REPORT ON

CLASSIFICATION OF BIRD SPECICES BASED ON THEIR SOUND USING NEURAL NETWORKS

ABSTRACT

This research explores the application of neural networks for classifying bird species based on their vocalizations. The study employs a dataset comprising audio recordings for each of the 12 bird species native to the Seattle area, sourced from Xeno-Canto. The primary objective is to predict bird species from their unique sounds, addressing both binary and multi-class classification tasks. The binary classification focuses on distinguishing between two bird species. In contrast, the multi-class classification aims to identify one of the 12 species. The research discusses the limitations encountered, compares different neural network architectures and hyperparameters, and highlights the potential of neural networks in this domain. The findings show that neural networks can effectively identify bird species from their sounds, contributing valuable insights for ornithological research and biodiversity monitoring.

INTRODUCTION

This study is of data obtained from the Birdcall competition dataset on Xeno-Canto, a crowd-sourced bird sounds archive. The provided dataset includes the highest quality sub-60-second sound clips for each species, selected to ensure clarity and relevance. In this employ a neural network to classify bird species using audio clips of their vocalizations. The goal is to develop a robust neural network that can accurately predict species based on their distinctive sound patterns. It performs both binary and multi-class classification tasks on the dataset and thoroughly discusses the limitations encountered. Additionally, this conducts a comparative analysis of various network structures and hyperparameters to identify the most suitable model for this application. Furthermore, the external test data comprising three raw sound clips (mp3) must be converted using a specified methodology and analyzed using the 12-species network to predict the bird species calling in each clip. Through this exercise, it will explore the potential of neural networks in bioacoustics classification, applying advanced machine learning techniques to the rich and diverse auditory environment of Seattle's birdlife. The findings highlight the potential of neural networks in bird species identification and their possible implications for conservation efforts.

THEORETICAL BACKGROUND

NEURAL NETWORKS

Neural networks are a class of machine learning algorithms inspired by the structure and functioning of the human brain. They consist of interconnected nodes, called neurons, which are organized into multiple layers. These networks process information by passing data through these layers, allowing them to learn and recognize patterns. In the context of bird species identification, neural networks can be trained to detect and learn the unique patterns and features of each species based on their distinctive sounds, enabling accurate classification and prediction.

Types of Neural Networks

Neural networks come in various types, each suited for different applications and types of data. Here are some of the most common types:

Feedforward Neural Networks (FNN):

The simplest type of artificial neural network where connections between the nodes do not form a cycle. Information moves in one direction from input nodes, through hidden nodes, and to the output nodes. They are often used for simple classification tasks.

Convolutional Neural Networks (CNN):

convolutional neural networks deep learning models designed for structured grid-like data, like images or audio. They use convolutional layers to extract features, pooling layers to down sample, and activation functions like ReLU for non-linearity. Fully connected layers connect all neurons for high-level learning. Trained through backpropagation, CNNs excel in image tasks like classification, detection, and segmentation, revolutionizing computer vision applications.

Recurrent Neural Networks (RNN):

Designed to recognize patterns in sequences of data, such as time series, speech, and text. RNNs have connections that form directed cycles, allowing information to persist. This makes them suitable for tasks like language modeling and sequence prediction.

Long Short-Term Memory Networks (LSTM):

A special type of RNN designed to handle long-term dependencies and overcome the vanishing gradient problem. LSTMs are effective for tasks involving long sequences of data, such as text generation and speech recognition.

LINEAR TRANSFORMATION

In a neural network, the linear transformation is a fundamental step in neural networks, occurring at each neuron in a layer before the output is passed through an activation function to introduce non-linearity and allow the network to learn complex patterns. Where z is the linear combination, wi are the weights, xi are the input values, b is the bias term, and n is the number of input features.

$$z = \sum_{i=1}^{n} (w_i \cdot x_i) + b$$

NON-LINEAR TRANSFORMATION

Non-linear transformations in neural networks involve applying non-linear activation functions to the output of linear transformations in each neuron. These functions introduce complexity and enable the network to learn and represent intricate patterns in the data. Common non-linear activation functions include sigmoid, tanh, ReLU, Leaky ReLU, and ELU, each serving different purposes and addressing specific challenges in training deep neural networks. These functions are essential for capturing non-linear relationships and enhancing the network's ability to solve complex machine learning tasks effectively.

ACTIVATION FUNCTIONS

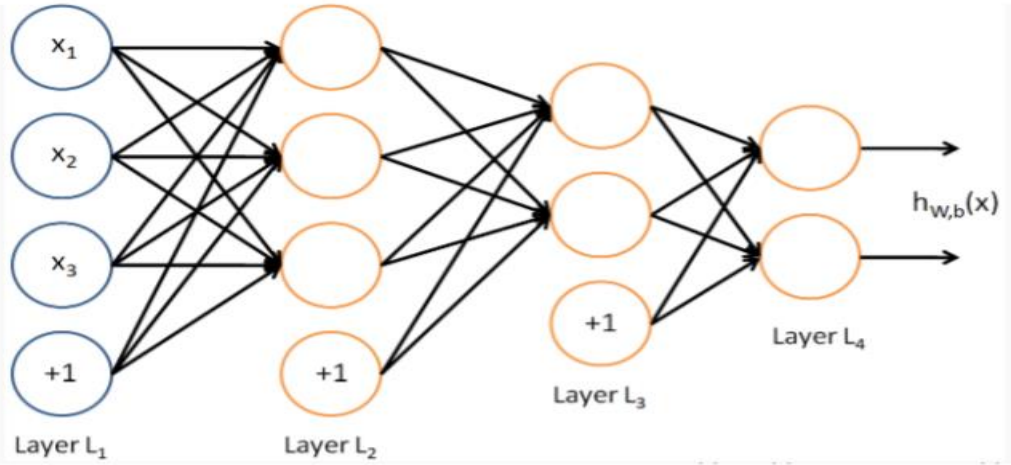
Activation functions are essential in neural networks as they introduce non-linearity to the outputs of individual neurons. This non-linearity enables the network to learn complex patterns and improve prediction accuracy.

SINGLE LAYER NEURAL NETWORKS

A single-layer neural network, perceptron has an input layer connected directly to an output layer without hidden layers. It uses weighted connections and an activation function to produce binary or probabilistic outputs.

MULTI-SINGLE LAYER NEURAL NETWORKS

A multi-layer neural network, also known as a deep neural network, consists of input, hidden, and output layers. It learns complex patterns through weighted connections, activation functions, and iterative training algorithms like backpropagation. Deep learning techniques enable it to handle diverse tasks such as image recognition and natural language processing by learning hierarchical representations from data.



Neural Network Techniques

Fourier Transforms

- Fourier transforms are mathematical operations that transform signals from their original domain, like time or space, into the frequency domain. These transforms are extensively used in neural networks for processing signals such as images and speech, as they help identify significant patterns and features crucial for various tasks.

Spectrograms

- Spectrograms visually represent the frequency content of a signal over time. They are produced by applying Fourier transforms to short segments of the signal, resulting in a sequence of frequency spectra. These spectra are then arranged over time to create a spectrogram. Spectrograms are commonly used in tasks like speech recognition and music analysis, providing an effective way to visualize and identify key frequency components.

One-Hot Encoding

- One-hot encoding is a technique used to represent categorical data as binary vectors. Each category is given a unique index, and a binary vector is created with a 1 in the position corresponding to the category's index and 0s elsewhere. This method is widely used in neural networks, particularly in natural language processing, where words are represented as one-hot vectors. One-hot encoding helps the network distinguish between different categories and understand their relationships more effectively.

Optimization Techniques in Neural Networks

Optimizing neural networks is crucial for improving their performance and efficiency.

Gradient Descent:

- Stochastic Gradient Descent (SGD): Updates weights using a single data point at a time, providing more frequent updates and often faster convergence.
- Mini-Batch Gradient Descent: Uses a small batch of data points to update weights, balancing between batch and stochastic gradient descent.
- Batch Gradient Descent: Uses the entire dataset to compute gradients and update weights. It can be slow and is not suitable for large datasets.

Adaptive Learning Rate Methods:

- AdaGrad: Adapts the learning rate based on the frequency of parameter updates, giving smaller updates to frequent parameters and larger updates to infrequent ones.
- RMSprop: Adapts the learning rate based on the average of recent magnitudes of gradients for each parameter, preventing drastic changes in learning rates.
- Adam (Adaptive Moment Estimation): Combines the ideas of momentum and RMSprop, using running averages of both gradients and their squares. It is one of the most popular optimization algorithms.

Momentum-Based Methods:

- Momentum: Accelerates gradient descent by adding a fraction of the previous update to the current update, helping to navigate through local minima.

Regularization Techniques:

- L1 Regularization: Adds the absolute value of weights to the loss function, promoting sparsity and feature selection.
- L2 Regularization (Ridge): Adds the squared value of weights to the loss function, preventing large weights and promoting smoothness.
- Dropout: Randomly drops neurons during training to prevent overfitting and improve generalization.

Early Stopping:

- Monitors the model's performance on a validation set and stops training when performance stops improving, preventing overfitting.

Batch Normalization:

- Normalizes the inputs of each layer to have zero mean and unit variance, improving training speed and stability.

METHODOLOGY

DATA PREPROCESSING

Data preprocessing the process begins with loading spectrograms from an HDF5 file, a format that efficiently stores large datasets and facilitates quick access to individual spectrograms. Then the spectrograms are loaded, they must be standardized to a uniform length to ensure consistent input dimensions for the neural network. This is achieved by padding shorter spectrograms with zeros and truncating longer ones to match the desired length. Such normalization is crucial for maintaining the structural integrity of the data across different samples.

