DATA5322 Written Homework 3
Ting-Yu, Lin

Analysis of Seattle Bird Sounds: Deep Learning

Abstract

Birds play a vital role in ecological research and are indicators of habitat quality and environmental health. In this project, we utilize spectrograms extracted from audio clips and employ a neural network model to analyze bird sounds. This collection contains recordings of 12 bird species, derived from the Xeno-Canto Bird Sound Archive. Our approach involves building binary and multi-class classification models to identify bird species. Additionally, we apply a trained model to predict the species of bird call segments in Seattle. Results showed high accuracy for binary classification but low accuracy for multi-class classification, demonstrating the complexity of distinguishing multiple species. Challenges encountered include model biases and computational limitations. Despite these challenges, neural networks provide valuable insights into bird species identification and habitat monitoring.

Introduction

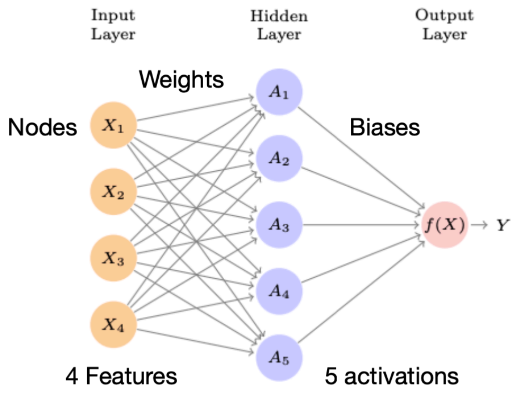
There are more than 10,000 species of birds in the world, and they are distributed in every corner. Birds play an important role in nature, their habitats can be used as excellent ecological indicators and also reflect the degree of environmental pollution.

The project aims to train neural network models based on the Birdcall competition dataset, which originally came from Xeno-Canto's, a crowd-sourced bird sounds archive. The provided dataset contains spectrograms of 10 mp3 sound clips extracted from 12 species of birds. These spectrograms allow us to train models to identify different bird species based on their unique acoustic signatures. Our approach consists of building a binary model and multi-category model. And use a multi-class model to predict bird species on three bird call clips from Seattle.

This analysis will help ecological researchers better understand changes in habitat quality, pollution levels, and facilitate the progress of restoration efforts.

Background

A neural network is a network of neurons organized in layers. Figure 1 shows example of a neural network structure. It processes data through a combination of linear and nonlinear transformations. Initially, inputs are combined linearly, integrating observations into the network. Then, hidden layers apply nonlinear activation functions to transform these inputs. Modern neural networks typically have more than one hidden layer. Finally, the output is derived from these transformations, providing a simplified representation of the data. The process will follows the formula : $A_k=g(w_{k0}+\sum w_{kj}X_j)$, where A_k is the output of neuron k, X_j are the inputs,w_{k0} is the bias term,w_{kj} are the weights, and g() is an activation function chosen based on the specific model. Figure 2 shows some activation function. In the project, we will use sigmoid for binary model, and softmax for multi-class model.



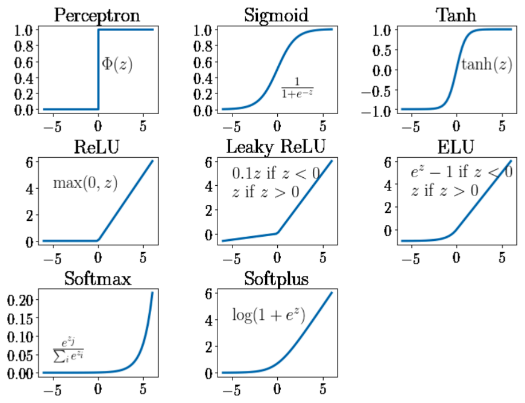


Figure 1. Example of a neural network structure.

Figure 2. Activation function.

Neural networks are extremely flexible and work on a wide variety of problems. A Convolutional Neural Network (CNN) is a common architecture used for image classification. It consists of convolution layers, where filters slide across an image to compute features, and pooling layers to condense information.Convolutional filters, the model weights, are learned during training, typically using the ReLU activation function. Pooling layers, like max pooling, summarize information by taking max values in image blocks.

Fourier transform is commonly paired with neural networks for tasks such as signal processing and feature extraction. It converts signals from the time or spatial domain into the frequency domain, enhancing neural networks' ability to grasp underlying patterns and structures within the data.

When building the neural network model, we can add the Dropout layer. In each training iteration, a certain percentage of neurons will be randomly dropped out, aiding in preventing the model from overly relying on specific neurons. We can also adjust the number of neurons in the Dense layer for both input and output. Additionally, Conv2D and Pooling layers can be included. The Conv2D layer enables the model to capture local features from different regions, while the Pooling layer is typically placed after the convolutional layer to reduce the spatial dimensions of the feature maps, while retaining the most important features.[1]

When training the neural network model, we can adjust parameters such as epochs (the number of complete passes through the entire training dataset) and batch size (the number of training examples utilized in one iteration). By adjusting these parameters and plotting the loss and accuracy, we can assess whether the model is overfitting and determine the best configuration for the model.

