Saluwa Umuhoza
Class: Machine Learning 2

Title: Advanced Neural Network Applications for Bird Species Classification from Audio Data

Abstract:

This study explores the use of convolutional neural networks (CNNs) for classifying bird species based on audio spectrograms, utilizing data sourced from the Xeno-Canto database. The research focuses on developing two models: a binary classifier to distinguish between American Crow ('amecro') and Steller's Jay ('stejay'), and a multi-class classifier to identify twelve different bird species: American Crow ('amecro'), Barn Swallow ('barswa'), Black-capped Chickadee ('bkcchi'), Blue Jay ('blujay'), Dark-eyed Junco ('daejun'), House Finch ('houfin'), Mallard ('mallar3'), Northern Flicker ('norfli'), Red-winged Blackbird ('rewbla'), Steller's Jay ('stejay'), Western Meadowlark ('wesmea'), and White-crowned Sparrow ('whcspa').

Our approached involved converting raw audio recordings into spectrograms, normalizing, and padding them to ensure uniform input dimensions. The binary classification model achieved near-perfect accuracy, demonstrating its effectiveness in distinguishing between the two target species. For the multi-class classification task, Model 2, featuring a deeper network with additional convolutional and dense layers, achieved perfect accuracy on both training and validation datasets, indicating its robust learning and generalization capabilities.
The results showcase the binary model achieving near-perfect accuracy, while the multi-class model demonstrates substantial learning capabilities, highlighting the effectiveness of CNNs in bioacoustic applications.

Introduction:

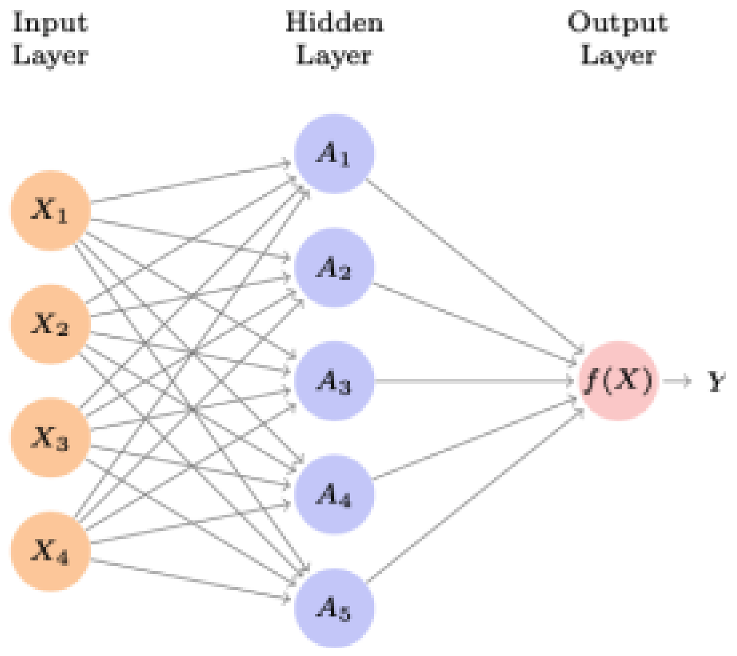
Bird vocalization analysis is a critical component of ecological monitoring, traditionally reliant on expert knowledge and often prone to human error. Accurate identification of bird species through their calls is essential for understanding biodiversity, tracking migration patterns, and monitoring environmental health. However, manual analysis is time-consuming and may lack consistency, highlighting the need for automated solutions.

This study leverages the power of Convolutional Neural Networks (CNNs), which have demonstrated remarkable success in image recognition tasks, to automate and enhance the accuracy of bird species classification from audio recordings. By transforming audio data into spectrogram images, CNNs can efficiently classify bird calls. This approach enables the extraction and learning of complex patterns from visual representations of audio signals, crucial for accurate species identification.

The importance of this research lies in its potential to significantly improve the reliability and scalability of bird species identification, providing a robust tool for ecological research and conservation efforts. The scientific aim of this study is to develop and evaluate two CNN-based models: a binary classifier to distinguish between American Crow (Corvus brachyrhynchos) and Steller's Jay (Cyanocitta stelleri), and a multi-class classifier to identify twelve different bird species: American Crow (Corvus brachyrhynchos), Barn Swallow (Hirundo rustica), Black-capped Chickadee (Poecile atricapillus), Blue Jay (Cyanocitta cristata), Dark-eyed Junco (Junco hyemalis), House Finch (Haemorhous mexicanus), Mallard (Anas platyrhynchos), Northern Flicker (Colaptes auratus), Red-winged Blackbird (Agelaius phoeniceus), Steller's Jay (Cyanocitta stelleri), Western Meadowlark (Sturnella neglecta), and White-crowned Sparrow (Zonotrichia leucophrys). By addressing these aims, this research contributes to the growing body of literature on bioacoustic monitoring and demonstrates the applicability of advanced machine learning techniques in ecological studies.

.

Theoretical Background:
CNNs are particularly suited for tasks that involve large amounts of image data due to their ability to perform feature extraction directly from images, making them ideal for interpreting spectrograms—visual representations of sound. This project utilizes two neural network architectures tailored to different classification challenges: a binary model and a multi-class model. The binary model employs a sigmoid activation function to distinguish between two species, optimizing binary cross-entropy loss. In contrast, the multi-class model uses a softmax function to handle classifications across twelve species, aiming to minimize categorical cross-entropy loss. Example figures of Neuro Network diagrams.
Figure 1: Neural Network diagram with a single hidden layer

