Transfer Learning for Identification of Rare Bird Species

I. Installing and Importing Libraries

```
# --- Installing and importing libraries---

# Installing libraries
#!pip install birdnetlib
#!pip install librosa==0.9.2
#!pip install opensoundscape
#!pip install git+https://github.com/kitzeslab/bioacoustics-model-zoo
#!pip install tensorflow tensorflow-hub
#!pip install pydub
#!pip install joblib

# Import packages
import os
import bioacoustics_model_zoo as bmz
import joblib

from pydub import AudioSegment
from pydub.utils import mediainfo
from google.colab import drive
from bioacoustics_model_zoo import BirdNET
from birdnetlib import Recording
from birdnetlib.analyzer import Analyzer
from datetime import datetime
from collections import defaultdict
from collections import Counter

from google.colab import drive

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.model_selection import StratifiedKFold
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics.pairwise import cosine_similarity
```

```
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDrive2ImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDriveImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _BokehImportHook.find_spec() no
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDrive2ImportHook.find_spec()
```

```
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDriveImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _BokehImportHook.find_spec() no
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDrive2ImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDriveImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _BokehImportHook.find_spec() no
```

II. Pre-Processing Audio Files

```
# ---Transform audio to standard WAV and padding to 3 seconds---

# Mount Google Drive
drive.mount('/content/drive')

# Input folders (original audio)
input_folders = {
    "NEG": "/content/drive/My Drive/AlternativeBirds/",
    "POS": "/content/drive/My Drive/OnlyAudioDoliornis/",
    "OPOS": "/content/drive/My Drive/OnlyAudioHapalopsittaca/"
}

# Output folders (converted WAV files)
output_folders = {
    "NEG": "/content/drive/My Drive/AlternativeBirds_wav/",
    "POS": "/content/drive/My Drive/OnlyAudioDoliornis_wav/",
    "OPOS": "/content/drive/My Drive/OnlyAudioHapalopsittaca_wav/"
}

# Function to convert and pad audio
def convert_and_pad_to_3s(input_path, output_path):
    try:
        audio = AudioSegment.from_file(input_path)
        duration_ms = 3000  # 3 seconds

        if len(audio) < duration_ms:
            silence = AudioSegment.silent(duration=duration_ms - len(audio))
            audio = audio + silence

        # Let pydub handle the writing
        audio.export(output_path, format="wav")

        print(f"Processed and saved: {output_path}")
    except Exception as e:
        print(f"Error processing {input_path}: {e}")

# Process each folder
for key in input_folders:
    input_dir = input_folders[key]
    output_dir = output_folders[key]
    os.makedirs(output_dir, exist_ok=True)

    audio_files = [
        os.path.join(input_dir, f)
        for f in os.listdir(input_dir)
```

```
        for f in os.listdir(input_dir)
        if f.lower().endswith(('.mp3', '.wav', '.flac', '.ogg'))
    ]

    for input_path in audio_files:
        filename = os.path.splitext(os.path.basename(input_path))[0] + ".wav"
        output_path = os.path.join(output_dir, filename)
        convert_and_pad_to_3s(input_path, output_path)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call driv
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC187 - Grey-bre
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC2069 - Plum-cr
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC3470 - Blue-ba
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC2019 - Plum-cr
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC2012 - Jameson
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC1598 - Blue-ba
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC20706 - Blue-t
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC14584 - Grey-b
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC62895 - Black
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC178 - Jameson_
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/AlternativeBirds_wav/XC2915 - Andean
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC10781 - Bay-
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC20615 - Bay-
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC41676 - Bay-
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC65640 - Bay-
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC142403 - Bay
```

```
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC222460 - Bay
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC222458 - Bay
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
Processed and saved: /content/drive/My Drive/OnlyAudioDoliornis_wav/XC222459 - Bay
```

