BioDCASE 2025 CHALLENGE TECHNICAL REPORT

Technical Report

Toby Martin*

Zorneding, Germany
toby@lesmartins.net

ABSTRACT

1. INTRODUCTION

This project focuses on detecting the presence of bird vocalisations, specifically those of the yellowhammer (Emberiza citrinella) . The task is a binary classification problem: each audio clip either contains a bird call or it doesn't.
The initial model architecture was based on a MobileNet-style convolutional neural network (CNN). While it achieved reasonably high accuracy and was relatively compact, it was still larger than necessary for this domain specific task and not well suited for deployment on resource constrained devices like embedded microcontrollers or edge AI systems.
The objective of this exercise was to optimise the model to reduce inference memory and computational requirements, and to maintain or improve performance on the validation set.
This document describes the process undertaken to improve the model, focusing on reducing its size and computational load while either preserving or improving accuracy. The rationale behind each modification is explained, the validation methodology presented, and the results of experiments comparing the original and optimised versions are given.

2. PROBLEM CONTEXT AND ORIGINAL MODEL OVERVIEW

Bird vocalisation detection is a binary classification problem where the goal is to evaluate whether a given segment of audio contains a vocalisation of interest. The input to the model is a mel-spectrogram representation of short audio clips.
The original model used a two-layer convolutional structure based on MobileNet primitives:

```
x = _conv_block(inputs, filters=n_filters_1, alpha=1, kernel=(10, 4), strides=(5, 2))
x = _depthwise_conv_block(x, pointwise_conv_filters=n_filters_1, alpha=1, block_id=1)
x = layers.GlobalMaxPooling2D(keepdims=True)(x)
x = layers.Dropout(dropout, name="dropout1")(x)
x = layers.Flatten()(x)
x = layers.Dense(2)(x)
outputs = layers.Softmax()(x)
```

While this architecture was effective, it had a relatively high number of parameters and required unnecessary computation for a task that involved detecting a single class.

3. INPUT DATA AND PREPROCESSING

The model operates on mel-spectrograms derived from raw audio clips. A mel-spectrogram is a time-frequency representation of audio that maps energy across frequency bands on a perceptual scale (the mel scale), better aligning with human and animal hearing. It is a common input format for audio classification tasks, especially when using 2D CNNs.
Based on domain knowledge and literature [1][2], yellowhammer vocalizations primarily occur between 3 kHz and 8 kHz, occasionally extending up to 9.8 kHz. To capture this range while reducing unnecessary high-frequency noise, the following feature extraction parameters are used:

```
feature_extraction:
  window_len: 4096
  window_stride: 512
  window_scaling_bits: 12
  mel_n_channels: 64    # 40
  mel_low_hz: 3000   # 125
  mel_high_hz: 7500 # 7500
  mel_post_scaling_bits: 6
```

4. ORIGINAL & OPTIMISED MODELS

The goal was to make the model smaller and more efficient, while preserving or improving its accuracy. To achieve this the following strategies were employed.
The original architecture followed a MobileNet-style depthwise separable CNN. These networks are known for achieving a good balance between accuracy and efficiency by replacing standard convolutions with depthwise and pointwise convolutions.
The original model used 32 filters in the initial convolution and 64 pointwise filters in the depthwise layer, followed by flattening and a dense classification head. While accurate, this model was relatively large and not well-tuned for binary classification or memory efficiency.
The optimised model makes several architectural changes to improve performance while reducing complexity.

4.1. Reduced Filter Counts

The number of filters was reduced to:

n_filters_1 = 24
n_filters_2 = 32

This cuts down the number of convolutional parameters and intermediate activations, significantly reducing RAM usage and model size.

```
x = _conv_block(inputs, filters=n_filters_1, alpha=1, kernel=(6, 3), strides=(4, 2))
x = _depthwise_conv_block(x, pointwise_conv_filters=n_filters_1, alpha=1, block_id=1)
```

4.2. Simpler Pooling-Based Feature Aggregation

Instead of flattening high-dimensional tensors into a long vector with Flatten(), a global pooling strategy was used.

```
x = layers.GlobalMaxPooling2D(keepdims=True)(x)
x = layers.Dropout(dropout, name="dropout1")(x)
```

Global pooling reduces the output of convolutional layers to a fixed-size representation regardless of the spatial dimensions, and it's computationally efficient.

4.3. Dual-Layer Output Structure with 2 Units

The classification head now includes a fully connected dense layer with 2 units and a Conv2D layer with 2 filters and a (1,1) kernel.

Finally a global average pooling operation to reduce the output to 1D.

```
x = layers.GlobalAveragePooling2D()(x)
outputs = layers.Activation("sigmoid")(x)
```

This structure appears redundant but likely reflects either a requirement to retain feature separation across two channels before final pooling, or a design constraint from downstream tasks, such as supporting multilabel detection even in a binary scenario.
Despite the 2-unit structure, the model remains efficient because these layers are applied to already pooled or low-dimensional tensors.

5. EVALUATION AND RESULTS

The optimized model was evaluated alongside the original using the same dataset, training parameters, and validation set. The key metric for performance comparison was validation average precision (PR AUC), which captures the tradeoff between precision and recall — particularly important in imbalanced classification problems like bird call detection.

Model	Avg Precision	Smoothed	Computation Time
Original	0.958	0.8884	27934 µs
Optimised	0.9858	0.9849	15177 µs

6. CONCLUSION

Through architectural simplification, parameter reduction, and loss-aware design, the efficiency and predictive performance of the bird call detection model was improved. The use of MobileNet-style blocks, global pooling, and sigmoid activation proved effective for this task. These improvements bring the model closer to being deployable on low-power, real-world devices such as the ESP32 while maintaining strong validation metrics.
This exercise also demonstrates that domain informed architectural tuning, even without deep ML specialisation, can yield significant gains when guided by sound principles and evaluation.

7. REFERENCES

[1] https://biodcase.github.io/challenge2025/submission.
[2] http://www.ieee.org/web/publications/rights/copyrightmain.html