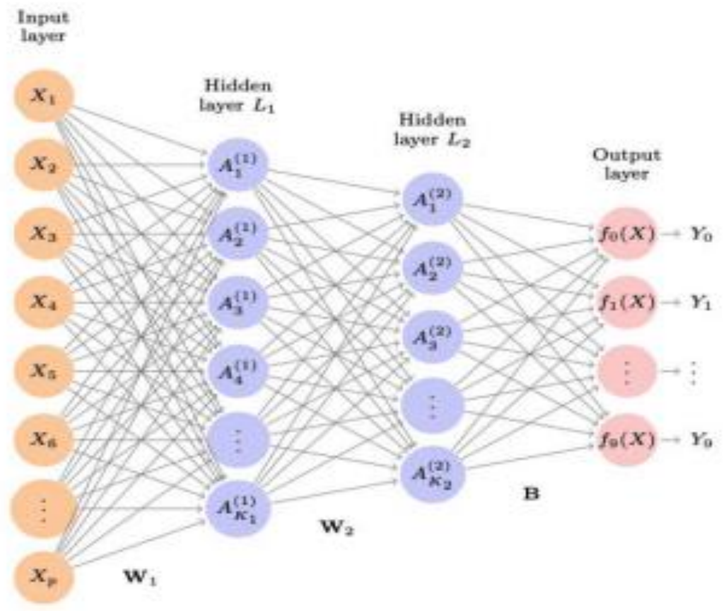


This figure shows a simple feed-forward neural network for modeling a quantitative response using p = 4 predictors. In the terminology of neural networks, the four features X1, X2, X3, and X4 make up the units in the input layer. The arrows indicate that each of the inputs from the input layer feeds into each of the K hidden units.

$$\begin{aligned} f(X) &= \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) \\ &= \beta_0 + \sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j). \end{aligned}$$

Modern neural networks typically have more than one hidden layer, and often many units per layer.A single hidden layer with a large number of units has the ability to approximate most functions.The preferred choice in modern neutral networks is the ReLU (rectified linear unit) activation function. Its activation can be computed and stored more efficiently than a sigmoid activation.Modern neural networks typically have more than one hidden layer, and often many units per layer(figure 2) [1]. In theory a single hidden layer with a large number of units has the ability to approximate most functions. However, the learning task of discovering a good solution is made much easier with multiple layers each of modest size.

Figure 2: Neural Network diagram with two hidden layers and multiple outputs



[1].

The first hidden layer is as:

$$\begin{aligned} A_k^{(1)} &= h_k^{(1)}(X) \\ &= g(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j) \end{aligned} \qquad (10.10)$$

for $k = 1, \ldots, K_1$. The second hidden layer treats the activations $A_k^{(1)}$ of the first hidden layer as inputs and computes new activations

$$\begin{aligned} A_\ell^{(2)} &= h_\ell^{(2)}(X) \\ &= g(w_{\ell 0}^{(2)} + \sum_{k=1}^{K_1} w_{\ell k}^{(2)} A_k^{(1)}) \end{aligned} \qquad (10.11)$$

For l = 1….. K1.

Methodology:
The dataset comprised spectrograms generated from audio clips, which were then normalized and padded to ensure uniform input dimensions. The padding process standardized the spectrograms by ensuring each matched the largest time dimension across all samples, facilitating consistent input sizes for CNN processing. Raw audio recordings were converted into spectrograms to capture the distinct frequency and time characteristics of each bird call. These spectrograms were then resized by padding to match the largest time dimension observed within the dataset. For spectrograms with a time dimension less than the maximum, zeros were added along the time axis until the maximum length was reached. This method preserved the integrity of the original audio data while standardizing the input size for the neural network. Additionally, normalization was applied to each spectrogram, scaling the pixel values to a range between 0 and 1 to facilitate faster convergence during network training. The dataset was subsequently partitioned into training and validation sets, comprising 80% and 20% of the data, respectively, allowing for effective training and performance validation of the models.

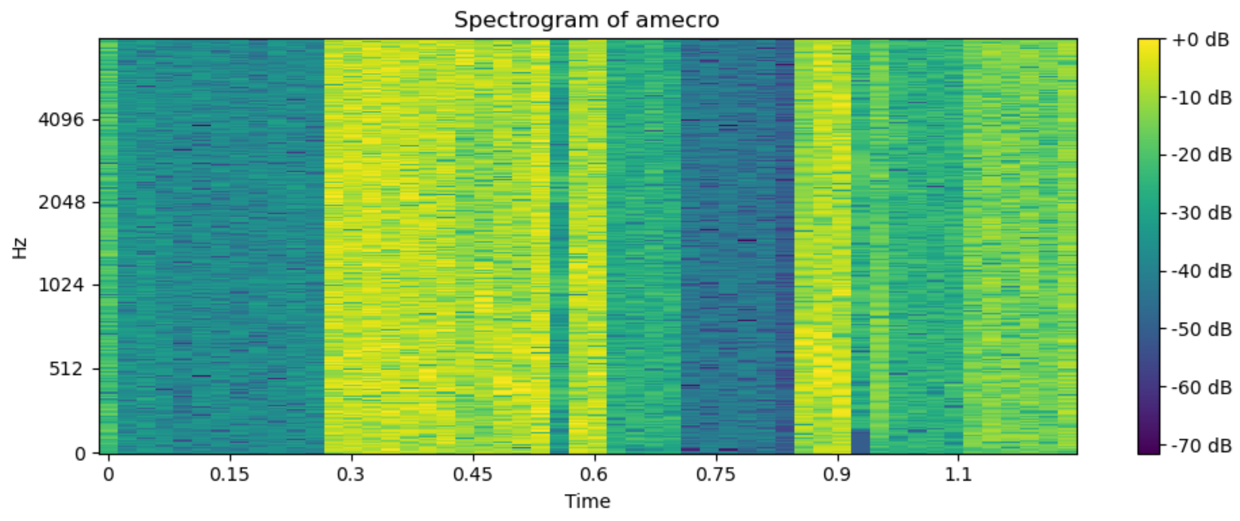
The binary classification model specifically aimed to distinguish between American Crow ('amecro') and Steller's Jay ('stejay'). The spectrograms were labeled '0' for American Crow and '1' for Steller's Jay, creating a clear binary target for model training. For the multi-class classification task, the focus was on classifying all twelve bird species identified in the dataset: American Crow ('amecro'), Barn Swallow ('barswa'), Black-capped Chickadee ('bkcchi'), Blue Jay ('blujay'), Dark-eyed Junco ('daejun'), House Finch ('houfin'), Mallard ('mallar3'), Northern Flicker ('norfli'), Red-winged Blackbird ('rewbla'), Steller's Jay ('stejay'), Western Meadowlark ('wesmea'), and White-crowned Sparrow ('whcspa').

