Bird Species Identification using Deep Learning on Spectrograms of Sound

Abstract

This is a project that entails bird species classification into their own species based on their unique vocalizations presented in the form of grayscale spectrogram. The dataset consists of 12 bird species with varying numbers of audio-based spectrograms created (128, 517). These images convey frequency patterns that are utilized as discriminative features for species identification. A convolutional neural network (CNN) model is developed and trained to recognize these frequency-based representations for successful classification.

Introduction

The general aim here is to train a Convolutional Neural Network (CNN) from scratch to identify bird species based on their sounds. The dataset used has already preprocessed spectrogram for 12 species, with each spectrogram being of size (128, 517), even though the number of samples for each species differs. This preprocessed data was taken from a public HDF5 file on GitHub, and the raw recordings were taken from the Xeno-Canto Bird Recordings Extended dataset on Kaggle, which had 240 bird species sounds in total. The data used here are not normalized, and the final objective is also to classify the bird species from a direct MP3 file input. This approach allows us to explore ways in which features of sound can be used economically for species classification based on deep learning.

Theoretical Background

Neural networks are computational models inspired by the way biological brains process information. They consist of layers of interconnected units called neurons that compute weighted sums of their inputs, add a bias term, and apply a nonlinear activation function. Different network architectures serve different tasks. Feed forward networks map inputs directly to outputs through a series of hidden layers. Recurrent networks introduce feedback connections that allow the model to capture temporal dependencies in sequential data. Modular networks combine specialized subnetworks to tackle complex problems. Convolutional neural networks extend these ideas by exploiting the spatial structure present in inputs.

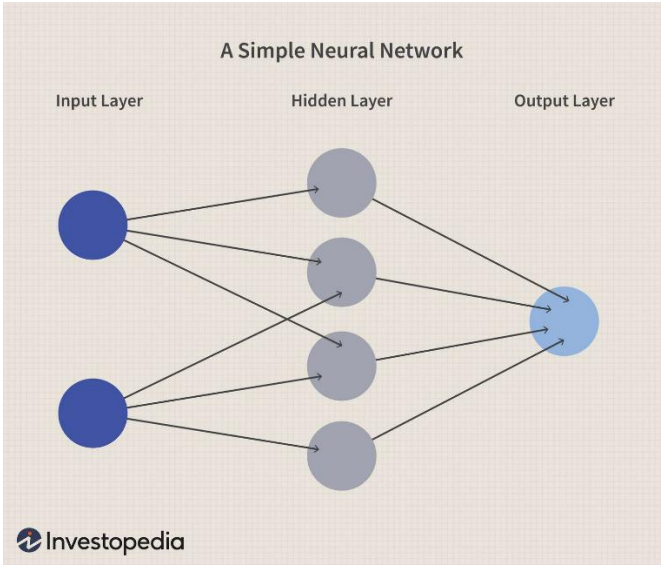


Figure 1 [5]

Single-layer networks consist of just an input layer and an output layer, with no hidden layers in between. In this setup each input node connects directly to each output node through a weight, and the network can only learn patterns that are linearly separable. Simple classification tasks can be handled by a single-layer perceptron network as shown in figure 1

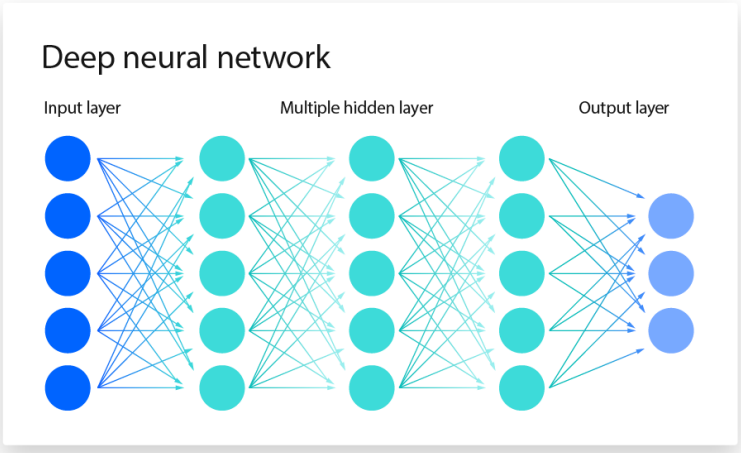


Figure 2 [6]

Multi-layer networks add one or more hidden layers between input and output, allowing them to learn far more complex relationships. Each hidden layer applies its

own set of weights and nonlinear activation functions before passing information onward. The result is a deep network that can extract hierarchical features from raw inputs, as illustrated by the deep neural network in figure 2.

A convolutional neural network begins with an input layer shaped to match the data, for example a Mel-scale spectrogram of size 128 by 517 with one channel. Convolutional layers then apply small learnable filters that slide across the spectrogram and extract local features such as frequency sweeps or harmonic textures. Pooling layers follow by reducing the spatial dimensions, retaining only the strongest activations to lower computational cost and add tolerance to slight shifts. After several convolution and pooling stages, feature maps are flattened and fed into fully connected layers before a final sigmoid or softmax output produces class probabilities.

