Deep Learning for Predicting Bird Species Based on Audio Spectrograms with a Convolutional Architecture

Elling Payne, Seattle University

May 2025

1 Introduction

Identifying bird species is of interest to ecologists, biologists, and ornithologists who need to collect data about the species present in a given area. Amateur bird watchers may also enjoy the ability to quickly identify species they do not recognize on their own. The problem is not trivial due to the type of data that would be readily available in these scenarios, images and audio, rather than structured data such as specific relevant characteristics identified by an expert. The relationships between the available data and bird species are not straight forward or linear, and present challenges to many simple machine learning models. This presents an opportunity for deep learning models, which can perform well at identifying complex relationships. A number of recent studies have used deep learning architectures to predict bird species. Farman et al. used a convolutional neural network (CNN) to predict species based on images [11]. Wang et al. proposed using mel-spectrograms of audio samples along transformations of mel-spectrograms as inputs to a network with a recurrent, long-term short-term architecture [10]. Eichinski et al. found some success with training a model to predict among seven species based on limited examples [9]. They first employed a method to produce more labeled audio examples, and then applied a convolutional network on the spectrograms.

In this study, audio data for 12 bird species was taken from the Xeno-Canto Bird Recordings Extended (A-M) Kaggle dataset [7]. This data was obtained initially from the Xeno-Canto Archive [16]. The sound recordings were then processed into mel-spectrograms by Dr. Ariana Mendible of Seattle University [18]. The dataset contains 1,981 spectrogram samples for 12 species. An additional three audio clips were obtained from Dr. Mendible and similarly processed into spectrograms for prediction. Models are considered to solve two specific problems. The first is identifying species from among the 12 based on the spectrogram data. The second is to distinguish between the American Robin and the Dark-Eyed Junco.

2 Technical Background: Deep Learning

In many real-world problems, the true relationship between a response and predictive features will include complexities such as nonlinearity, non-monotonicity, or feature interactions. These often cause simpler models such as linear regression, support vector machines, and linear discriminant analysis to perform poorly. With enough data, deep learning models are able to learn incredibly intricate and non-intuitive relationships between the response and features, sometimes outperforming simpler models.

2.1 Basic Neural Networks (NNs)

2.1.1 Structure of a basic NN (Layers)

- *Input Layer*: Training data for a NN must be numeric. If a training example for a given problem has shape $(d_1, d_2, ..., d_s)$, then the input layer will have $m = \prod_{i=1}^{s} d_i$ nodes $\{x_1, ..., x_m\}$, each containing a value from the input.

- *First Hidden Layer*: The number of nodes in a hidden layer must be chosen to balance model bias and variance. Suppose K nodes are chosen for the hidden layer. During forward-feeding of the input values through the model, the values $\{A_1, ..., A_k\}$, called the activation values of the hidden nodes, are computed by transforming an affine function of the input layer. The coefficients and intercept of the affine function differ for each hidden node, and are called weights.

- *Additional Hidden Layers*: An arbitrary number of layers may be used. The activations of deeper layers are computed in the same basic way

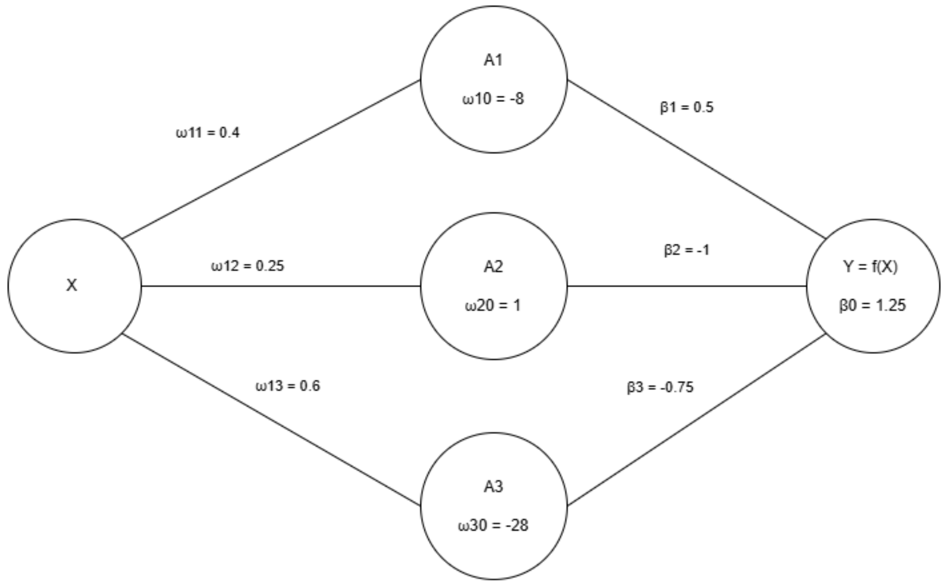

Figure 1: An example of a single-layer, feed-forward neural network with a single node in the input layer and a single node in the output layer, with three nodes in one hidden layer. Produced with draw.io [13]

as the first layer, but using the activations of the previous hidden layer as the inputs. Each layer will use a different set of weights and may use a different transformation of the affine function.

- *Output Layer*: The number of nodes in the output layer depends on the task. For numeric prediction, the output will contain as many nodes as predictions being made. For binary classification, the output will most likely contain one node for the positive classification probability. For multiclassification, the output will have one node to represent the probability of each response category. Similar to previous layers, the output nodes also apply an affine function to the the final hidden activation values. In the case of classification, a nonlinear transformation is then applied. In the output layer, the coefficients and intercepts used in the affine functions are called biases instead of weights.