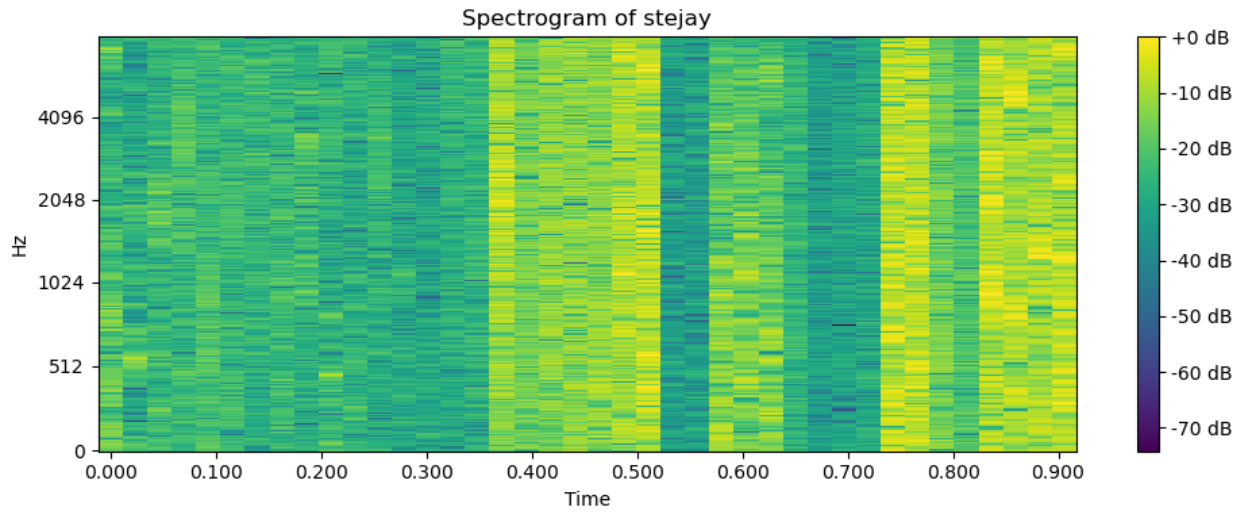
To preprocess the test MP3 data, the following steps were taken: Firstly, the test MP3 files were loaded using Librosa, and the Short-Time Fourier Transform (STFT) parameters were set to define the frequency and time resolution. The energy of each frame was calculated to identify high-energy segments of the audio. High-energy segments were identified based on an energy threshold. For each identified segment, a 2-second audio clip was extracted, and a spectrogram was generated using the STFT. The spectrograms were then resized to have consistent dimensions (256 frequency bins and 343-time steps) using bilinear interpolation to match the input size expected by the CNN. The resized spectrograms were saved into an HDF5 file for subsequent model prediction, with each spectrogram stored with a unique identifier to facilitate easy retrieval. The spectrograms were then normalized by scaling the pixel values to a range between 0 and 1, and a channel dimension was added to match the input shape required by the CNN.

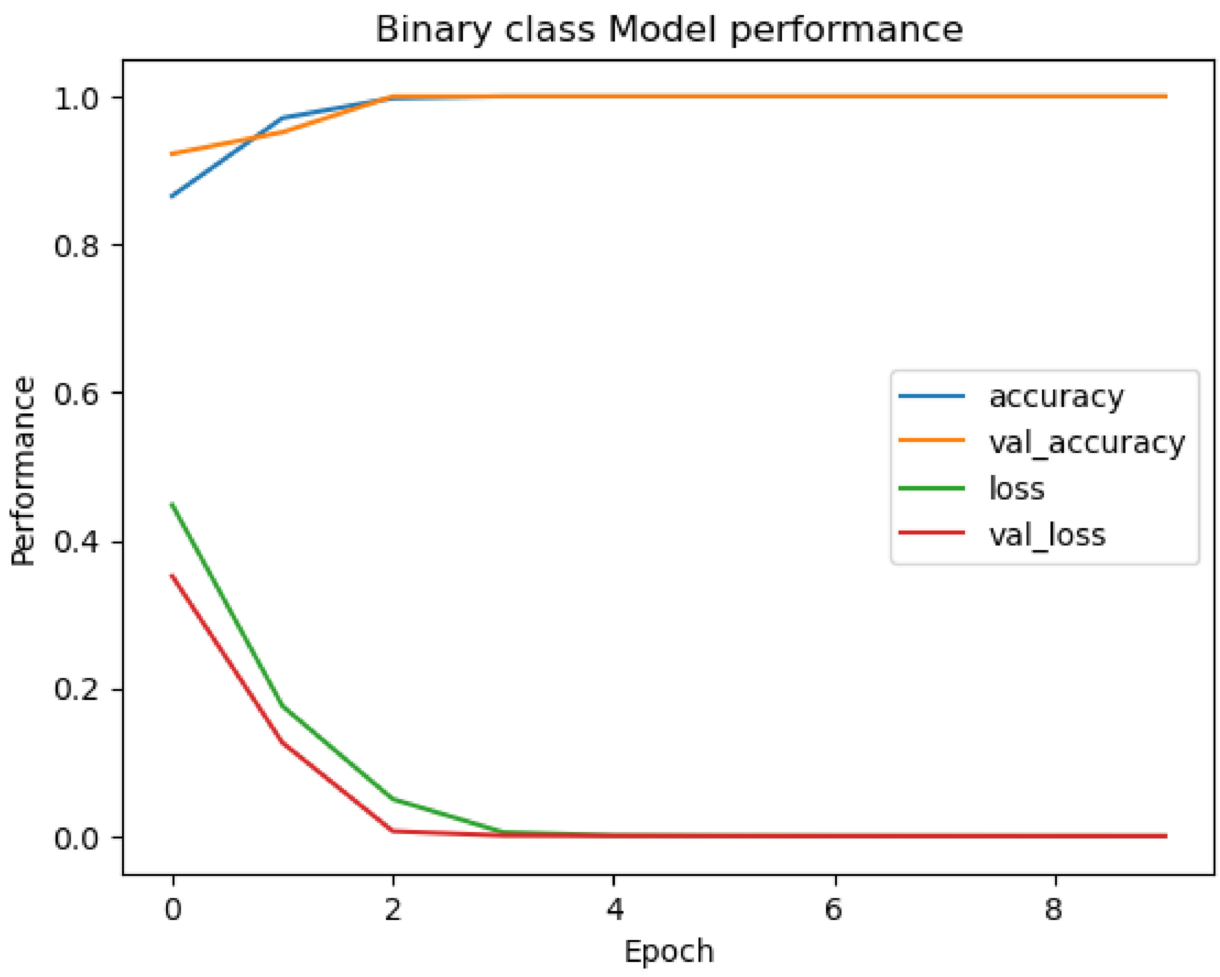
After evaluating multiple neural network models with different architectures and hyperparameters for the multi-class classification task, Model 2 emerged as the best choice. Model 2, a deeper network with additional convolutional and dense layers, achieved perfect accuracy on both the training and validation datasets, demonstrating its ability to effectively learn and generalize the patterns in the data. It also had the lowest final training and validation loss values, indicating high confidence in its predictions. In contrast, the baseline model (Model