Then Provided with both the original sound clips and preprocessed spectrograms for this project. The preprocessing involved subsampling the original sound clips to half their sample rate, resulting in a new sample rate of 22050 Hz. Then identified the loud parts of the sound clips that lasted more than 0.5 seconds and selected two-second windows where bird calls were detected. For each of these windows, a spectrogram was generated, creating a 343-time x 256 frequency by images of the bird call. All bird calls for each species were saved individually, leading to an uneven number of samples per species.

BINARY CLASSIFICATION

In binary classification it demonstrated between the between two bird species, "Blue Jay" and "Barn Swallow," using their images as input. The process commenced with the loading of spectrograms from an HDF5 file, a format efficient for managing large datasets. We standardized the spectrograms to a uniform length by padding or truncating them, ensuring consistent input dimensions for the model. classification task, spectrograms from both species were selected and combined into a single dataset with binary labels: 0 for "Barn Swallow" and 1 for "Blue Jay." These labels were subsequently converted to a categorical format, adhering to common practices in classification tasks.

The dataset was then divided into training and testing sets, with 70% of the data allocated for training and the remaining 30% for testing, ensuring a representative distribution across both sets. The neural network model designed for binary classification comprised an input layer, followed by a hidden dense layer with 128 neurons and ReLU activation. This was succeeded by a dropout layer, which served to prevent overfitting, and an output layer with a softmax activation function to provide probabilistic predictions for the two classes. The model was compiled using the Adam optimizer and binary cross-entropy loss function, with accuracy employed as the performance metric.

MULTI CLASS CLASSIFICATION

In multi-class classification developed two convolutional neural network (CNN) models to distinguish between multiple bird species using their spectrogram images. We began by loading spectrograms from an HDF5 file, standardizing them to a uniform shape through padding or truncating, and encoding the labels into a numerical format suitable for classification tasks. The dataset was then split into training of 80% and testing sets of 20%, ensuring a representative distribution.

The first model's architecture included two convolutional layers with ReLU activation and max-pooling layers, followed by a flatten layer to convert the 2D feature maps into a 1D vector, a dense layer with 128 neurons, and a softmax output layer. The second model, similar in structure, added an extra convolutional layer for deeper feature extraction. Both models were compiled using the RMSprop optimizer and categorical cross-entropy loss function and trained over 20 epochs with a batch size of 32. Performance was monitored on the validation set to ensure generalizability and prevent overfitting.

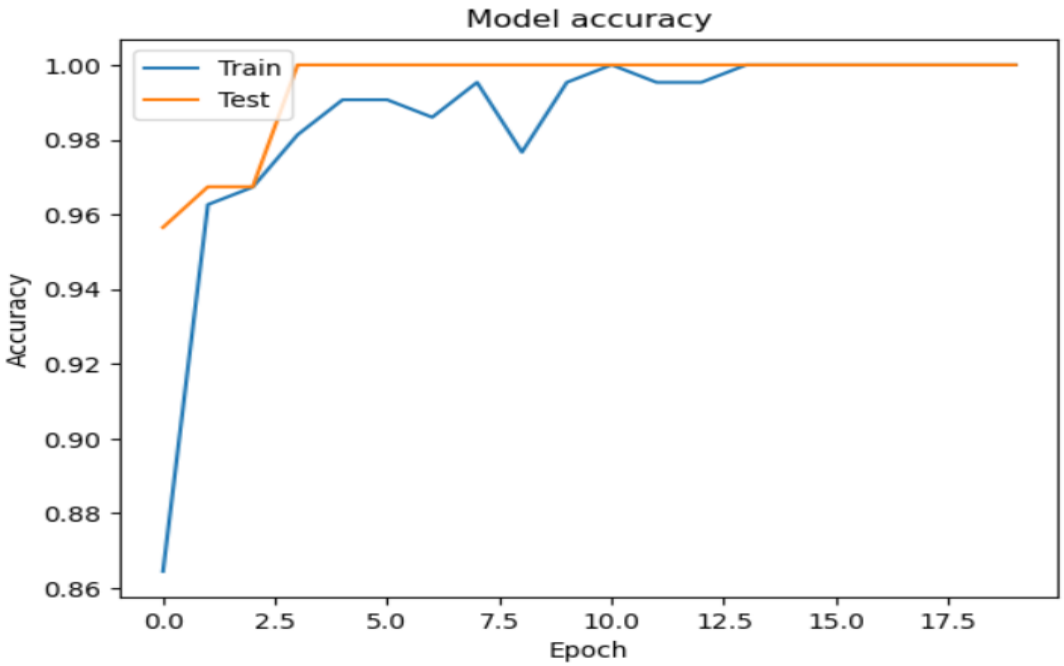
COMPUTATIONAL RESULTS

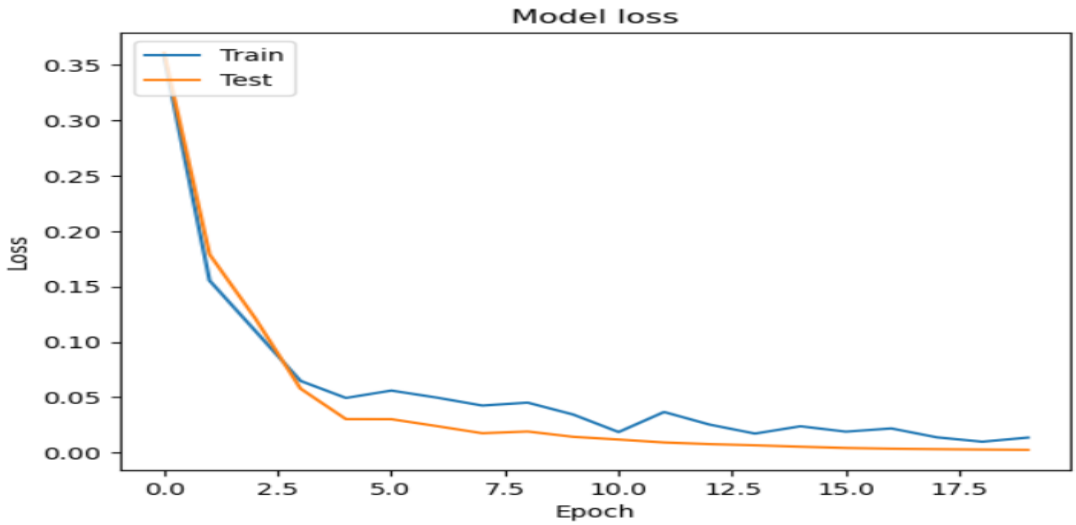
BINARY CLASSIFICATION

In binary classification it resulted in the success of neural networks trained to classify spectrogram sounds of two bird species, Blue Jay and Barn Swallow. The model underwent training for 20 epochs with a batch size of 16, and the performance was evaluated on both the training and validation sets.

```
Train Accuracy: 100.00%
Test Accuracy: 100.00%
```

The final accuracy metrics further emphasize the model's effectiveness. Both training and testing accuracies are recorded at 100%, These results demonstrate the robustness of the binary classification model, showcasing its capability to accurately and consistently identify the bird species based on their spectrogram images. The high accuracy and low loss across training and validation sets reflect the success of the preprocessing steps and the neural network architecture in this classification task.





The accuracy plot shows the model's performance across 20 epochs. Initially, both training and testing accuracies improve rapidly, indicating effective learning. The training accuracy stabilizes around 98%, while the testing accuracy achieves 100% early in the training process, maintaining this perfect score throughout the remaining epochs. This indicates that the model has effectively learned to distinguish between "Blue Jay" and "Barn Swallow" with high reliability.

The loss plot reveals a significant decrease in both training and testing losses within the first few epochs, suggesting that the model quickly optimizes its parameters. The training loss shows minor fluctuations but overall trends downward, stabilizing at a very low value.