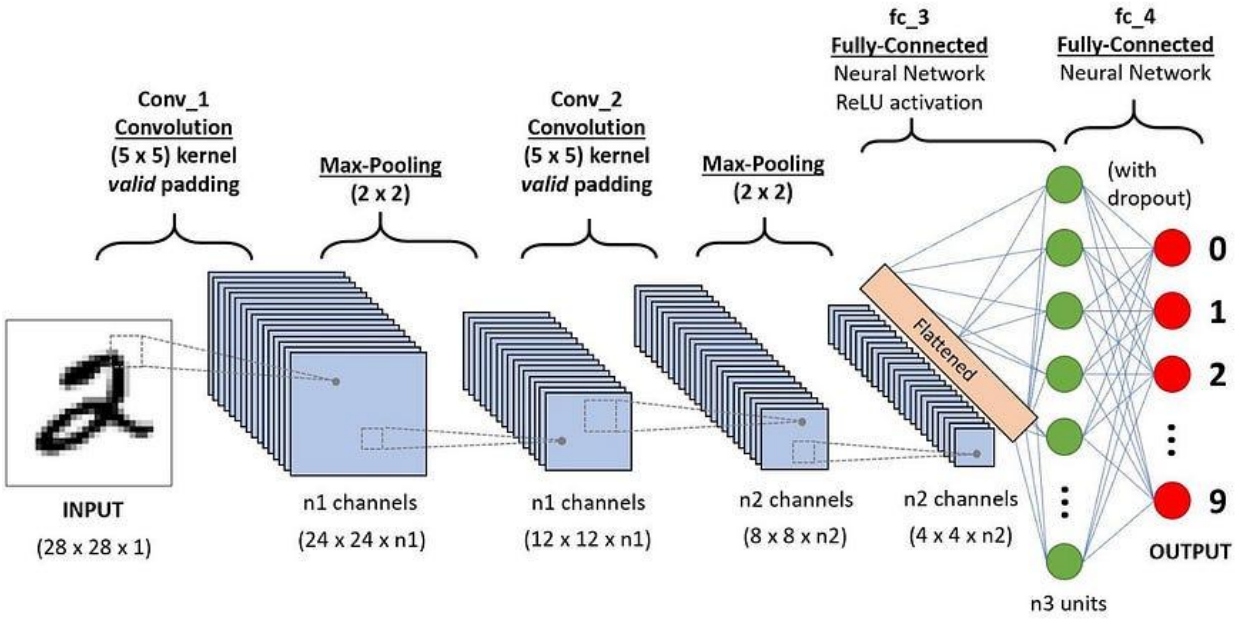


Figure 3 CNN layers [3]

Figure 3 shows a straightforward example of a convolutional neural network. It begins with an input layer that accepts a single-channel, 28×28 image. The first

convolutional layer applies several 5×5 filters across the input, detecting simple features like edges or corners. A max-pooling layer then scans each 2×2 region and keeps the highest value, which shrinks the feature maps and makes the network more robust to small shifts.

Next comes a second convolutional layer with the same 5×5 filters, which builds on the first layer's output to capture more complex patterns. Another 2×2 max-pooling layer reduces the spatial size further. At this point the network has a set of small feature maps that are flattened into a one-dimensional vector. This vector feeds into a fully connected layer, where each neuron looks at the entire set of features to learn high-level combinations.

A dropout layer follows the first fully connected layer, randomly turning off a fraction of neurons during training to prevent overfitting. Finally, the last fully connected layer produces one output per class (ten in this diagram). A softmax activation then converts these outputs into probabilities, allowing the network to make its final prediction.

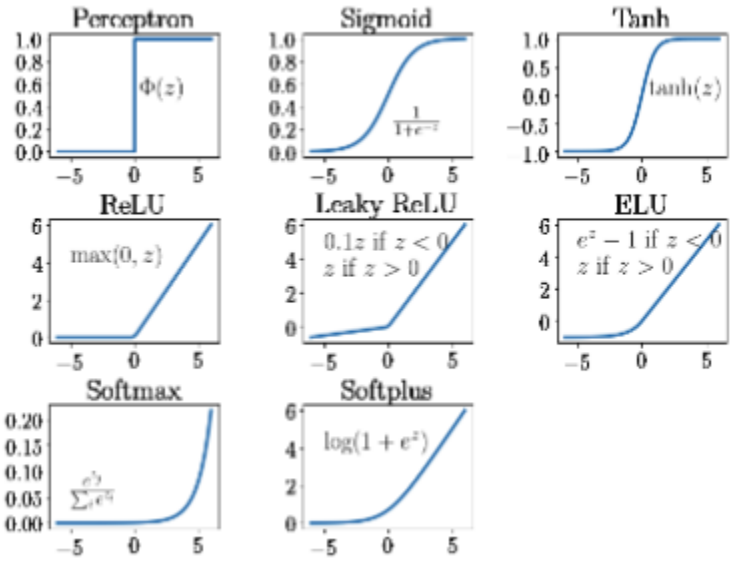


Figure 4 [7]