III. Verifying if BirdNET Identifies our Rare Species

```
# --- Check if BirdNET identifies our rare species---

def analyze_audio_folder(folder_path):
    analyzer = Analyzer()  # Load BirdNET model

    audio_files = [
        f for f in os.listdir(folder_path)
        if f.lower().endswith((".mp3", ".wav"))
    ]

    for filename in audio_files:
        file_path = os.path.join(folder_path, filename)
        print(f"\nAnalyzing:")

        try:
            recording = Recording(analyzer, file_path)
            recording.analyze()

            detections = recording.detections

            if detections:
                print("Detections:")
                for d in detections:
                    print(f"  - {d['common_name']} ({d['confidence']:.2f})")
            else:
                print("No detections found.")

        except Exception as e:
            print(f"Error analyzing {filename}: {e}")

#For Doliornis
print("Identification results corresponding to Doliornis:")
analyze_audio_folder(input_folders["POS"])

#For Hapalopsittaca
print("\nIdentification results corresponding to Hapalopsittaca:")
analyze_audio_folder(input_folders["OPOS"])
```

```
Identification results corresponding to Doliornis:
Labels loaded.
```

```
load model True
Model loaded.
Labels loaded.
load_species_list_model
Meta model loaded.

Analyzing:
read_audio_data
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDrive2ImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDriveImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _BokehImportHook.find_spec() no
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDrive2ImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDriveImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _BokehImportHook.find_spec() no
read_audio_data: complete, read  1 chunks.
analyze_recording XC10781 - Bay-vented Cotinga - Doliornis sclateri.mp3
No detections found.

Analyzing:
read_audio_data
read_audio_data: complete, read  9 chunks.
analyze_recording XC20615 - Bay-vented Cotinga - Doliornis sclateri.mp3
Detections:
  - Dot-winged Antwren (0.50)
  - Bertoni's Antbird (0.19)
  - White-bellied Antbird (0.21)
  - Human non-vocal (0.34)
  - American Bullfrog (0.16)

Analyzing:
read_audio_data
read_audio_data: complete, read  40 chunks.
analyze_recording XC41676 - Bay-vented Cotinga - Doliornis sclateri.mp3
Detections:
  - Western Rock Nuthatch (0.13)
  - Gray Flycatcher (0.12)
  - Turquoise Jay (0.26)
  - White-throated Tyrannulet (0.11)
  - Red-necked Phalarope (0.11)
  - White-throated Tyrannulet (0.51)
  - White-throated Tyrannulet (0.22)
  - Barred Fruiteater (0.48)
  - Turquoise Jay (0.20)
  - Tschudi's Tapaculo (0.46)
  - Tschudi's Tapaculo (0.64)
  - Unicolored Tapaculo (0.42)
  - Citrine Warbler (0.21)
  - European Goldfinch (0.18)
  - White-throated Tyrannulet (0.49)
  - White-throated Tyrannulet (0.86)
  - White-throated Tyrannulet (0.12)

Analyzing:
read_audio_data
read_audio_data: complete, read  41 chunks.
analyze_recording XC65640 - Bay-vented Cotinga - Doliornis sclateri.mp3
```

RESULT: BirdNET does not identify our rare species (Doliornis sclateri or Hapalopsittaca melanotis)

melanotis)

