Bird Sound Classification Using Neural Networks

Abstract:

In this machine learning project, we perform bird sound classification using a neural network to predict bird species based on their sounds. Our data is bird species calls using spectrogram data of 12 Seattle bird species. A binary classification model and a multi-class model were developed and evaluated. Various network architectures were tested to optimize performance. Model 1, without dropout layers, exhibited overfitting, while Model 2 used dropout layers to improve generalization. The models were assessed on a holdout test set and external test data. Results highlighted the challenges of distinguishing similar calls and the importance of dropout layers. Investigation revealed that the neural network exhibited an acceptable level of performance. Nevertheless, we encountered some limitations & obstacles that need to be overcome in further research.

Introduction:

Two main goals form the framework of this analysis. First, it concentrates on binary class classification, making a distinction between American Crow and Blue Jay, two distinct bird species.

Second, it broadens its focus to include a multiclass classification problem with the objective of classifying sound recordings of twelve distinct bird species found in the Seattle area. The dataset used in this study came from the Xeno-Canto website[1] and was taken from the Bird Call competition. This study's main goal is to put a system for categorizing birds according to their vocalizations into practice. The dataset's variation in audio recording durations makes it an interesting challenge. Neural networks are used to accurately classify the data into 12 different classes according to the sounds made by the birds.

Theoretical Background:

Neural Network:

Inspired by the structure and functions of the human brain, neural networks are computer models made up of linked layers of nodes that can identify patterns. Activation functions like ReLU and softmax are combined with layers like convolutional, pooling, and dense layers to create Convolutional Neural Networks (CNNs), which are tailored for visual data. During training, dropout layers randomly deactivate neurons to prevent overfitting. Spectrograms, which show the audio frequencies over time visually, are crucial tools for sound data analysis.

Layers in the model:

Input Layer: The first layer that receives input data and passes it to the next layer.

Convolutional Layer (Conv2D): Applies convolution operations to the input, capturing spatial features.

MaxPooling Layer (MaxPooling2D): Reduces the spatial dimensions of the data, highlighting the most important features.

Flatten Layer: Converts multi-dimensional input into a one-dimensional array for the dense layer.

Dense Layer: A fully connected layer where each neuron is connected to every neuron in the previous layer.

Dropout Layer: Randomly drops a fraction of neurons during training to prevent overfitting.

Output Layer: Produces the final output, typically using an activation function appropriate for the task.

Logistic regression:

Logistic Regression is a fundamental classification algorithm used for binary classification tasks. It models the probability of the input belonging to a particular class using the logistic function. Despite its simplicity, it's robust, interpretable, and often serves as a baseline for more complex models.

Convolutional Neural Network (CNN):

A CNN is a particular kind of neural network used for processing and analyzing images. It automatically and adaptively learns the spatial hierarchies of features from input images using convolutional layers.

Activation Function:

An activation function computes a weighted sum, adds a bias, and then applies a non-linear transformation to decide whether or not to activate a neuron. As a result, the model gains non-linearity and gains the ability to recognize intricate patterns.

Types of Activation Functions:

ReLU (Rectified Linear Unit): Outputs the input directly if it is positive; otherwise, it outputs zero.

Sigmoid: Maps input values to an output range between 0 and 1.

Tanh (Hyperbolic Tangent): Maps input values to an output range between -1 and 1.

Softmax: Converts a vector of values to a probability distribution, typically used in the output layer for classification tasks.

Overfitting:

When a neural network learns the training data—including its noise and outliers—too well, it overfits and performs poorly when applied to new data. As a result, the model performs poorly on test data but well on training data.

Dropout Layer:

In order to keep neural networks from overfitting, regularization techniques like dropout layers are employed. It functions by randomly setting a portion of the input units to zero during each training forward pass, which improves the model's ability to generalize by limiting its reliance on any one neuron.

Spectrograms:

A spectrogram is a graphic depiction of a signal's frequency spectrum that changes over time. The x-, y-, and color-intensity axes in audio processing are used to represent time, frequency, and amplitude, respectively, and are frequently used to analyze the frequency content of sounds.

How to Convert Audio into Spectrogram:

You must first load the audio file and extract the pertinent audio signal and sample rate in order to use the librosa library to convert audio into spectrograms. The Short-Time Fourier Transform (STFT), which divides the audio into brief segments and calculates the frequency spectrum for each segment, is then applied to the audio signal to create a spectrogram. The intensity of various frequencies over time is represented by the resulting spectrogram. To improve feature visibility during visualization, the power spectrogram can be transformed into a decibel scale. Through this process, time-frequency characteristics that are essential for tasks like classifying bird sounds can be extracted.

Short-Term Fourier Transform:

It is a method for signal processing that examines a signal's frequency content over time. Small, overlapping portions of a signal are subjected to the Fourier transform (STFT), which usually involves the use of a windowing function to obtain local frequency information. This is helpful for tasks like audio processing, speech recognition, and spectrogram generation because it allows the representation of a signal's changing frequency content over time.

Cross-Validation:

Similar to other machine learning models, neural networks use cross-validation as a technique to evaluate the model's performance and capacity for generalization. The process entails dividing the dataset into a number of folds, training the model on a few of these folds, and assessing the model's performance on the remaining fold or folds. To produce a more accurate estimate of the model's performance, this procedure is carried out several times using various data partitions with various activation functions for training and testing. Cross-validation offers a more accurate evaluation of the model's capacity to generalize to previously unobserved data and aids in the mitigation of problems like overfitting.