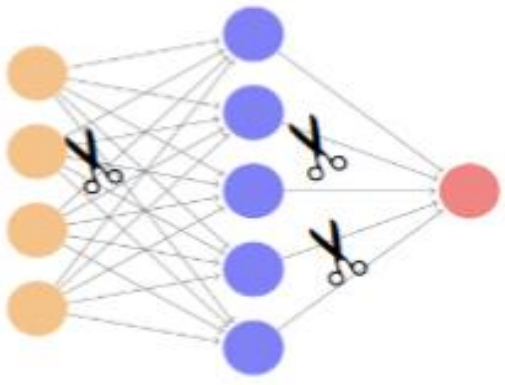
Activation functions are the nonlinear transformations that give neural networks their power to learn complex patterns. Without them, a network would collapse into a simple linear model regardless of its depth. In hidden layers, the Rectified Linear

Unit (ReLU) is widely used because its zeros out negative inputs and lets positive values pass unchanged, which helps prevent vanishing gradients. Variants like Leaky ReLU allow a small, non-zero slope for negative inputs, and Exponential Linear Units (ELU) introduce smooth negative outputs that can speed up convergence. The hyperbolic tangent function (tanh) maps inputs to the range -1 to 1, centering data around zero, while the classic sigmoid squeezes values into 0 to 1, making it suitable for binary decisions.

Loss Function:

Loss functions tell the network how well it is doing and guide its learning. For two classes, binary cross-entropy looks at how far the predicted probability is from the true label and gives bigger “mistakes” when the model is very confident but wrong. For more than two classes, categorical cross-entropy compares the full set of predicted probabilities to the actual class and measures the overall difference. Both of these functions change smoothly as predictions get better or worse, so the network can adjust its weights little by little.

Hyperparameter tuning decides how the network learns from the data. The learning rate sets how big each update to the weights is. If it is too high, the model may jump past good solutions. If it is too low, the model may learn very slowly. The batch size determines how many samples the model processes before it updates its weights. Smaller batches add some randomness to learning, which can help the model find better solutions. Larger batches give more consistent updates but use more memory. The number of epochs is the count of how many times the model sees the entire training set. Early stopping can end training early when the model stops improving, which helps prevent overfitting.



Regularization techniques help prevent overfitting and improve generalization. Dropout randomly disables a percentage of neurons during each training step, forcing the network to distribute learned features more broadly. L2 weight decay adds a penalty proportional to the square of each

weight, discouraging excessively large parameters and smoothing the loss surface. Callbacks such as early stopping halt training when validation performance plateaus, and learning-rate schedulers reduce the step size when progress stalls, ensuring efficient and robust convergence.

Methodology:

The initial data is a collection of audio clips converted into a spectrogram representing the bird species. Upon loading the spectrograms, we first clipped their decibel values to the range [–80, 0] dB and scaled them to [0, 1]. The normalized spectrograms were then transposed to (samples, 128, 517), cast to float32, and reshaped to (samples, 128, 517, 1) to match our CNN input requirements. Species labels were assigned programmatically and the entire dataset was shuffled to randomize batch composition.

Mel-scale spectrograms were loaded from an HDF5 archive, clipped to [–80, 0] dB and scaled to [0, 1] via the normalize_db_scale function. Each spectrogram was then transposed to (samples, 128, 517), cast to float32, and reshaped to (samples, 128, 517, 1) for CNN input. Species labels were assigned programmatically, the dataset shuffled, and then split into 60 % training, 20 % validation, and 20 % test sets.

Hyperparameter configuration was consistent across both tasks: Adam optimizer (learning rate $1x10^{-4}$), L2 regularization ($\lambda = 0.02$ for binary; 0.01 for multi-class), dropout (0.5 in binary; 0.3 in multi-class), early stopping (patience = 5 -10 epochs), ReduceLROnPlateau (factor = 0.5 after five stagnant epochs), and model checkpointing based on validation loss.

Binary classification between 'amerob' and 'rewbla'.

A compact CNN with two convolutional blocks (16 → 32 filters), each followed by max-pooling and dropout, was trained, flattened into a dense layer, and finished with a sigmoid output. Test-set performance was reported via accuracy, AUC, precision, recall, and F_1 score.

Multi-class classification across twelve species.

A deeper CNN with four convolutional blocks (32 → 64 → 128 → 256 filters), global average pooling, and a softmax output was trained. Categorical accuracy and AUC metrics were computed on the held-out test partition.

Classification of external test clips.

The unseen three-second audio segments were processed identically and passed through both models. Top-prediction agreement and confidence scores were analyzed to assess binary and multi-class bird identification.