2.1.2 Activation Functions

Common Hidden Layer Activation Functions:

In the hidden layers, activation functions introduce a nonlinear component to the model that results in greater flexibility. The affine functions of the previous layer effectively shift and stretch the activation function, so that different hidden nodes in the layer may capture different aspects of the relationship between response and features. The sigmoid function was popular, but the Rectified Linear Unit (ReLU) transformation is popular for computational efficiency when computing gradients.

$$\text{Sigmoid: } g(z) = \frac{1}{1+e^{-z}} \tag{1}$$

$$\text{ReLU: } g(z) = \begin{cases} 0 & \text{if } z < 0 \\ x & \text{otherwise.} \end{cases} \tag{2}$$

Common Output Activation Functions:

For numeric prediction, no activation function is necessary in the output beyond the affine combination of the final hidden activations. For classification, activation functions serve the purpose of transforming the numeric output of the final hidden layer into a set of valid probabilities for the response categories. Therefore, the choice of activation function depends on the type of response. For a binary response, a sigmoid function is used, just as in logistic regression. For classification with multiple response classes, a probability for each response category is required. The softmax function is used to ensure that the output probabilities sum to one, which tends to result in better predictions. Unlike the previously mentioned activation functions, the softmax function requires as input all the outputs from the previous layer. Suppose there are K categories in the response class. Then let $Z = \{z_1, ...z_K\}$ be the inputs to each output node, after applying the affine function based on the relevant biases.

$$\text{Softmax: } g_k(Z) = \frac{e^{z_k}}{\sum_{z\in Z} e^z}, k \in \{1, ..., K\} \tag{3}$$

2.1.3 Backpropogation (Optimization)

For simplicity, assume a model with a single output and arbitrary inputs, $f(X) = y$. If we consider all of the weights and biases as a single parameter vector, $\vec{\theta}$, then the problem of optimization becomes one of minimization over a loss function L, given a set of inputs, $\mathbb{X}$, and outputs, $\vec{y}$. That is,

$$\min_{\vec{\theta}\in\Theta} L_{\mathbb{X},\vec{y}}(\vec{\theta}) \tag{4}$$

The general method for solving this problem is *gradient descent*, where the model changes the weights and biases iteratively based on the gradient of the loss function. In practice, *stochastic gradient descent* (SGD), where each iteration uses only a sample of $\mathbb{X}$ and $\vec{y}$ to compute the loss function and the gradient, makes the process more computationally feasible. Various optimizers have been created to perform SGD, which each have their own advantages and disadvantages. On of the most common in use in Python [14] (TensorFLow [5] and Keras [6]) is RMSprop [4], which can automatically adjust the learning rate for gradient descent during training. For numeric responses, the loss function is usually based on the sum of squared errors for the training data. In classification problems, since the model predicts probabilities, the loss is instead typically calculated as the negative log-likelihood of the training data given the parameters $\vec{\theta}$. In the case of binary classification, this is known as *binary cross-entropy.* Suppose n training examples.
Squared Error Loss:

$$L_{\mathbb{X},\vec{y}}(\vec{\theta}) = \sum_{X\in\mathbb{X},y\in\vec{y}} (f_{\vec{\theta}}(X) - y)^2 \tag{5}$$

Binary Cross-Entropy:

$$L_{\mathbb{X},\vec{y}}(\vec{\theta}) = -\frac{1}{n} \sum_{X\in\mathbb{X},y\in\vec{y}} [y\log(f_{\vec{\theta}}(X)) + (1-y)\log(1-f_{\vec{\theta}}(X))] \tag{6}$$

For more than two response categories, *categorical cross-entropy* is used. once again, suppose n data examples and suppose K response categories. Let $x_i \in \mathbb{X}$ represent a feature example. Further, let $y_{ik} \in \mathbb{Y}$ indicate whether example i truly belongs to category k of the response class, and let $f_k(x_i)$ be the models prediction for the probability that the i^{th} example belongs to

category k. Then the loss for the categorical model is the following:
Categorical Cross-Entropy:

$$L_{\mathbb{X},\mathbb{Y}}(\vec{\theta}) = -\sum_{i=1}^{n}\sum_{k=0}^{K} y_{ik}\log(f_k(x_i)) \tag{7}$$

2.1.4 Use Case and Limitations, and Tuning

Because of their ability to use layers of linear and nonlinear interactions to identify hard to see but potentially important features of the training data, neural networks can uncover trends that elude other models. However, they typically require a lot of data to do so and often overfit because of the high number of model parameters relative to the available data. Beyond ensuring sufficient data, there are a number of hyper-parameters that affect model performance and overfitting. The most obvious include the number of epochs or iterations used for training, the batch size used for stochastic gradient descent, the number of hidden layers and the size of each hidden layer. In some situations, changing the activation function in a hidden layer, or changing the optimization protocol for gradient descent, may also affect performance. There are also regularization methods which may be applied to ameliorate overfitting in a particular layer. The most popular are dropout regularization, which chooses a random subset of outputs from the layer to ignore for each training iteration, and lasso or ridge regularization. Lasso regularization applies a penalty based on the norm of the vector of weights in a given layer. If dropout or lasso regularization are used, the dropout rate and penalty level λ, respectively, become tuning parameters.

2.2 Convolutional Neural Networks (CNNs)

In image processing, convolutional networks make use of convolutional filters to identify important, simpler features from the image, which can then be used to train a traditional network to make the relevant prediction. They can improve performance when a basic neural network would require far more nodes to learn the same information and potentially be more prone to overfitting as a result. This is one of the general reasons for using multiple hidden layers in a model as well, but it is especially true here.

2.2.1 Convolutional Filters

A convolutional filter in this context is a real-valued matrix of size $r \times q$. Suppose a data example of size $m{*}n$, where the elements represent a grayscale value. The filter passes over the example image and results in a new $(m-r+1)\times(n-q+1)$ sized convolved image, which is typically padded to retain the original size. Each element in the new image is a measure of how similar the corresponding region of the original data was to the filter. This is illustrated with the example below, where E is the result of applying convolution filter C to the data D:

$$C = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \tag{8}$$

$$D = \begin{bmatrix} e & f & g \\ h & i & j \\ k & l & m \end{bmatrix} \tag{9}$$

$$E = \begin{bmatrix} ae + bf + ch + di & af + bg + ci + dj \\ ah + bi + ck + dl & ai + bj + cl + dm \end{bmatrix} \tag{10}$$