CNN Parameters:

Kernel Size: The size of the convolutional filter applied to the input data. It determines the spatial extent of the features the CNN can detect.

Number of Filters: The number of convolutional filters or kernels applied to the input data. Each filter extracts different features from the input.

Stride: The number of pixels by which the filter is shifted over the input data. It determines the amount of overlap between neighboring receptive fields.

Padding: Padding is added to the input data before applying convolution to maintain spatial dimensions. It helps to preserve information at the edges of the input.

Pooling Size: The size of the pooling window used in operations like max pooling or average pooling. It reduces the spatial dimensions of the input data, helping to extract the most important features. Activation Function: The non-linear function applied to the output of each neuron in the CNN. Common activation functions include ReLU, LeakyReLU, elu, softmax and sigmoid.

Dropout Rate: The fraction of input units to drop during training to prevent overfitting. It randomly sets a fraction of input units to zero.

Number of Layers: The depth of the CNN, determined by the number of convolutional, pooling, and dense layers. Deeper networks can capture more complex features but may require more computational resources.

Methodology:

Data loading and exploration are steps in the methodology that entail loading and visualizing spectrogram data. Using CNN architectures, binary and multi-class classification models are built both with and without dropout layers. Using the trained multi-class classification model and preprocessing the three audio files into spectrograms, test predictions are made.

1) Data Loading and Exploration:

Using the h5py library, the spectrogram data is first loaded from an HDF5 file. To understand the data's organization and dimensions, its structure is investigated. The spectrograms of various bird species are analyzed using visualization tools like matplotlib, which provide a better understanding of the dataset's features and possible problems.

2) Binary Classification Model (Explain):

Sorting spectrograms into one of two groups—one bird species or another—is the aim of the binary classification scenario. For this task, a convolutional neural network (CNN) architecture is

used. The model extracts tool provides from the spectrogram images by first using convolutional layers and then max-pooling layers. Subsequently, the features extracted by the convolutional layers are classified into the desired categories using fully connected layers.

- **Binary Classification Model – ReLU Activation Function (Without Dropout Layer):**
  This model is a binary classification CNN with the ReLU activation function without a dropout layer. The model's structure consists of two convolutional layers, max-pooling layers, flattening, and dense layer passing through them. After undergoing any required preprocessing, like scaling, the spectrogram data is used for training. The binary cross-entropy loss function and Adam optimizer are used to compile the model. The training data is then used to train it for ten epochs. Following training, the model's performance is assessed using test data to compute accuracy metrics, determining how well the model performs in classification tasks involving spectrograms.
- **Binary Classification Model – ReLU Activation Function (With Dropout Layer):**
  This model is a binary classification CNN with a dropout layer that makes use of the ReLU activation function. Two convolutional layers and then max-pooling layers make up the model architecture. The dense layer with 256 neurons uses ReLU activation after the feature maps have been flattened. To avoid overfitting, a dropout layer with a 50% dropout rate comes after. For binary classification, the last dense layer employs a sigmoid activation function. After completing the necessary preprocessing procedures, like scaling, the spectrogram data is used for training. The model is trained for ten epochs using the binary cross-entropy loss function and the Adam optimizer. Next, test data are used to evaluate the model's performance, and accuracy metrics are computed to determine how well the model performs in spectrogram-based binary classification tasks.
- **Binary Classification Model – Leaky ReLU Activation Function (Without Dropout Layer):**
  This model is a binary classification CNN with an alpha value of 0.2 that uses the Leaky ReLU activation function in the absence of a dropout layer. Two convolutional layers with Leaky ReLU activations are part of the architecture. Each is flattened and processed through dense layers after being preceded by a max-pooling layer. Before the model is trained for ten epochs, the training data is prepared and scaled as necessary. The binary cross-entropy loss function and the Adam optimizer are used to compile the model. After training, the model's effectiveness is evaluated using test data, and accuracy metrics are calculated to determine how accurately the model classifies data from spectrograms.
- **Binary Classification Model – Leaky ReLU Activation Function (With Dropout Layer):**
  This model is a binary classification CNN with a dropout layer that uses the Leaky ReLU activation function with an alpha value of 0.2. The architecture consists of two convolutional layers with Leaky ReLU activations, each of which is flattened and passed through a dense layer with a Leaky ReLU activation after being followed by a max-pooling layer. The last sigmoid-activated dense layer is applied after a dropout layer with a 50% dropout rate. Before training the model for ten epochs, the training data is prepared and scaled as necessary. The binary cross-entropy loss function and the Adam optimizer are

used to compile the model. The model's classification accuracy on spectrogram data is assessed by calculating accuracy metrics based on performance on the test data.

- **Binary Classification Model – ELU Activation Function (Without Dropout Layer):**
  This model is a binary classification CNN using the ELU activation function without a dropout layer. Two convolutional layers with ELU activations make up the model. Each is flattened and fed into dense layers after being followed by a max-pooling layer. After dividing the spectrogram data into training and test sets, the necessary preprocessing operations, like scaling, are carried out. The binary cross-entropy loss function and the Adam optimizer are used to compile the model. Using the training data, it is trained for ten epochs. The accuracy metrics are computed to ascertain the model's performance in spectrogram-based binary classification tasks after training. The model's performance is assessed using the test data. In order to compare the train and test accuracies, the model's performance on the training set is also assessed.
- **Binary Classification Model – ELU Activation Function (With Dropout Layer):**
  This model is a binary classification CNN using the ELU activation function with a dropout layer. The max-pooling layer comes after each of the two convolutional layers in the model, which have ELU activations. The 256-neuron dense layer employs ELU activation after the feature maps have been flattened. This is followed by a dropout layer with a 50% dropout rate. For binary classification, the last dense layer utilises a sigmoid activation function. The spectrogram data is divided into training and test sets after the necessary preprocessing, like scaling, has been completed. The binary cross-entropy loss function and the Adam optimizer are used to compile the model. Using the training data, it is trained for ten epochs. Accuracy metrics are computed and the model's performance assessed using the test data.