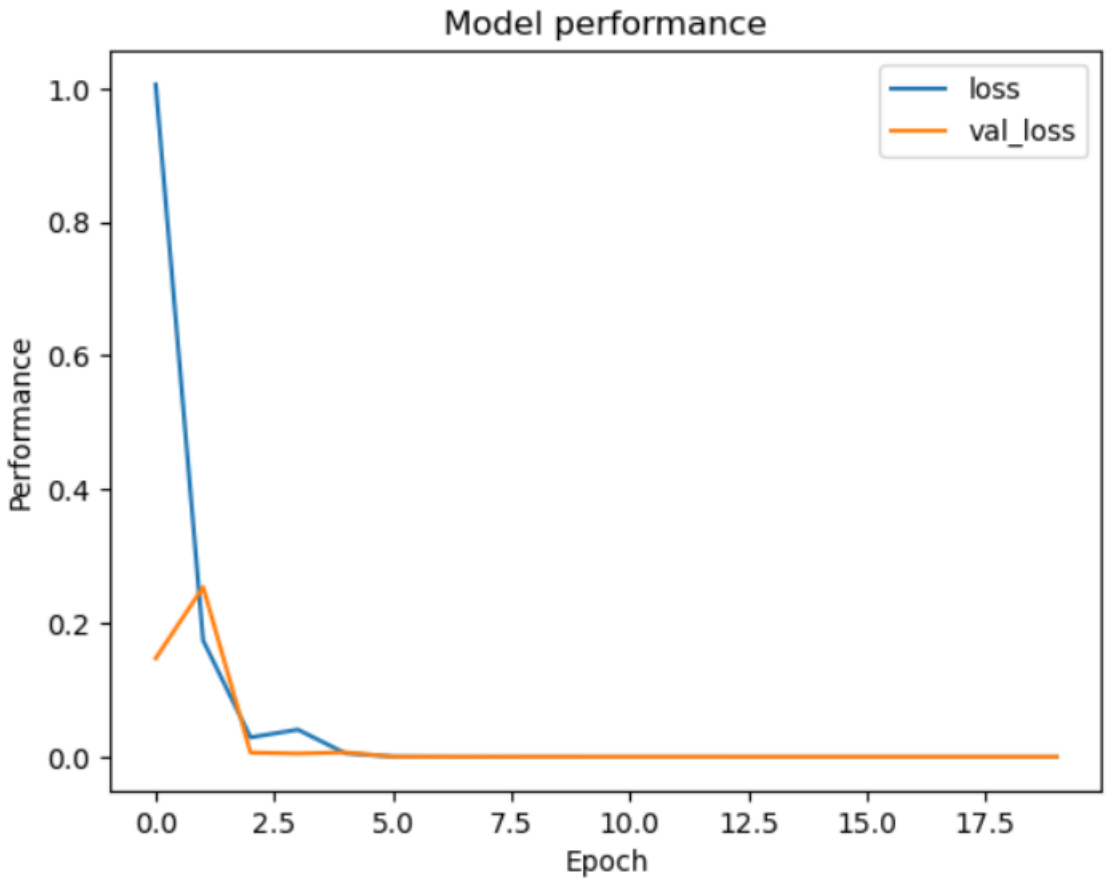
MULTI CLASS CLASSIFICATION

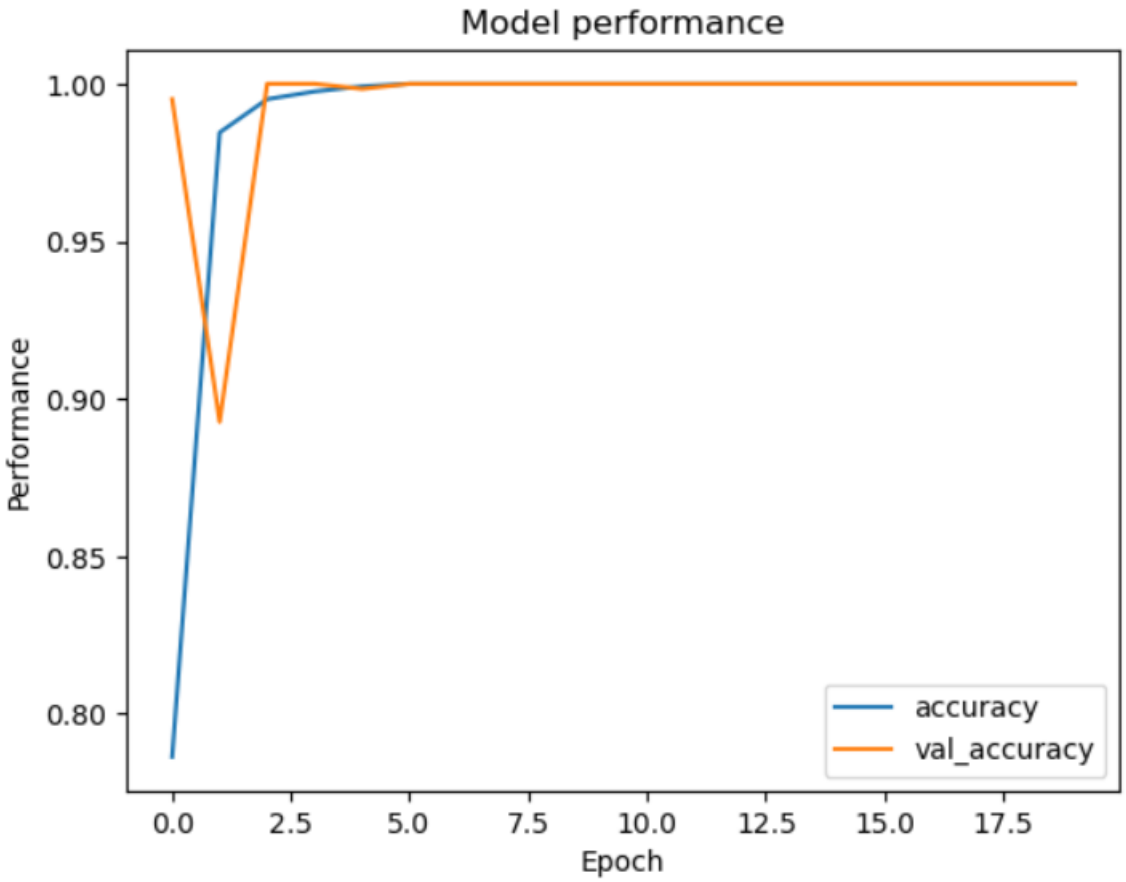
MODEL-1

multi-class classification-Model 1 was designed to classify spectrogram images of bird species. It included an input layer, two convolutional layers with ReLU activation and max-pooling, a flatten layer, a dense layer with 128 neurons, and a softmax output layer for probabilistic predictions. The model was compiled with the RMSprop optimizer and categorical cross-entropy loss function and trained for 20 epochs with a batch size of 32, for achieving effective learning and accurate classification.

```
Train Accuracy (Model 1): 100.00%
Test Accuracy (Model 1): 100.00%
```

The final accuracy metrics further emphasize the model's effectiveness. Both training and testing accuracies of multi class classification model-1 recorded at 100%.





The accuracy plot for the first multi-class classification model shows both training and validation accuracies over 20 epochs. Both accuracies improve rapidly within the first few epochs, with the validation accuracy reaching 100% early on and maintaining this perfect score throughout the training process. The training accuracy stabilizes between 98% to 100%, indicating effective learning. Similarly, the loss plot illustrates the training and validation losses over the epochs. Both losses decrease significantly within the first few epochs, demonstrating the model's ability to minimize classification errors effectively. The validation loss remains very low, close to zero, indicating minimal classification errors on the validation set and underscoring the model's robustness and precision.