2.2.2 Pooling

Pooling is a way of reducing the size of the outputs of convolutions. A pooling size and method is chosen, and the outputs of the convolutions are then partitioned based on the pooling sized. Each partition is pooled to a single value. Then the smaller outputs are passed to the next layer. The most common type is max pooling, in which the largest value within the pool area is selected. Because the largest value of the convolution corresponds to the section of the pool area that was most similar to the filter, max pooling has the nice property of reducing the importance of the exact location of key features in the input. This can result in better generalization. An example of max pooling is given below, where the matrix A is reduced to B based on a 2×2 max pooling area.

$$A = \begin{bmatrix} 1 & 3 & 4 & 2 \\ 2 & 1 & 1 & 0 \\ 0 & 3 & 0 & 2 \\ 5 & 2 & 3 & 2 \end{bmatrix}, \ B = \begin{bmatrix} 3 & 4 \\ 5 & 3 \end{bmatrix} \tag{11}$$

2.2.3 CNN Structure

A convolutional layer is made up of a group of different convolutional filters. Inputs to convolutional layers discussed to far have had only one channel, as is the case in greyscale images and sonic spectrograms. In other cases, such as color images, the input must be passed to the convolutional layer as multiple channels. The filters produce a different output for each channel, which increases the number of parameters to train. A typical approach to including convolutional layers is to include several convolutional layers of increasing size, with pooling layers after each. With this combination, each subsequent convolutional layer is able to learn a more complicated set of features that distinguish different training examples. After sufficient detection of useful features, the output of the convolutional layers is flattened and passed to a fully connected layer (or several) for the final prediction.

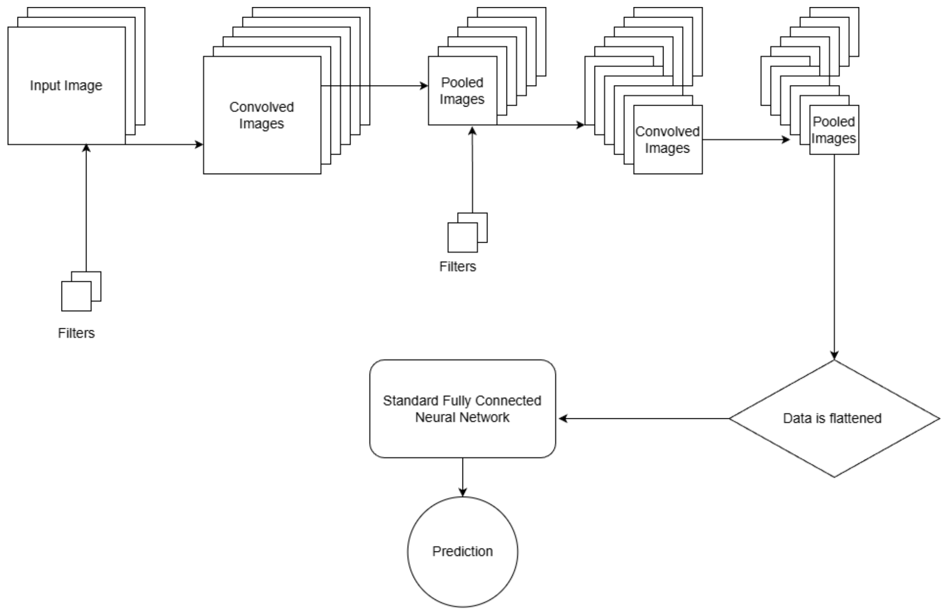

Figure 2: An example of a neural network with two convolutional layers and two pooling layers.

2.2.4 Tuning

The number of convolutional filters in a layer is an important parameter. More filters can learn a larger set of features, but result in much greater complexity. Filters also work best when sized similarly to the types of features present in the data. However, additional convolutional layers with pooling layers can also somewhat make up for a filter that is sized too small.

2.3 Recurrent Neural Networks (RNNs)

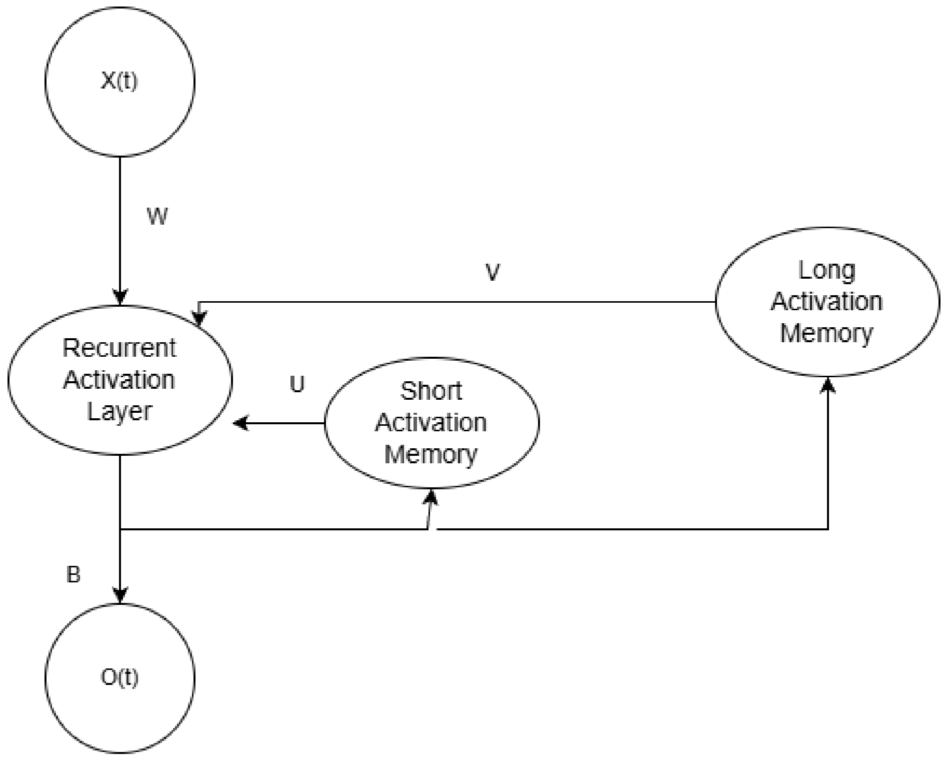

Figure 3: An example of a recurrent neural network structure with two memory tracks

One potential shortcomings of the models discussed so far is that they are not optimized to capture sequential relationships in data. While convolutional layers can improve a models ability to understand the local context in an input, this is not the same as understanding the order of the data. Recurrent network architectures seek to address this issue and therefore often excel at tasks processing documents or timeseries, including audio timeseries.