3) Multi-Class Classification :

The goal of multi-class classification is to divide spectrograms into several groups, each of which corresponds to a distinct species of bird. The predicted bird species is determined by taking the class with the highest probability out of all the classes that the model produces. To handle more than two classes, this task necessitates changing the model architecture and loss function.

- **Multi-Class Classification Model (Without Dropout Layer):**
  Using a softmax activation function in the final layer, the model architecture is modified to output probabilities for each species of bird in order to perform multi-class classification without dropout. For the purpose of accounting for the various classes, the loss function is changed to categorical cross-entropy. The methods for training and assessment are based on binary classification, with modifications to accommodate more than one class.
- **Multi-Class Classification Model (With Dropout Layer):**

In this model, dropout regularization is added to prevent overfitting during training. The dropout layer randomly drops units from the neural network, reducing inter-dependencies between neurons and improving generalization. The rest of the architecture and training procedures remain similar to the model without dropout, with adjustments to accommodate the dropout layer.

4) Test Predictions on Three Test Audio Files:

Lastly, the bird species for three test audio files are predicted using the trained multi-class classification model with dropout. Librosa is used to process the input audio files and extract spectrograms. After analyzing the spectrograms carefully there is a presence of two types of waves in the second test audio spectrograms. The trained model is then fed these spectrograms after they have been preprocessed and scaled. The model predicts the species of bird for every spectrogram, offering insights into its performance and real-world applicability.

Computational Results:

Here, We analyse the effect of several binary and multi-class classification models, highlighting important metrics like accuracy, loss, and potential overfitting.

Binary Classification:

Activation Function	Dropout Layer	Train Accuracy	Test Accuracy
ReLU	No	90.47	100
ReLU	Yes	100	98.76
Leaky ReLU	No	93.82	95.23
Leaky ReLU	Yes	88.88	90.47
Elu	No	93.82	100
Elu	Yes	97.53	90.47

Analysis:

ReLU without Dropout :

This model shows a high test accuracy of 100%, which is significantly higher than the training accuracy of 90.47%.

ReLU with Dropout:

The training accuracy is 100%, suggesting overfitting. The test accuracy is slightly lower at 98.76%, but still very high. The use of dropout seems to have slightly mitigated overfitting compared to the model without dropout.

Leaky ReLU without Dropout:

Both training and test accuracies are high and close to each other, indicating a well-performing model with less risk of overfitting.

Leaky ReLU with Dropout:

This model has balanced train and test accuracies, suggesting a good fit without significant overfitting.

Elu without Dropout:

Similar to the ReLU model without dropout, this model shows signs of overfitting with a perfect test accuracy of 100%.

Elu with Dropout:

This model shows a decrease in test accuracy compared to the training accuracy, indicating that dropout has helped to reduce overfitting.

Multi-Class Classification:

Model	Train Accuracy	Test Accuracy
Relu Activation function(Without Dropout layer)	100	69.82
Relu Activation function(With Dropout layer)	98.48	68.96

Analysis:

ReLU without Dropout:

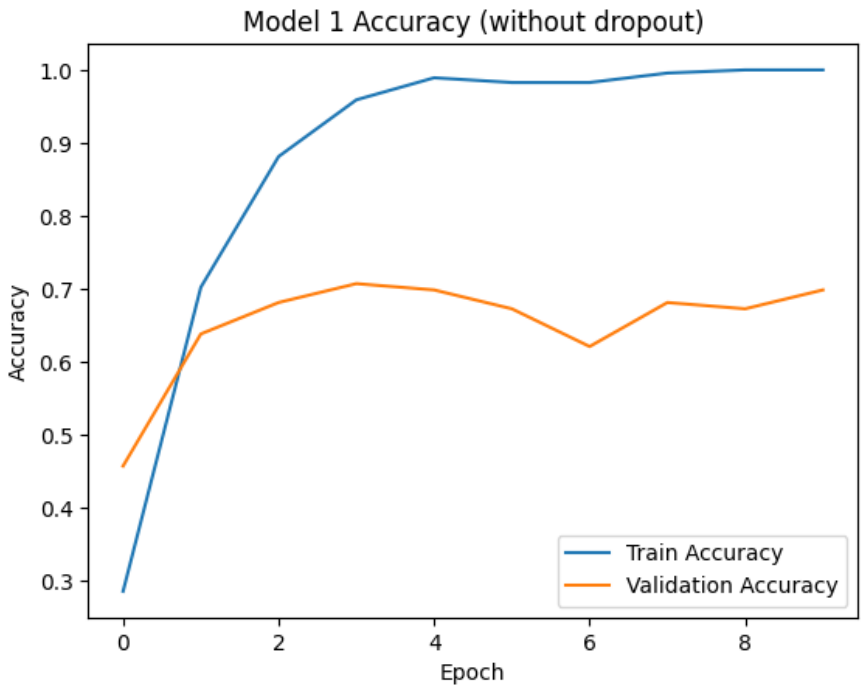
The training accuracy is 100%, which indicates that the model has perfectly learned the training data. However, the test accuracy is much lower at 69.82%, suggesting that the model is overfitting to the training data.