IV. Obtaining Embeddings from BirdNET for Transfer Learning

```
# --- Get the embeddings of our rare species from BirdNET for transfer learning ---

# Initialize BirdNET model
model = bmz.BirdNET()

# Create function for extracting embeddings
import numpy as np
def extract_embeddings(audio_folder, output_folder):
    os.makedirs(output_folder, exist_ok=True)

    audio_files = [
        os.path.join(audio_folder, f)
        for f in os.listdir(audio_folder)
        if f.lower().endswith(('.mp3', '.wav'))
    ]

    if not audio_files:
        print(f"No audio files found in {audio_folder}")
        return

    print(f"Embedding {len(audio_files)} files from: {audio_folder}")

    # Run embedding
    embeddings_df = model.embed(audio_files)

    # Group by filename (from MultiIndex)
    grouped = embeddings_df.groupby(level=0)

    for file_name, group in grouped:
        emb_array = group.to_numpy()  # shape: (n_chunks, 1024)
        base_name = os.path.splitext(os.path.basename(file_name))[0]
        save_path = os.path.join(output_folder, base_name + ".npy")
        np.save(save_path, emb_array)
        print(f"Saved: {save_path} with shape {emb_array.shape}")

# Run the function
audio_to_embedding_folders = {
    "/content/drive/My Drive/AlternativeBirds_wav/": "/content/drive/My Drive/Alter
    "/content/drive/My Drive/OnlyAudioDoliornis_wav/": "/content/drive/My Drive/Onl
    "/content/drive/My Drive/OnlyAudioHapalopsittaca_wav/": "/content/drive/My Driv
}

for audio_folder, embedding_folder in audio_to_embedding_folders.items():
    extract_embeddings(audio_folder, embedding_folder)
```

```
File BirdNET_GLOBAL_6K_V2.4_Labels_af.txt already exists; skipping download.
downloading model from URL...
File BirdNET_GLOBAL_6K_V2.4_Model_FP16.tflite already exists; skipping download
```

```
File BirdNET_GLOBAL_6K_V2.4_Model_FP16.tflite already exists, skipping download.
Embedding 11 files from: /content/drive/My Drive/AlternativeBirds_wav/
/usr/local/lib/python3.11/dist-packages/opensoundscape/ml/cnn.py:599: UserWarning:
                    This architecture is not listed in opensoundscape.ml.cnn_archi
                    It will not be available for loading after saving the model wi
                    To make it re-loadable, define a function that generates the a
                    then use opensoundscape.ml.cnn_architectures.register_architec

                    The function can also set the returned object's .constructor_n
                    to avoid this warning and ensure it is reloaded correctly by o

                    See opensoundscape.ml.cnn_architectures module for examples of

  warnings.warn(
/usr/local/lib/python3.11/dist-packages/opensoundscape/ml/cnn.py:623: UserWarning:
  warnings.warn(
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC14584 - Grey-breasted Mounta
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC1598 - Blue-banded Toucanet
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC178 - Jameson_s Snipe - Gall
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC187 - Grey-breasted Mountain
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC2012 - Jameson_s Snipe - Gal
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC2019 - Plum-crowned Parrot -
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC2069 - Plum-crowned Parrot -
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC20706 - Blue-throated Piping
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC2915 - Andean Cock-of-the-ro
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC3470 - Blue-banded Toucanet
Saved: /content/drive/My Drive/AlternativeBirds_Emb/XC62895 - Black Tinamou - Tina
Embedding 12 files from: /content/drive/My Drive/OnlyAudioDoliornis_wav/
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC142403 - Bay-vented Coting
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC20615 - Bay-vented Cotinga
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC222458 - Bay-vented Coting
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC222459 - Bay-vented Coting
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC222460 - Bay-vented Coting
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC296816 - Bay-vented Coting
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC296818 - Bay-vented Coting
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC296819 - Bay-vented Coting
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC41676 - Bay-vented Cotinga
Saved: /content/drive/My Drive/OnlyAudioDoliornis_Emb/XC65640 - Bay-vented Cotinga
Embedding 11 files from: /content/drive/My Drive/OnlyAudioHapalopsittaca_wav/
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC151165 - Black-winged
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC151172 - Black-winged
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC16129 - Black-winged
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC2004 - Black-winged P
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC2005 - Black-winged P
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC2070 - Black-winged P
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC349146 - Black-winged
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC350403 - Black-winged
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC354158 - Black-winged
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC354160 - Black-winged
Saved: /content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/XC821443 - Black-winged
```

```
# --- Check number of embbedings per folder---

# Folder dictionary
folders = {
    0: "/content/drive/My Drive/AlternativeBirds_Emb/",
    1: "/content/drive/My Drive/OnlyAudioDoliornis_Emb/"
```

```
    1: "/content/drive/My Drive/OnlyAudioBolitoris_Emb/",
    2: "/content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/"
}

# Count embeddings
embedding_counts = {}

for label, folder_path in folders.items():
    total_embeddings = 0
    for file in os.listdir(folder_path):
        if file.endswith('.npy'):
            filepath = os.path.join(folder_path, file)
            data = np.load(filepath)

            # If 1D, count as 1 embedding; otherwise, use number of rows
            if data.ndim == 1:
                total_embeddings += 1
            else:
                total_embeddings += data.shape[0]

    embedding_counts[label] = total_embeddings

# Print results
for label, count in embedding_counts.items():
    print(f"Class {label}: {count} embeddings")
```

```
Class 0: 102 embeddings
Class 1: 154 embeddings
Class 2: 74 embeddings
```

```
# --- Check the number of embeddings in one recording ---
# Embeddings are of 3 seconds, therefore, some recordings may have more than one embed

# Choose a folder and file
folder = "/content/drive/My Drive/AlternativeBirds_Emb/"
files = [f for f in os.listdir(folder) if f.endswith('.npy')]

# Load one file (e.g., the first one)
file_path = os.path.join(folder, files[0])
data = np.load(file_path)

# Show info
print(f"File: {files[0]}")
print("Shape:", data.shape)
print("Rows:\n", data)
```

```
File: XC14584 - Grey-breasted Mountain Toucan - Andigena hypoglauca.npy
Shape: (4, 1024)
Rows:
 [[1.00081    0.05319615 0.35048535 ... 0.82700044 0.6854153  0.01185115]
 [0.85154    0.08075234 0.6724665  ... 1.2943727  1.0188434  0.04159411]
 [0.14932956 0.05190347 0.4674158  ... 0.9484196  1.5497198  0.0411665 ]
 [0.5318307  0.         0.5636771  ... 1.0350096  1.2452976  0.00568366]]
```