2.3.1 RNN Structure

In a recurrent structure, each training example X is really a sequence $\{X(t)|t \in \{1, ..., T\}\}$. To predict a response based on the sequence, the recurrent network iterates over each element of the sequence, producing a series of activation values and outputs indexed by t. The key point is that recurrent activation layers at each step in the sequence receive information not only about the current sequence element but also about one or more previous activation states. In the simplest form, the recurrent activation layer receives the current sequence element and the previous activation state. A more typical approach is to use multiple memory tracks that can identify both short and long-term relationships. Each activation memory track used by the network has its own set of additional weights to train. The recurrent network uses the same weights over the entire sequence for each memory track. Optimization and gradient descent are performed based on the final outputs of each example sequence. The choice of what memory tracks to use can be important to model performance. Recurrent networks are often fed back into fully connected standard layers for final prediction. Another common practice is to feed convolutional networks into recurrent networks to identify key features of the sequence beforehand.

3 Methods

3.1 Data Preparation

All audio clips were truncated to the first three seconds and then converted into mel-spectrograms. The librosa Python package [17] was used to process the mp3 files into spectrogram data. the final shape of the spectrograms was 517 timesteps by 128 frequency buckets.

3.2 Initial Exploration

Initially, several different architectures were considered and compared primarily based on plots of training and test accuracy vs epoch, as in figure 4. The first structure was a network with several two dimensional convolutional and pooling layers. The second was a model utilizing recurrent layers with long term and short term memory tracks. The third and fourth structures also utilized LSTM layers, but first passed the data through one-dimensional and two-dimensional convolutional layers, respectively.

3.3 CNN Tuning and Model Evaluation

The recurrent layers did not improve performance over the convolutional model during exploration, so a convolutional model was selected and tuned further. Then the filter size, number of filters, dropout rate, and training epochs were all tuned based on validation accuracy for both the binary and multiclass models. Stochastic gradient descent batch size was selected to provide 6-8 batches per epoch based on the size of training data, which appeared to perform sufficiently. The final models were evaluated primarily through a combination of accuracy, precision, recall, and F1-scores.

4 Results

4.1 Final Network Structure and Tuning

The same model structure was selected for both binary and multiple classification, excepting the output layer. A successful recurrent structure was not identified. The selected structure included four two-dimensional convolutional layers, each followed by a 2×2 max pooling layer. A 7×7 kernel size was selected during tuning, along with a count of 16 filters in the first layer. Subsequent layers each contained twice the number of filters as in the previous layer. After the convolutional, pooling layers, and flattening, a ten percent dropout rate was used for regularization before passing the data through a final hidden layer with 512 nodes. 40 epochs were selected for fitting the final binary classification model, while 35 epochs were used in training the final multiple classification model.

4.2 Binary Classification Performance

The binary model performs reasonable well, as table 1 shows performance well above that of random chance, as well as fairly balanced performance between the two response categories.

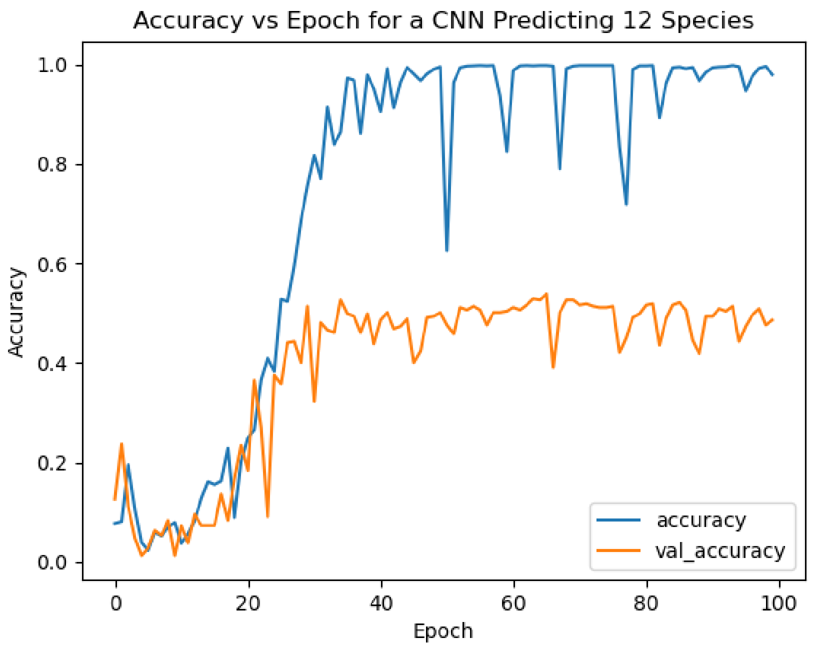

Figure 4: A plot of training and validation accuracy vs epoch for the final training of the multiclass model. These plots are useful for identifying the optimal number of training epochs.

Accuracy	Precision	Recall	F1-score
0.783	0.779	0.786	0.78

Table 1: Final metrics for the binary classification model based on the validation set. Precision, recall, and F1-score are reported as the average per class and rounded to three significant figures.

4.3 Multiple Classification Performance

While the multiple classification model performed better than random chance, performance was lacking. Table 2 shows that accuracy is less than 50%, while other metrics are even worse. The uneven performance between the response species is further illustrated by figure 5. The black-capped chickadee and northern flicker had F1-scores of zero, and the model fails to identify them. This could be due to these species being the least represented in the dataset. However, the model performs quite well for the American Crow, which is the third least represented species, suggesting there may be other factors at play. It may be that the American Crow, despite being underrepresented, has a very distinct sound.

Accuracy	Precision	Recall	F1-score
0.469	0.359	0.358	0.342

Table 2: Final metrics for the multiple classification model based on the validation set. Precision, recall, and F1-score are reported as the average per class and rounded to three significant figures.