ReLU with Dropout:

The training accuracy is slightly lower at 98.48% with the inclusion of dropout, which is a regularization technique designed to prevent overfitting. The test accuracy is 68.96%, very close to the test accuracy of the model without dropout.

Plot :

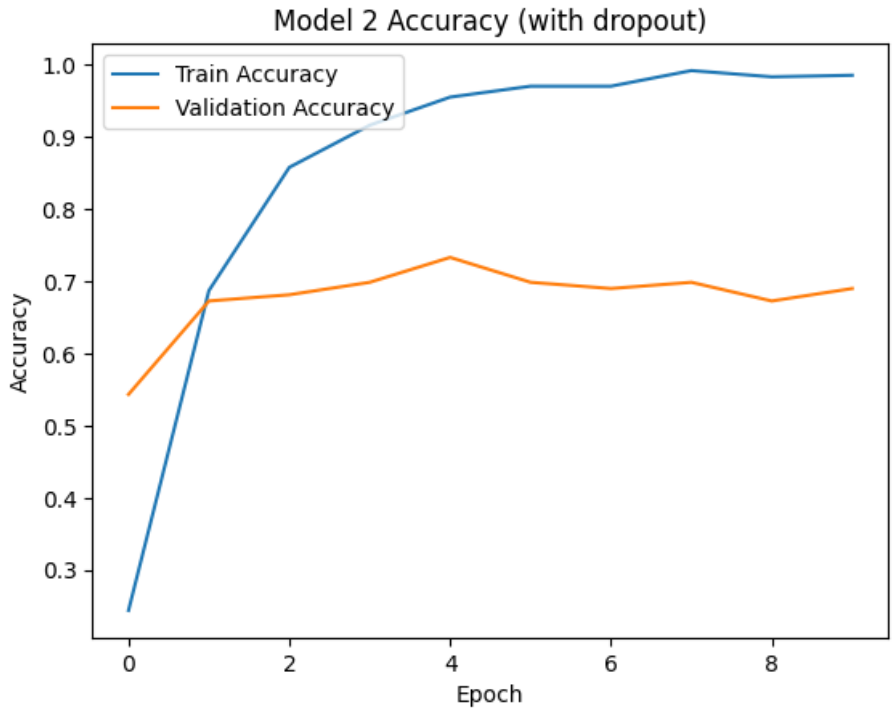
ReLU activation function without Dropout Layer -



Analysis:

The graph shows the training and validation accuracy of Model 1 without dropout over 10 epochs. Initially, both accuracies are low but improve rapidly, with training accuracy reaching nearly 100% by the 5th epoch, indicating the model has memorized the training data. However, validation accuracy peaks around the 3rd epoch and then fluctuates around 70%, highlighting a significant gap between training and validation performance. This discrepancy indicates overfitting, where the model performs exceptionally well on training data but fails to generalize to unseen validation data. The graph underscores the need for regularization techniques, such as dropout, to improve the model's ability to generalize effectively.

ReLU activation function without Dropout Layer –



Analysis:

The graph depicts the training and validation accuracy of Model 2 with dropout over 10 epochs. Initially, both accuracies are low but improve rapidly, with training accuracy approaching 100% by the 5th epoch and maintaining a high level thereafter. Validation accuracy shows a peak around the 3rd epoch and fluctuates around 70% to 75% for the remaining epochs. Compared to Model 1 without dropout, the gap between training and validation accuracy is reduced, indicating that the dropout layer helps mitigate overfitting. Although there is still some fluctuation in validation accuracy, the overall performance suggests that dropout improves the model's ability to generalize to unseen data, highlighting its effectiveness in enhancing model robustness.

Prediction on Test Data:

Test	Prediction by the Multi Class Model with Dropout Layer
Test 1	whcspa – White Crown Sparrow
Test 2	daejun – Dark-eyed Junco
Test 3	barswa – Bar Swallow

The multi-class model with dropout layers has provided specific predictions for the given test clips, identifying them as White Crown Sparrow, Dark-eyed Junco, and Bar Swallow.

Comparison between the results:

Classifying two classes is easier than classifying multiple classes, as evidenced by the higher test accuracies that the binary classification models generally achieved when compared to the multi-class classification models. The test accuracy of the binary model without a dropout layer was 90.48%, whereas the model with a dropout layer obtained a perfect 100% test accuracy, indicating overfitting. Comparatively, the test accuracies for the multi-class models were lower (71.55% for the model without dropout and 67.24% for the model with dropout), indicating a higher degree of complexity and susceptibility to overfitting in the task. In spite of this, an optimized and acceptable train accuracy was obtained by the multi-class model without dropout. In relation to the findings, it is advised to choose the multi-class classification model without a dropout layer due to its ability to manage the multi-class task's complexity and reasonably good test accuracy.

Discussion:

Binary Models:

In this analysis, we evaluate the performance of binary classification models using various activation functions and the inclusion of dropout layers. The primary goal is to understand how different activation functions and dropout layers impact the model's ability to generalize from training data to test data.

