

## **Deep Learning Kaggle BirdCLEF Final Project**

**Names : Seth Born & Precious Obaseki**

### **TASK AND APPROACH**

The goal of this project is to develop a machine learning model capable of accurately identifying bird species from audio recordings in the Western Ghats, a biodiversity hotspot. The dataset consists of audio recordings labeled with various bird species. Each audio clip is processed to extract meaningful features that are then fed into a machine-learning model. A successful model would be able to identify endemic bird species of the sky-islands of the Western Ghats using soundscape data, detect and classify endangered bird species, especially those for species which there is limited training data available, and detect and classify nocturnal bird species, which are often less studied and understood.

The approach of involves preprocessing the audio data to enhance quality and consistency, implement feature extraction using techniques like Mel-frequency cepstral coefficients (MFCCs) to capture the unique aspects of each bird call, and train different machine learning models to classify the bird species based on these features.

### **CODE WORKFLOW DESCRIPTION**

The project code is to identify the call of different bird species. First, Python libraries are installed including torch, transformers, datasets, librosa, and soundfile, which are used for handling machine learning models, audio processing, and data manipulation. We use OpenAI's Whisper model, which is designed for audio processing. We then download the BIRDCLEF data from Kaggle. For data preparation we start a preprocessing function to load audio files. The audio data is then resampled to a consistent rate and formatted to suit the input requirements of our models. Next, a custom BirdCLEFDataset class is defined using pytorch's data loader. This class is designed to enhance batch processing efficiency during model training. Our models, called AudioCNN2D(1-5) are coded to interpret 2D audio spectrograms. The network comprises a sequence of convolutional layers paired with max-pooling and dense layers. Training and validation functions are then coded for training. The model iterates over data batches to optimize its parameters, concurrently validating its predictive prowess against an unseen subset of the data.

## **MODEL OVERVIEW**

Our project explored various CNN architectures, each designed with different complexities and feature extraction capabilities, to classify bird species from audio spectrograms efficiently. The models varied in layer depths, filter sizes, and dropout rates to understand their impact on learning and generalization.

## **TRAINING RESULT**

### **Model 1: AudioCNN2D1**

- Architecture: This model includes three convolutional layers with increasing filter sizes (16, 32, 64) and max-pooling layers, followed by two fully connected layers with a dropout of 0.5 to combat overfitting.
- Training Dynamics: Started with a training loss of 4.2528 and validation loss of 3.5149, showing steady progress in fitting the training data while the validation accuracy showed initial improvements.
- Performance: The model achieved a training accuracy of 97.73% by the 50th epoch. However, validation accuracy was around 46.54%, indicating potential overfitting despite high training performance.

### **Model 2: AudioCNN2D2**

- Architecture Adjustments: A simpler structure with reduced convolutional filters (8, 16, 32) aimed at reducing model complexity to improve generalization.
- Training and Validation: Showed a less aggressive learning curve with the final training accuracy reaching 43.85% and validation accuracy at approximately 37.35%.

### **Model 3: AudioCNN2D3**

- Architecture Adjustments: Further adjusted dropout rates to 0.3 in an attempt to provide more regularity and less loss of important features during training.
- Performance: Had final training accuracy at 92.13% and validation accuracy increasing to around 38.89%, showing some robustness.

#### **Model 4: AudioCNN2D4**

- Complexity: Added an additional convolutional layer and pool (128 filters), increasing the model's ability to learn more complex features but also increasing the risk of overfitting.
- Training Results: Initiated training with high loss but managed to lower it significantly, ending with a training accuracy of 59.32% and a validation accuracy that stayed around 37.72%, suggesting some capacity to generalize despite its complexity.