4.4 External Audio Clips

Upon an initial airing of the three mp3 files obtained form Dr. Mendible, the author noted that the first two clips sounded as if there may be multiple birds present, while the third clip seemed to contain one distinct, long, and repeated call. In the first clip, the differing sound did not appear until around halfway through the recording. In the second clip, the author noted that two potentially distinct sounds seemed present form the start. However, the probabilities given by the model suggest a much greater confidence in the species assignment for the first clip. This might contradict the author's observations. However, this may result from the second bird call being truncated during preprocessing. The model's predictions lend some weak support to the idea that the second clip has two birds. The model has difficulty deciding between the house sparrow and the red-winged blackbird, which might suggest that features from multiple bird calls are present in the spectrogram. The model's low confidence in predicting the species of the third sample might suggest multiple birds are presents. However, it may also suggest that the bird in the clip is one of the underrepresented species, or

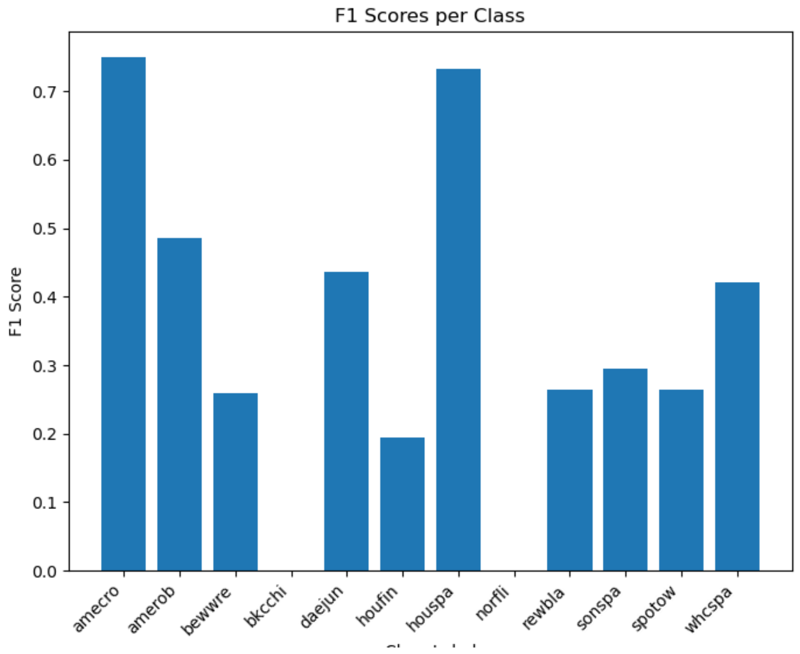

Figure 5: F1 scores for each of the response species.

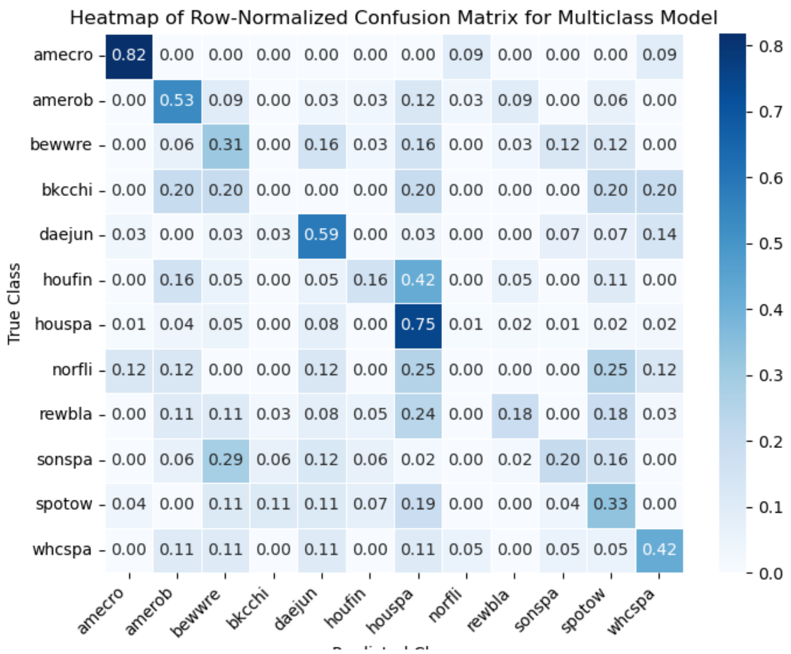

Figure 6: A heatmap of the normalized confusion matrix for the multiclass model shows which classifications are most common for each bird species.

may simply be due to the poor fitting of the model. It's worth noting that in both the second and third clip, one of the species that split the probability was the house sparrow. The house sparrow is overrepresented in the dataset, and many species are most commonly confused with the house sparrow by the model, shown in figure 6. Table 3 shows the model's predictions for each of the three clips.

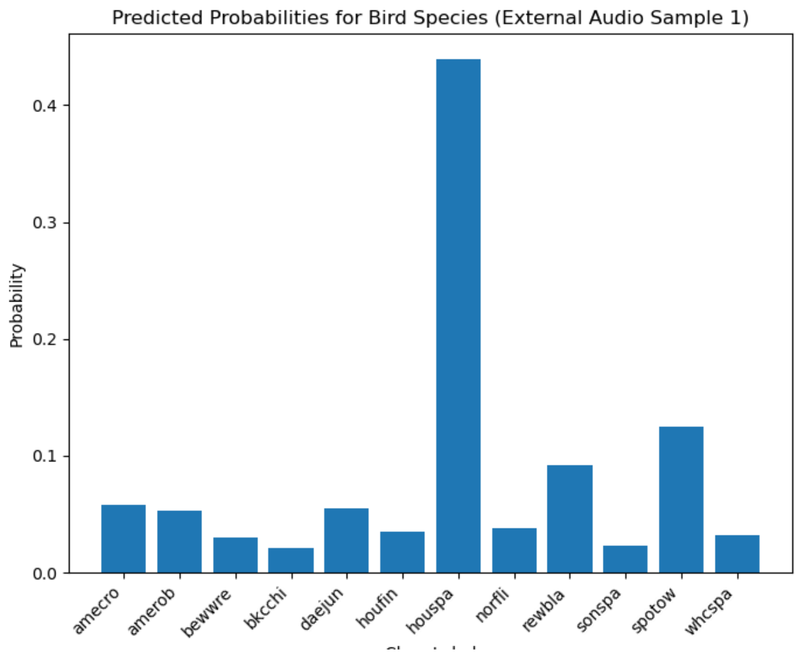

Figure 7: Predicted probabilities for the first external mp3 sample.