Methodology

Data Preparation

We selected 10 sound clips to be the highest quality available sub-60-second clips for each species. Each sound clip is first subsampled to a sample rate of 22,050 Hz. Then, segments of the audio that are "loud" and last longer than 0.5 seconds are identified. From these, 2-second windows are selected where bird calls are clearly detected. For each 2-second segment, a spectrogram is generated with 343 pixels in time and 256 pixels in frequency, representing a bird's call. Finally, these spectrograms are saved individually for each species, resulting in a dataset with a variable number of samples per species.

Binary Model

We selected two bird species, ‘amecro’ and ‘barswa’, to build a neural network for binary classification. Each spectrogram was labeled according to its species, amecro assigned a label of 0 and barswa a label of 1. We then concatenated the labeled data from both species to create a unified dataset. This dataset was split into training and testing subsets at an 80:20 ratio.

Next, we constructed the model by using a sequential model architecture. The model consists of a flattening layer to convert 2D spectrogram data into 1D, followed by two dense layers(128 units and 64 units). Each dense layers followed by a dropout layer (40% in the first dropout and 30% in the second). The output layer used a sigmoid activation function for binary classification.

Finally, we compiled the model with the RMSprop optimizer and binary crossentropy as the loss function. We tested various numbers of epochs (5, 10, 20, and 50) and batch sizes (32, 128, and 256), using a validation split of 20% to monitor overfitting. Then, we evaluated the accuracy on the test set and created plots of the training and validation loss and accuracy to identify the optimal model configuration.

Multi-class Model

To build a neural network that classifies between all 12 bird species, we first generated a unique label for each species and concatenated the labeled data from all species to

create a unified dataset. The labels were then one-hot encoded to prepare for neural network training. The dataset was subsequently split into training and testing subsets at an 80:20 ratio.
Next, we built a model similar to the binary model but replaced the sigmoid activation function with a softmax activation function to handle multiple classes. Additionally, we constructed another model with two convolutional layers, each followed by a max pooling layer. This was followed by a flattening layer, a dense layer with 128 units, and a dropout layer(50% dropout rate). The output layer used a softmax activation function suitable for multi-class classification.

Finally, we compiled the model with the RMSprop optimizer and categorical crossentropy as the loss function. It was trained for 50 epochs with a batch size of 256, and used a validation split of 20% to monitor overfitting. Then, we evaluated the accuracy on the test set and created plots of the training and validation loss and accuracy to find the optimal model configuration.

External Test Data

The three bird call clips undergo the data preparation steps mentioned above, produced a spectrogram with 343x256 pixels. These spectrograms were then analyzed using a multi-class model to predict the species of birds contained in the clips.

Results

Figure 3 shows the spectrograms of 12 species, from which we can see that different species have distinct waveform patterns.