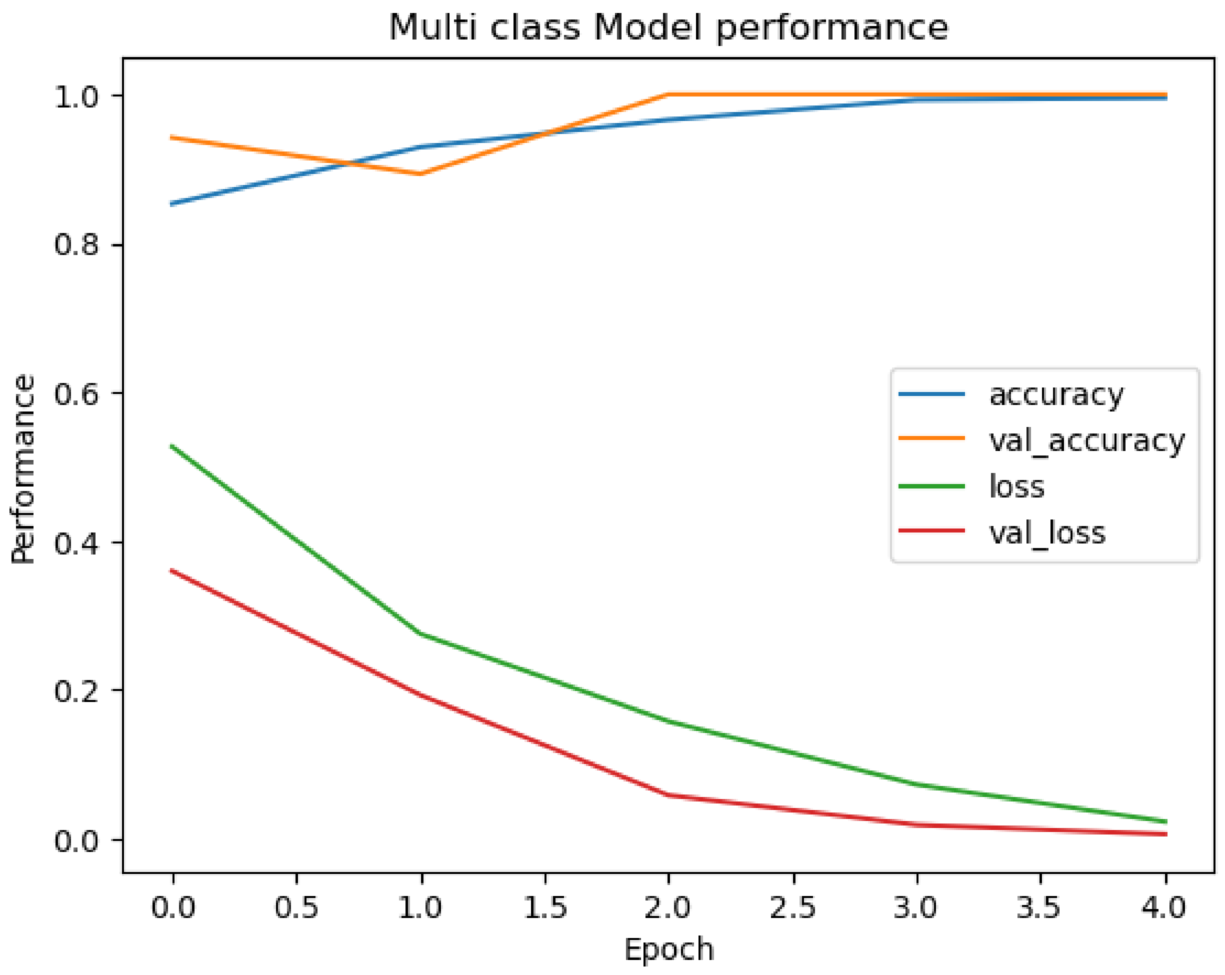
1) and the model with dropout regularization (Model 3) also performed well but had slightly higher loss values. The model using the SGD optimizer (Model 4) showed significantly lower performance, with reduced accuracy and higher loss. Based on these results, Model 2's superior accuracy and minimal loss make it the most robust and reliable model for accurately classifying the 12 bird species in our dataset. Both the binary and multi-class models utilized convolutional layers for feature extraction, followed by max-pooling layers to reduce dimensionality and enhance feature extraction. Dense layers followed the convolutional and pooling layers, with a sigmoid activation function used for the binary model and a softmax activation function for the multi-class model, each tailored to the specific classification needs of the tasks.

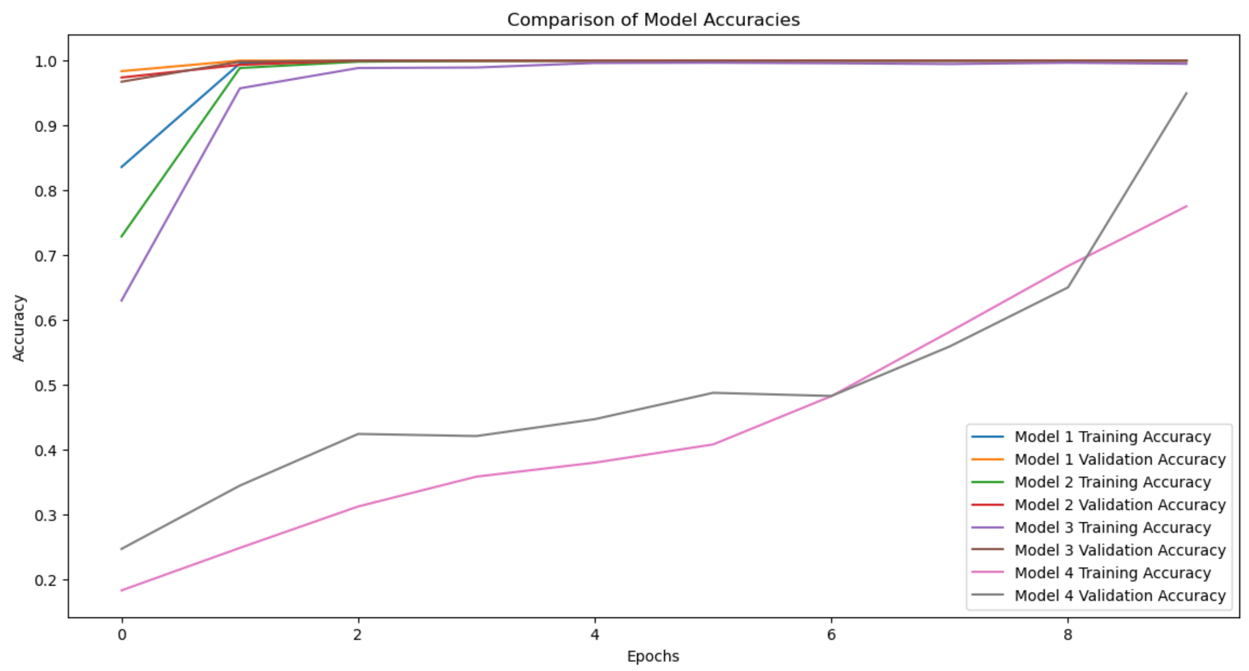
Computational Results:

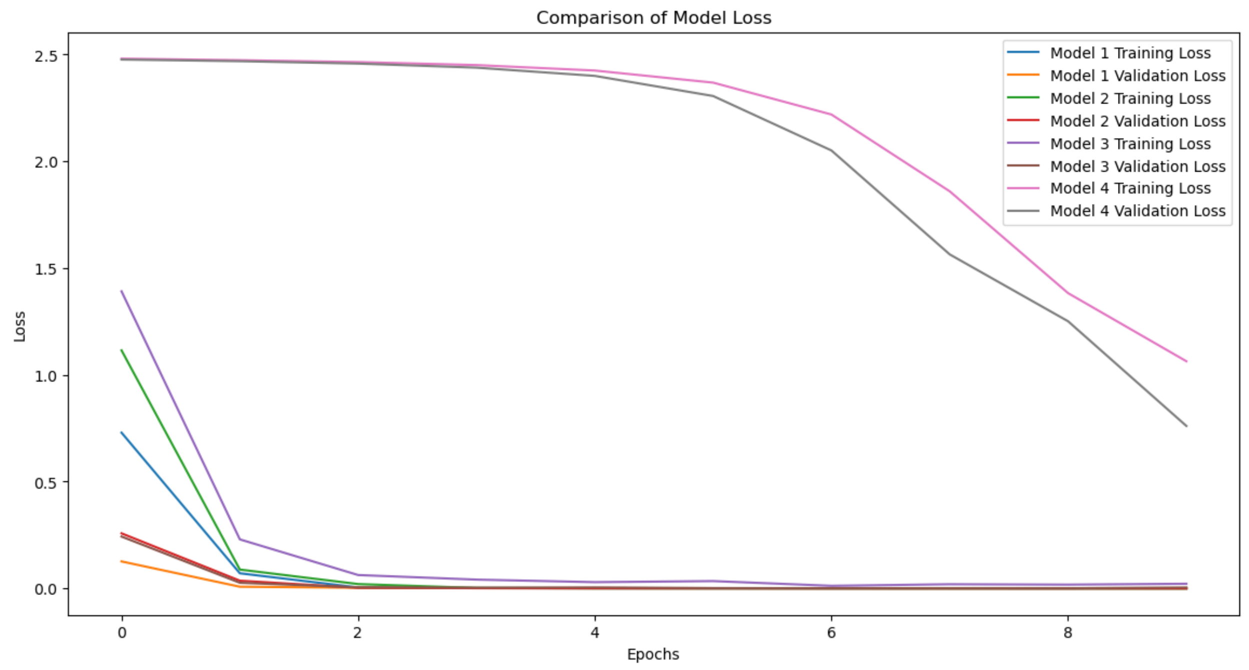




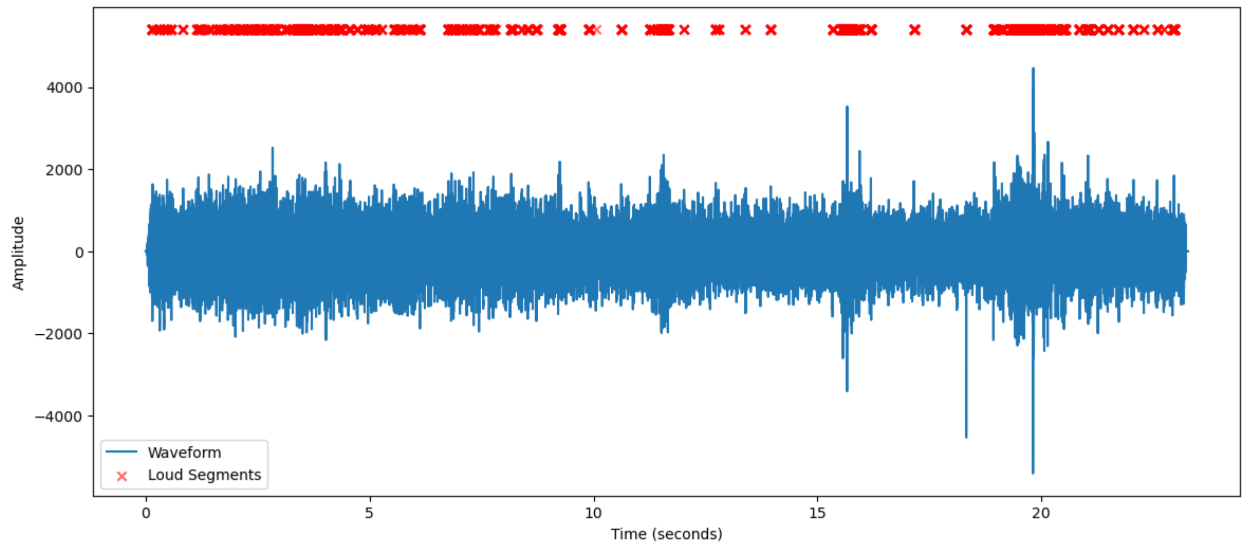

Binary class Model performance
Performance
Epoch
1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
accuracy
val_accuracy
loss
val_loss

