EFFICIENT BIRD SOUND DETECTION ON THE BELA EMBEDDED SYSTEM

Alexandru-Marius Solomes and Dan Stowell

Machine Listening Lab, Centre for Digital Music, Queen Mary University of London

ABSTRACT

Monitoring wildlife is an important aspect of conservation initiatives. This paper proposes an automatic detection algorithm for the Bela embedded Linux device for wildlife monitoring. The program is capable of computing on-board detection using convolutional neural networks (CNNs) with an AUC score of 82.5% on the testing set of an international data challenge. This paper details how the model is exported to work on the Bela Mini in C++, with the spectrogram generation and the implementation of the feed-forward network, and evaluates its performance on the Bird Audio Detection challenge 2018 DCASE data.

Index Terms— embedded, bioacoustics, acoustic, detection, deep learning

1. INTRODUCTION

The distribution of birds within a habitat is expected to change in number and density as the impacts of climate change and land use come about in future years [1]. In the UK alone, there are over 150 species of birds requiring monitoring with over 60 needing the highest conservation priority [2]. Many birds are easily detectable by their calls, making acoustic monitoring appropriate. The advent of machine learning for big data gave rise to open-source initiatives that have pushed the state-of-the-art in acoustic detection to new heights. In the field of bird audio detection, data challenges such as those organized by DCASE,[1] have produced state-of-the-art results, with multiple methods attaining a performance of around 88% area under the receiver operating characteristic (ROC) curve (AUC) [3].

Nonetheless, outside-of-the-lab solutions for implementing these models are still behind this trend. Model training, in the lab, is usually performed in high-level programming languages such as Python. When the end-goal for such a model is applying its learned knowledge to the environment around it, machine learning intersects with the Internet of Things (IoT) [4]. For efficiency and speed on a lower spec resource, the model must be transferred to a lower-level language.

In this field, the CMSIS-NN[2] library aims to be the on-the-edge detection standard, providing a C API for Cortex-M processors. Using this library, [5], train a convolutional neural network (CNN) on the CIFAR-10 dataset using the Caffe API, load the learned weights onto a Nucleo-f746zg[3] and evaluate its performance. Due to the memory constraints of the embedded system, they use the partial *im2col* data structure, which leaves a smaller memory footprint. In another paper, Lay et al. present a framework based on the CMSIS-NN aimed for on-the-edge learning [6]. They focus on optimizing floating-point errors and show that their solution, KANJI, can remove deployment challenges.

The literature on embedded systems for the task of wildlife monitoring, however, has not yet adopted this framework. Prior work runs a Python-based model directly, despite the power and computational overhead this implies [7]; or alternatively stores/transmits the data for processing off-site [8]. In translating a model to a low-power system, the constraints mainly lie in implementing a system capable of not only performing inference but also transforming the input signal before the inference (i.e. feature extraction and preprocessing) within the limitations of a low power system. An example of an initiative that comes close is the AudioMoth [4] designed by Open Acoustic Devices. The AudioMoth features powerful algorithms used for frequency-based detection where the input signal is scanned using Goertzel filters [9]. This device works well when separating signal from background noise can be done at the frequency level (bats, insects). When the signal is more complex, however, devices like the AudioMoth suffer from high memory requirements.

This paper implements state-of-the-art bird detection algorithms on the Bela Mini embedded system. The Bela Mini is a device created at Augmented Instruments Laboratory[5], in the Centre for Digital Music at Queen Mary University of London. It features a custom audio processing environment, an audio driver using the Programmable Realtime Unit (PRU), running the Xenomai real-time Linux extensions [10]. This system architecture allows the Bela to prioritize audio

Supported by EPSRC fellowship EP/L020505/1.

[1]`http://dcase.community/challenge2018/task-bird-audio-detection`

[2]CMSIS-NN: `http://www.keil.com/pack/doc/CMSIS_Dev/NN/html/index.html`

[3]`https://www.st.com/en/evaluation-tools/nucleo-f746zg.html`

[4]AudioMoth: `https://www.openacousticdevices.info/audiomoth`

[5]AIL: `http://instrumentslab.org/`

code other system tasks, making ideal for sound applications. Moreover, Bela's 512 MB of RAM; CPU speed of up to 1GHz; and onboard IDE make it highly appropriate for the prototyping and deployment of deep learning algorithms.