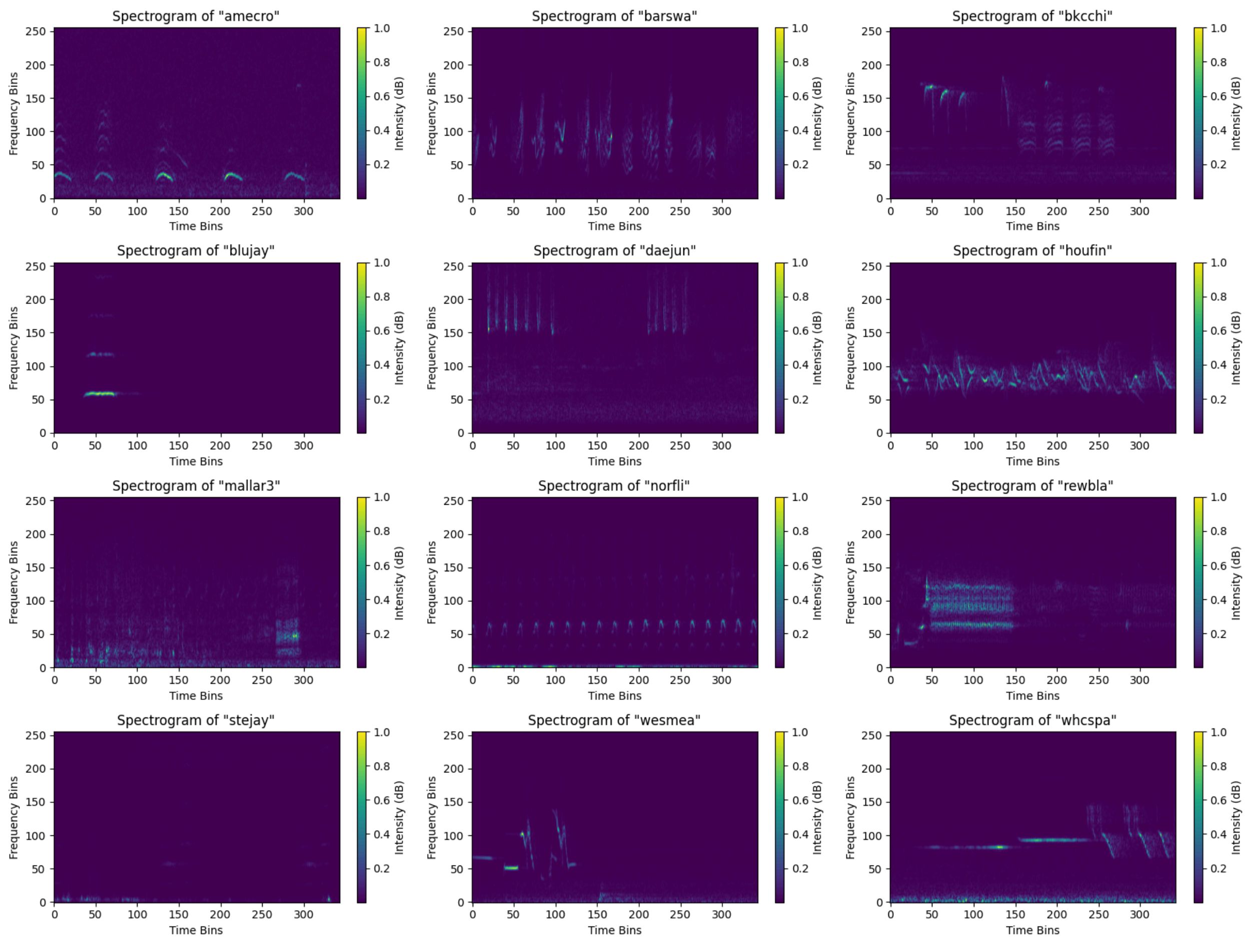


Figure 3. Spectrograms of 12 bird species

Binary Model

Figure 4 displays the training and validation loss and accuracy across different batch sizes. The plots show that the model with a batch size of 128 achieves lower training and validation losses, and slightly higher accuracy compared to other batch sizes. We have selected a batch size of 128 as the training parameter for subsequent models.

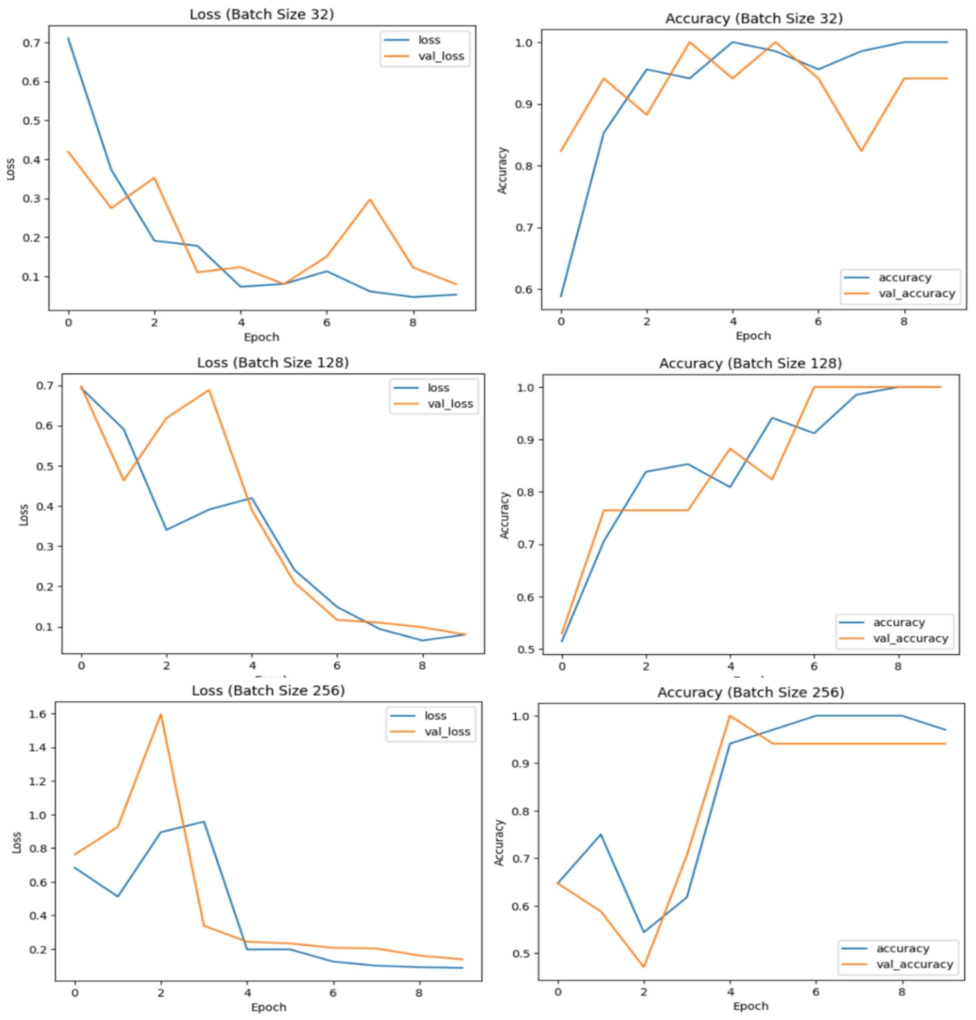


Figure 4. The training and validation loss and accuracy with different batch sizes.

