Optimising the Engine Model for Frequency Band Removal

Jessica Stinson
224576666

Introduction

Founded in 2022, Project Echo is dedicated to developing a sound classification model to facilitate the monitoring of endangered species in the Otways, Victoria. The current model can classify 118 species within the area, with ongoing efforts to expand the database to include all regional species. In the long term, the project aims to broaden the model's capabilities to recognise a diverse range of animal vocalisations and environmental sounds. Project Echo provides a non-intrusive approach to tracking both endangered species and their predators to support the work of animal conservationists. Although the model demonstrates high accuracy in classifying animal sounds under optimal conditions, its performance is notably reduced when processing audio files with missing frequency bands. This analysis seeks to evaluate the current model pipeline and identify potential areas for improvement in the training process.

Assessment of the Current Training Pipeline

The current engine model classifies audio files with extremely high accuracy (>99%), however, previous testing showed a significant drop in accuracy when frequency masking was applied to test audio files. Upon reviewing the model pipeline, it was discovered that although frequency masking was included in the training pipeline, both the minimum and maximum frequency values were set to zero, so the model was not applying the augmentation during training (see figure 1, below). Additionally, the frequency masking implementation applied a fixed range to all audio files, limiting its effectiveness in simulating real-world variability.

```
# Create melspectrograms
min_freq_remove = 0
max_freq_remove = 0

        # Apply frequency masking
        if min_freq_remove is not None and max_freq_remove is not None:
            mask = (mel_freqs >= min_freq_remove) & (mel_freqs <= max_freq_remove)
            image[mask, :] = 0
```

Figure 1. Frequency masking as defined in the original model pipeline

To ensure that frequency masking is effectively applied during training, the function responsible for this augmentation was modified. The minimum and maximum frequency values are no longer hardcoded in the pipeline. As a result, if the model is built without specifying these parameters, it will automatically apply frequency masking using randomly selected frequency bands. This provides flexibility, allowing users to optionally define specific frequency ranges for future applications. By default, when no minimum and maximum values are provided, each audio file has a 25% chance of being selected for frequency masking, with a random 1,000 Hz band applied to the corresponding mel spectrogram. The updated frequency masking method is shown in Figure 2 below.

```
# Apply random frequency masking
if min_freq is not None and max_freq is not None:
    mask = (mel_freqs >= min_freq) & (mel_freqs < max_freq)
    image[mask, :] = 0
elif random.random() < 0.25:
    start_freq_options = range(0, 12001, 1000)  # Possible starting frequencies (0, 1000, 2000, ..., 12000)
    if start_freq_options:
        min_freq = random.choice(start_freq_options)
        max_freq = min_freq + 1000
        mask = (mel_freqs >= min_freq) & (mel_freqs < max_freq)
        image[mask, :] = 0
image = np.moveaxis(image, 1, 0)
sample = image[0:expected_clip_samples,:]
```

Figure 2. Updated training pipeline to apply frequency masking

In addition to modifying the frequency masking, several changes were implemented to improve the robustness and repeatability of the pipeline. The original pipeline split the complete dataset into three subsets: training, testing, and validation. While this approach ensures that the model is evaluated on unseen data, it lacked a mechanism to save these splits for future reuse. As a result, future users were required to re-split the dataset or test on the entire dataset, potentially introducing bias into the observed accuracy during testing. To address this, the create_datasets function was updated to first check for an existing split file. If a split file is found, the predefined splits are loaded; otherwise, the function creates new dataset splits and saves them for future use and distribution (see Figure 3 below).

A similar improvement was applied to the model build process to avoid unnecessary reinitialization. The updated pipeline checks whether a model path already exists before creating a new build, allowing the notebook to be run multiple times without overwriting existing models.

```python
# Function to create the datasets
def create_datasets(audio_dir, train_split=0.8, val_split=0.19, split_file = None, save_splits_to = None):
    # Check for a split file and load dataset splits if available
    if split_file:
        print("Loading dataset splits from saved file")
        with open(split_file, "r") as f:
            data = json.load(f)
        file_paths, labels = zip(*data["all_data"])
        file_paths = list(file_paths)
        labels = list(labels)
        class_names = data["class_names"]

    else:
        # create new dataset splits if a split file is not available
        print("Creating new dataset splits")
        file_paths, labels, class_names = index_directory(audio_dir)

        all_data = list(zip(file_paths, labels))

        random.seed(42)
        random.shuffle(all_data)

        file_paths, labels = zip(*all_data)
        file_paths = list(file_paths)
        labels = list(labels)

        # Save splits for future use and distribution
        if save_splits_to:
            os.makedirs(os.path.dirname(save_splits_to), exist_ok=True)
            with open(save_splits_to, "w") as f:
                json.dump({
                    "all_data": list(zip(file_paths, labels)),
                    "class_names": class_names
                }, f, indent=2)
```