[illegible]

V. Evaluation of Models for Transfer Learning

5.1 Data Pre-Processing for Model Evaluation

```
# --- Data preprocessing for model evaluation ---

# Load embeddings and labels
X = []
y = []
file_ids = []

folders = {
    0: "/content/drive/My Drive/AlternativeBirds_Emb/",
    1: "/content/drive/My Drive/OnlyAudioDoliornis_Emb/",
    2: "/content/drive/My Drive/OnlyAudioHapalopsittaca_Emb/"
}

for label, folder_path in folders.items():
    for file in os.listdir(folder_path):
        if file.endswith('.npy'):
            filepath = os.path.join(folder_path, file)
            data = np.load(filepath)
            for row in data:
                X.append(row)
                y.append(label)
                file_ids.append(file)  # Track filename

X = np.array(X)
y = np.array(y)
file_ids = np.array(file_ids)

X_all_data = X.copy()
y_all_data = y.copy()

# Split based on unique files to avoid data leakage.
# Reason: Embeddings from the same audio file are correlated.
# Including some for training and others for testing can inflate performance.

unique_files = np.unique(file_ids)
file_labels = [y[file_ids == f][0] for f in unique_files]

# Train and test split
train_files, test_files = train_test_split(
    unique_files, test_size=0.2, stratify=file_labels, random_state=42
)

train_mask = np.isin(file_ids, train_files)
test_mask = np.isin(file_ids, test_files)

X_train = X[train_mask]
```

```
X_train = X[train_mask]
y_train = y[train_mask]
X_test = X[test_mask]
y_test = y[test_mask]

# Normalize embeddings using StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Check results
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (249, 1024)
y_train shape: (249,)
X_test shape: (81, 1024)
y_test shape: (81,)
```

5.2 Evaluation of Multinomial Logistic Regression (Standard)

```
# --- Evaluation of multinomial logistic regression for prediction of rare species-

# Train the model
clf = LogisticRegression(solver='lbfgs', max_iter=500) #multinomial
clf.fit(X_train, y_train)

# Evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred))

acc_mnl = round(accuracy_score(y_test, y_pred),4)
print("\nAccuracy:", acc_mnl)
```

```
                precision    recall  f1-score   support

             0       1.00      0.84      0.91        25
             1       0.78      1.00      0.88        42
             2       0.67      0.29      0.40        14

      accuracy                           0.83        81
     macro avg       0.81      0.71      0.73        81
  weighted avg       0.83      0.83      0.80        81


Accuracy: 0.8272
```