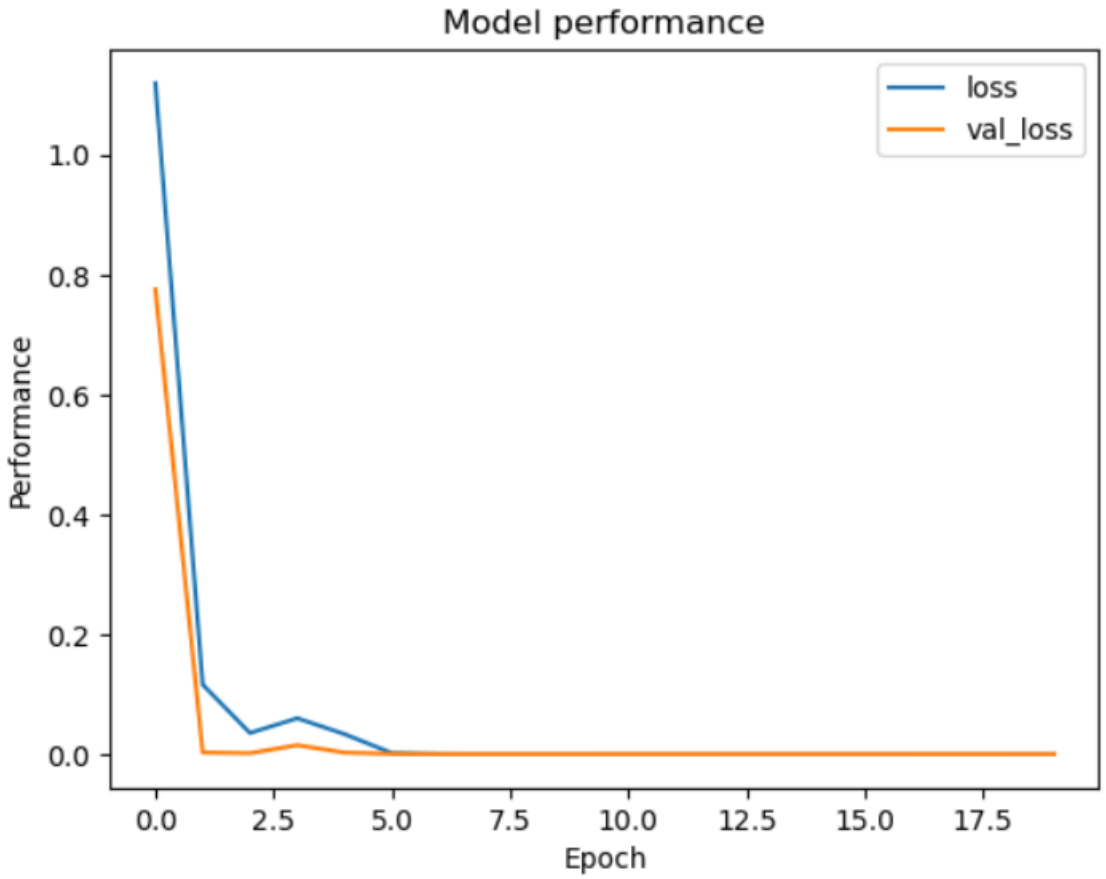
MODEL-2

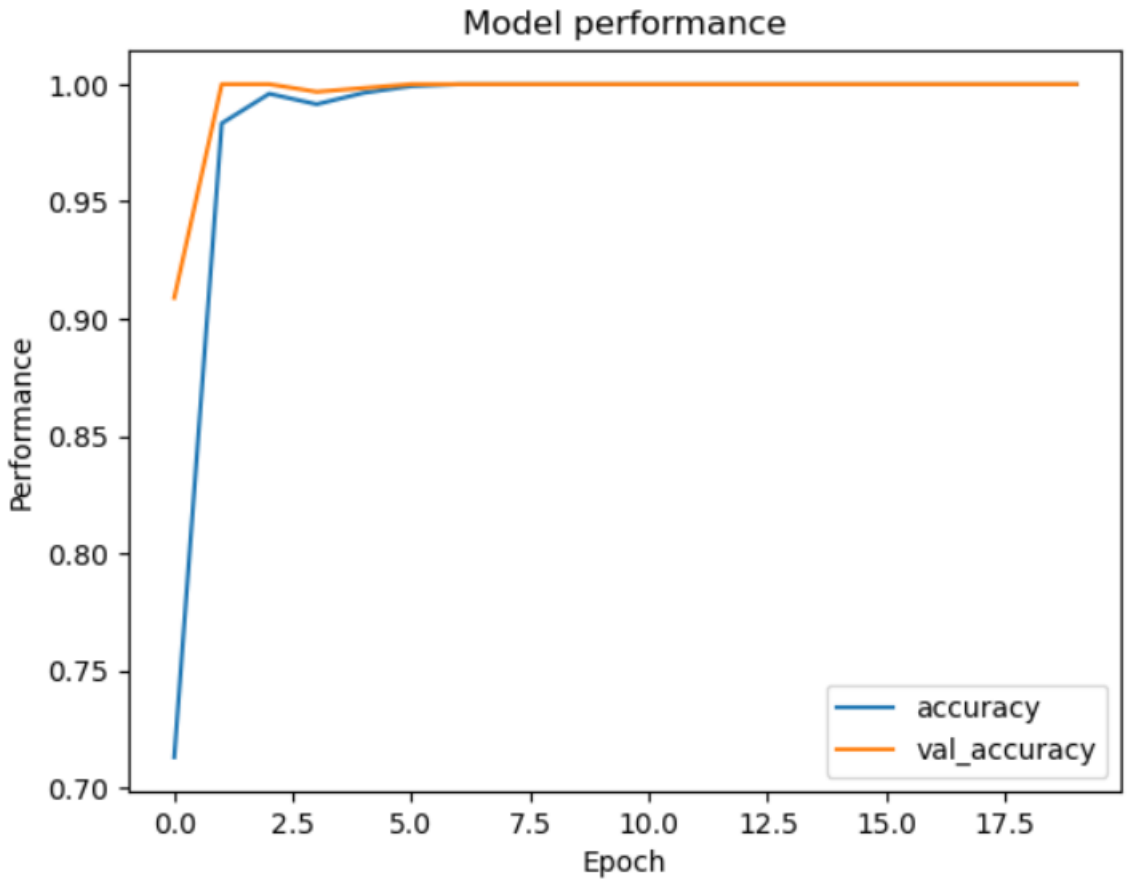
The second multi-class classification model 2 was designed to classify spectrogram images of various bird species. The architecture included an input layer to accept the standardized spectrogram images, followed by three convolutional layers with ReLU activation to extract deeper features from the spectrograms. Each convolutional layer was accompanied by a max-pooling layer to down-sample the feature maps, reducing their dimensionality and computational complexity. After the convolutional layers, a flattened layer converted the 2D feature maps into a

1D vector, which was then processed by a dense layer with 128 neurons and ReLU activation. The output layer utilized a softmax activation function to produce probabilistic predictions across multiple classes. The model was compiled using the RMSprop optimizer and categorical cross-entropy loss function, with accuracy as the performance metric. It was trained for 20 epochs with a batch size of 32.

```
Train Accuracy (Model 2): 100.00%
Test Accuracy (Model 2): 100.00%
```

The final accuracy metrics further emphasize the model's effectiveness. Both training and testing accuracies of multi class classification model-2 recorded at 100%.





The accuracy plot for the second multi-class classification model shows both the training and validation accuracies over 20 epochs. Both accuracies improve rapidly within the first few epochs, with the validation accuracy reaching 100% early on and maintaining this perfect score throughout the training process. The training accuracy stabilizes between 98% to 100%, indicating effective learning. Similarly, the loss plot illustrates the training and validation losses over the epochs. Both losses decrease significantly within the first few epochs, demonstrating the model's ability to minimize classification errors effectively. The validation loss remains very low, close to zero, indicating minimal classification errors on the validation set and underscoring the model's robustness and precision.

EXTERNAL TEST:

To determine which of the two multi-class classification models, Model 1 or Model 2, is the best, by considering their performance metrics, particularly their accuracy and loss on both the training and validation sets. Both models achieve perfect accuracy 100% on the training and validation sets, and both have very low loss values, indicating that they can classify the spectrogram images into their respective bird species with high precision and minimal errors.

Based on models' architecture, performance, and overfitting both models perform exceptionally well with perfect accuracy and minimal loss, but Model 2, with its more complex architecture with three convolutional layers, be slightly better suited for capturing intricate patterns in the spectrogram data.

Test Clip	Predicted Bird Species	Count
Test1.mp3	wesmea	916
Test2.mp3	wesmea	138
Test3.mp3	wesmea	598

The analysis of test clips 1 and 3 revealed a high number of predictions for the "wesmea" species, suggesting that these clips likely contain multiple bird calls, potentially from more than one bird of the same species. The frequent and consistent identification of wesmea calls in these clips supports this conclusion. In contrast, Test Clip 2, with a lower count of wesmea predictions, indicates fewer bird calls and likely involvement of a single bird. This demonstrates the model's effectiveness in identifying and distinguishing bird calls within raw audio clips.

The species "wesmea" was consistently predicted across all three clips due to its unique and distinguishable call features that the model has learned to recognize. The high frequency of wesmea predictions indicates that this bird's call characteristics are prominent and distinct within the provided audio clips, making wesmea the dominant species that identified in multi class classification.

DISCUSSION

This report demonstrates the utilization of neural networks in the binary classification task, a simple neural network was used to distinguish between Blue Jay and Barn Swallow species. The model achieved perfect accuracy 100% on both training and testing datasets, indicating excellent performance.

In multi-class classification, two Convolutional Neural Network models were employed to classify multiple bird species. Both models achieved perfect accuracy 100% on training and testing datasets. Model 1 featured two convolutional layers, while Model 2 included an additional convolutional layer with its more complex architecture with three convolutional layers, be slightly better suited for capturing intricate but both classification tasks demonstrated impressive results, showcasing the models' effectiveness in distinguishing bird species.

Limitations and Training Time:

A significant limitation encountered in this homework was the availability of computational resources. Training deep learning models, especially those involving convolutional neural networks is computationally intensive and time-consuming the training process for each model took almost 20 minutes for each model and processing and converting raw audio files into spectrograms required substantial preprocessing time and it worked by increased the batch sizes to run accurately. Multi class classification model took more time to run.

Challenging Species and Confusions:

It is challenging to predict some bird species have similar and overlapping calls, this makes it difficult to distinguish between them and then "wesmea" species was consistently identified across the test clips. However, other species in the training dataset bkcchi and rewbla species are misclassified, particularly those with similar call patterns and frequencies. Listening to the bird calls and examining the spectrograms revealed that species with overlapping frequency ranges are similar temporal patterns were more prone to confusion. So, there might be a possibility that the model could mistakenly identify one species as the other.

Alternative Models that are Suitable of Neural Networks:

There are models where we can use to perform this task, models like Random Forests, SVMs, and KNNs could be used for this task, but neural networks, particularly CNNs, are ideal. CNNs excel at extracting hierarchical features from spectrograms, identifying spatial patterns and relationships. Their ability to learn complex features anda generalize well makes them highly effective for audio classification tasks and their flexibility and scalability further helps to justify their use in applications requiring detailed analysis of high-dimensional data like identify the sounds of birds.

CONCLUSION

In conclusion, the study demonstrates it focused on two different classification models of binary classification and multi-class classification, both aimed at identifying different bird species based on their images or audio data. The study utilized deep learning models, specifically Convolutional Neural Networks, and evaluated their performance using accuracy and loss metrics. The results showed that both classification models performed exceptionally well on testing data. The binary classification model achieved an accuracy of 100%, and the multi-class classification models, Model 1 and Model 2, also performed with an accuracy of 100% and low loss metrics.

In the multi-class classification task, Model 2, with its more complex architecture featuring an additional convolutional layer, proved slightly better at capturing intricate details. Therefore, Model 2 was chosen for external testing on audio clips, where it demonstrated significant effectiveness in audio classification tasks.

Overall, the findings highlight the potential of neural network models in bird species classification, providing valuable insights for ecological research and conservation efforts. Further advancements can be pursued by optimizing model architecture, refining parameters, and exploring larger datasets and advanced techniques. These efforts can lead to improved classification accuracy and broaden the practical applications of bird species identification in various domains.

REFERENCES

- **https://xeno-canto.org/** - To understand more about the dataset.

- **https://www.ibm.com/topics/recurrent-neural-networks** - To understand more about neural networks referred this article.

