Bird Sound Classification Using MEL Spectrograms

Anurag Surve

Supervised by Amir Jafari

Spring 2025

1. Introduction

Growing up in a small village in western Maharashtra, I often woke to the calls of Indian Peafowl (Morni) near our home. Distinguishing each sound became difficult when other ambient sounds like wind, distant machinery, or chatter—masked the peafowl's song. That was the reason I suggested this project to my groupmates i.e BirdCLEF 2025 dataset: converting its audio clips into Mel-spectrograms, feeding them through an EfficientNet-B0 model, and wrapping everything in a lightweight Streamlit app for instant predictions. In doing so, I learned firsthand how spectrograms reveal time–frequency features, how augmentations like Mixup and

SpecAugment improve robustness to noisy recordings, and how quickly a functional demo can be deployed using Streamlit.

2. Description of the Individual Work

2.1 Streamlit App Development

- **Interactive demo:** Built a Streamlit front end that lets users upload up to 60 s of audio, auto-trim or pad it to a 5 s window, then visualize waveform and Mel-spectrogram.
- **Real-time inference:** Integrated the trained EfficientNet-B0 model (loaded via PyTorch/Timm) to output top-5 species predictions with probabilities.

3. Detailed Description of the Contributions

3.1 Preprocessing Pipeline

```
def preprocess(y):  1 usage
    # trim or pad
    max_len = cfg.FS * MAX_DURATION
    if len(y) > max_len:
        y = y[:max_len]
    elif len(y) < cfg.FS * cfg.WINDOW_SIZE:
        pad = cfg.FS*cfg.WINDOW_SIZE - len(y)
        y = np.pad(y, pad_width: (0,pad), mode='constant')
    spec = to_melspec(y)
    spec = cv2.resize(spec, cfg.TARGET_SHAPE, interpolation=cv2.INTER_LINEAR)
    return spec.astype(np.float32)
```

- Ensured uniform 256×256 input shape.
- Normalized decibel values to [0,1] for stable training.

3.2 Model Implementation & Training

- Wrapped efficientnet_b0 from Timm in a PyTorch nn.Module, replacing its head with a linear layer for 206 species.

- Conducted 5-fold stratified cross-validation (stratified on primary_label and noise_level).
- Used AdamW (lr = 5e-4, weight_decay = 1e-5) with CosineAnnealingLR over 10 epochs per fold.
- Incorporated Mixup (α = 0.5) and SpecAugment (time/frequency masking) in the training pipeline.

4. Results

4.1 Cross-Validation Performance

Fold	ROC-AUC	Final Val Loss
0	0.9451	0.237
1	0.9513	0.221
2	0.9438	0.245
3	0.9475	0.229
4	0.9501	0.232
Mean	**0.9476**	**0.233**

The model consistently achieved ~0.948 macro-AUC, demonstrating balanced performance across common and rare species.

4.2 Demo Prediction

On an external 10 s sample ("sample_01.mp3"):

View all results

	species_id	common_name	class_name	probability
198	yecspi2	Yellow-chinned Spinetail	Aves	0.5257
170	srwswa1	Southern Rough-winged Swallow	Aves	0.4326
192	whwswa1	White-winged Swallow	Aves	0.2308
141	rebbla1	Red-breasted Meadowlark	Aves	0.1558
70	bkcdon	Black-capped Donacobius	Aves	0.1394
116	grnkin	Green Kingfisher	Aves	0.0657
113	grekis	Great Kiskadee	Aves	0.0451
135	piwtyr1	Pied Water-Tyrant	Aves	0.0313
189	whmtyr1	White-headed Marsh Tyrant	Aves	0.0289
166	soulap1	Southern Lapwing	Aves	0.023

- **Top 1:** Yellow-chinned Spinetail (50.0 %)
- **Top 2:** Southern Rough-winged Swallow (42.0 %)

The 8% gap and sharp probability drop after rank 2 indicate strong class separation.

5. Summary and Conclusions

- **Key outcome:** Our pipeline—combining dynamic spectrogram generation, Mixup/SpecAugment, and EfficientNet-B0—achieved a robust 0.9476 Macro-averaged ROC-AUC.
- **What I learned:**
    - **How spectrograms work:** Gained hands-on understanding of STFT, Mel filter banks, decibel scaling, and normalization—observing how time–frequency representations capture bird-call features.
    - **How to build a Streamlit app:** Learned Streamlit's caching, layout, and widget APIs to create an interactive demo handling audio, preprocessing controls, real-time inference, and dynamic plotting.
- **Future work:** Implement sliding-window inference with vote aggregation; explore lightweight attention modules; apply model quantization for on-device deployment.

6. Code Attribution Percentage

- Lines adapted from online examples: ≈300
- Lines I modified: ≈200
- Original lines I wrote: ≈150

$$Attribution\,\% = \frac{300 - 200}{300 + 150} \times 100 \approx 22\%$$

7. References

1. Cornell Lab of Ornithology. (n.d.). *Birds of the World*. Retrieved April 30, 2025, from [URL]
2. Kaggle. (2025). *BirdCLEF 2025 competition data*. Retrieved April 30, 2025, from [URL]
3. Sharma, A. (2024, April 12). *EfficientNetB0 architecture: Stem layer*. Image Processing with Python (Medium). Retrieved April 30, 2025, from [URL]
4. Vidhya, A. (2024, January 15). *Understanding the Mel-spectrogram*. Analytics Vidhya (Medium). Retrieved April 30, 2025, from [URL]