Iteration 4

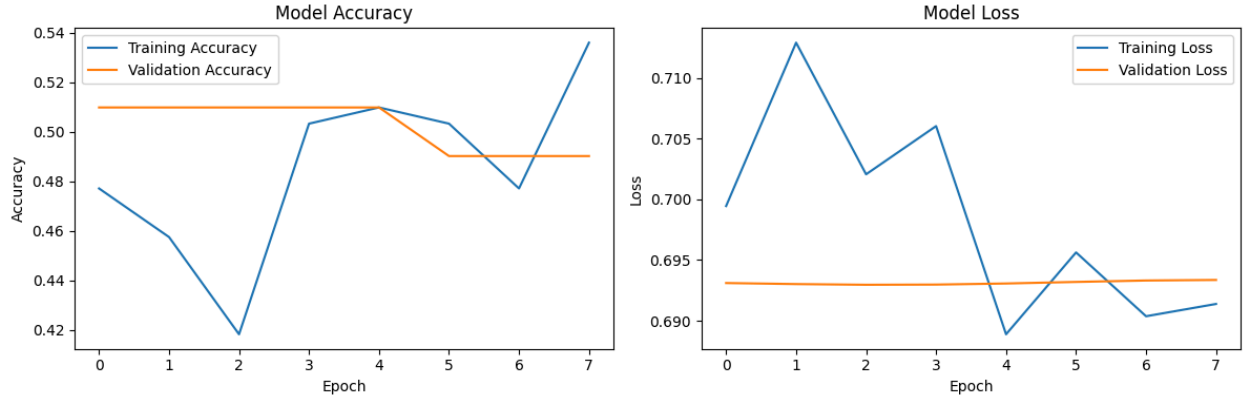

Figure 6: Binary classification training accuracy/loss - Iteration 5

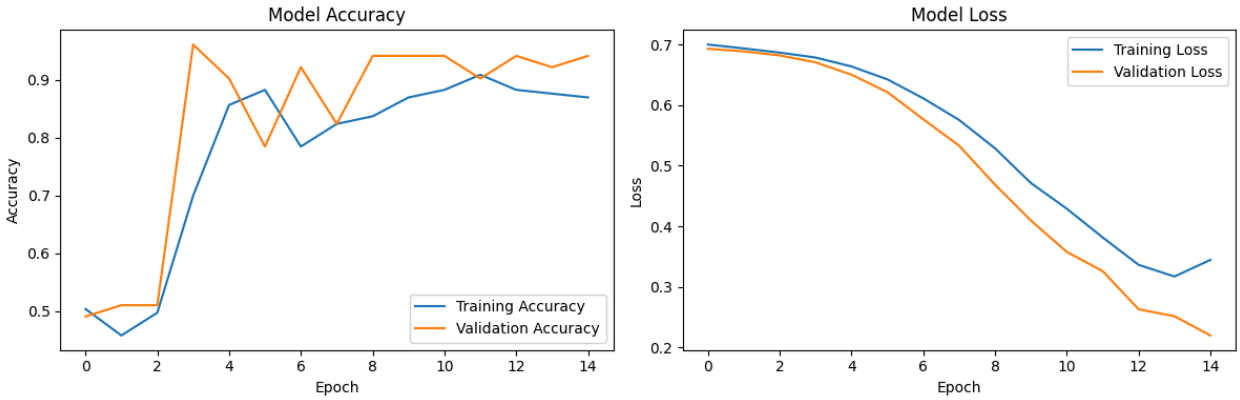

Figure 7: Binary classification training accuracy/loss - Final Iteration

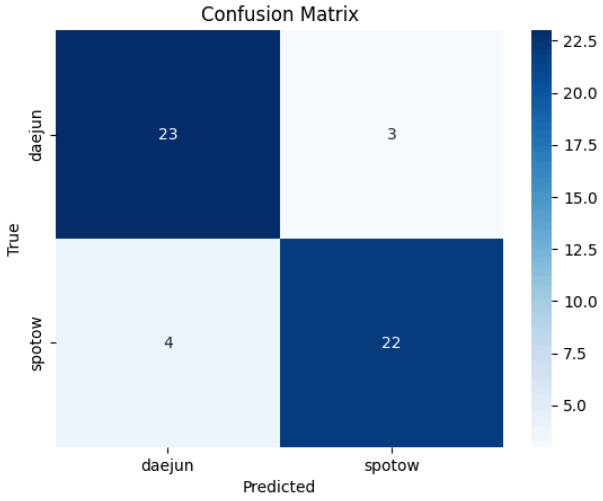

Figure 8: Confusion matrix for binary classification on Test Dataset

Multiclass Classification

The multiclass task was more challenging. I ran two training iterations. The first one showed some improvement, but not enough. The second run, I tuned the hyperparameters and it made a huge difference. Validation accuracy steadily improved. By the end of training, validation

accuracy peaked at 96%, and the model was producing high F1-scores across most classes.