We tested different numbers of epochs, as shown in Figure 5. Initially, training and validation losses fluctuated significantly, likely due to model weight initialization and learning rate settings. As training progressed, the model adapted better, and losses stabilized, indicating effective learning. Models with 20 and 50 epochs showed consistently low losses; however, the 20-epoch model displayed a slight decline in validation accuracy towards the end, suggesting overfitting. Conversely, the 50-epoch model maintained high and stable accuracy. Therefore, we have chosen 50 epochs as the optimal training parameter for our models. The model trained with 50 epochs and 128 batch size can get nearly 100% test accuracy. This indicates that the model can accurately distinguish between the calling sounds of ‘amecro’ and ‘barswa’.

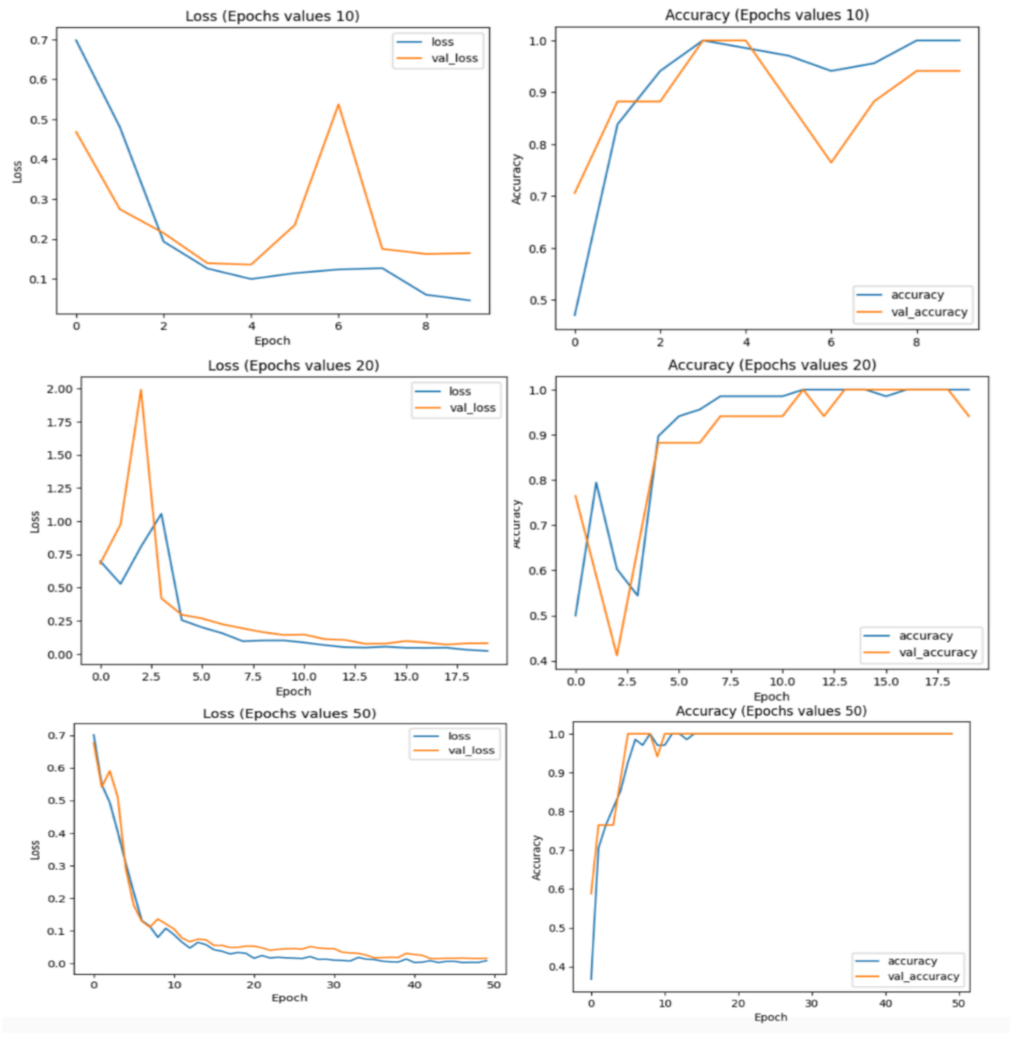


Figure 5. The training and validation loss and accuracy with various numbers of epochs.