Clip 1	Clip 2	Clip 3
house sparrow	red-winged blackbird	spotted towhee

Table 3: Predictions for the external test mp3s based on the multiclassification model.

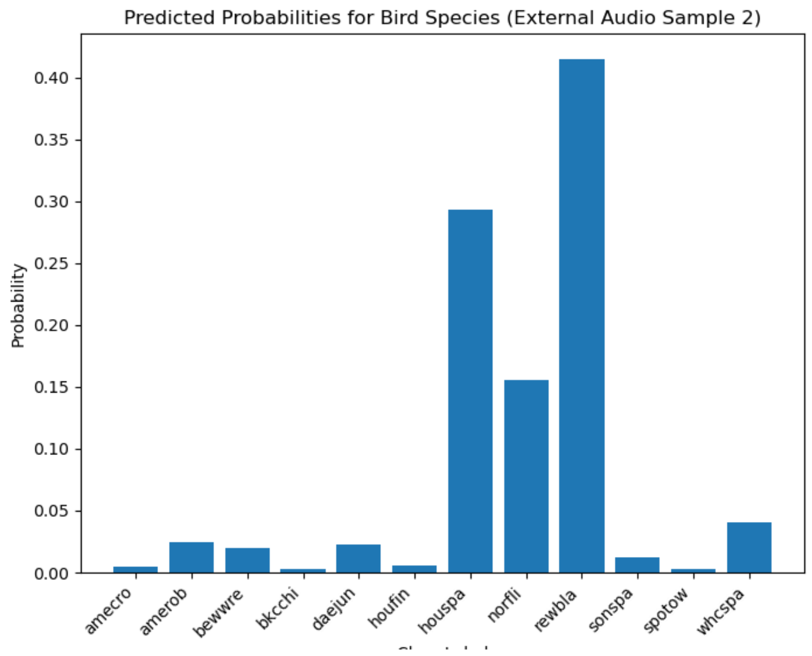

Figure 8: Predicted probabilities for the second external mp3 sample.

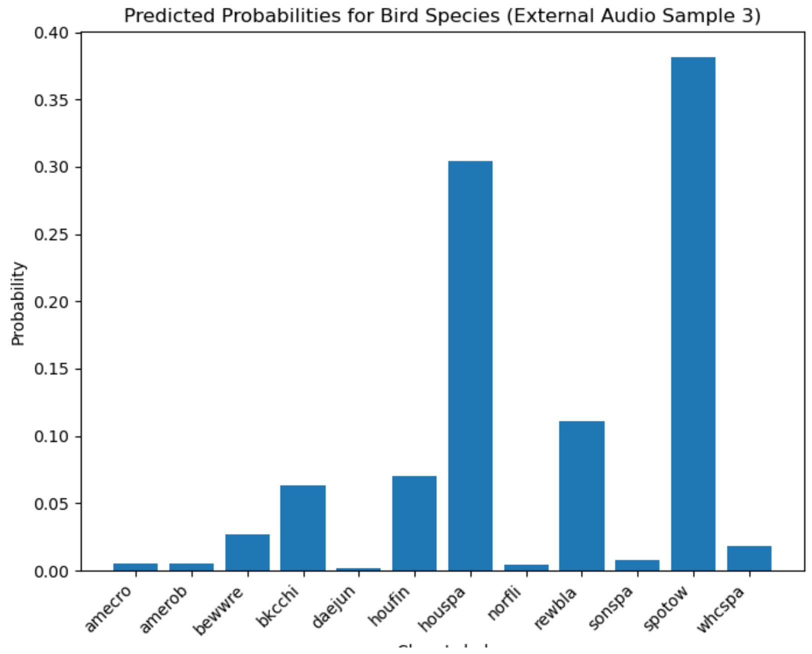

Figure 9: Predicted probabilities for the third external mp3 sample.

5 Discussion

5.1 Challenges and Limitations

During exploration and training of various models, several limitations became apparent, including the small size of the training set, the noisiness of the data, the number of tuning parameters to consider, and the computational expense of training deep neural networks. When training on a smaller subset of the data, the multiple classification methods attempted in this study achieved no accuracies above 25-30%. One the full dataset, the same multiclass model achieved a validation accuracy of 46.9%. This suggests that with more training data, the same model could likely perform much better. Crowd-sourced audio data also has a lot of noise, and bird calls do not occur in the same time windows, presenting a challenge for any candidate model. Finally, the computational cost of training the models during tuning, even on a much smaller subset of the data, meant that many architectures and hyper-parameter combinations were out of the author's reach during the time-frame of the study. For the multi classification, versions could take several minutes to train on the limited data set. Tuning for each the binary and multiple classification problems took several hours of run time.

5.2 Model Predictions

One of the more common misclassifications for many species is the House Sparrow. This makes sense since the House Sparrow was overrepresented in the training and testing data. However, figure 6 also suggests some more interesting misclassifications. The Song Sparrow is often mistaken for the Bewick's Wren, and the House Finch is more often classified as the House Sparrow than itself. Looking at figures 10 and 11, one might note that the calls of both the Song Sparrow and Bewick's Wren have spectrograms characterized by sharp upward points and vertical lines. While the similarities between figures 12 and 13 are less obvious, the spectrograms of both calls appear to contain similar vertical zigzags in the peaks. The fact that the house finch is a often misclassified while the House Sparrow is never classified as the House Finch may suggest that the misclassification results from a combination of similarity in their calls and the overrepresentation of the House Sparrow.