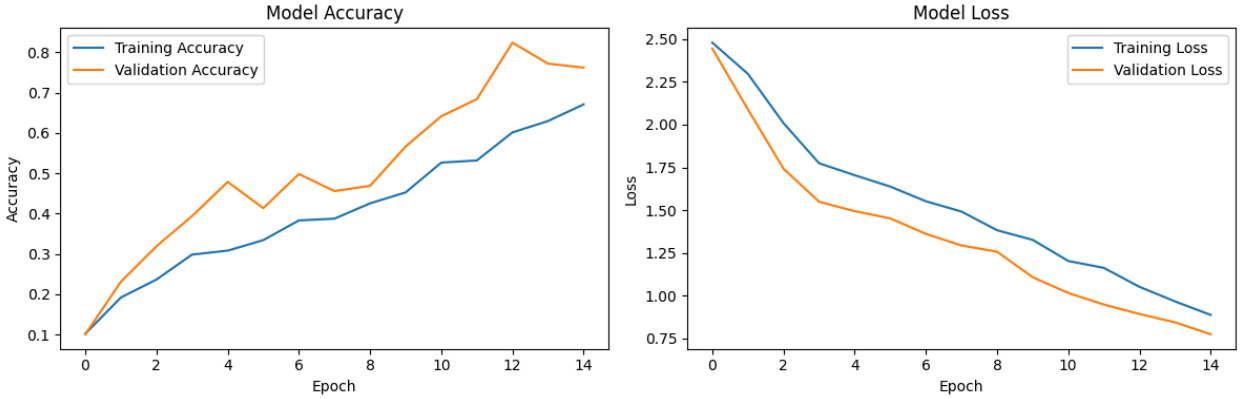

Figure 9: Multiclass classification training loss

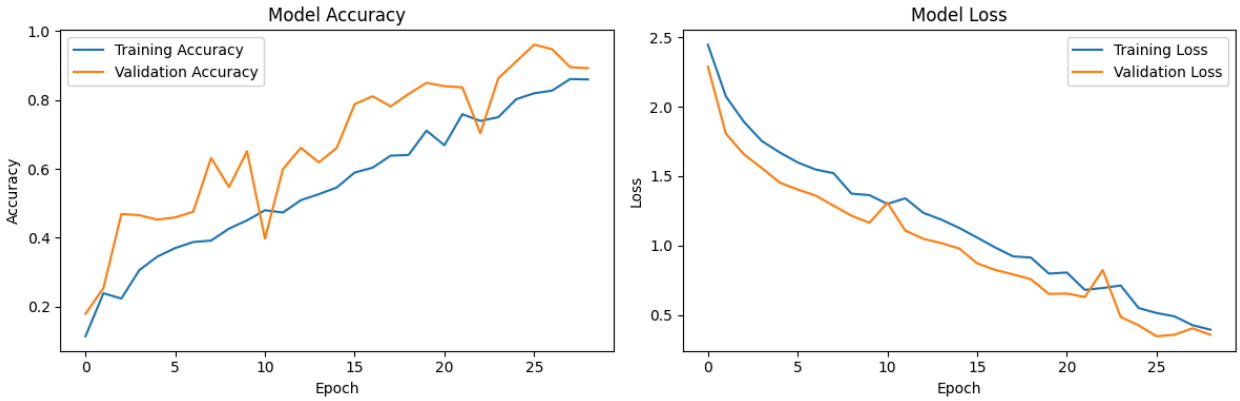

Figure 10: Confusion matrix for multiclass classification

External Test Clips

Once I had finished training and validating my models, the final challenge was predicting three mystery bird calls from raw mp3 audio clips. These weren't included in the training or validation sets, and they didn't come preprocessed like the rest of the dataset. To prepare the clips, I wrote a new preprocessing function to convert each mp3 file into a Mel spectrogram. After generating the spectrograms, I normalized each one individually and padded them to match the shape expected by my model. Then I passed them through my final multiclass classifier to get predictions.