Multi-class Model

First, we built model 1 with parameters similar to those of the binary model but replaced the sigmoid activation function with a softmax activation function. We then trained model 1 for 50 epochs with a batch size of 128. The results are shown in Figure 6. When it reached 50 epochs, both the training and validation loss and accuracy values were very stable.

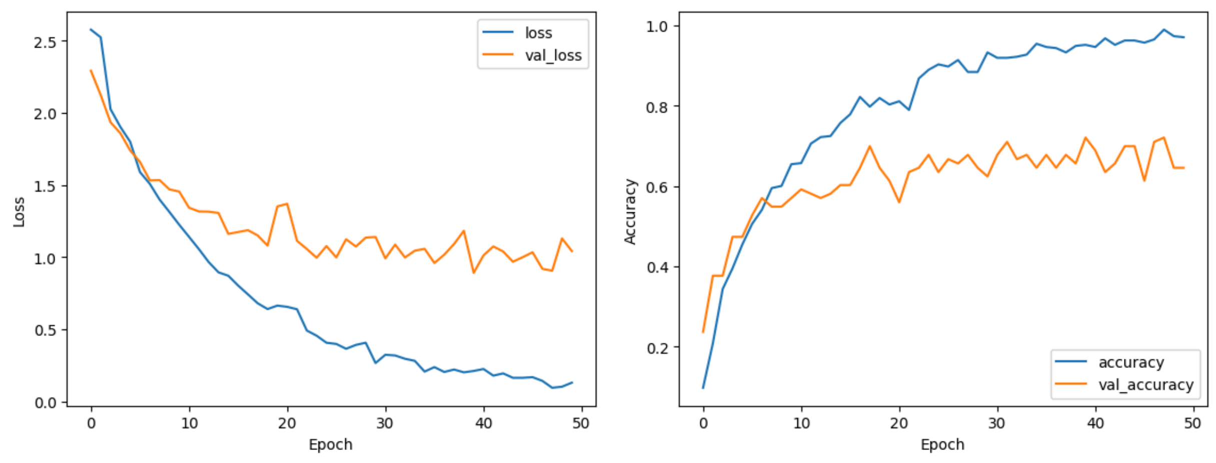


Figure 6. The training and validation loss and accuracy of model 1.

Next, we built model 2 by adding two convolutional layers and max pooling layers. The output layer used a softmax activation function. And trained model 2 for 50 epochs with a batch size of 128. The results are shown in Figure 7. The result similar to model 1.

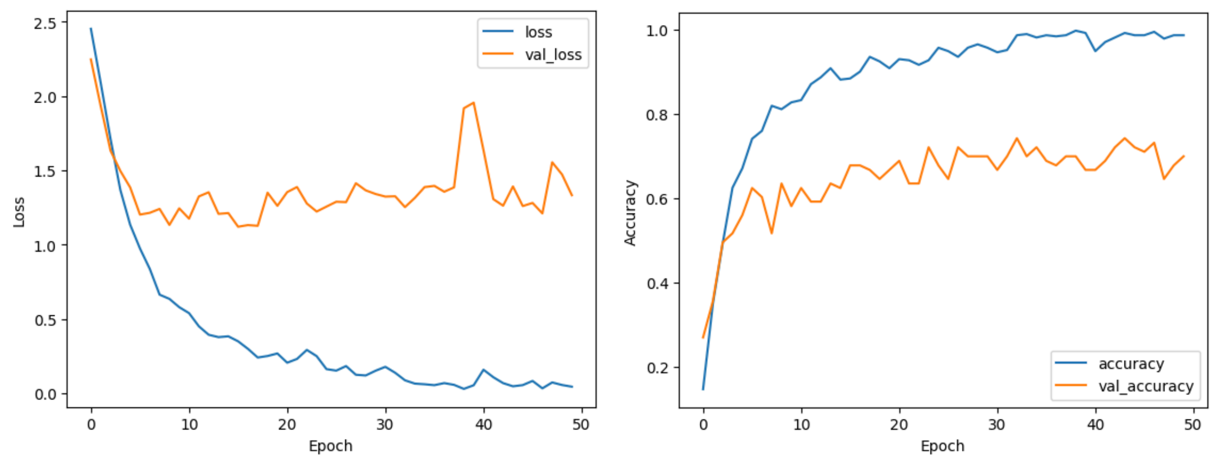


Figure 7. The training and validation loss and accuracyof model 2.

We used Model 1 and Model 2 to make predictions on the test data, achieving test accuracies of 64.66% and 67.24%, respectively. Therefore, we have chosen Model 2 to make predictions on external test data.

External Test Data

Spectrograms of three bird call clips showns in Figure 8. We can see the distinct waveform patterns.