Figure 3. First section of the create_datasets function to check for a split file and load datasets

After implementing these updates, the revised pipeline was used to create and save a new model along with new dataset splits, which are available for further testing. The model was first evaluated using the test dataset, consisting of 76 randomly selected audio files, with no frequency masking applied. Following this, a random sample of 25 test audio files was extracted for additional evaluation involving frequency masking. For this test, each audio file was masked using a 1,000 Hz frequency band applied at varying frequencies ranging from 0 to 14,000 Hz. The model then classified the resulting masked spectrograms, and the prediction accuracy was calculated for each frequency band. A plot was generated to visually illustrate the impact of different masked frequency ranges on model accuracy.

Discussion

Testing of the updated model and training pipeline revealed that the classification system is now significantly more resilient to accuracy loss caused by frequency masking. When evaluated on the original, unmasked test dataset, the model achieved excellent performance, with an accuracy of 100% for the given data splits. When frequency masking was applied to a subset of the test data, the model still demonstrated strong performance, maintaining an accuracy above 95% for masked frequency bands ranging from 1,000 Hz to 14,000 Hz (see Figure 4 below).

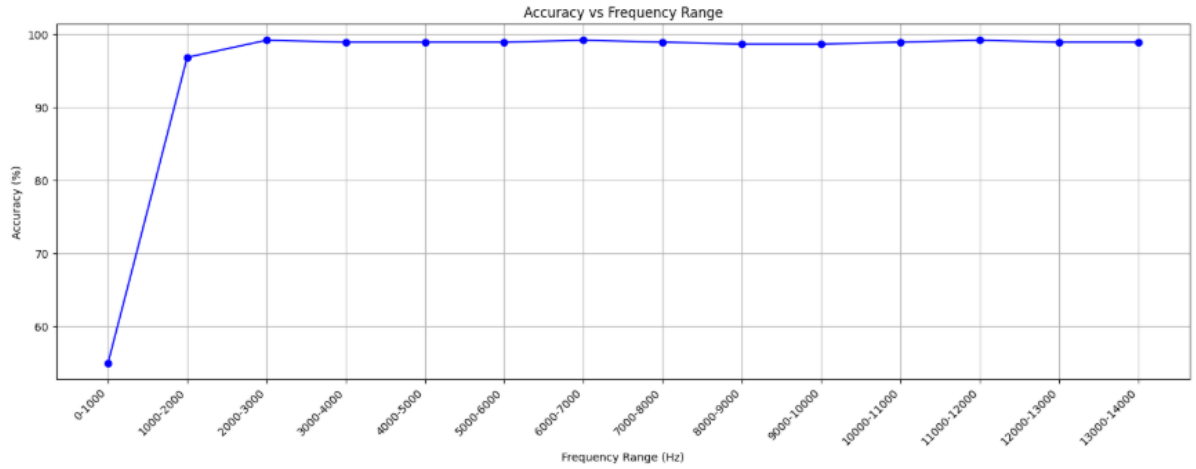

Figure 4. The model accuracy for each range of frequency masking

It is important to note that the accuracy of the model declined significantly when frequency masking was applied within a range of 0 to 1,000 Hz. This range yielded an accuracy of just 54.95%, suggesting that the model may need more training with data in this frequency range. The current dataset primarily contains mid to high range frequency audio from bird vocalisations with minimal low-frequency representation. To enable the model to generalise more effectively across the full audio spectrum, particularly when frequency masking is applied, it must be exposed to a broader range of audio during training. Without adequate representation of low-frequency audio, the model may not learn to recognise or compensate for missing information in that range.

The long-term goal of Project Echo is to deploy this model as a conservation tool capable of classifying a wide range of animal audio recordings. Currently, the available dataset does not fully reflect the diversity of audio signals the model is expected to encounter in real-world applications. Continued efforts by the Project Echo team to source additional training and testing samples will be critical to improving model performance and ensuring broad applicability. In particular, it is recommended that future data collection focuses on increasing the representation of low-frequency audio, including bird territorial and mating calls as well as mammalian vocalizations, to improve the model's adaptability and overall robustness.

Conclusion

The changes made to the model training pipeline have significantly improved the robustness and repeatability of the audio classification model. The model continues to perform with high accuracy on clean, unmasked data and has demonstrated increased resilience when frequency masking is applied during testing. This analysis has identified a key limitation of the current model due to insufficient exposure to low-frequency audio during training. To ensure the long-term success and reliability of the model in practical applications, it is essential that efforts continue to expand the diversity of the training dataset, particularly by incorporating more low-frequency audio samples. Despite this limitation, the Project Echo classification model shows strong potential for its intended role as a conservation tool and remains a promising foundation for future development.