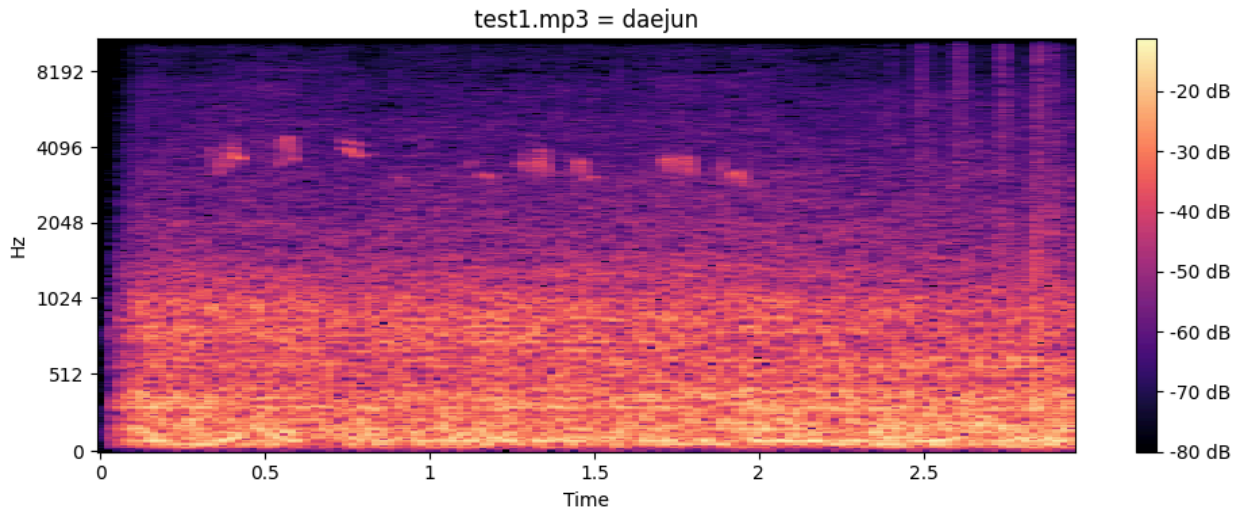

Figure 11: External Data 1 Prediction

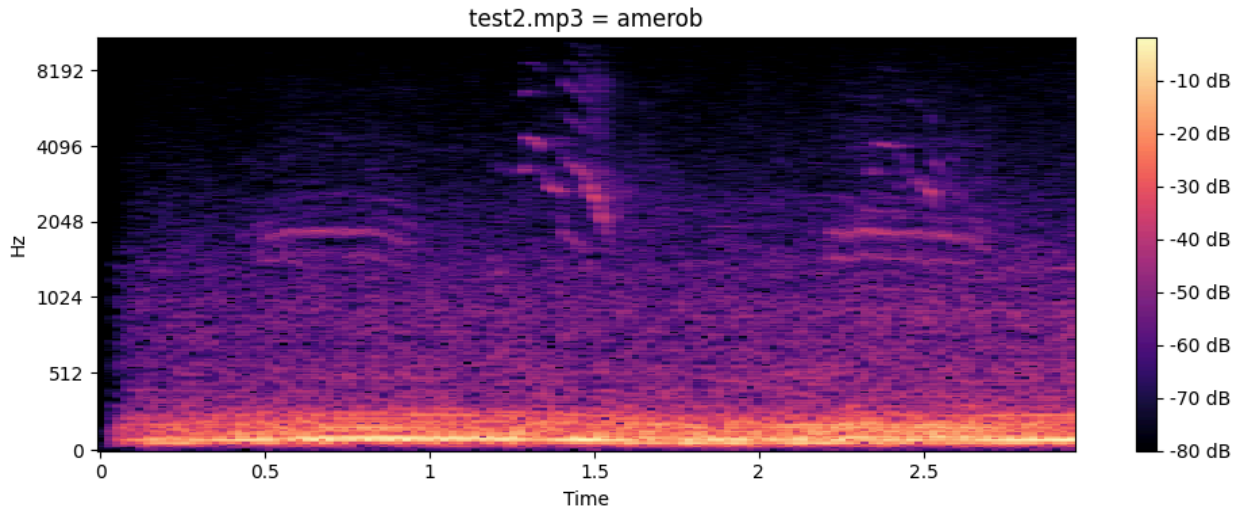

Figure 12: External Data 2 Prediction

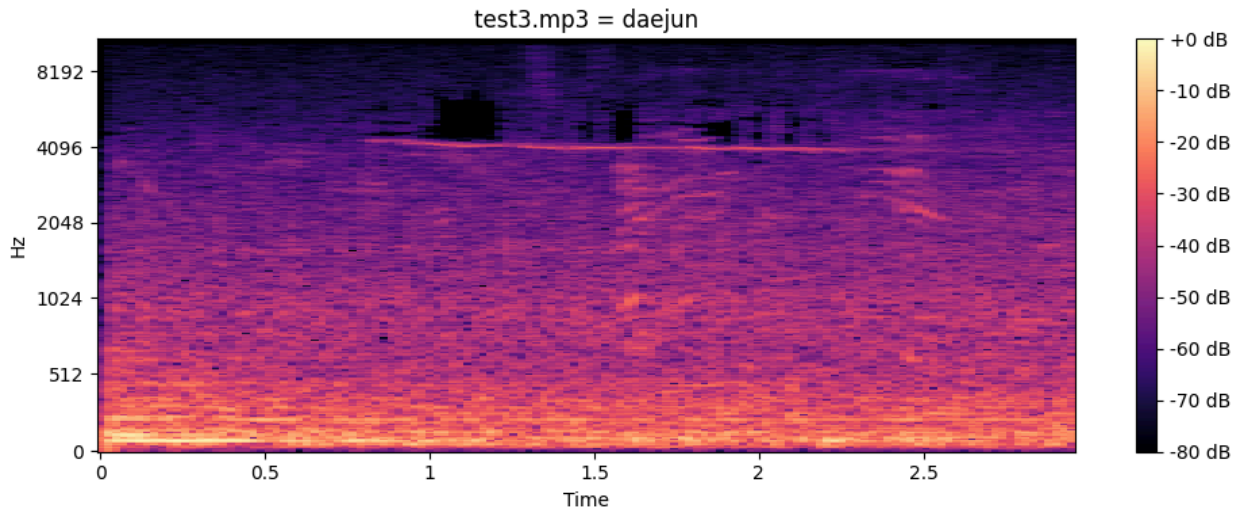

Figure 13: External Data 3 Prediction

Clip 1 had a low-confidence prediction for "*daejun"*, and other species like "*amerob"* and "*whcspa"* weren't far behind. The spectrogram had scattered energy across different bands, which could mean overlapping background sounds or more than one bird. Clip 2 was much more straightforward. The model strongly predicted "*amerob"* with 74% confidence, and the spectrogram showed a clean, consistent pattern that matched what I'd seen from "*amerob"* samples during training. Clip 3 was tricky. *Daejun* came out on top again, but the prediction confidence was low (35%) and "*amerob"* and "*spotow"* also scored fairly high. The spectrogram had multiple frequency ranges active at once, which made me think there may have been more than one bird in that clip.

Discussion

Several key challenges and decisions affected the performance of both the binary and multiclass models. In the binary classification task, the model initially failed to learn. Accuracy hovered around 50%, and the loss remained flat. After inspecting the inputs, I discovered that many spectrograms were low-intensity and appeared almost blank, which made it difficult for the model to extract meaningful features. To fix this, I normalized each spectrogram individually using its own minimum and maximum values. This adjustment revealed the structure in the frequency data and allowed the model to start learning. I also padded the spectrograms to ensure that all inputs had the same shape, which enabled consistent batching during training. Once these preprocessing steps were applied, the model began to improve and achieved good results on both the validation and test sets.