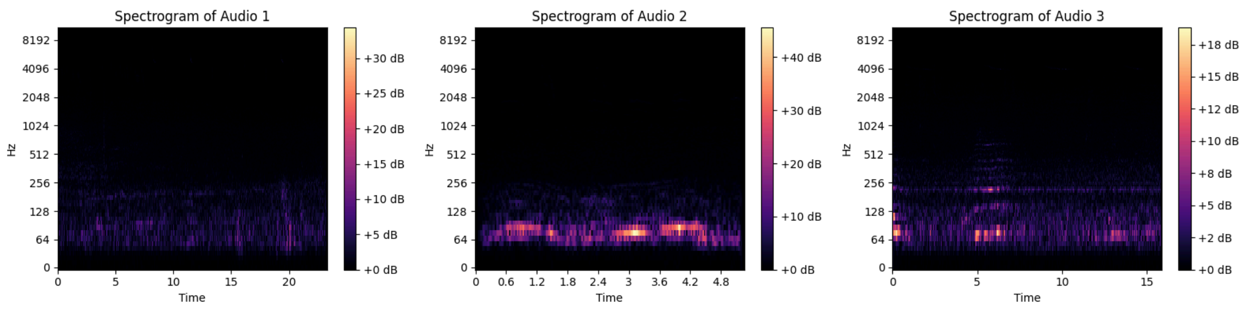


Figure 8 Spectrograms of three bird call clips.

We used model 2 to predict three bird call clips, with the results shown in Figure 9. All three bird call clips were predicted to contain calls from the ‘norfli’ species, and Audio 2 also includes calls from the ‘daejun’ species.

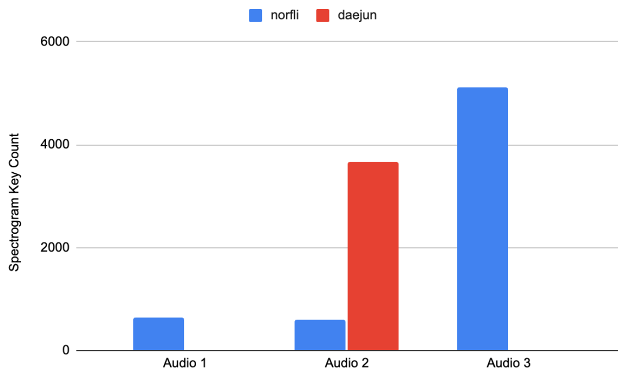


Figure 9: Predictions for Three Bird Call Clips.

Discussion

In this project, the binary model we used was able to achieve close to 100% test accuracy and was able to accurately distinguish between ‘amecro’ and ‘barswa’ bird sounds. However, in the multi-class model, the highest value of test accuracy is only about 67%. This may be because the number of bird species is increased from two to twelve, which greatly increases the complexity of the data, resulting in a lower accuracy rate. In addition, if the model is too simple, it may not be able to effectively capture the complex features in the data. By adding more layers or increasing the number of neurons, the model can be helped to learn more detailed features, thereby improving the performance of the model.

The model predicts the spectrograms of bird audio clips 1 and 3 as ‘norfli’, but after listening to these audio clips, it is obvious that they are two different sounds. This indicates a bias in the model's predictive ability, possibly due to insufficient model accuracy or insufficient diversity in the spectrogram data. On the other hand, for bird sound audio clip 2, it can be clearly heard that it contains two different bird sounds, which is consistent with our prediction results. This reflects that the model is able to accurately classify different bird sounds in some cases, but it still needs to further improve its prediction accuracy.

Looking at the spectrograms of 12 bird species, we can find that the spectrograms of ‘houfin’ and ‘rewbla’ show complex and dense wave patterns, which easily overlap with the characteristics of other bird species, leading to classification mistake.

SVM can also be used to perform the classification task in this project. Since SVM does not directly process image data, the spectrogram data must be converted into one-dimensional feature vectors. Furthermore, the data should be standardized or normalized because SVM is highly sensitive to the scale of the features. Neural networks are extremely effective in tasks involving images and their extension to spectrograms. They can automatically learn and detect complex patterns in data, making them ideal for identifying a wide range of bird calls.

During the process of compiling this report, I frequently encountered RAM limitations that forced tasks originally run on a CPU had to be switched to a GPU. However, the GPU resources available on Colab are limited, which became a significant limitation during the analysis. Training the model typically takes about five minutes, with run times varying based on the number of epochs and batch size used. Convolutional neural network (CNN) requires a large amount of computing resources due to its complex architecture (including multiple convolutional layers, pooling layers and fully connected layers). Furthermore, each training iteration involves forward and backward propagation, which is particularly time-consuming in networks with multiple layers.