5.3 Evaluation of Multinomial Logistic Regression with Cross Validation

```
# --- Evaluation of multinomial logistic regression with CV for prediction of rare

# # Cross-validation setup
X = []
y = []
groups = []  # file-level grouping

for label, folder_path in folders.items():
    for file in os.listdir(folder_path):
        if file.endswith('.npy'):
            filepath = os.path.join(folder_path, file)
            data = np.load(filepath)
            for row in data:
                X.append(row)
                y.append(label)
                groups.append(file)  # group by filename

X = np.array(X)
y = np.array(y)
groups = np.array(groups)

cv = StratifiedGroupKFold(n_splits=5)

# Evaluate model with cross validation
accuracies = []
fold = 1
for train_idx, test_idx in cv.split(X, y, groups):
    print(f"\n=== Fold {fold} ===")

    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    clf = LogisticRegression(solver='lbfgs', max_iter=500) #multinomial
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    print(classification_report(y_test, y_pred))
    fold += 1

# Print average accuracy
avg_acc_mnl = round(np.mean(accuracies), 4)
print("Average Accuracy:", avg_acc_mnl)
```

```
=== Fold 1 ===
              precision    recall  f1-score   support

           0       0.80      0.21      0.33        19
           1       0.68      1.00      0.81        40
           2       1.00      0.64      0.78        14
```

```
    accuracy                           0.73        73
   macro avg       0.83      0.62      0.64        73
weighted avg       0.77      0.73      0.68        73


=== Fold 2 ===
              precision    recall  f1-score   support

           0       0.94      1.00      0.97        17
           1       1.00      1.00      1.00        40
           2       1.00      0.93      0.96        14

    accuracy                           0.99        71
   macro avg       0.98      0.98      0.98        71
weighted avg       0.99      0.99      0.99        71


=== Fold 3 ===
              precision    recall  f1-score   support

           0       0.78      1.00      0.88        29
           1       1.00      0.83      0.91        24
           2       0.90      0.64      0.75        14

    accuracy                           0.87        67
   macro avg       0.89      0.83      0.85        67
weighted avg       0.89      0.87      0.86        67


=== Fold 4 ===
              precision    recall  f1-score   support

           0       1.00      0.85      0.92        20
           1       0.69      1.00      0.82        25
           2       0.67      0.29      0.40        14

    accuracy                           0.78        59
   macro avg       0.79      0.71      0.71        59
weighted avg       0.79      0.78      0.75        59


=== Fold 5 ===
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        17
           1       0.96      1.00      0.98        25
           2       0.94      0.94      0.94        18

    accuracy                           0.97        60
   macro avg       0.97      0.96      0.96        60
```

5.4 Evaluation of KNN model

```
# --- Evaluation of KNN for prediction of rare species---

# Train the model (K=3)
```

```
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)

# Evaluate
y_pred_knn = knn.predict(X_test)
print(classification_report(y_test, y_pred_knn))

acc_knn = round(accuracy_score(y_test, y_pred),4)
print("\nAccuracy:", acc_knn)
```

```
                  precision    recall  f1-score   support

               0       0.93      0.76      0.84        17
               1       0.58      1.00      0.74        25
               2       0.67      0.11      0.19        18

        accuracy                           0.67        60
       macro avg       0.73      0.63      0.59        60
    weighted avg       0.71      0.67      0.60        60


    Accuracy: 0.9667
```

5.4 Evaluation of KNN model with Cross Validation

```
# --- Evaluation of KNN with CV for prediction of rare species---

# Set up Stratified Group K-Fold CV
cv = StratifiedGroupKFold(n_splits=5)

accuracies = []
fold = 1
for train_idx, test_idx in cv.split(X, y, groups):
    print(f"\n=== Fold {fold} ===")

    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    # Train and evaluate KNN
    knn = KNeighborsClassifier(n_neighbors=3)
    knn.fit(X_train, y_train)
    y_pred_knn = knn.predict(X_test)

    acc = accuracy_score(y_test, y_pred_knn)
    accuracies.append(acc)

    print(classification_report(y_test, y_pred_knn, zero_division=0))
    fold += 1

# Print average accuracy
avg_acc_knn = round(np.mean(accuracies), 4)
print("Average Accuracy:", avg_acc_knn)
```

```
=== Fold 1 ===
              precision    recall  f1-score   support

           0       1.00      0.05      0.10        19
           1       0.62      1.00      0.77        40
           2       1.00      0.57      0.73        14

    accuracy                           0.67        73
   macro avg       0.88      0.54      0.53        73
weighted avg       0.79      0.67      0.59        73


=== Fold 2 ===
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        17
           1       0.91      1.00      0.95        40
           2       1.00      0.79      0.88        14

    accuracy                           0.94        71
   macro avg       0.97      0.91      0.93        71
weighted avg       0.95      0.94      0.94        71


=== Fold 3 ===
              precision    recall  f1-score   support

           0       0.93      0.97      0.95        29
           1       0.70      0.96      0.81        24
           2       1.00      0.29      0.44        14

    accuracy                           0.82        67
   macro avg       0.88      0.74      0.73        67
weighted avg       0.86      0.82      0.79        67


=== Fold 4 ===
              precision    recall  f1-score   support

           0       1.00      0.55      0.71        20
           1       0.57      1.00      0.72        25
           2       1.00      0.29      0.44        14

    accuracy                           0.68        59
   macro avg       0.86      0.61      0.63        59
weighted avg       0.82      0.68      0.65        59


=== Fold 5 ===
              precision    recall  f1-score   support

           0       0.93      0.76      0.84        17
           1       0.58      1.00      0.74        25
           2       0.67      0.11      0.19        18

    accuracy                           0.67        60
```

```
        macro avg       0.73      0.63      0.59        60
```