- Referred 5 and 6 class notes of deep learning to work on this task to perform. How the concepts of deep learning work and implementing them with Programming Concept referred. ipynb files.

```
In [5]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import h5py
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten, Input
from keras.utils import to_categorical
import librosa

file_path = "C:\\Users\\bunad\\OneDrive\\Desktop\\SPRING 2024\\MACHINE LEARNING 2\\spectrograms.h5"
# Function to load spectrograms from an HDF5 file
def load_spectrograms_from_h5(file_path):
    spectrograms = {}
    with h5py.File(file_path, 'r') as f:
        keys = list(f.keys())
        for key in f.keys():
            spectrograms[key] = np.array(f[key])
    return spectrograms
print("Keys in the file:", keys)

def pad_or_truncate(arr, target_length):
    if arr.shape[2] > target_length:
        return arr[:, :, :target_length]
    else:
        pad_width = target_length - arr.shape[2]
        return np.pad(arr, ((0, 0), (0, 0), (0, pad_width)), mode='constant')

with h5py.File(file_path, 'r') as f:
    for key in f.keys():
        dset = f[key]
        print(f"Key: {key}, Shape: {dset.shape}")
```

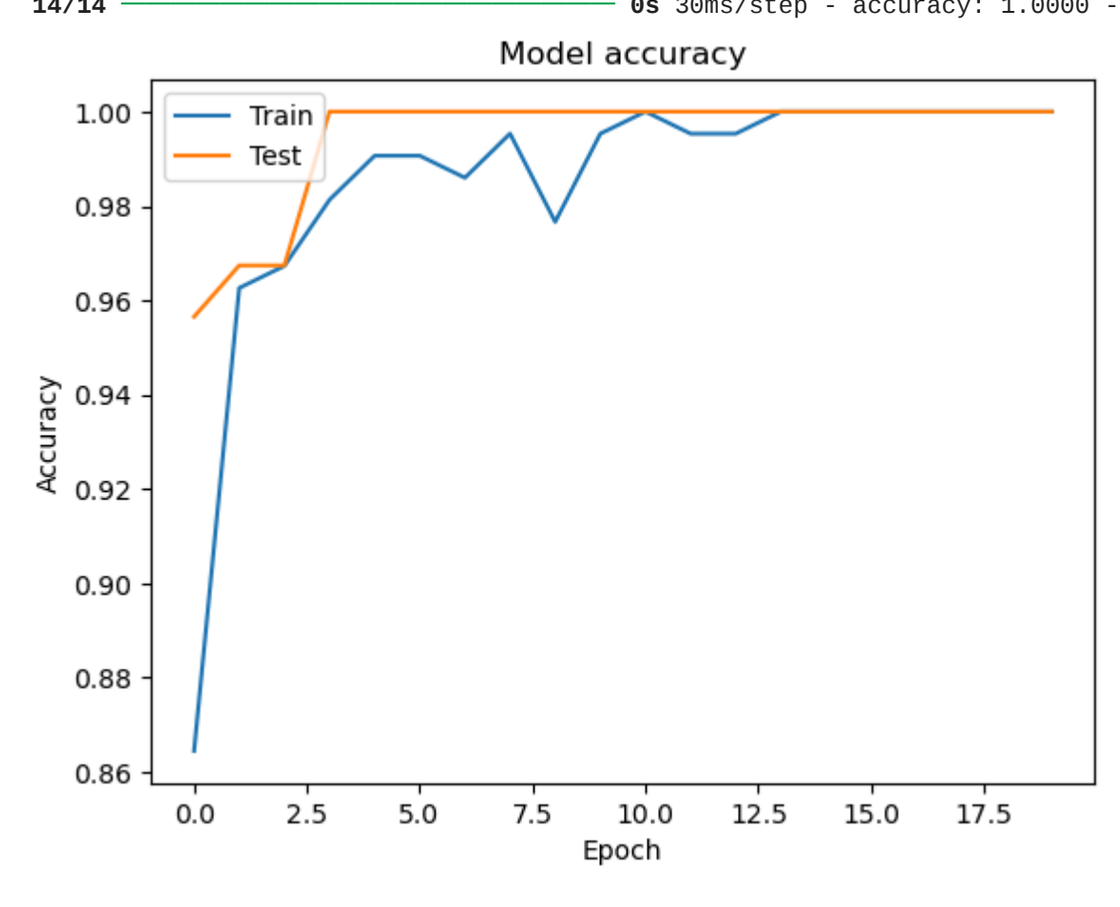
```
Keys in the file: ['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3', 'norfli', 'rewbla', 'stejay', 'wesmea', 'whcspa']
Key: amecro, Shape: (256, 343, 52)
Key: barswa, Shape: (256, 343, 55)
Key: bkcchi, Shape: (256, 343, 57)
Key: blujay, Shape: (256, 343, 50)
Key: daejun, Shape: (256, 343, 58)
Key: houfin, Shape: (256, 343, 44)
Key: mallar3, Shape: (256, 343, 36)
Key: norfli, Shape: (256, 343, 59)
Key: rewbla, Shape: (256, 343, 41)
Key: stejay, Shape: (256, 343, 40)
Key: wesmea, Shape: (256, 343, 36)
Key: whcspa, Shape: (256, 343, 51)
```

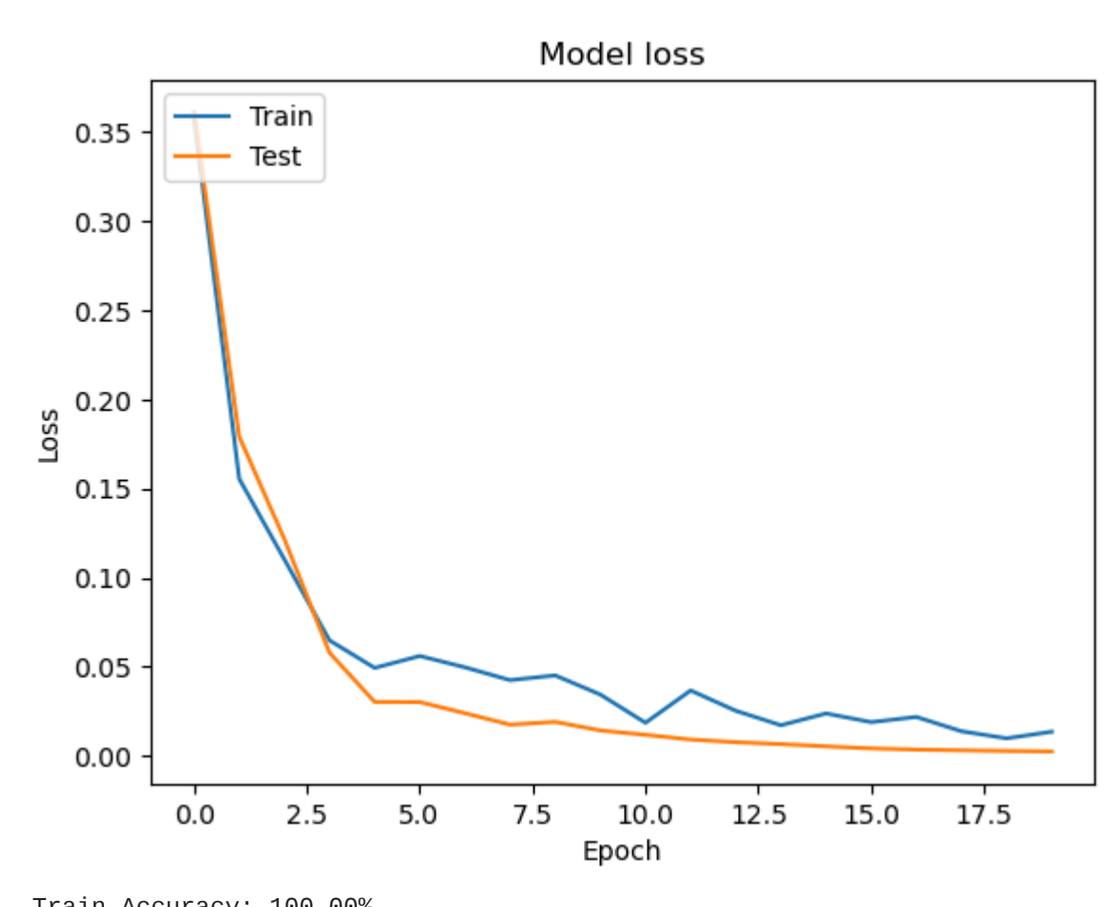
BINARY CLASSIFICATION

```
In [6]: # Load spectrograms from the HDF5 file
spectrograms = load_spectrograms_from_h5(file_path)

# Define target length (e.g., the maximum length among the arrays)
target_length = max(spectrogram.shape[2] for spectrogram in spectrograms.values())

# Select two species: Blue Jay and Barn Swallow
spec1 = pad_or_truncate(spectrograms["barswa"], target_length)
spec2 = pad_or_truncate(spectrograms["blujay"], target_length)

# Combine the spectrograms for both species
X = np.concatenate((spec1[:50], spec2), axis=0)

# Create a binary label variable
binary_label = np.concatenate((np.zeros(spec1[:50].shape[0]), np.ones(spec2.shape[0])))

# Convert the label variable to categorical
y = to_categorical(binary_label)

# Split the data into training and testing sets
np.random.seed(123)
index = np.random.choice(range(X.shape[0]), size=int(X.shape[0] * 0.7), replace=False)
X_train, X_test = X[index], X[np.setdiff1d(range(X.shape[0]), index)]
y_train, y_test = y[index], y[np.setdiff1d(range(X.shape[0]), index)]

# Flatten the data
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)
```

```
In [40]: # Define the binary model architecture
binary_model = Sequential()
binary_model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
binary_model.add(Dropout(0.5))
binary_model.add(Dense(2, activation='softmax'))

# Compile the binary model
binary_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the binary model
binary_history = binary_model.fit(X_train, y_train, epochs=20, batch_size=16, validation_data=(X_test, y_test))

# Plot the training history
plt.plot(binary_history.history['accuracy'])
plt.plot(binary_history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

plt.plot(binary_history.history['loss'])
plt.plot(binary_history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()

# Evaluate the binary model
train_score = binary_model.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy: {:.2f}%".format(train_score[1] * 100))

test_score = binary_model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: {:.2f}%".format(test_score[1] * 100))
```

```
C:\Users\bunad\install\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 2s 49ms/step - accuracy: 0.7232 - loss: 0.4580 - val_accuracy: 0.9565 - val_loss: 0.3612
Epoch 2/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9437 - loss: 0.1906 - val_accuracy: 0.9674 - val_loss: 0.1791
Epoch 3/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - accuracy: 0.9769 - loss: 0.1030 - val_accuracy: 0.9674 - val_loss: 0.1217
Epoch 4/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.9692 - loss: 0.0788 - val_accuracy: 1.0000 - val_loss: 0.0579
Epoch 5/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9968 - loss: 0.0450 - val_accuracy: 1.0000 - val_loss: 0.0303
Epoch 6/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9848 - loss: 0.0712 - val_accuracy: 1.0000 - val_loss: 0.0303
Epoch 7/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.9849 - loss: 0.0483 - val_accuracy: 1.0000 - val_loss: 0.0240
Epoch 8/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9924 - loss: 0.0438 - val_accuracy: 1.0000 - val_loss: 0.0176
Epoch 9/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 0.9680 - loss: 0.0545 - val_accuracy: 1.0000 - val_loss: 0.0192
Epoch 10/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.9969 - loss: 0.0376 - val_accuracy: 1.0000 - val_loss: 0.0144
Epoch 11/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 1.0000 - loss: 0.0164 - val_accuracy: 1.0000 - val_loss: 0.0119
Epoch 12/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.9956 - loss: 0.0339 - val_accuracy: 1.0000 - val_loss: 0.0093
Epoch 13/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 0.9991 - loss: 0.0193 - val_accuracy: 1.0000 - val_loss: 0.0078
Epoch 14/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - accuracy: 1.0000 - loss: 0.0152 - val_accuracy: 1.0000 - val_loss: 0.0067
Epoch 15/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 1.0000 - loss: 0.0268 - val_accuracy: 1.0000 - val_loss: 0.0054
Epoch 16/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - accuracy: 1.0000 - loss: 0.0167 - val_accuracy: 1.0000 - val_loss: 0.0043
Epoch 17/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 33ms/step - accuracy: 1.0000 - loss: 0.0218 - val_accuracy: 1.0000 - val_loss: 0.0036
Epoch 18/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 42ms/step - accuracy: 1.0000 - loss: 0.0124 - val_accuracy: 1.0000 - val_loss: 0.0032
Epoch 19/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 1s 34ms/step - accuracy: 1.0000 - loss: 0.0104 - val_accuracy: 1.0000 - val_loss: 0.0028
Epoch 20/20
14/14 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - accuracy: 1.0000 - loss: 0.0118 - val_accuracy: 1.0000 - val_loss: 0.0026
```






```
Train Accuracy: 100.00%
Test Accuracy: 100.00%
```