Results:

Binary classification between 'amerob' and 'rewbla'.

In order to find the best binary classifier for amerob versus rewbla calls, four CNN variants were tested under different hyperparameter settings. Model 3 was trained for 100 epochs with a $5x10^{-4}$ learning rate, 0.3 dropout and 0.02 L2 regularization, yielding strong recall but only moderate precision. Model 4 also ran for 100 epochs but used a lower $1x10^{-4}$ learning rate, higher 0.4 dropout and 0.01 regularization, which produced the highest area under the ROC curve. Model 5, with a smaller batch size of 16, a 0.5 dropout rate and only 20 epochs, lagged behind in overall accuracy. In contrast, Model 6 trained for 20 epochs with a $1x10^{-4}$ learning rate, 0.2 dropout and 0.01 regularization reached 75 percent test accuracy and balanced precision and recall at 0.76 each, all while showing smooth, stable learning curves. Based on its superior accuracy, balanced performance metrics and efficient training profile, Model 6 was selected as the final binary classifier.

Model Name	Accuracy	AUC	Precision	Recall	F1 Score
model_1_binary_cnn	0.7222	0.7663	0.7647	0.6842	0.7222
model_2_binary_cnn	0.5833	0.6788	0.5645	0.9211	0.7000
model_3_binary_cnn	0.7222	0.7639	0.6731	0.9211	0.7778
model_4_binary_cnn	0.6667	0.7848	0.6750	0.7105	0.6923
model_5_binary_cnn	0.5694	0.6424	0.5660	0.7895	0.6593
model_6_binary_cnn	0.7500	0.7585	0.7632	0.7632	0.7632

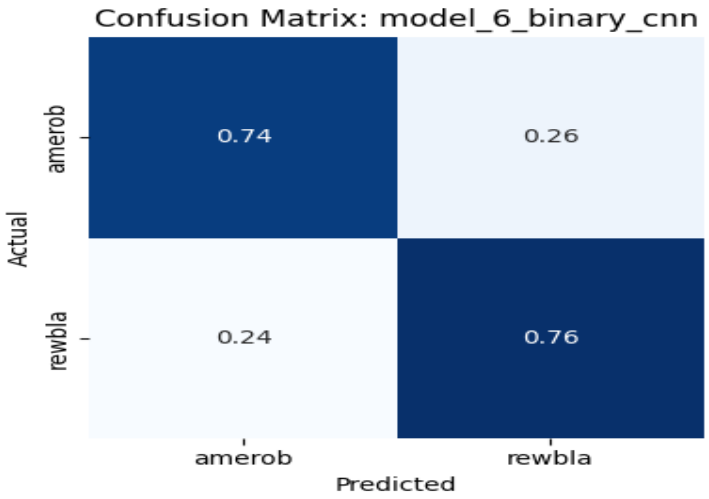


Figure 6

As shown in the confusion matrix for model_6 (Figure 6), it correctly classifies 74 % of amerob calls and 76 % of rewbla calls, with nearly equal rates of false positives (26 %) and false negatives (24 %), demonstrating that it does not favor one class over the other.

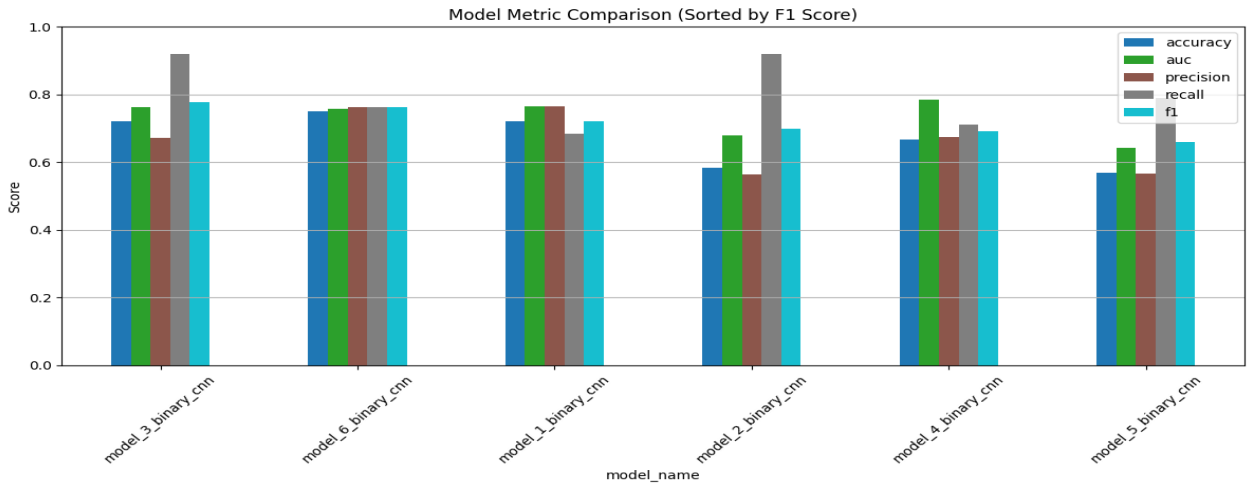


Figure 7

Figure 7 provides a side-by-side comparison of accuracy, AUC, precision, recall, and F1 scores for all six binary CNN models, ordered by their F1 performance.