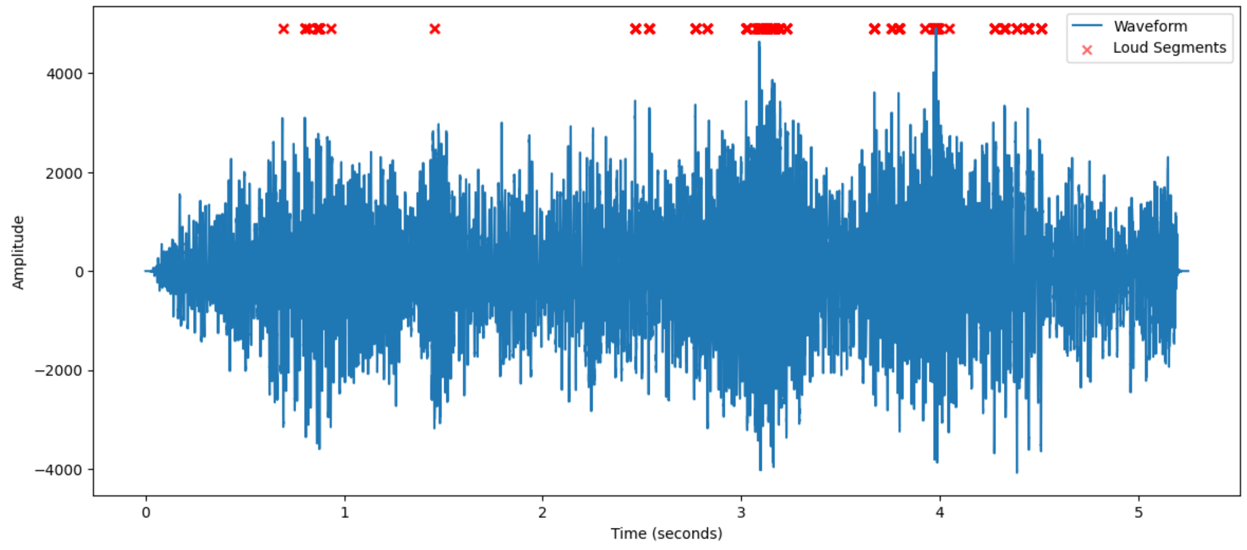

Multi class Model performance
Performance
Epoch
accuracy
val_accuracy
loss
val_loss
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

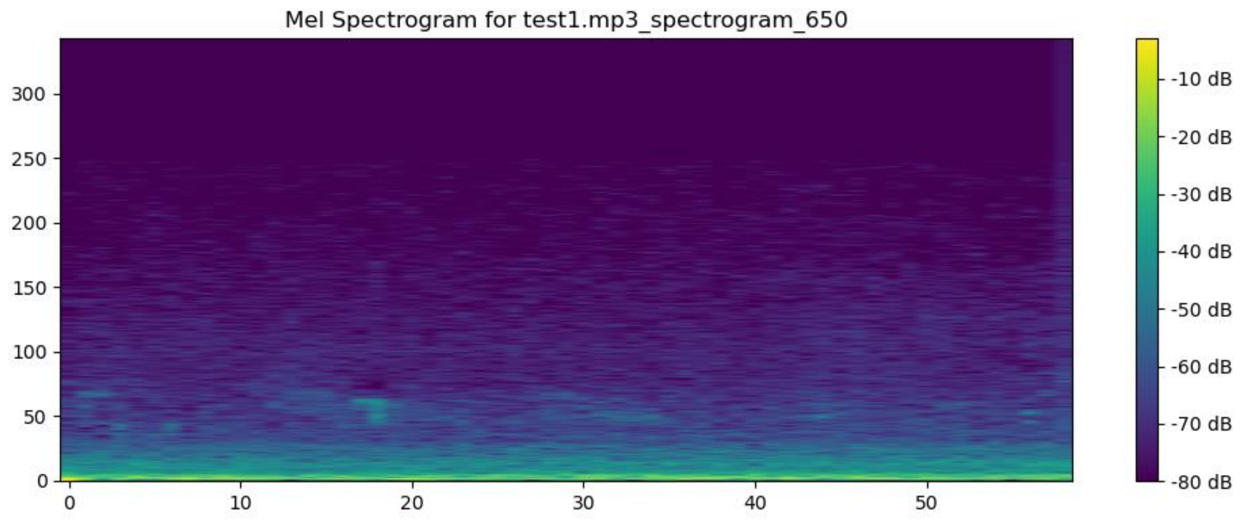

Comparison of Model Accuracies
Accuracy
Epochs
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2
4
6
8
Model 1 Training Accuracy
Model 1 Validation Accuracy
Model 2 Training Accuracy
Model 2 Validation Accuracy
Model 3 Training Accuracy
Model 3 Validation Accuracy
Model 4 Training Accuracy
Model 4 Validation Accuracy


Comparison of Model Loss
Loss
Epochs
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
Model 1 Training Loss
Model 1 Validation Loss
Model 2 Training Loss
Model 2 Validation Loss
Model 3 Training Loss
Model 3 Validation Loss
Model 4 Training Loss
Model 4 Validation Loss

Highlighted Loud Segments and Generated Spectrograms for Test Audio Files







Discussion:
The binary model’s rapid achievement of near-perfect accuracy demonstrates its significant potential utility in targeted ecological monitoring tasks, where specific species differentiation is crucial. The model successfully distinguished between American Crow and Steller's Jay, achieving high accuracy and minimal loss on both training and validation datasets. This result suggests that the binary model captured the distinguishing features of the two bird species effectively, highlighting the robustness and reliability of CNNs for binary classification tasks in bioacoustic applications.

However, the multi-class model presented more complex challenges. Despite this, Model 2 emerged as the best-performing architecture among the evaluated models, achieving perfect accuracy on both training and validation datasets. The deep architecture of Model 2, featuring additional convolutional and dense layers, enabled it to effectively learn and generalize the patterns in the data. The model demonstrated the lowest final training and validation loss values, indicating high confidence in its predictions. This performance underscores the effectiveness of a well-designed CNN architecture in handling multi-class classification tasks, even with diverse and complex datasets.