#### **Model 5: AudioCNN2D5**

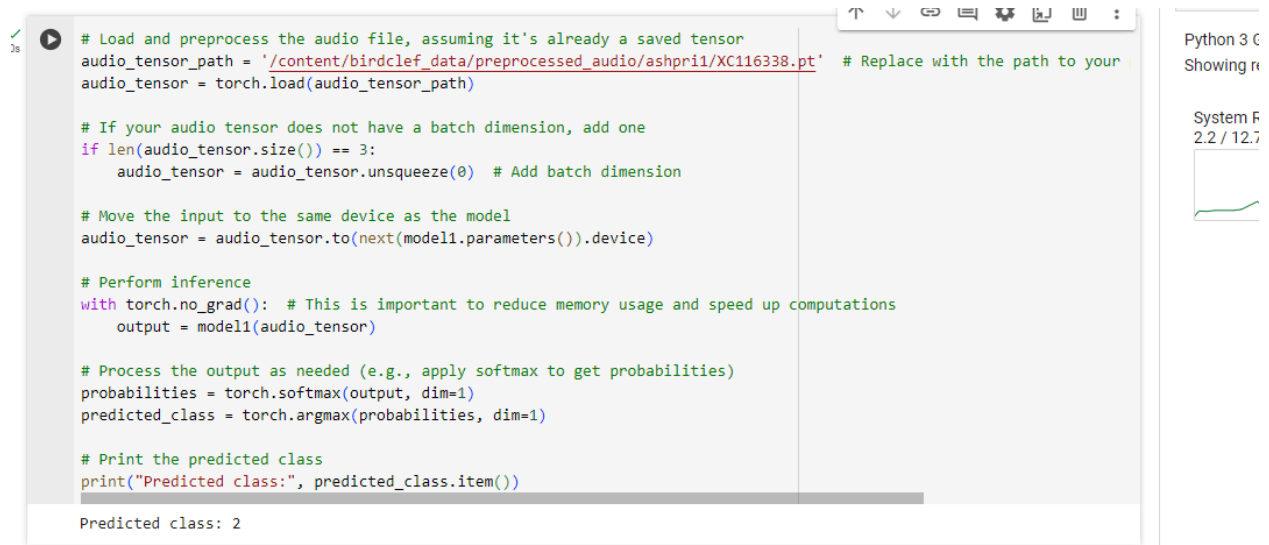
- Architecture: Included ReLU activations and a unique pooling strategy to experiment with different non-linearities and subsampling techniques.
- Outcomes: This model demonstrated a different learning pattern with somewhat lower final accuracies (55.86% training and 36.43% validation), indicating that while the model learned adequately, it might not have captured all necessary patterns effectively as the others.

#### **COMPARATIVE ANALYSIS**

- General Trends: Higher complexity in models did not necessarily translate to better validation performance, indicating a trade-off between model capacity and overfitting.
- Given its learning capability, Model 1 stands out as the best candidate for further development and optimization. Future efforts should focus on enhancing its generalization to unseen data while preserving its strong pattern recognition capacity. This could involve experimenting with less aggressive pooling, introducing batch normalization, or employing different dropout configurations.

## TESTING

The last thing we did was test a few bird calls on the model 1 and attached below is an image of a correct prediction.



```
# Load and preprocess the audio file, assuming it's already a saved tensor
audio_tensor_path = '/content/birdclef_data/preprocessed_audio/ashpri1/XC116338.pt' # Replace with the path to your
audio_tensor = torch.load(audio_tensor_path)

# If your audio tensor does not have a batch dimension, add one
if len(audio_tensor.size()) == 3:
    audio_tensor = audio_tensor.unsqueeze(0) # Add batch dimension

# Move the input to the same device as the model
audio_tensor = audio_tensor.to(next(model1.parameters()).device)

# Perform inference
with torch.no_grad(): # This is important to reduce memory usage and speed up computations
    output = model1(audio_tensor)

# Process the output as needed (e.g., apply softmax to get probabilities)
probabilities = torch.softmax(output, dim=1)
predicted_class = torch.argmax(probabilities, dim=1)

# Print the predicted class
print("Predicted class:", predicted_class.item())
```

Predicted class: 2

Python 3.6.5  
Showing 1/1 cells  
System F  
2.2 / 12.7

## CHALLENGES

A key challenge encountered across our model was the balance between model complexity and the risk of overfitting. Increasing the complexities of the models did not always improve the validation performance. Despite high training accuracies all the models struggled with generalization. Another challenge was the limited GPU and runtime capabilities which restricted the ability to experiment with various code as one model could take hours to run.