5.5 Evaluation of Few Shot Learning

```
# --- Evaluation of few-shot learning for prediction of rare species---
# The simpliest version using nearest neighbor with cosine similarity.

def few_shot_predict(X_train, y_train, X_query, k=3):
    similarities = cosine_similarity(X_query, X_train)
    predictions = []
    for sim_row in similarities:
        top_k_indices = sim_row.argsort()[-k:][::-1]
        top_k_labels = y_train[top_k_indices]
        predicted_label = Counter(top_k_labels).most_common(1)[0][0]
        predictions.append(predicted_label)
    return predictions

y_pred_fewshot = few_shot_predict(X_train, y_train, X_test, k=3)
print(classification_report(y_test, y_pred_fewshot))

acc_fewshot = round(accuracy_score(y_test, y_pred),4)
print("\nAccuracy:", acc_fewshot)
```

```
                  precision    recall  f1-score   support

               0       0.94      0.94      0.94        17
               1       0.83      0.96      0.89        25
               2       0.93      0.72      0.81        18

        accuracy                           0.88        60
       macro avg       0.90      0.87      0.88        60
    weighted avg       0.89      0.88      0.88        60


    Accuracy: 0.9667
```

5.6 Evaluation of Few Shot Learning with Cross Validation

```
# --- Evaluation of few-shot learning with CV for prediction of rare species---

cv = StratifiedGroupKFold(n_splits=5)
accuracies = []
fold = 1
for train_idx, test_idx in cv.split(X, y, groups):
    print(f"\n=== Fold {fold} ===")
    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]

    y_pred = few_shot_predict(X_train, y_train, X_test, k=3)
    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)
```

```
    print(classification_report(y_test, y_pred, zero_division=0))
    fold += 1


# Print average accuracy
avg_acc_fewshot = round(np.mean(accuracies), 4)
print("Average Accuracy:", avg_acc_fewshot)
```

```
=== Fold 1 ===
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        19
           1       0.63      1.00      0.78        40
           2       1.00      0.71      0.83        14

    accuracy                           0.68        73
   macro avg       0.54      0.57      0.54        73
weighted avg       0.54      0.68      0.59        73


=== Fold 2 ===
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.98      1.00      0.99        40
           2       1.00      0.93      0.96        14

    accuracy                           0.99        71
   macro avg       0.99      0.98      0.98        71
weighted avg       0.99      0.99      0.99        71


=== Fold 3 ===
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        29
           1       0.86      1.00      0.92        24
           2       1.00      0.71      0.83        14

    accuracy                           0.93        67
   macro avg       0.94      0.89      0.91        67
weighted avg       0.93      0.93      0.92        67


=== Fold 4 ===
              precision    recall  f1-score   support

           0       1.00      0.90      0.95        20
           1       0.68      1.00      0.81        25
           2       1.00      0.29      0.44        14

    accuracy                           0.80        59
   macro avg       0.89      0.73      0.73        59
weighted avg       0.86      0.80      0.77        59
```

```
=== Fold 5 ===
              precision    recall  f1-score   support

           0       0.94      0.94      0.94        17
           1       0.83      0.96      0.89        25
           2       0.93      0.72      0.81        18

    accuracy                           0.88        60
   macro avg       0.90      0.87      0.88        60
```