Conclusions

This project used neural networks to predict the species of a bird's sound. In the binary model, the model successfully identified ‘amecro’ and ‘barswa’ with almost perfect accuracy, highlighting the effectiveness of neural networks in simple classification tasks. However, in the multi-class model, the accuracy was lower, showing the complexity of classifying between multiple breeds. Predictions using a multi-class model on three external test data all showed birds of the "norfli" variety, but clearly sounded different, indicating a bias in the model's predictive. The model needs to be improved or the dataset examined. Neural networks are very effective for analyzing images such as spectrograms. But it is characterized by being slow and computationally expensive, sometimes making the analysis process difficult. Improving these constraints and improving the ability to analyze and predict bird sounds will help ecological researchers better understand changes in habitat quality, pollution levels, and facilitate progress in restoration efforts.

References

[1] Convolution & Pooling Layers, Nikita Malviya, 2023
[2] Cornell Birdcall Identification, kaggle, mohan_220656, 2023
[3] Xeno-Canto Bird Sound Archive, 2005-2024
[4] Xeno-Canto Bird Recordings Extended, rohanrao vopani, 2020

Appendix

```
# Library
import h5py
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
from keras.utils import to_categorical
import matplotlib.pyplot as plt
from matplotlib import image
from tensorflow.keras.utils import to_categorical
import pandas as pd
from scipy.ndimage import zoom
import librosa

# Binary Model(test batch_sizes)
f = h5py.File('/content/drive/MyDrive/Colab Notebooks/spectrograms.h5', 'r')
# Load and transpose data
amecro_data = f['amecro'][:].transpose((2, 0, 1))
barswa_data = f['barswa'][:].transpose((2, 0, 1))
# Create labels, amecro as 0, barswa as 1
labels_amecro = np.zeros(amecro_data.shape[0])
labels_barswa = np.ones(barswa_data.shape[0])
# Merge data
data = np.concatenate((amecro_data, barswa_data), axis=0)
labels = np.concatenate((labels_amecro, labels_barswa), axis=0)

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=42)
# Batch sizes to test
batch_sizes = [32, 128, 256]
```

```
for batch_size in batch_sizes:
 # Build the model
  model = Sequential([
      Flatten(input_shape=(256, 343)),
      Dense(128, activation='relu'),
      Dropout(0.4),
      Dense(64, activation='relu'),
      Dropout(0.3),
      Dense(1, activation='sigmoid') ])
  # Compile the model
  model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
  # Train the model
  history = model.fit(X_train, y_train, epochs=10, batch_size=batch_size
                      , validation_split=0.2, verbose=0)
  # Evaluate the model
  score = model.evaluate(X_test, y_test, verbose=0)
  print("Test Accuracy: {:.2f}%".format(score[1]*100))
  # Make plots
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title("Loss (Batch Size {})".format(batch_size))
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['loss', 'val_loss'], loc='upper right')
  plt.show();
  plt.plot(history.history['accuracy'])
  plt.plot(history.history['val_accuracy'])
  plt.title("Accuracy (Batch Size {})".format(batch_size))
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['accuracy', 'val_accuracy'], loc='lower right')
  plt.show();

# Binary Model(test epochs)
f = h5py.File('/content/drive/MyDrive/Colab Notebooks/spectrograms.h5', 'r')
# Load and transpose data
amecro_data = f['amecro'][:].transpose((2, 0, 1))
barswa_data = f['barswa'][:].transpose((2, 0, 1))
# Create labels, amecro as 0, barswa as 1
labels_amecro = np.zeros(amecro_data.shape[0])
```

```
labels_barswa = np.ones(barswa_data.shape[0])
# Merge data
data = np.concatenate((amecro_data, barswa_data), axis=0)
labels = np.concatenate((labels_amecro, labels_barswa), axis=0)
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
random_state=42)
# epochs_number to test
epochs_number = [10, 20, 50]
for epochs in epochs_number:
  # Build the model
  model = Sequential([
      Flatten(input_shape=(256, 343)),
      Dense(128, activation='relu'),
      Dropout(0.4),
      Dense(64, activation='relu'),
      Dropout(0.3),
      Dense(1, activation='sigmoid') ])
  # Compile the model
  model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accuracy'])
  # Train the model
  history = model.fit(X_train, y_train, epochs=epochs, batch_size=128
                      , validation_split=0.2, verbose=0)
  # Evaluate the model
  score = model.evaluate(X_test, y_test, verbose=0)
  print("Test Accuracy: {:.2f}%".format(score[1]*100))
  plt.plot(history.history['loss'])