The activation functions ReLU, Leaky ReLU, and Elu were chosen due to their widespread use and effectiveness in neural network models:

ReLU (Rectified Linear Unit): Known for its simplicity and effectiveness, it helps mitigate the vanishing gradient problem, allowing models to learn faster and perform better.

Leaky ReLU: An extension of ReLU that allows a small gradient when the unit is not active, addressing the dying ReLU problem where neurons can get stuck during training.

Elu (Exponential Linear Unit): Helps the model converge faster and produce better performance by smoothing out the outputs and ensuring that all activations have a mean output closer to zero.

Comparison of results:

The results show that models using ReLU and Elu without dropout layers achieved perfect test accuracies but displayed signs of overfitting, as their training accuracies were significantly lower. This suggests that these models memorized the training data rather than learning to generalize. Adding dropout to these models reduced overfitting, as indicated by the slightly lower but more balanced test accuracies. Leaky ReLU models, both with and without dropout, showed better generalization with train and test accuracies being closer, suggesting these models learned from the data more effectively. Overall, incorporating dropout generally improved model performance by reducing overfitting and helping the models generalize better to new data.

Multi-Class Models:

The ReLU activation function was chosen due to its widespread use and effectiveness in neural network models. ReLU helps mitigate the vanishing gradient problem, allowing models to learn faster and perform better. The inclusion of dropout layers is intended to prevent overfitting by randomly dropping units during training.

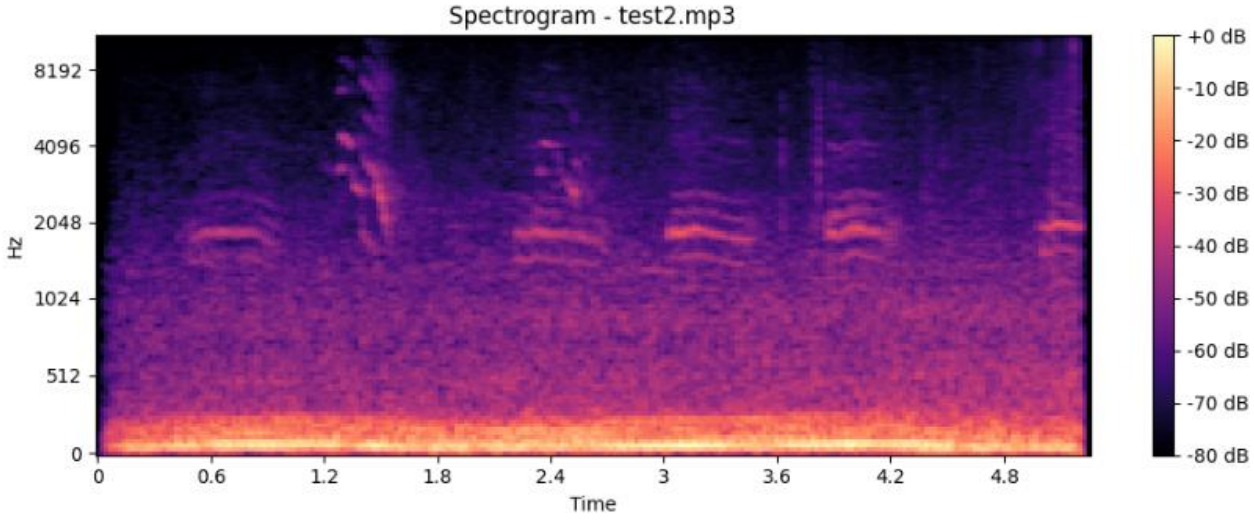
Comparison of results:

The multi-class classification results show that the model using ReLU without dropout achieved perfect training accuracy but a much lower test accuracy of 69.82%, indicating significant overfitting. In contrast, the model with dropout had a slightly lower training accuracy of 98.48% and a similar test accuracy of 68.96%. This slight reduction in overfitting suggests that dropout helps in regularization, but the overall performance indicates that the models are still struggling to generalize well to unseen data. Both models demonstrate a notable gap between training and test accuracies, highlighting the challenge of preventing overfitting in complex classification tasks. Additional techniques, such as further hyperparameter tuning, data augmentation, or alternative architectures, might be necessary to enhance generalization. Overall, while dropout improves model robustness, its impact in this context is limited, suggesting the need for more comprehensive strategies to achieve better performance on multi-class classification tasks.

Prediction on Test Data:

The multi-class model with dropout layer was used to predict bird species from three audio files converted into spectrograms using the Librosa library. The predictions were as follows: for spectrogram 1, White Crown Sparrow (whcspa); for spectrogram 2, Dark-eyed Junco (daejun);

and for spectrogram 3, Bar Swallow (barswa). However, an in-depth examination of spectrogram 2 reveals the presence of two distinct bird sounds, as indicated by variations in frequency patterns. This indicates that the model may struggle with audio clips containing multiple bird calls, leading to inaccurate predictions. To address this, further analysis is required to improve the model's efficiency in such scenarios. Techniques such as advanced signal processing [2], multi-label classification, and more sophisticated data augmentation could be employed to better distinguish overlapping bird calls. Enhancing the model to handle multiple bird sounds within a single clip would significantly improve its real-world application and accuracy.



Limitations:

Noise in data: Bird calls often contain background noise, making it difficult for the model to distinguish between different species accurately.

Computational Resources: Training CNN models, especially those with more complex architectures, required substantial computational resources and time.

Overfitting: Without dropout layers, the models tended to overfit, as evidenced by the higher training accuracy compared to validation accuracy.