5.7 Evaluation of Neural Network (MLP)

```
# --- Evaluation of MLP (Multilayer Perceptron) for prediction of rare species---
# A type of feedforward neural network suitable for small samples

# Train the model
clf = MLPClassifier(hidden_layer_sizes=(256, 128), max_iter=300, random_state=42)
clf.fit(X_train, y_train)

# Evaluate
print(classification_report(y_test, clf.predict(X_test)))

acc_mlp = round(accuracy_score(y_test, y_pred),4)
print("\nAccuracy:", acc_mlp)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.83      1.00      0.91        25
           2       1.00      0.72      0.84        18

    accuracy                           0.92        60
   macro avg       0.94      0.91      0.92        60
weighted avg       0.93      0.92      0.91        60


Accuracy: 0.8833
```

5.8 Evaluation of Neural Network (MLP) with Cross Validation

```
# --- Evaluation of MLP (Multilayer Perceptron) with CV for prediction of rare spec
import pandas as pd

fold = 1
accuracies = []
for train_idx, test_idx in cv.split(X, y, groups):
    print(f"\n=== Fold {fold} ===")

    X_train, X_test = X[train_idx], X[test_idx]
    y_train, y_test = y[train_idx], y[test_idx]
```

```
    clf = MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=300, random_state=42
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    accuracies.append(acc)

    print(pd.DataFrame(classification_report(y_test, y_pred, output_dict=True)).tra
    fold += 1

# Print average accuracy
avg_acc_mlp = round(np.mean(accuracies), 4)
print("\nAverage Accuracy:", avg_acc_mlp)
```

```
=== Fold 1 ===
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDrive2ImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _PyDriveImportHook.find_spec()
<frozen importlib._bootstrap>:1047: ImportWarning: _BokehImportHook.find_spec() no
              precision    recall  f1-score    support
0              0.833333  0.263158  0.400000  19.000000
1              0.714286  1.000000  0.833333  40.000000
2              0.909091  0.714286  0.800000  14.000000
accuracy       0.753425  0.753425  0.753425   0.753425
macro avg      0.818903  0.659148  0.677778  73.000000
weighted avg   0.782631  0.753425  0.714155  73.000000

=== Fold 2 ===
              precision    recall  f1-score    support
0              0.944444  1.000000  0.971429  17.000000
1              1.000000  0.975000  0.987342  40.000000
2              0.928571  0.928571  0.928571  14.000000
accuracy       0.971831  0.971831  0.971831   0.971831
macro avg      0.957672  0.967857  0.962447  71.000000
weighted avg   0.972613  0.971831  0.971943  71.000000

=== Fold 3 ===
              precision    recall  f1-score    support
0              0.805556  1.000000  0.892308  29.000000
1              1.000000  0.875000  0.933333  24.000000
2              0.900000  0.642857  0.750000  14.000000
accuracy       0.880597  0.880597  0.880597   0.880597
macro avg      0.901852  0.839286  0.858547  67.000000
weighted avg   0.894942  0.880597  0.877268  67.000000

=== Fold 4 ===
              precision    recall  f1-score    support
0              1.000000  0.800000  0.888889  20.000000
1              0.714286  1.000000  0.833333  25.000000
2              0.500000  0.285714  0.363636  14.000000
accuracy       0.762712  0.762712  0.762712   0.762712
macro avg      0.738095  0.695238  0.695286  59.000000
weighted avg   0.760291  0.762712  0.740712  59.000000

=== Fold 5 ===
              precision    recall  f1-score    support
0              0.944444  1.000000  0.971429  17.000000
```

```
1              1.000000  0.960000  0.979592  25.000000
2              1.000000  1.000000  1.000000  18.000000
accuracy       0.983333  0.983333  0.983333   0.983333
macro avg      0.981481  0.986667  0.983673  60.000000
weighted avg   0.984259  0.983333  0.983401  60.000000

Average Accuracy: 0.8704
```

VI. Summary of Model Evaluation Results

```
# Summary of accuracy results
results_table = pd.DataFrame({
    "Model (with Cross Validation)": ["Multinomial Logistic", "K-Nearest Neighbors"
    "Avg Accuracy": [avg_acc_mnl, avg_acc_knn, avg_acc_fewshot, avg_acc_mlp]
})

# Print results
print(results_table)
```

```
  Model (with Cross Validation)  Avg Accuracy
0          Multinomial Logistic        0.8648
1           K-Nearest Neighbors        0.7561
2             Few-Shot Learning        0.8552
3         Multilayer Perceptron        0.8704
```