The baseline model (Model 1), the model with dropout regularization (Model 3), and the model using the SGD optimizer (Model 4) were all multi-class models designed to predict the following 12 bird species: American Crow ('amecro'), Barn Swallow ('barswa'), Black-capped Chickadee ('bkcchi'), Blue Jay ('blujay'), Dark-eyed Junco ('daejun'), House Finch ('houfin'), Mallard ('mallar3'), Northern Flicker ('norfli'), Red-winged Blackbird ('rewbla'), Steller's Jay ('stejay'), Western Meadowlark ('wesmea'), and White-crowned Sparrow ('whcspa'). Among these models, Models 1 and 3 performed well but had slightly higher loss values compared to Model 2, indicating slightly less confidence in their predictions. Model 4 showed significantly lower performance, with reduced accuracy and higher loss. This disparity highlights the importance of selecting appropriate optimizers and model architectures to achieve optimal performance in neural network training.

While both models achieved high validation accuracies, suggesting effective capture of distinguishing features of each bird species, the journey to these results highlighted several critical aspects. The binary model’s rapid convergence to high accuracy indicates that distinguishing between two species is relatively straightforward for CNNs, provided the data is well-preprocessed and labeled. In contrast, the multi-class model's development underscored the inherent challenges in broader bioacoustic applications. The need for comprehensive training and carefully tuned network architectures became evident to handle the diversity within the dataset.

The pre-trained Keras model, best_bird_species_classifier.keras, was successfully loaded and used to predict bird species from external test audio files (test1.mp3, test2.mp3, and

test3.mp3). The audio files were processed by converting them to NumPy arrays, detecting loud segments based on an energy threshold, and generating Mel spectrograms for these segments. The spectrograms were resized and normalized before being fed into the model for predictions. The model consistently predicted "wesmea" (Western Meadowlark) for all spectrograms with a probability of 1.0, suggesting a strong bias or overfitting towards this class. This could be due to an imbalance in the training data or feature similarities in the audio clips. Further investigation, including re-training with a balanced dataset and thorough model evaluation, is recommended to address this bias.
Challenging Species and Confusions: The model consistently predicted "wesmea," indicating it might be biased or overfitted. Listening to the bird calls and examining the spectrograms, it appears that the model did not differentiate between different bird species. This suggests that other species' calls were either too similar to the "wesmea" call in the feature space the model learned or that the model was not exposed adequately to diverse examples during training.

Alternative Models and Neural Network Suitability: Other models that could be used for this task include Support Vector Machines (SVM), Random Forests, and Gradient Boosting Machines (GBM). However, neural networks, especially Convolutional Neural Networks (CNNs), are well-suited for this application because they can effectively capture and learn from the complex patterns in spectrograms, which are essentially images representing the frequency content of the audio signals over time. Neural networks can automatically learn hierarchical features from the data, making them powerful for tasks involving image-like data, such as spectrograms in audio classification.

Conclusion:
This study evaluates the use of Convolutional Neural Networks (CNNs) for classifying bird species using audio spectrograms from the Xeno-Canto database. Two models were developed: a binary classifier to distinguish between American Crow and Steller's Jay, and a multi-class classifier to identify twelve bird species. The binary model achieved near-perfect accuracy, while the multi-class model, particularly the deeper Model 2, achieved perfect accuracy on both training and validation datasets.

The process involved converting audio recordings into spectrograms, normalizing, and padding them to ensure uniform input dimensions. The binary model's high accuracy demonstrates CNNs' effectiveness in specific species differentiation. In contrast, the multi-class model successfully handled the complexity of identifying multiple species, proving the robustness of well-designed CNN architectures.

However, challenges were noted, such as the model's consistent prediction of "wesmea" (Western Meadowlark) for all test samples, indicating potential biases or overfitting. This issue may stem from imbalanced training data or insufficient exposure to diverse examples, necessitating further investigation and re-training with a balanced dataset.

In conclusion, this research highlights the potential of CNNs in bioacoustic applications, offering reliable and scalable methods for bird species identification. The findings contribute to ecological monitoring and conservation efforts, providing valuable tools for understanding and preserving biodiversity. Future work should focus on refining these models, addressing biases, and exploring additional architectures to improve accuracy and reliability in automated bird call classification systems.

Citation:

1. James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. Retrieved from https://hastie.su.domains/ISLP/ISLP_website.pdf.download.html