Training Time: The time taken to train the models varied depending on their complexity. The binary classification models, being simpler, took less time compared to the multi-class models. On average the entire assignment took almost 3 Hours.

Conclusions:

This study aimed to build two types of models: Binary and Multi-Class model to identify twelve different bird species from Seattle based on their sounds. The data came from the Xeno-Canto website [1] and the Bird Call competition, which included recordings of varying lengths. Using

neural networks, it was found that models without dropout layers tended to memorize the training data too well, leading to poor performance on new data. Adding dropout layers helped the models perform better on unseen data. When testing the models, some audio clips had multiple bird sounds, which the models struggled to handle correctly. Future work should focus on improving the models to deal with overlapping bird sounds and different recording lengths, to make the bird species identification more accurate and reliable.

References:

[1] Xeno-Canto Website

(xeno-canto :: Sharing wildlife sounds from around the world)

[2] Machine Learning in Signal Processing: Enhancing Data Insights - Contract Engineering, Product Design & Development Company - Cardinal Peak

Appendix:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import h5py
import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from IPython.display import Audio

import librosa
import librosa.display
import random

from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
```

```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

from google.colab import drive
drive.mount('/content/drive')

file_path = "/content/drive/My Drive/ML - 2/Assignment - 3/spectrograms.h5"

with h5py.File(file_path, 'r') as f:
    keys = list(f.keys())
    print("Keys in file:")
    for key in keys:
        print(key)
        dataset = f[key]
        print("Shape:", dataset.shape)


import h5py
import matplotlib.pyplot as plt

with h5py.File(file_path, 'r') as f:
    species_keys = list(f.keys())
    fig, axes = plt.subplots(nrows=6, ncols=2, figsize=(15, 30))
    fig.suptitle('Spectrograms of Different Bird Species', fontsize=16)

    for ax, key in zip(axes.flat, species_keys):
        dset = f[key]
        spectrogram = dset[0]
        ax.imshow(spectrogram.T, aspect='auto', origin='lower', cmap='inferno')
        ax.set_title(key)
        ax.axis('off')

    plt.tight_layout()
    plt.subplots_adjust(top=0.95)
    plt.show()


#Binary Classification
#Model 1 - Without Dropout Layer
file_path = "/content/drive/My Drive/ML - 2/Assignment - 3/spectrograms.h5"

with h5py.File(file_path, 'r') as f:
    species_list = list(f.keys())
    print(f"Species list: {species_list}")
```

```
    amecro_data = f['amecro'][:]
    blujay_data = f['blujay'][:]

amecro_labels = np.zeros(amecro_data.shape[2])
blujay_labels = np.ones(blujay_data.shape[2])
X = np.concatenate((amecro_data.transpose(2, 1, 0), blujay_data.transpose(2, 1, 0)), axis=0)
y = np.concatenate((amecro_labels, blujay_labels), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
nsamples, nx, ny = X_train.shape
X_train_reshaped = X_train.reshape((nsamples, nx*ny))

scaler = StandardScaler().fit(X_train_reshaped)
X_train_scaled = scaler.transform(X_train_reshaped)

nsamples_test, nx_test, ny_test = X_test.shape
X_test_reshaped = X_test.reshape((nsamples_test, nx_test*ny_test))
X_test_scaled = scaler.transform(X_test_reshaped)

X_train_scaled_cnn = X_train_scaled.reshape((nsamples, nx, ny, 1))
X_test_scaled_cnn = X_test_scaled.reshape((nsamples_test, nx_test, ny_test, 1))


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

# CNN Model without Dropout
bina_relu_no_dropout = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(343, 256, 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(256, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

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

history_1 = bina_relu_no_dropout.fit(X_train_scaled_cnn, y_train, validation_data=(X_test_scaled_cnn,
y_test), epochs=10)

test_loss_1, test_accuracy_1 = bina_relu_no_dropout.evaluate(X_test_scaled_cnn, y_test)
test_accuracy_1 = test_accuracy_1 * 100
print(f"CNN (without dropout) test accuracy: {test_accuracy_1:.4f}")
print(f"CNN (without dropout) test loss: {test_loss_1:.4f}")
```

```
train_loss_1 = history_1.history['loss'][-1]
train_accuracy_1 = history_1.history['accuracy'][-1] * 100
print(f"CNN (without dropout) train accuracy: {train_accuracy_1:.4f}")
print(f"CNN (without dropout) train loss: {train_loss_1:.4f}")

#Model 2 - With Dropout Layer


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

# CNN Model with Dropout
bina_relu = tf.keras.Sequential([
   tf.keras.layers.Input(shape=(343, 256, 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(256, activation='relu'),
   tf.keras.layers.Dropout(0.5),
   tf.keras.layers.Dense(1, activation='sigmoid')
])

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

history_11 = bina_relu.fit(X_train_scaled_cnn, y_train, validation_data=(X_test_scaled_cnn, y_test),
epochs=10)

test_loss_11, test_accuracy_11 = bina_relu.evaluate(X_test_scaled_cnn, y_test)
test_accuracy_11 = test_accuracy_11 * 100
test_loss_11 = test_loss_11* 100
print(f"Test accuracy: {test_accuracy_11:.4f}")
print(f"Test loss: {test_loss_11:.4f}")

train_loss_11 = history_11.history['loss'][-1]
train_accuracy_11= history_11.history['accuracy'][-1] * 100
train_loss_11= train_loss_11 * 100
print(f"Train accuracy: {train_accuracy_11:.4f}")
print(f"Train loss: {train_loss_11:.4f}")