Best Model Curve:

The accuracy and loss curves for the best-performing model demonstrate stable and consistent improvement over epochs, indicating effective learning without significant overfitting.

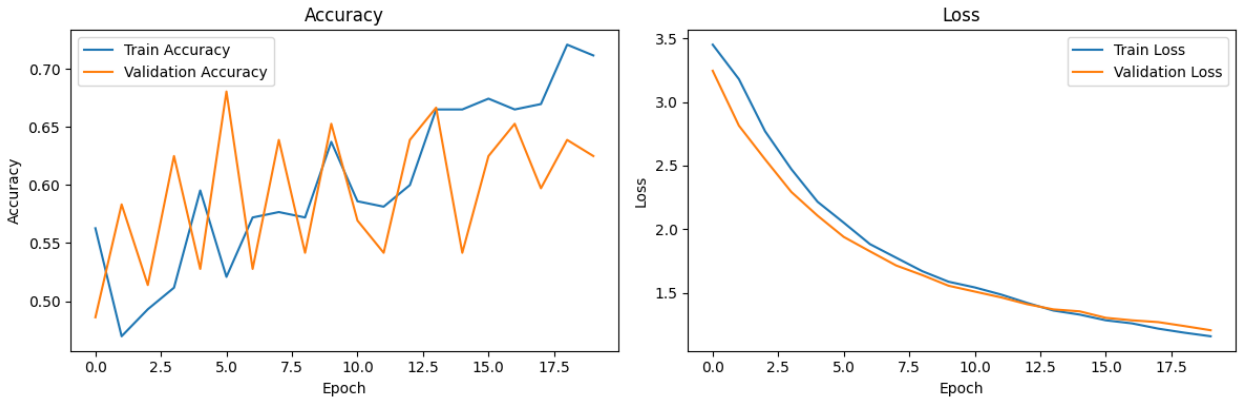


Figure 8

Poorly Performing Model Curve:

In contrast, the weaker model's accuracy curve exhibits large fluctuations and instability throughout training, suggesting difficulties in generalizing from the imbalanced data.

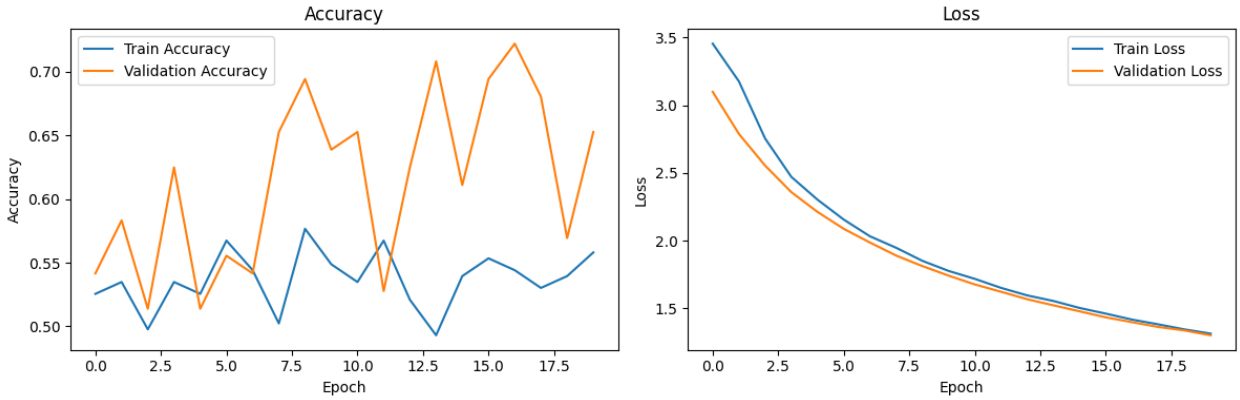


Figure 9

Multi-class classification across twelve species.

I tested six different CNN configurations for the multi-class bird classification task. The first model, **cnn_multiclass_model1**, was trained for 50 epochs with a batch size of 32, learning rate of 1e-4, dropout rate of 0.3, and L2 regularization of 0.01. It achieved a test accuracy of **35.26%** and an AUC score of **0.7945**, making it one of the lower-performing models. On the other hand, **cnn_multiclass_model5**, which also used a batch size of 32 and learning rate of 1e-4 but trained for more epochs and applied a dropout rate of 0.4, reached the highest test accuracy of **39.04%** and the highest AUC of **0.8145**. Considering both accuracy and AUC, **cnn_multiclass_model5** was selected as the final model for multi-class classification.