  plt.plot(history.history['val_loss'])
  plt.title("Loss (Epochs values {})".format(epochs))
  plt.ylabel('Loss')
  plt.xlabel('Epoch')
  plt.legend(['loss', 'val_loss'], loc='upper right')
  plt.show();
  plt.plot(history.history['accuracy'])
  plt.plot(history.history['val_accuracy'])
  plt.title("Accuracy (Epochs values {})".format(epochs))
  plt.ylabel('Accuracy')
  plt.xlabel('Epoch')
  plt.legend(['accuracy', 'val_accuracy'], loc='lower right')
  plt.show();
```

```
# Figure 8
# Load the audio file
y1, sr1 = librosa.load('/content/drive/MyDrive/Colab Notebooks/test_birds/test1.mp3',
sr=22050)
y2, sr2 = librosa.load('/content/drive/MyDrive/Colab Notebooks/test_birds/test2.mp3',
sr=22050)
y3, sr3 = librosa.load('/content/drive/MyDrive/Colab Notebooks/test_birds/test3.mp3',
sr=22050)
# Perform Short-Time Fourier Transform to generate a spectrogram
D1 = librosa.stft(y1)
S_db1 = librosa.amplitude_to_db(np.abs(D1), ref=np.max)
D2 = librosa.stft(y2)
S_db2 = librosa.amplitude_to_db(np.abs(D2), ref=np.max)
D3 = librosa.stft(y3)
S_db3 = librosa.amplitude_to_db(np.abs(D3), ref=np.max)
# Create subplots
plt.figure(figsize=(16, 4))
# Plot for Audio 1
plt.subplot(1, 3, 1)
librosa.display.specshow(D1, sr=sr1, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram of Audio 1')
# Plot for Audio 2
plt.subplot(1, 3, 2)
librosa.display.specshow(D2, sr=sr2, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram of Audio 2')
# Plot for Audio 3
plt.subplot(1, 3, 3)
librosa.display.specshow(D3, sr=sr3, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram of Audio 3')
# Adjust layout for better visualization
plt.tight_layout()
# Show the plot
plt.show()

# External Test Data
# Load audio file at half the sampling rate
```

```
y, sr = librosa.load('/content/drive/MyDrive/Colab Notebooks/test_birds/test1.mp3',
sr=22050)
n_fft = 512  # Keeps frequency bins around 257, which is close to 256
hop_length = max(1, int((len(y) / sr) / 343))  # Adjust hop_length to achieve ~343 time
steps
# Generate the spectrogram again with adjusted parameters
frame_length  = n_fft
energy = np.array([
    sum(abs(y[i:i+frame_length]**2))
    for i in range(0, len(y), hop_length)])
# Find frames with energy higher than a threshold
threshold = np.max(energy) * 0.5
high_energy_frames = np.where(energy > threshold)[0]
# Convert frames to time
loud_times = librosa.frames_to_time(high_energy_frames, sr=sr,
hop_length=hop_length)
def resize_spectrogram(spectrogram, target_shape):
    y_ratio = target_shape[0] / spectrogram.shape[0]
    x_ratio = target_shape[1] / spectrogram.shape[1]
    resized_spectrogram = zoom(spectrogram, (y_ratio, x_ratio), order=1)  # Using
bilinear interpolation (order=1)
    return resized_spectrogram
# Function to generate a spectrogram and save it
def generate_spectrogram(audio, start_time, sr, filename):
    # Extract 2-second clip
    start_sample = int(start_time * sr)
    end_sample = start_sample + 2 * sr
    clip = audio[start_sample:end_sample]
    # Generate spectrogram
    D = librosa.stft(clip)
    S_db = librosa.amplitude_to_db(np.abs(D), ref=np.max)
    # Resize the spectrogram if necessary
    if S_db.shape != (256, 343):
      S_db = resize_spectrogram(S_db, (256, 343))

            # Save spectrogram to HDF5
    with h5py.File('/content/drive/MyDrive/Colab
Notebooks/test_birds/test1_spectrograms.h5', 'a') as file:
        file.create_dataset(filename, data=S_db)
    return S_db
```

```
# Process each loud section and visualize the last one
for i, time in enumerate(loud_times):
    S_db = generate_spectrogram(y, time, sr, f'spectrogram_{i}')
librosa.display.specshow(S_db, sr=sr, x_axis='time', y_axis='log')
plt.colorbar(format='%+2.0f dB')
plt.title('Spectrogram')
plt.show()
def prepare_spectrogram(s_db):
    # Assuming the training data was normalized or preprocessed similarly
    s_db_normalized = s_db / np.max(np.abs(s_db))
    return np.expand_dims(s_db_normalized, axis=-1)  # Add channel dimension if
needed
# Open the file with test spectrograms
with h5py.File('/content/drive/MyDrive/Colab
Notebooks/test_birds/test1_spectrograms.h5', 'r') as file:
    # List all datasets (spectrograms) in the file
    spectrogram_keys = list(file.keys())
    test_spectrograms = [prepare_spectrogram(file[key][:]) for key in spectrogram_keys]
predictions = model.predict(np.array(test_spectrograms))
predicted_classes = np.argmax(predictions, axis=1)
predicted_birds = [bird_types[i] for i in predicted_classes]
# Create a DataFrame to display results
results_df = pd.DataFrame({
    'Spectrogram Key': spectrogram_keys,
    'Predicted Bird': predicted_birds})
# Display the DataFrame
print(results_df)
```