#Model with Leaky ReLU Activation
#without dropout

# CNN Model with Leaky ReLU Activation and No Dropout
bina_leaky_relu_no_dropout = tf.keras.Sequential([
   tf.keras.layers.Input(shape=(343, 256, 1)),
   tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.2)),
```

```
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.2)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation=tf.keras.layers.LeakyReLU(alpha=0.2)),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

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

history_2 = bina_leaky_relu_no_dropout.fit(X_train_scaled_cnn, y_train, epochs=10, batch_size=32,
validation_data=(X_test_scaled_cnn, y_test))

test_loss_2, test_accuracy_2 = bina_leaky_relu_no_dropout.evaluate(X_test_scaled_cnn, y_test)
print("Test Loss (without dropout):", test_loss_2)
print("Test Accuracy (without dropout):", test_accuracy_2)

train_loss_2, train_accuracy_2 = bina_leaky_relu_no_dropout.evaluate(X_train_scaled_cnn, y_train)
print("Train Loss (without dropout):", train_loss_2)
print("Train Accuracy (without dropout):", train_accuracy_2)

#Leaky ReLU with dropout

bina_leaky_relu_dropout = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(343, 256, 1)),
    tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.2)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation=tf.keras.layers.LeakyReLU(alpha=0.2)),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation=tf.keras.layers.LeakyReLU(alpha=0.2)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

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

history_22 = bina_leaky_relu_dropout.fit(X_train_scaled_cnn, y_train, epochs=10, batch_size=32,
validation_data=(X_test_scaled_cnn, y_test))

test_loss_22, test_accuracy_22 = bina_leaky_relu_dropout.evaluate(X_test_scaled_cnn, y_test)
print("Test Loss:", test_loss_22)
print("Test Accuracy:", test_accuracy_22)

train_loss_22, train_accuracy_22 = bina_leaky_relu_dropout.evaluate(X_train_scaled_cnn, y_train)
print("Train Loss:", train_loss_22)
print("Train Accuracy:", train_accuracy_22)
```

```
#Model with ELU Activation
#Without Dropout


import matplotlib.pyplot as plt

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

bina_elu_no_dropout = tf.keras.Sequential([
   tf.keras.layers.Input(shape=(343, 256, 1)),
   tf.keras.layers.Conv2D(32, kernel_size=(3, 3)),
   tf.keras.layers.Activation('elu'),
   tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
   tf.keras.layers.Conv2D(64, kernel_size=(3, 3)),
   tf.keras.layers.Activation('elu'),
   tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
   tf.keras.layers.Flatten(),
   tf.keras.layers.Dense(256),
   tf.keras.layers.Activation('elu'),
   tf.keras.layers.Dense(1, activation='sigmoid')
])

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

history_3 = bina_elu_no_dropout.fit(X_train_scaled_cnn, y_train, epochs=10, batch_size=32,
validation_data=(X_test_scaled_cnn, y_test))

test_loss_3, test_accuracy_3 = bina_elu_no_dropout.evaluate(X_test_scaled_cnn, y_test)
print("Test Loss:", test_loss_3)
print("Test Accuracy:", test_accuracy_3)

train_loss_3, train_accuracy_3 = bina_elu_no_dropout.evaluate(X_train_scaled_cnn, y_train)
print("Train Loss:", train_loss_3)
print("Train Accuracy:", train_accuracy_3)

#Elu with dropout layer

import tensorflow as tf
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

bina_elu_dropout = tf.keras.Sequential([
   tf.keras.layers.Input(shape=(343, 256, 1)),
   tf.keras.layers.Conv2D(32, kernel_size=(3, 3), activation='elu'),
   tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
   tf.keras.layers.Conv2D(64, kernel_size=(3, 3), activation='elu'),
```

```
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='elu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

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

history_33 = bina_elu_dropout.fit(X_train_scaled_cnn, y_train, epochs=10, batch_size=32,
validation_data=(X_test_scaled_cnn, y_test))

test_loss_33, test_accuracy_33 = bina_elu_dropout.evaluate(X_test_scaled_cnn, y_test)
print("Test Accuracy:", test_accuracy_33)

train_loss_33, train_accuracy_33 = bina_elu_dropout.evaluate(X_train_scaled_cnn, y_train)
print("Train Accuracy:", train_accuracy_33)

#Multi-Class
#Model 1 - without Dropout Layer

import h5py
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load data
file_path = "/content/drive/My Drive/ML - 2/Assignment - 3/spectrograms.h5"

with h5py.File(file_path, 'r') as f:
    species_names_multiclass = ['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3', 'norfli',
'rewbla', 'stejay', 'wesmea', 'whcspa']
    data = []
    labels = []
    for species in species_names_multiclass:
        species_data = f[species][:]
        species_label = [species_names_multiclass.index(species)] * species_data.shape[2]
        data.append(species_data.transpose(2, 1, 0))
        labels.extend(species_label)
X = np.concatenate(data, axis=0)
y = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(X_train.shape[0], -1)).reshape(X_train.shape)
X_test_scaled = scaler.transform(X_test.reshape(X_test.shape[0], -1)).reshape(X_test.shape)
```