VII. Selection of the Best Model Using Transfer Learning for Rare Species Prediction

Based on cross-validated performance using BirdNET embeddings, the Multilayer Perceptron (MLP) achieved the highest average accuracy and is selected as the most reliable model for identifying our rare bird species.

```
# Train the final model on all data
final_mlp = MLPClassifier(hidden_layer_sizes=(128, 64), max_iter=300, random_state=
final_mlp.fit(X_all_data, y_all_data)

# Save the model
joblib.dump(final_mlp, "mlp_model_birdnet.pkl")
print("MLP model saved as 'mlp_model_birdnet.pkl'")
```

```
MLP model saved as 'mlp_model_birdnet.pkl'
```

VIII. Example: Applying the Transfer Learning Model for Rare Bird Species Identification

```
# Prepare the recording
input_file = "XC20615 - Bay-vented Cotinga - Doliornis sclateri.mp3"
output_file = "XC20615 - Bay-vented Cotinga - Doliornis sclateri.wav"

# Pre-process the recording
convert_and_pad_to_3s(input_file, output_file)
```

```
Processed and saved: XC20615 - Bay-vented Cotinga - Doliornis sclateri.wav
<ipython-input-2-3223057341c3>:31: ResourceWarning: unclosed file <_io.BufferedRan
  audio.export(output_path, format="wav")
```

```
# Get embeddings from BirdNET
model = bmz.BirdNET()
embedding1 = model.embed(output_file)
X_input = embedding1.reset_index(drop=True).values
print("\nThe embeddings are:\n", X_input)
```

```
File BirdNET_GLOBAL_6K_V2.4_Labels_af.txt already exists; skipping download.
downloading model from URL...
File BirdNET_GLOBAL_6K_V2.4_Model_FP16.tflite already exists; skipping download.
/usr/local/lib/python3.11/dist-packages/opensoundscape/ml/cnn.py:599: UserWarning:
                    This architecture is not listed in opensoundscape.ml.cnn_archi
                    It will not be available for loading after saving the model wi
                    To make it re-loadable, define a function that generates the a
                    then use opensoundscape.ml.cnn_architectures.register_architec

                    The function can also set the returned object's .constructor_n
                    to avoid this warning and ensure it is reloaded correctly by o

                    See opensoundscape.ml.cnn_architectures module for examples of

  warnings.warn(
/usr/local/lib/python3.11/dist-packages/opensoundscape/ml/cnn.py:623: UserWarning:
  warnings.warn(

The embeddings are:
 [[0.03338824 0.8628469  0.5336458  ... 0.609214   0.07550486 0.37615904]
 [1.0899563  0.1301161  0.08624835 ... 0.32889202 0.61821526 0.38222626]
 [0.8048906  0.5017663  0.         ... 0.16950338 0.43497276 0.18075454]
 ...
 [0.         0.9102369  0.27659604 ... 1.2303518  0.17949228 0.        ]
 [0.18699229 0.32541603 0.18230295 ... 1.7507849  0.         0.01350991]
 [1.0755877  0.4611834  0.09565047 ... 0.15522702 0.07813891 0.19568026]]
```

```
# Load the saved model for prediction
model_transferlearning = joblib.load("mlp_model_birdnet.pkl")
y_pred = model_transferlearning.predict(X_input)

# Mapping of label to species name
label_to_species = {
    0: "No species identified",
    1: "Doliornis sclateri"
```

```
    1: "Doliornis sclateri",
    2: "Hapalopsittaca melanotis"
}

# Convert predictions to species names
species_pred = [label_to_species[label] for label in y_pred]

# Print or use the result
print("Predictions per audio segment:", species_pred)

# Most frequent prediction
most_common = Counter(species_pred).most_common(1)[0][0]
print("\nFinal predicted species:", most_common)
```

```
Predictions per audio segment: ['Doliornis sclateri', 'Doliornis sclateri', 'Dolio

Final predicted species: Doliornis sclateri
```