Model Name	Accuracy	AUC
cnn_multiclass_model3	0.3904	0.8084
cnn_multiclass_model5	0.3904	0.8145
cnn_multiclass_model4	0.3753	0.8108
cnn_multiclass_model6	0.3602	0.8112
cnn_multiclass_model1	0.3526	0.7945
cnn_multiclass_model2	0.2846	0.7399

Based on the results, the **best performing model** for multi-class bird species classification is **cnn_multiclass_model5**. It achieved the **highest accuracy (39.04%)** and **highest AUC score (0.8145)** among all models, showing strong ability to correctly distinguish between different bird species.

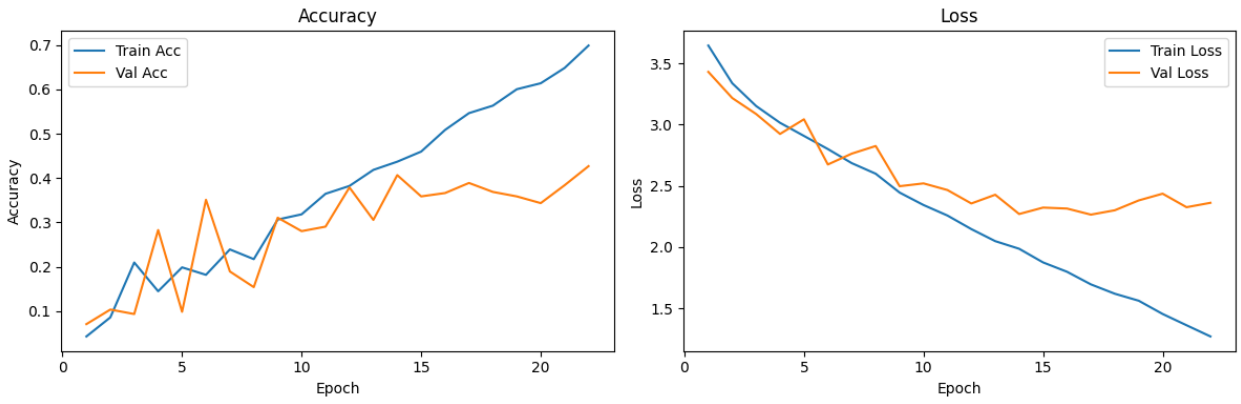


Figure 10

The **worst performing model** is **cnn_multiclass_model2**, which had the **lowest accuracy (28.46%)** and the **lowest AUC score (0.7399)**. It struggled with many classes and showed less reliable learning behavior compared to the others.