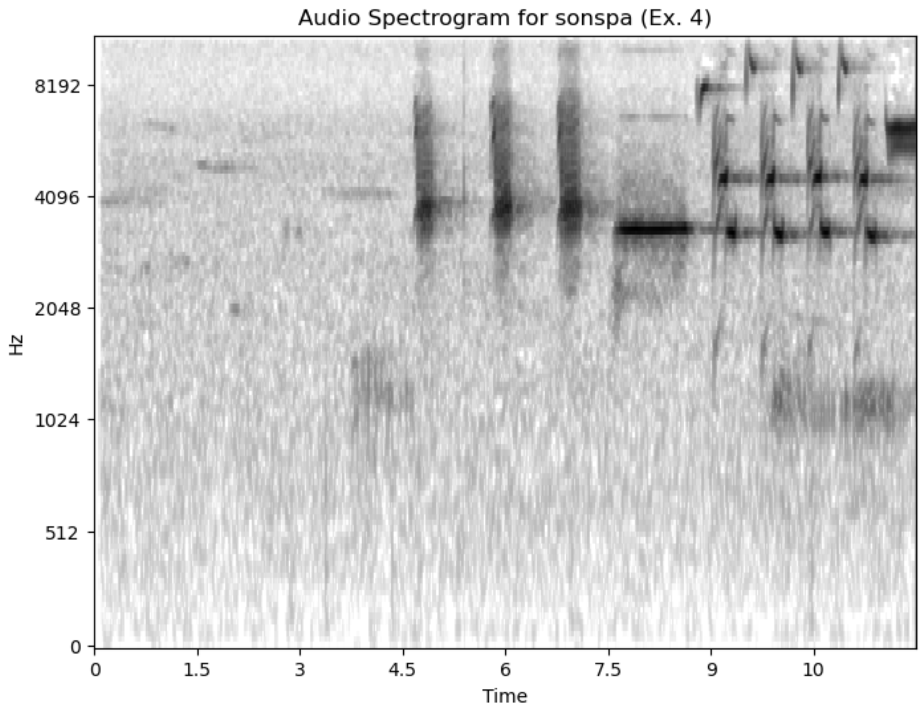

Figure 10: An example sonic spectrogram for the Song Sparrow

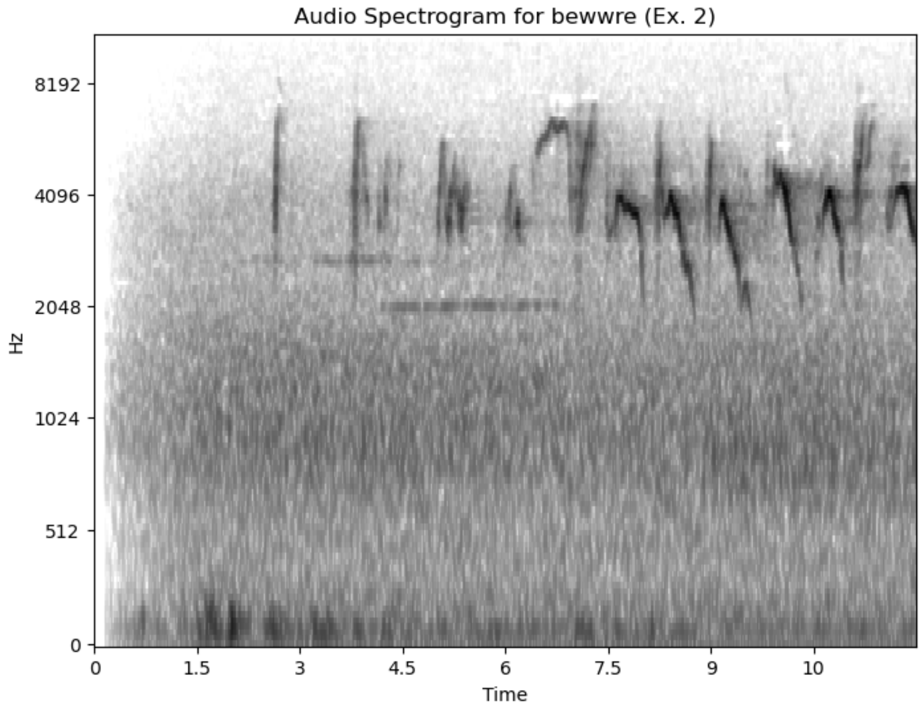

Figure 11: An example sonic spectrogram for the Bewick's Wren. Note the similarities to figure 10