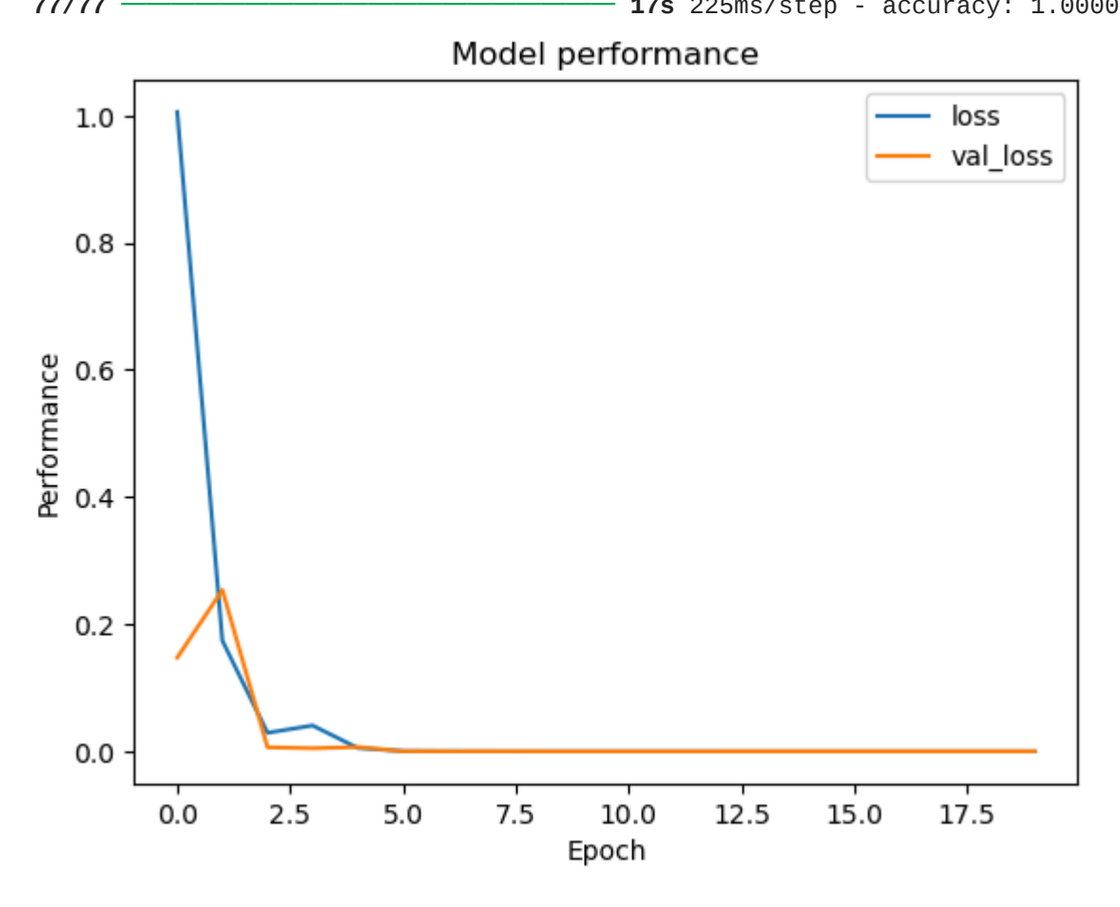
MULTI-CLASS CLASSIFICATION

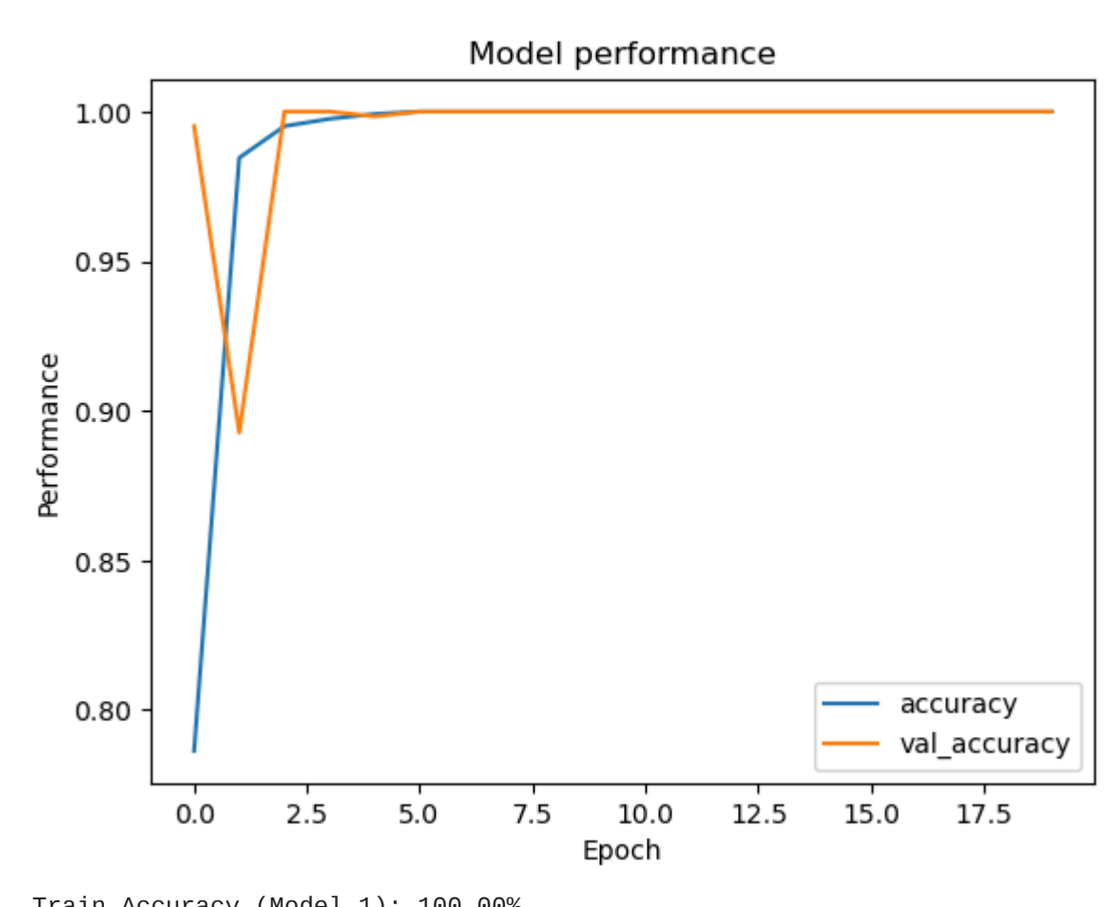
```
In [7]: # Function to pad or truncate arrays to the same shape
def pad_or_truncate(arr, target_shape):
    if arr.shape[0] > target_shape[0]:
        arr = arr[:target_shape[0], :]
    if arr.shape[1] > target_shape[1]:
        arr = arr[:, :target_shape[1]]
    pad_height = target_shape[0] - arr.shape[0]
    pad_width = target_shape[1] - arr.shape[1]
    return np.pad(arr, ((0, pad_height), (0, pad_width)), mode='constant')

# Load the spectrograms from the HDF5 file
file_path = "C:\\Users\\bunad\\OneDrive\\Desktop\\SPRING 2024\\MACHINE LEARNING 2\\spectrograms.h5"
spectrogram_data = []
labels = []

with h5py.File(file_path, 'r') as f:
    keys = list(f.keys())
    # Determine the target shape by finding the max dimensions
    max_shape = (0, 0)
    for species_key in keys:
        spectrograms = f[species_key]
        for spectrogram in spectrograms:
            max_shape = (max(max_shape[0], spectrogram.shape[0]), max(max_shape[1], spectrogram.shape[1]))

    for species_key in keys:
        spectrograms = f[species_key]
        for spectrogram in spectrograms:
            padded_spectrogram = pad_or_truncate(spectrogram, max_shape)
            spectrogram_data.append(padded_spectrogram)
            labels.append(species_key)

spectrogram_data = np.array(spectrogram_data)

# Encode the labels
encoder = LabelEncoder()
labels = encoder.fit_transform(labels)
labels = to_categorical(labels, num_classes=len(keys))

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(spectrogram_data, labels, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# Reshape the data for Conv2D layer
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