For the multiclass classification task, I conducted two training iterations. In the first iteration, I used proper padding and applied per-spectrogram normalization to ensure that all inputs were

consistent and informative. These preprocessing improvements allowed the model to begin learning effectively, and validation accuracy reached around 76%. However, the model showed signs of overfitting, and I wanted to see if I could push accuracy higher. In the second iteration, I focused on tuning the training process. I increased the number of training epochs from 15 to 30 to give the model more time to learn, and I added dropout regularization to reduce overfitting. These changes helped the model generalize better and maintain stable learning, though performance gains were more incremental.

The external test clips added another layer of complexity. These were raw mp3 audio files, unlike the preprocessed spectrograms used during training. This required building a new preprocessing step where I converted the audio into Mel spectrograms before normalizing and padding them. After running predictions through the multiclass model, I found that one clip had a high-confidence result, while the other two were more ambiguous, likely due to overlapping calls or background noise. Despite the added noise and variability, the model still produced reasonable predictions, which showed it was able to generalize beyond the training data.

Looking ahead, there are several ways the model could be improved further. One direction would be to tune hyperparameters, even more that I did, such as learning rate, batch size, dropout rate, and kernel size. This could be done using grid search or random search to explore combinations and find more optimal configurations. In terms of architecture, adjusting the number of filters in each convolutional layer or increasing the size and depth of dense layers might help the model learn more complex patterns.One of the main limitations of this project was the training time, especially for the multiclass classification task. Because I was training on all available samples across 12 species, each epoch took a long time to complete, and the full training process was slow. If I were to do this again, I would experiment with training on balanced subsets of the data to speed things up and make iteration more manageable.

Conclusion

This project showed that a well-preprocessed dataset and a basic CNN model can go a long way in classifying bird calls. Once I fixed the normalization and padding issues, the models were able to learn effectively. The binary model reached 87% accuracy, and the multiclass model hit over 80% validation accuracy. These results show that even with simple tools, it’s possible to build reliable models for identifying bird species by sound.This kind of work matters, especially for people in fields like conservation, wildlife monitoring. Tools like this can help automate the process of identifying bird calls in large datasets or in real-time monitoring environments which would usually take a lot of time and expertise to do manually. It could also help track species presence, migration patterns, or detect changes in biodiversity over time.

References

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: With Applications in R* (2nd ed.). Springer. https://www.statlearning.com/

Rao, R. (2020). *Xeno-Canto Bird Recordings Extended (A–M)* [Dataset]. Kaggle. https://www.kaggle.com/datasets/rohanrao/xeno-canto-bird-recordings-extended-a-m