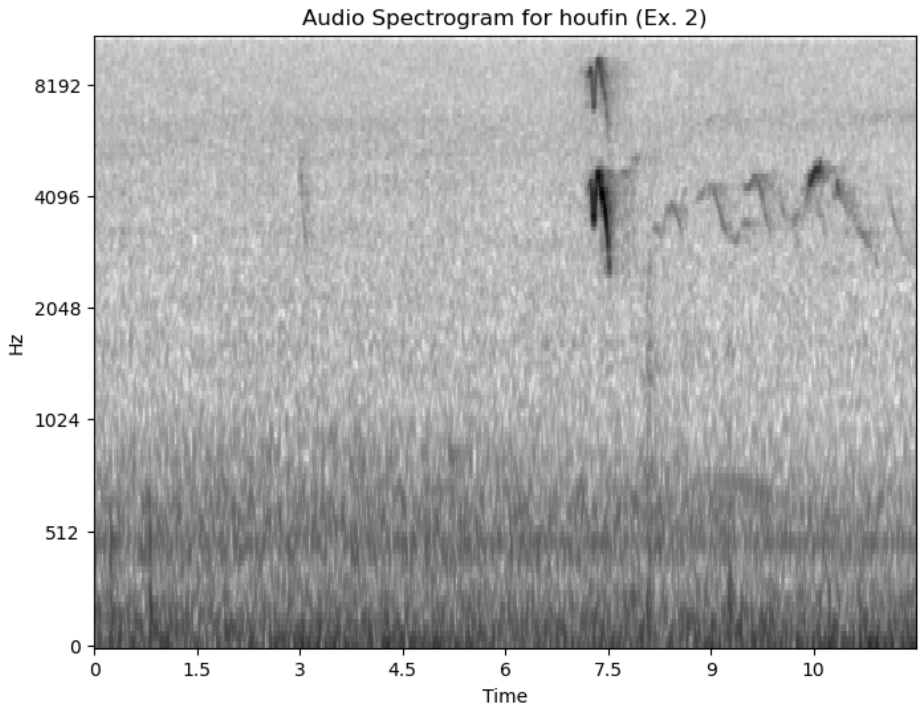

Figure 12: An example sonic spectrogram for the House Finch

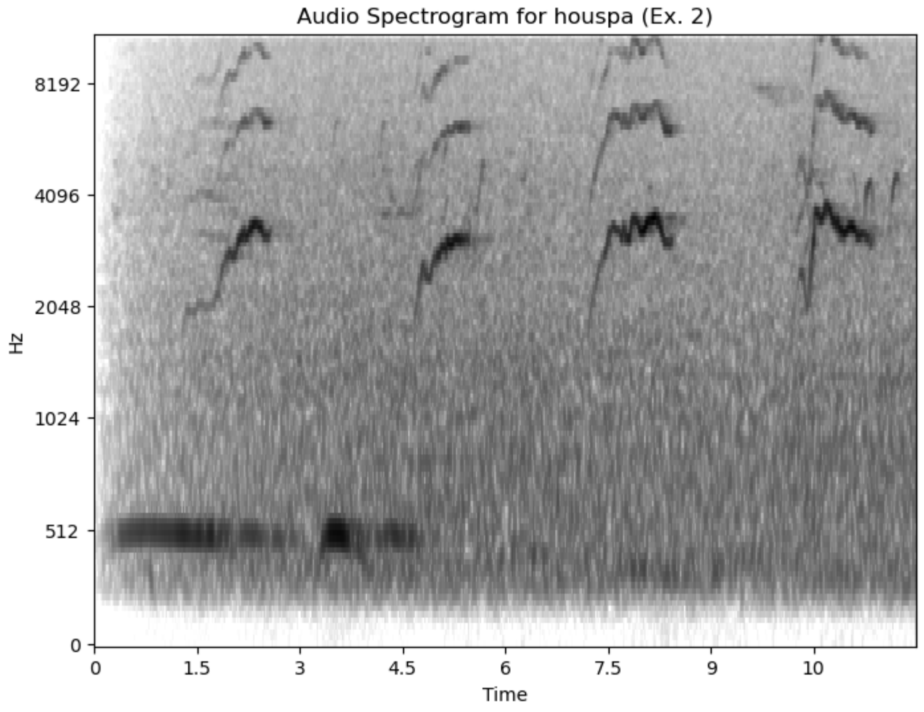

Figure 13: An example sonic spectrogram for the House Sparrow. Note the similarities to figure 12

5.3 Conclusion and Justification for Deep Learning vs Other Models

While the noisiness of the dataset presents challenges, it also presents an opportunity for a convolutional neural network with pooling. Pooling has been shown to induce less sensitivity toward the position of features in the spectrograms, while convolutional filters designed to identify the kind of visual features present in the spectrograms. Simpler models such as logistic regression and support vector machine may have trouble interpreting the visual information, since they will just treat each spectrogram as a vector. Two spectrograms which show the same call in different places will appear entirely distinct to these models unless some preprocessing is done first to either identify or shift the relevant features. With more data and computational resources devoted to tuning, a convolutional model is well positioned to make the predictions relevant to this study. The data obtained also support the potential of a convolutional neural network. Given that the spectrograms can be interpreted as a sequence with 517 steps and 128 frequency features, a network using of a recurrent structure would be expected to perform well. The poor performance of the recurrent structures attempted here are likely due to insufficient tuning and selection of the other layers. With more data, augmented or original, further tuning, and potentially the inclusion of other layer structures, a deep learning model shows promise in identifying birds based on their calls.

References

[1] J.D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: `10.1109/MCSE.2007.55`.

[2] Wes McKinney. *Data Structures for Statistical Computing in Python.* 2010. URL: `https://pandas.pydata.org/`.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. *Scikit-learn: Machine Learning in Python.* 2011. URL: `https://scikit-learn.org/`.

[4] Geoffrey Hinton. *Lecture 6.5 - RMSProp.* Coursera: Neural Networks for Machine Learning. 2012. URL: `https://www.coursera.org/learn/neural-networks`.

[5] Martín Abadi, Ashish Agarwal, Paul Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL: `https://www.tensorflow.org/`.

[6] François Chollet. *Keras: Deep Learning for Python*. 2015. URL: `https://keras.io/`.

[7] Vopani Rohanroa. *Xeno-Canto Bird Recordings Extended (A-M)*. Version 11. Kaggle, 2019. URL: `https://www.kaggle.com/datasets/rohanrao/xeno-canto-bird-recordings-extended-a-m`.

[8] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, et al. *Array programming with NumPy*. 2020. URL: `https://numpy.org/`.

[9] Philip Eichinski et al. "A convolutional neural network bird species recognizer built from little data by iteratively training, detecting, and labeling". In: *Frontiers in Ecology and Evolution* 10 (2022), p. 810330.

[10] Hanlin Wang et al. "An efficient model for a vast number of bird species identification based on acoustic features". In: *Animals* 12.18 (2022), p. 2434.

[11] Hira Farman et al. "Deep learning based bird species identification and classification using images". In: *Journal of Computing & Biomedical Informatics* 6.01 (2023), pp. 79–96.

[12] G. James et al. *An Introduction to Statistical Learning with Applications in Python*. Springer, 2023.

[13] JGraph Ltd. *draw.io*. 2023. URL: `https://www.drawio.com/`.

[14] Python Core Team. *Python: A dynamic, open source programming language*. Python Software Foundation. 2023. URL: `https://www.python.org/`.

[15] Andrew Collette et al. *h5py: Pythonic interface to HDF5*. 2025. URL: `https://www.h5py.org`.

[16] Xeno-canto Foundation. *Xeno-canto: Crowdsourced Bird Sound Archive*. 2025. URL: `https://www.xeno-canto.org`.

[17] Brian McFee et al. *Librosa: Python library for audio and music analysis*. 2025. DOI: `https://doi.org/10.5281/zenodo.15006942`. URL: `https://librosa.org/`.

[18] Ariana Mendible. *5322*. 2025. URL: `https://github.com/mendible/5322`.

[19] Michael Waskom. *Seaborn: Statistical Data Visualization*. 2025. URL: `https://seaborn.pydata.org`.