```
(2457, 343, 59) (615, 343, 59) (2457, 12) (615, 12)
```

```
In [43]: # Define the first multi-class model architecture
multi_model_1 = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2], 1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(keys), activation='softmax')
])

# Compile the first multi-class model
multi_model_1.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the first multi-class model
history_1 = multi_model_1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=32)

# Plot the accuracy and loss for the first multi-class model
plt.plot(history_1.history['loss'], label='loss')
plt.plot(history_1.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()

plt.plot(history_1.history['accuracy'], label='accuracy')
plt.plot(history_1.history['val_accuracy'], label='val_accuracy')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()

train_score_1 = multi_model_1.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy (Model 1): {:.2f}%".format(train_score_1[1] * 100))

test_score_1 = multi_model_1.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy (Model 1): {:.2f}%".format(test_score_1[1] * 100))
```

```
Epoch 1/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 19s 232ms/step - accuracy: 0.5765 - loss: 1.6604 - val_accuracy: 0.9951 - val_loss: 0.1473
Epoch 2/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 218ms/step - accuracy: 0.9857 - loss: 0.1516 - val_accuracy: 0.8927 - val_loss: 0.2537
Epoch 3/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 223ms/step - accuracy: 0.9903 - loss: 0.0442 - val_accuracy: 1.0000 - val_loss: 0.0063
Epoch 4/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 220ms/step - accuracy: 0.9972 - loss: 0.0301 - val_accuracy: 1.0000 - val_loss: 0.0048
Epoch 5/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 224ms/step - accuracy: 0.9992 - loss: 0.0056 - val_accuracy: 0.9984 - val_loss: 0.0066
Epoch 6/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 222ms/step - accuracy: 1.0000 - loss: 5.0406e-04 - val_accuracy: 1.0000 - val_loss: 1.4080e-04
Epoch 7/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 18s 228ms/step - accuracy: 1.0000 - loss: 1.3784e-04 - val_accuracy: 1.0000 - val_loss: 7.3082e-05
Epoch 8/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 226ms/step - accuracy: 1.0000 - loss: 6.8701e-05 - val_accuracy: 1.0000 - val_loss: 5.8152e-05
Epoch 9/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 220ms/step - accuracy: 1.0000 - loss: 5.0189e-05 - val_accuracy: 1.0000 - val_loss: 3.2719e-05
Epoch 10/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 218ms/step - accuracy: 1.0000 - loss: 3.0348e-05 - val_accuracy: 1.0000 - val_loss: 3.2684e-05
Epoch 11/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 220ms/step - accuracy: 1.0000 - loss: 2.8626e-05 - val_accuracy: 1.0000 - val_loss: 2.7996e-05
Epoch 12/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 217ms/step - accuracy: 1.0000 - loss: 1.9296e-05 - val_accuracy: 1.0000 - val_loss: 1.8082e-05
Epoch 13/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 220ms/step - accuracy: 1.0000 - loss: 1.4578e-05 - val_accuracy: 1.0000 - val_loss: 1.8240e-05
Epoch 14/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 217ms/step - accuracy: 1.0000 - loss: 1.2204e-05 - val_accuracy: 1.0000 - val_loss: 1.2315e-05
Epoch 15/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 215ms/step - accuracy: 1.0000 - loss: 9.5016e-06 - val_accuracy: 1.0000 - val_loss: 1.1304e-05
Epoch 16/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 218ms/step - accuracy: 1.0000 - loss: 1.1996e-05 - val_accuracy: 1.0000 - val_loss: 1.2783e-05
Epoch 17/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 226ms/step - accuracy: 1.0000 - loss: 7.5228e-06 - val_accuracy: 1.0000 - val_loss: 9.4450e-06
Epoch 18/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 218ms/step - accuracy: 1.0000 - loss: 7.5736e-06 - val_accuracy: 1.0000 - val_loss: 1.0474e-05
Epoch 19/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 220ms/step - accuracy: 1.0000 - loss: 8.5181e-06 - val_accuracy: 1.0000 - val_loss: 7.9232e-06
Epoch 20/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 225ms/step - accuracy: 1.0000 - loss: 5.1824e-06 - val_accuracy: 1.0000 - val_loss: 7.8897e-06
```

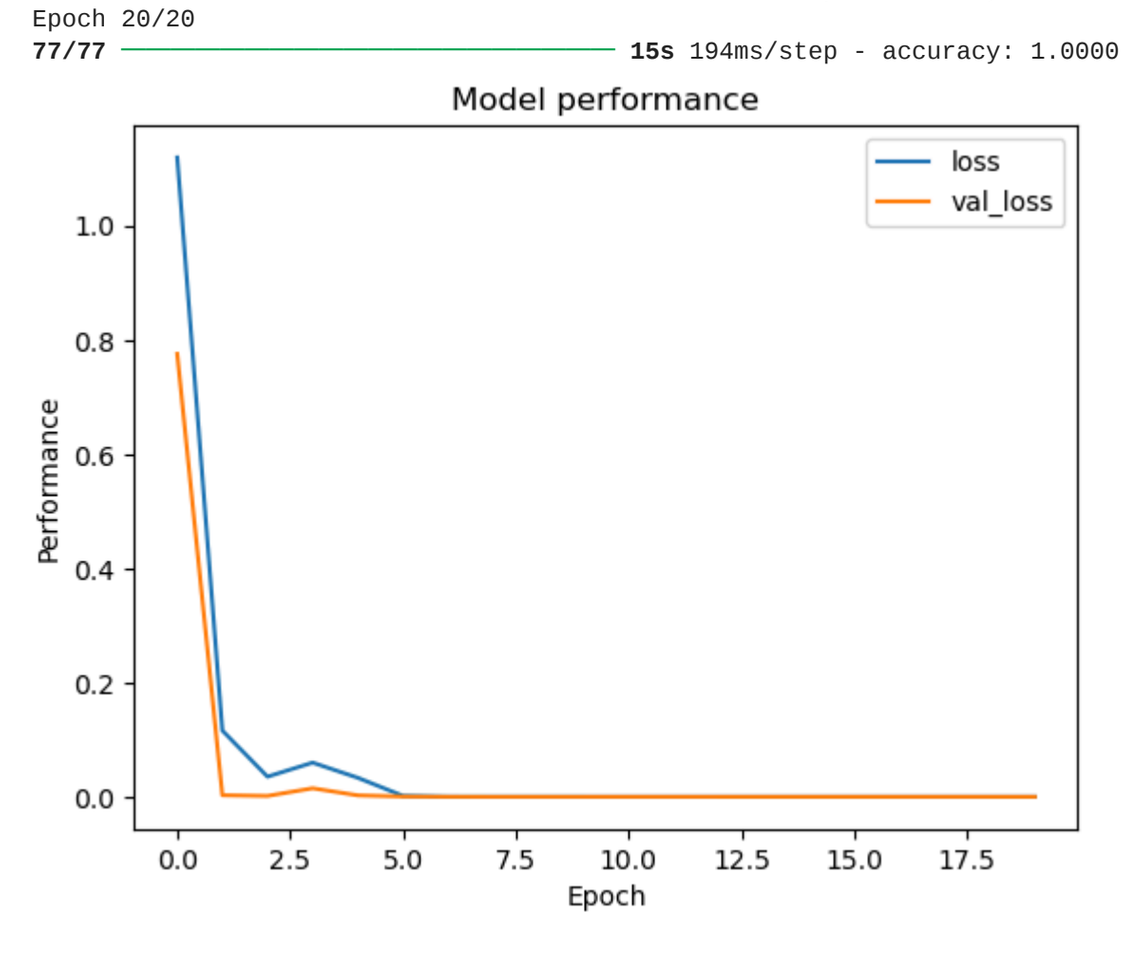


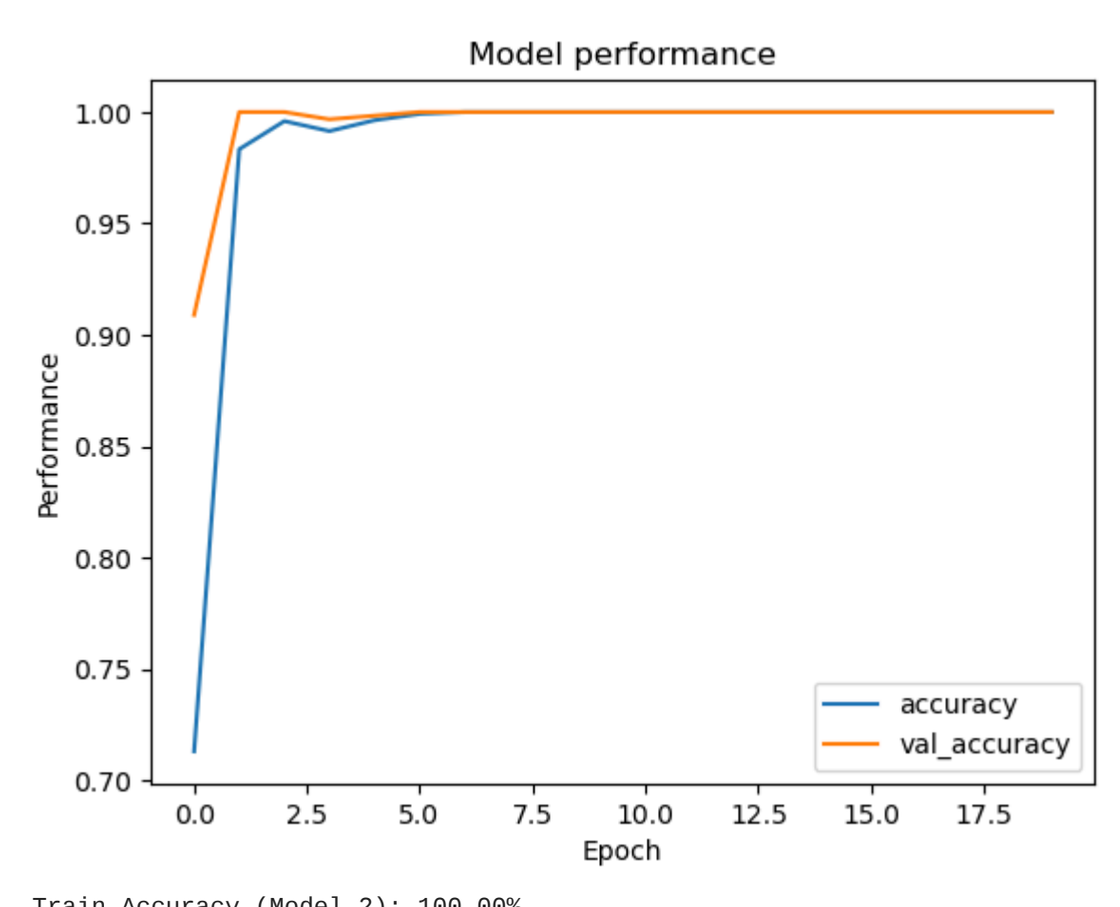