```
X_train_scaled = X_train_scaled[..., np.newaxis]
X_test_scaled = X_test_scaled[..., np.newaxis]

model1 = tf.keras.Sequential([
   tf.keras.layers.Input(shape=X_train_scaled.shape[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(256, activation='relu'),
   tf.keras.layers.Dense(len(species_names_multiclass), activation='softmax')
])

model1.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history1 = model1.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), epochs=10)

test_loss1, test_accuracy1 = model1.evaluate(X_test_scaled, y_test)
test_accuracy1 = test_accuracy1 * 100
print(f"Model 1 Test accuracy: {test_accuracy1:.4f}")
print(f"Model 1 Test loss: {test_loss1:.4f}")

train_loss1 = history1.history['loss'][-1]
train_accuracy1 = history1.history['accuracy'][-1] * 100
print(f"Model 1 Train accuracy: {train_accuracy1:.4f}")
print(f"Model 1 Train loss: {train_loss1:.4f}")

plt.plot(history1.history['accuracy'], label='Train Accuracy')
plt.plot(history1.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model 1 Accuracy (without dropout)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='lower right')

plt.plot(history1.history['loss'], label='Train Loss')
plt.plot(history1.history['val_loss'], label='Validation Loss')
plt.title('Model 1 Loss (without dropout)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper right')

#Model2 - Adding Dropout Layer

import h5py
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Load data
file_path = "/content/drive/My Drive/ML - 2/Assignment - 3/spectrograms.h5"

with h5py.File(file_path, 'r') as f:
    species_names_multiclass = ['amecro', 'barswa', 'bkcchi', 'blujay', 'daejun', 'houfin', 'mallar3', 'norfli',
'rewbla', 'stejay', 'wesmea', 'whcspa']
    data = []
    labels = []
    for species in species_names_multiclass:
        species_data = f[species][:]
        species_label = [species_names_multiclass.index(species)] * species_data.shape[2]
        data.append(species_data.transpose(2, 1, 0))
        labels.extend(species_label)
X = np.concatenate(data, axis=0)
y = np.array(labels)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.reshape(X_train.shape[0], -1)).reshape(X_train.shape)
X_test_scaled = scaler.transform(X_test.reshape(X_test.shape[0], -1)).reshape(X_test.shape)
X_train_scaled = X_train_scaled[..., np.newaxis]
X_test_scaled = X_test_scaled[..., np.newaxis]

model2 = tf.keras.Sequential([
    tf.keras.layers.Input(shape=X_train_scaled.shape[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(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(len(species_names_multiclass), activation='softmax')
])

model2.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history2 = model2.fit(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test), epochs=10)

test_loss2, test_accuracy2 = model2.evaluate(X_test_scaled, y_test)
test_accuracy2 = test_accuracy2 * 100
print(f"Model 2 Test accuracy: {test_accuracy2:.4f}")
print(f"Model 2 Test loss: {test_loss2:.4f}")

train_loss2 = history2.history['loss'][-1]
```

```
train_accuracy2 = history2.history['accuracy'][-1] * 100
print(f"Model 2 Train accuracy: {train_accuracy2:.4f}")
print(f"Model 2 Train loss: {train_loss2:.4f}")

plt.plot(history2.history['accuracy'], label='Train Accuracy')
plt.plot(history2.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model 2 Accuracy (with dropout)')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(loc='upper left')

plt.plot(history2.history['loss'], label='Train Loss')
plt.plot(history2.history['val_loss'], label='Validation Loss')
plt.title('Model 2 Loss (with dropout)')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper right')

#Test
import librosa
import librosa.display

def generate_spectrograms(audio_paths):
   spectrograms = []
   for audio_path in audio_paths:

      y, sr = librosa.load(audio_path)

      S = librosa.feature.melspectrogram(y=y, sr=sr)

      S_db = librosa.power_to_db(S, ref=np.max)

      spectrograms.append(S_db)

      plt.figure(figsize=(10, 4))
      librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='mel')
      plt.colorbar(format='%+2.0f dB')
      plt.title('Spectrogram - ' + audio_path.split('/')[-1])
      plt.tight_layout()
      plt.show()

   return spectrograms

audio_paths = ["/content/drive/My Drive/ML - 2/Assignment - 3/test_birds/test1.mp3",
           "/content/drive/My Drive/ML - 2/Assignment - 3/test_birds/test2.mp3",
           "/content/drive/My Drive/ML - 2/Assignment - 3/test_birds/test3.mp3"]

spectrograms = generate_spectrograms(audio_paths)
```

```
import numpy as np
from skimage.transform import resize
from sklearn.preprocessing import StandardScaler

spectrograms = generate_spectrograms(audio_paths)

desired_shape = (343, 256)

processed_spectrograms = [resize(spectrogram, desired_shape, anti_aliasing=True) for spectrogram in
spectrograms]

spectrograms = np.array(processed_spectrograms)

spectrograms_flat = spectrograms.reshape(spectrograms.shape[0], -1)

scaler = StandardScaler()
scaler.fit(X_train_scaled.reshape(X_train_scaled.shape[0], -1))
spectrograms_scaled_flat = scaler.transform(spectrograms_flat)

spectrograms_scaled = spectrograms_scaled_flat.reshape(spectrograms.shape)
spectrograms_scaled = spectrograms_scaled[..., np.newaxis]

predictions = model2.predict(spectrograms_scaled)

predicted_indices = np.argmax(predictions, axis=1)
predicted_species = [species_names_multiclass[idx] for idx in predicted_indices]

for i, species in enumerate(predicted_species):
    print(f"Prediction for spectrogram {i + 1}: {species}")
```