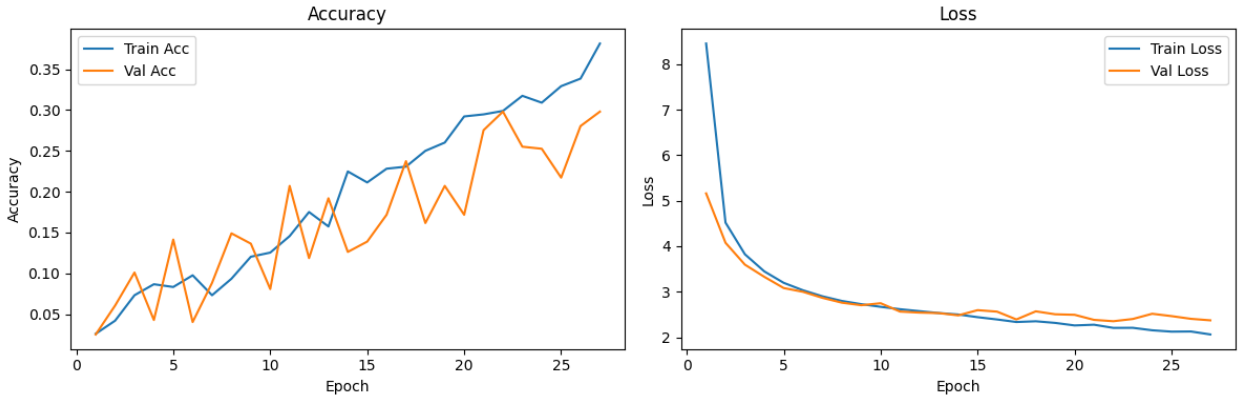


Figure 11

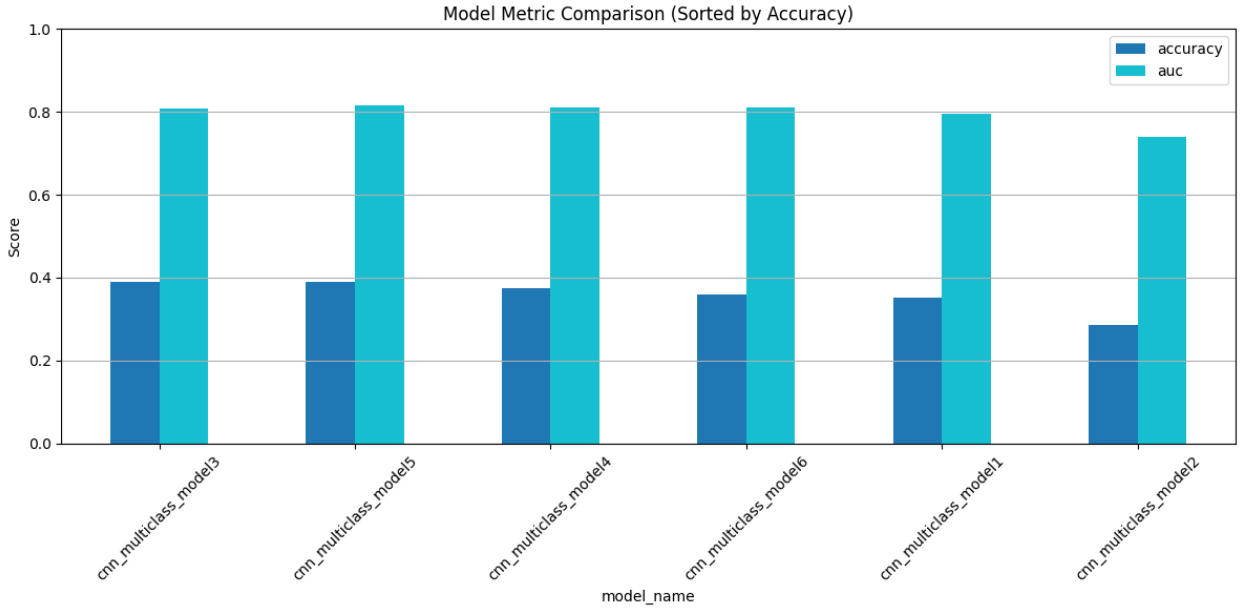


Figure 12

Figure 12 shows a comparison of multi-class CNN models based on their accuracy and AUC scores. From the chart, we can see that model 3 and 5 achieved the highest accuracy and AUC, indicating better overall classification performance.

Discussion

Limitations

A limitation of this study is that the dataset contained uneven sample sizes among the species, and class weights were not properly adjusted to account for this imbalance. For instance, certain species such as 'norfli' had as few as 37 samples, whereas 'houspa' had as many as 630 samples, causing bias in the model training. This uneven distribution contributed to unstable validation results, suggesting that the model had difficulty consistently predicting classes with fewer samples. To improve the model's performance and stability, future research should use class-weight adjustments, oversampling techniques, or additional data collection to create a more balanced training set.

Challenges Faced

Training Without Early Stopping:

Early stopping was not being utilized initially during training, and the model was trained for only 300 epochs. Without early stopping, training would have taken almost an hour, and the model's performance was suboptimal. The accuracy curve indicated evident overfitting, with the spikes in the training accuracy being very high compared to being flat in the validation accuracy. This discrepancy asserted that the model was not properly generalizing and was memorizing the training data instead. To counter this, early stopping was used to prevent overfitting, and training was halted when validation accuracy stopped improving.

Training Time:

The binary classification model was trained for approximately 23 minutes for 200 epochs with learning rate 1e-4 and early stopping patience of 5.

The multi-class classification model, however, took around 40 minutes to train with the same parameters but for 200 iterations as well. The increased training duration in the multi-class model is due to the greater number of classes and task complexity involved in the classification.

Hard Species to Predict

Certain species were more difficult to predict due to issues in data quality and overlapping characteristics:

norfli: Due to the low number of samples (37), the model struggled to generalize owing to the small amount of data.

bkcchi: Similarly, the small dataset (45 samples) made it difficult for the model to separate this species properly.

In the confusion matrix shown in Figure 13, which corresponds to the best-performing multi-class model, we can still observe that some bird species remain difficult to classify accurately. Although the model performs well overall, species like norfli (Northern Flicker) and bkcchi (Black-capped Chickadee) have very few correct predictions and are often confused with other classes. This suggests that even the most optimized model struggles to learn distinct patterns for these particular birds. One likely reason is the limited number of training samples for these species, which makes it harder for the model to generalize. The confusion matrix clearly highlights this challenge, showing that having an imbalanced dataset can impact the model's ability to differentiate between certain classes effectively.