```
Train Accuracy (Model 1): 100.00%
Test Accuracy (Model 1): 100.00%
```

```
In [44]: # Define the second multi-class model architecture
multi_model_2 = Sequential([
    Input(shape=(X_train.shape[1], X_train.shape[2], 1)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(len(keys), activation='softmax')
])

# Compile the second multi-class model
multi_model_2.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the second multi-class model
history_2 = multi_model_2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=20, batch_size=32)

# Plot the accuracy and loss for the second multi-class model
plt.plot(history_2.history['loss'], label='loss')
plt.plot(history_2.history['val_loss'], label='val_loss')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()

plt.plot(history_2.history['accuracy'], label='accuracy')
plt.plot(history_2.history['val_accuracy'], label='val_accuracy')
plt.title('Model performance')
plt.xlabel('Epoch')
plt.ylabel('Performance')
plt.legend()
plt.show()

train_score_2 = multi_model_2.evaluate(X_train, y_train, verbose=0)
print("Train Accuracy (Model 2): {:.2f}%".format(train_score_2[1] * 100))

test_score_2 = multi_model_2.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy (Model 2): {:.2f}%".format(test_score_2[1] * 100))
```

```
Epoch 1/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 21s 234ms/step - accuracy: 0.4649 - loss: 1.7836 - val_accuracy: 0.9089 - val_loss: 0.7756
Epoch 2/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 18s 232ms/step - accuracy: 0.9630 - loss: 0.2714 - val_accuracy: 1.0000 - val_loss: 0.0027
Epoch 3/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 18s 231ms/step - accuracy: 0.9967 - loss: 0.0312 - val_accuracy: 1.0000 - val_loss: 0.0015
Epoch 4/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 17s 224ms/step - accuracy: 0.9879 - loss: 0.0855 - val_accuracy: 0.9967 - val_loss: 0.0149
Epoch 5/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 16s 213ms/step - accuracy: 0.9963 - loss: 0.0287 - val_accuracy: 0.9984 - val_loss: 0.0025
Epoch 6/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 16s 204ms/step - accuracy: 0.9999 - loss: 5.4211e-04 - val_accuracy: 1.0000 - val_loss: 4.4130e-04
Epoch 7/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 16s 205ms/step - accuracy: 1.0000 - loss: 1.2105e-04 - val_accuracy: 1.0000 - val_loss: 5.8566e-05
Epoch 8/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 16s 208ms/step - accuracy: 1.0000 - loss: 3.9839e-05 - val_accuracy: 1.0000 - val_loss: 1.7593e-05
Epoch 9/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 196ms/step - accuracy: 1.0000 - loss: 2.1544e-05 - val_accuracy: 1.0000 - val_loss: 1.2813e-05
Epoch 10/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 192ms/step - accuracy: 1.0000 - loss: 1.9138e-05 - val_accuracy: 1.0000 - val_loss: 9.9807e-06
Epoch 11/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 192ms/step - accuracy: 1.0000 - loss: 1.0366e-05 - val_accuracy: 1.0000 - val_loss: 1.1981e-05
Epoch 12/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 191ms/step - accuracy: 1.0000 - loss: 7.2470e-06 - val_accuracy: 1.0000 - val_loss: 1.6111e-05
Epoch 13/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 201ms/step - accuracy: 1.0000 - loss: 1.1004e-05 - val_accuracy: 1.0000 - val_loss: 6.0038e-06
Epoch 14/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 16s 208ms/step - accuracy: 1.0000 - loss: 4.8535e-06 - val_accuracy: 1.0000 - val_loss: 5.7179e-06
Epoch 15/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 195ms/step - accuracy: 1.0000 - loss: 4.4922e-06 - val_accuracy: 1.0000 - val_loss: 4.8922e-06
Epoch 16/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 16s 205ms/step - accuracy: 1.0000 - loss: 4.1138e-06 - val_accuracy: 1.0000 - val_loss: 4.5938e-06
Epoch 17/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 194ms/step - accuracy: 1.0000 - loss: 3.1092e-06 - val_accuracy: 1.0000 - val_loss: 4.3358e-06
Epoch 18/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 199ms/step - accuracy: 1.0000 - loss: 3.0997e-06 - val_accuracy: 1.0000 - val_loss: 7.3862e-06
Epoch 19/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 201ms/step - accuracy: 1.0000 - loss: 4.0160e-06 - val_accuracy: 1.0000 - val_loss: 4.0326e-06
Epoch 20/20
77/77 ━━━━━━━━━━━━━━━━━━━━ 15s 194ms/step - accuracy: 1.0000 - loss: 2.4600e-06 - val_accuracy: 1.0000 - val_loss: 3.3440e-06
```






```
Train Accuracy (Model 2): 100.00%
Test Accuracy (Model 2): 100.00%
```

External test

```
In [53]: import os
import numpy as np
import librosa
from collections import Counter

# Function to pad or truncate arrays to the same shape
def pad_or_truncate(arr, target_shape):
    if arr.shape[0] > target_shape[0]:
        arr = arr[:target_shape[0], :]
    if arr.shape[1] > target_shape[1]:
        arr = arr[:, :target_shape[1]]
    pad_height = target_shape[0] - arr.shape[0]
    pad_width = target_shape[1] - arr.shape[1]
    return np.pad(arr, ((0, pad_height), (0, pad_width)), mode='constant')

# Function to process new audio clips and make predictions
def process_and_predict(sound_clip_paths, model, encoder):
    n_fft = 2048
    hop_length = 512
    window_size = 2  # seconds

    target_shape = multi_model_2.input_shape[1:4]  # Using the provided dimensions
    results = []

    for path in sound_clip_paths:
        spectrograms_list = []
        audio, sr = librosa.load(path, sr=22050)
        loud_parts = librosa.effects.split(audio, top_db=30)
        for interval in loud_parts:
            start = interval[0]
            end = interval[1]
            duration = end - start
            if duration >= 0.5:
                for i in range(int(start), int(end) - window_size * sr + 1, hop_length):
                    window = audio[i:i + window_size * sr]
                    spectrogram = librosa.feature.melspectrogram(y=window, sr=sr, n_fft=n_fft, hop_length=hop_length)
                    spectrogram = librosa.power_to_db(spectrogram, ref=np.max)
                    spectrogram = pad_or_truncate(spectrogram, target_shape)
                    spectrogram = spectrogram[..., np.newaxis]  # Add channel dimension
                    spectrograms_list.append(spectrogram)

        spectrograms_array = np.array(spectrograms_list)

        # Predict using the model
        predictions = model.predict(spectrograms_array)
        predicted_classes = np.argmax(predictions, axis=1)
        predicted_species = encoder.inverse_transform(predicted_classes)

        # Count occurrences of each species in this audio file
        species_count = Counter(predicted_species)
        results.append((path, species_count))

    return results

# Define paths to new sound clips
test_clips_folder = r"C:\Users\bunad\OneDrive\Desktop\SPRING 2024\MACHINE LEARNING 2\test_birds\test_birds"
sound_clip_paths = [os.path.join(test_clips_folder, file) for file in os.listdir(test_clips_folder) if file.endswith('.mp3')]

# Assuming encoder is defined and trained
# Predict using the second multi-class model
results = process_and_predict(sound_clip_paths, multi_model_2, encoder)

# Print the predictions and counts for each test clip
for path, species_count in results:
    print(f"Test Clip: {path}")
    print("Predicted Bird Species Count:")
    for species, count in species_count.items():
        print(f"{species}: {count} times")
    print()
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 1s 47ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 42ms/step
19/19 ━━━━━━━━━━━━━━━━━━━━ 1s 48ms/step
Test Clip: C:\Users\bunad\OneDrive\Desktop\SPRING 2024\MACHINE LEARNING 2\test_birds\test_birds\test1.mp3
Predicted Bird Species Count:
wesmea: 916 times

Test Clip: C:\Users\bunad\OneDrive\Desktop\SPRING 2024\MACHINE LEARNING 2\test_birds\test_birds\test2.mp3
Predicted Bird Species Count:
wesmea: 138 times
```

```
Test Clip: C:\Users\bunad\OneDrive\Desktop\SPRING 2024\MACHINE LEARNING 2\test_birds\test_birds\test3.mp3
Predicted Bird Species Count:
wesmea: 598 times
```