Appendix:

```
mport h5py
import numpy as np

def load_species_data(file_path, species_keys):
    data = {}
    with h5py.File(file_path, 'r') as file:
        for key in species_keys:
            data[key] = file[key][:]
    return data

# Specify the file path and the two species you are interested in
file_path = '/Users/saluwaumuhoza/Downloads/spectrograms.h5'  # Change this to your actual
file path



# Load the data function
def load_species_data(file_path, species_keys):
    data = {}
    with h5py.File(file_path, 'r') as file:
        for key in species_keys:
            data[key] = file[key][:]
    return data

# Padding function
def pad_spectrograms(data, target_shape):
    padded_data = []
    for spectrogram in data:
        padding = [(0, max(0, target_shape[0] - spectrogram.shape[0])),  # Height padding
                   (0, max(0, target_shape[1] - spectrogram.shape[1])),  # Width padding
                   (0, 0)]  # Channel padding, necessary if your data includes a channel dimension
        padded_spectrogram = np.pad(spectrogram, padding, mode='constant',
constant_values=0)
```

```
        padded_data.append(padded_spectrogram)
    return np.array(padded_data)

# Load and preprocess the data
file_path = '/Users/saluwaumuhoza/Downloads/spectrograms.h5'
binary_species = ['amecro', 'stejay']
data_binary = load_species_data(file_path, binary_species)

# Determine the maximum dimensions
max_height = max(data_binary[sp].shape[1] for sp in binary_species)
max_width = max(data_binary[sp].shape[2] for sp in binary_species)

# Pad and add a channel dimension if necessary
X_bin = np.concatenate([pad_spectrograms(data_binary[sp], (max_height, max_width)) for sp
in binary_species])
X_bin = np.array([np.expand_dims(s, -1) for s in X_bin])

# Prepare labels
y_bin = np.array([0] * len(data_binary['amecro']) + [1] * len(data_binary['blujay']))

# Normalize the data
X_bin = X_bin / 255.0

# Split data
X_train_bin, X_test_bin, y_train_bin, y_test_bin = train_test_split(X_bin, y_bin, test_size=0.2,
random_state=42)

# Define the neural network model
def create_binary_model(input_shape):
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        MaxPooling2D(2, 2),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    return model

# Create and compile the model
binary_model = create_binary_model((max_height, max_width, 1))
binary_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Train the model
history_binary = binary_model.fit(X_train_bin, y_train_bin, epochs=10,
validation_data=(X_test_bin, y_test_bin))

np.random.seed(123)
tf.random.set_seed(123)

b1 = tf.keras.Sequential([
  tf.keras.layers.Input(shape=(343,52,1)),
  tf.keras.layers.Conv2D(32, kernel_size=(3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
  tf.keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(pool_size=(2,2)),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(1, activation='sigmoid')
])

b1.compile(optimizer='adam', loss='binary_crossentropy',  metrics=['accuracy'])

history = b1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10)

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Binary class Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()
def create_multiclass_model(input_shape, num_classes):
  model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(num_classes, activation='softmax')  # Softmax for multi-class
  ])
  return model

# Create and compile the model
```

```
multiclass_model = create_multiclass_model((max_height_multi, max_width_multi, 1),
len(multi_species))
multiclass_model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Combine all spectrograms and labels
spectrograms = []
labels = []

for idx, species in enumerate(species_keys):
    spectrograms += list(all_species_data[species])
    labels += [idx] * len(all_species_data[species])  # Create labels based on the index in
species_keys

# Find the maximum length of the time dimension among all spectrograms
max_len = max(s.shape[1] for s in spectrograms)

# Pad each spectrogram to have the same time dimension
padded_spectrograms = [np.pad(s, pad_width=((0, 0), (0, max_len - s.shape[1])),
mode='constant', constant_values=0) for s in spectrograms]

# Convert lists to numpy arrays
padded_spectrograms = np.array(padded_spectrograms)
labels = np.array(labels)

# Split the dataset into training and testing for multi-class classification
X_train_multi, X_test_multi, y_train_multi, y_test_multi =
train_test_split(padded_spectrograms, labels, test_size=0.2, random_state=1)

# Set seeds for reproducibility
np.random.seed(123)
tf.random.set_seed(123)

# Build the model
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train_multi.shape[1], X_train_multi.shape[2], 1)),  # Adjusted
input shape
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(species_keys), activation='softmax')  # Output layer for multi-class
classification
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

#Baseline Model
model_1 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train_multi.shape[1], X_train_multi.shape[2], 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(species_keys), activation='softmax')
])

model_1.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

#Deeper model
model_2 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train_multi.shape[1], X_train_multi.shape[2], 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(len(species_keys), activation='softmax')
])

model_2.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

#Model with Dropout for regularization
model_3 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train_multi.shape[1], X_train_multi.shape[2], 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Dropout(0.25),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(species_keys), activation='softmax')
])

model_3.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

#Different oprimizer
model_4 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train_multi.shape[1], X_train_multi.shape[2], 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(128, kernel_size=(3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dense(len(species_keys), activation='softmax')
])

model_4.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

def plot_histories(histories, title, metric='accuracy'):
    plt.figure(figsize=(14, 7))

    for history, label in histories:
        plt.plot(history.history[metric], label=f'{label} Training {metric.capitalize()}')
        plt.plot(history.history[f'val_{metric}'], label=f'{label} Validation {metric.capitalize()}')

    plt.title(title)
    plt.xlabel('Epochs')
    plt.ylabel(metric.capitalize())
    plt.legend()
    plt.show()

# List of model histories and labels
```

```
histories = [
    (history_1, 'Model 1'),
    (history_2, 'Model 2'),
    (history_3, 'Model 3'),
    (history_4, 'Model 4')
]

# Plot training and validation accuracy for all models
plot_histories(histories, 'Comparison of Model Accuracies', metric='accuracy')
plot_histories(histories, 'Comparison of Model Loss', metric='loss')
# Directory containing the test audio files
test_dir = '/Users/saluwaumuhoza/Downloads/test_birds'

# List of test audio files
test_files = ['test1.mp3', 'test2.mp3', 'test3.mp3']
test_file_paths = [os.path.join(test_dir, file) for file in test_files]

# Load Model 2
best_model = tf.keras.models.load_model('best_bird_species_classifier.keras')

# Ensure the input shape matches the model's expected input shape
input_shape = (128, 343)

# Function to load and preprocess an audio file using pydub
def load_and_preprocess_audio(file_path, input_shape):
    try:
        print(f"Loading audio file: {file_path}")
        audio = AudioSegment.from_file(file_path)
        y = np.array(audio.get_array_of_samples(), dtype=np.float32)
        sr = audio.frame_rate
        print(f"Audio file {file_path} loaded, length: {len(y)}")
        if len(y) == 0:
            raise ValueError("Loaded audio file is empty.")
    except Exception as e:
        print(f"Error loading audio file {file_path}: {e}")
        return None

    # Convert audio to Mel spectrogram
    def extract_mel_spectrogram(y, sr, n_mels=128, hop_length=512):
        S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels, hop_length=hop_length)
        S_dB = librosa.power_to_db(S, ref=np.max)
        return S_dB

    mel_spectrogram = extract_mel_spectrogram(y, sr)
    print(f"Mel spectrogram shape: {mel_spectrogram.shape}")

    # Pad the spectrogram to the same length as the training data
    def pad_spectrogram(spectrogram, input_shape):
        target_shape = input_shape[1]
        if spectrogram.shape[1] < target_shape:
            padding_width = target_shape - spectrogram.shape[1]
            padded_spectrogram = np.pad(spectrogram, pad_width=((0, 0), (0, padding_width)),
mode='constant', constant_values=0)
        else:
            padded_spectrogram = spectrogram[:, :target_shape]
        return padded_spectrogram

    padded_spectrogram = pad_spectrogram(mel_spectrogram, input_shape)
    print(f"Padded spectrogram shape: {padded_spectrogram.shape}")

    # Reshape to add the channel dimension
    test_input = padded_spectrogram[np.newaxis, ..., np.newaxis]
    print(f"Test input shape: {test_input.shape}")
    return test_input

# Print model summary to understand input and output shapes
best_model.summary()

# Load and preprocess the first test file to get an initial input
initial_test_input = load_and_preprocess_audio(test_file_paths[0], input_shape)

# Predict using the initial input to call the model and define its input
if initial_test_input is not None:
    initial_prediction = best_model.predict(initial_test_input)
```