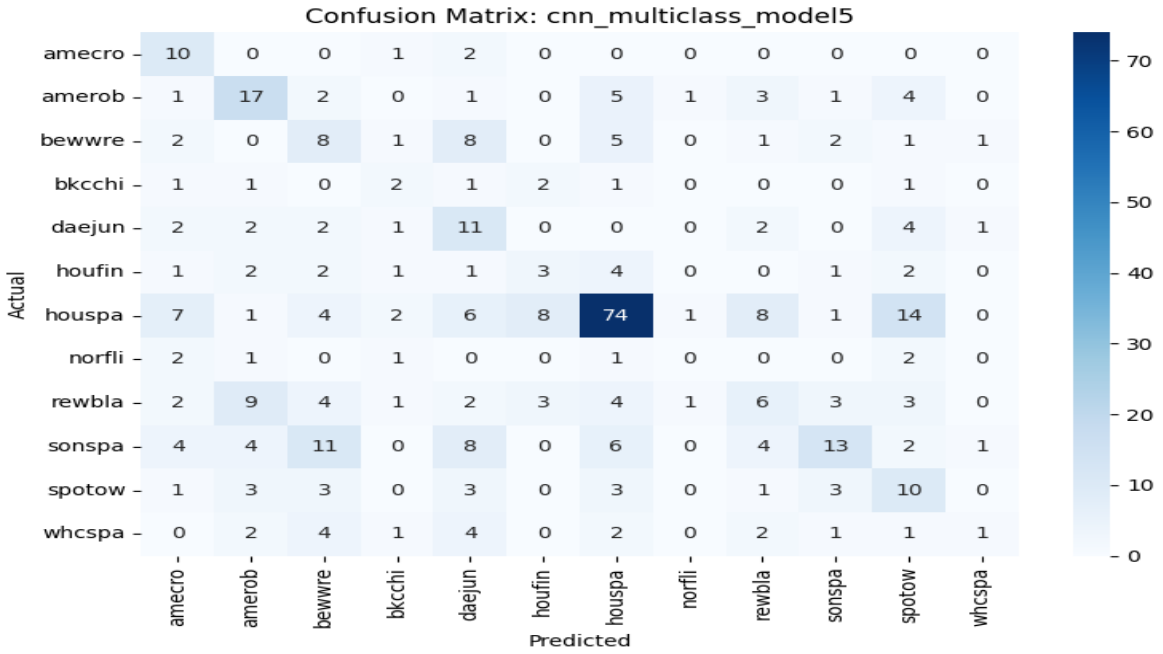


Figure 13

Alternative Models for Neural Networks

For this exercise, there could have been different models applicable for bird species classification from audio data, yet the most appropriate is a neural network, especially a CNN, due to the nature of data (spectrograms) and the task nature.

Pretrained Models

Transfer Learning with pretrained models is a robust approach. Pretrained models on large image datasets, such as VGG16, ResNet, or Inception, can be used to classify bird species. The pretrained models have already optimized layers that are excellent at extracting image features, which may be a good fit for spectrograms as they essentially translate audio information into 2D image-like objects.

This approach allows leveraging powerful pre-trained weights, saving training time, and perhaps improving accuracy since the model already knows how to extract important features from visual data.

Audio-Specific Deep Learning Models

Alternatively, models like WaveNet or Raw Audio Neural Networks, specializing in audio classification, can be used. Such models operate on raw audio data (e.g., waveforms) and do not involve the conversion of data to spectrograms. This may rule out information loss during the conversion phase to spectrograms, hence the model's performance should be improved while processing the audio data in its native form.

Dense Neural Networks:

A Dense Neural Network (DNN) would be another potential contender for this task. While CNNs are excellent at handling spatial data like images, a fully connected DNN can also be used, especially if the learned features from the spectrograms are already decent and well-preprocessed and dimensionally reduced. However, DNNs typically require more feature engineering and might not be as adept at handling tasks with spatial or sequential patterns like audio classification.

Why CNN Makes Sense:

Since the work is working with spectrograms, which are essentially 2D pictures that detail the frequency makeup of sound as a function of time, a

Convolutional Neural Network (CNN) is particularly well-adapted. CNNs are really designed for image classification, so they are ideally suited to work on tasks that involve inspecting 2D data such as spectrograms.

CNNs are good at recognizing local patterns within data using sliding windows (filters), making them perfect for working with spectrograms. They can learn spatial hierarchies automatically and also pull-out suitable features like edges and textures, which is of the utmost importance in distinguishing between species of birds by their respective call features.

Git Link for the project: Project

References:

[1] Early Stopping Usage
TensorFlow. EarlyStopping callback. Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

[2] Reducing Learning Rate on Plateau
TensorFlow. ReduceLROnPlateau callback. Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ReduceLROnPlateau

[3] Deep CNN Reference for Other Applications
Deep Convolutional Neural Networks in data classification and pattern recognition applications. https://www.mdpi.com/1099-4300/23/11/1507

[4] Theoretical Background on CNN Architectures
Convolutional Neural Networks (CNN) Architectures Explained. https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243

[5] Single Layer Neural Network Image
Investopedia. A Simple Neural Network. https://www.investopedia.com/terms/n/neuralnetwork.asp

[6] Deep Neural Network Image
IBM. Neural networks explained. https://www.ibm.com/think/topics/neural-networks

[7] Johson, J.(2020). Lecture 10: Training Neural Networks (Part 1). EECS 498/598: Deep Learning for Computer Vision, University of Michigan. https://web.eecs.umich.edu/~justincj/slides/eecs498/FA2020/598_FA2020_lecture10.pdf