The model architecture chosen is inspired by the winning entry for the Bird Audio Detection (BAD) challenge [3] — the 'bulbul' network submitted by Grill and Schlüter [11]. On top of this, we propose our version of the 'bulbul' network, which features fewer learnable parameters and achieves better results on the test set. We implement both models on the Bela Mini, providing a small-footprint C++ library that requires no external code libraries (i.e. aiming for a smaller footprint than CMSIS-NN). Finally, we evaluate their performance on the Bela Mini, with a focus on computation speed and floating-point errors.

2. MODEL ARCHITECTURE

The models we implement in the Bela Mini is based on the model introduced by Grill and Schlüter [11], the winners in the Bird Audio Detection Challenge. The table below illustrates the two very closely-related architectures we evaluated:

Table 1: Network Architectures

Model A		Model B	
Input	$1 \times 1000 \times 80$	Input	$1 \times 998 \times 80$
Convolution (3×3)	$16 \times 998 \times 78$	Convolution (3×3)	$16 \times 996 \times 78$
Max-pooling (3×3)	$16 \times 332 \times 26$	Max-pooling (3×3)	$16 \times 332 \times 26$
Convolution (3×3)	$16 \times 330 \times 24$	Convolution (3×3)	$16 \times 330 \times 26$
Max-pooling (3×3)	$16 \times 110 \times 8$	Max-pooling (3×3)	$16 \times 110 \times 8$
Convolution (3×1)	$16 \times 108 \times 8$	Convolution (3×3)	$16 \times 108 \times 6$
Max-pooling (3×1)	$16 \times 36 \times 8$	Max-pooling (3×1)	$16 \times 36 \times 6$
Convolution (3×1)	$16 \times 36 \times 8$	Convolution (3×3)	$16 \times 34 \times 4$
Max-pooling (3×1)	$16 \times 11 \times 8$	Max-pooling (3×1)	$16 \times 11 \times 4$
Dense	256	Dense	256
Dense	32	Dense	32
Dense	1	Dense	1
Trainable Parameters	373,713	Trainable Parameters	196,561

The input to both models is spectrogram images, normalized to 10 seconds in duration, with 80 triangular filters converted to Mel-scale. We use a window size of 1024 samples, computed at a sample rate of 44.1 kHz, with hop size of 1102 and a stride of 441 frames with a minimum frequency of 0 Hz and a maximum of 22.5 kHz. In model A, the Mel-frequency spectrograms pass through a convolution layer, where the bias and 16 kernels of congruent dimension 3, learn different spatial and temporal dependencies by convolving over the image, with a stride of 2. Next, the resulting output goes through a max-pooling layer, suppressing the noise with a sliding window of congruent dimension 3. These two steps are repeated one more time, by when the initial image shrunk to a size of $16 \times 110 \times 8$. The next two sets of convolution and max-pooling layers have sliding windows of dimension 3×1. Finally, three fully-connected layers are applied to give the prediction. All convolution and dense layers go through a leaky rectifier unit, except for the final layer - using a sigmoid instead. Model B has a similar architecture to model A, the main difference lies in third and fourth convolution layers which execute a 3×3 convolution, halving trainable parameters.

Model training was done using stochastic gradient descent on mini-batches of 64, using the same parameters as Grill and Schlüter [11], namely: ADAM update and 0.001 learning rate.

3. IMPLEMENTATION DETAILS

Transferring the model to the Bela requires a method of reading the weights, learned in Keras, in C++. We specified the following simple text file format:

Listing 1. Extended Backus-Naur representation of model weights in text file format (.txt)

```
num layers
{layer name
    d1
    d2
    d3
    d4
    {weights}
}
```

The start of the file states the number of layers the model contains. Each curly brace denotes a repetition, made up of a name-value, dimensions, and weights. This representation of the model allows a base four-dimensional C++ data structure to store these values inside of a vector container. This vector container is used by the feed-forward program to know which operation to execute at a time and makes use of pointer arithmetic to access each of the layer's input weights.

The Bela API consists of three functions: a setup function, a render function, and a cleanup function. The setup function initializes pointers to the objects used in the detection program, including spectrogram and classifier objects. The render function is the function that is regularly called by Bela's PRU [10]. In the render function, a state machine switches between recording a sufficient duration of audio, performing the deep learning inference and saving the sound signal (if a positive detection occurs). Finally, in the cleanup function, pointers created in setup are deleted.

4. EVALUATION

We evaluate the model in two stages: first on how well they perform detection according to the data sets of the the Bird Audio Detection Challenge, and second on how efficiently they perform on the Bela embedded Linux device. In the first step (Section 4.1), we use the AUC measure - the metric of choice in the challenge. The challenge evaluates submissions on their ability to perform general-purpose detection. The

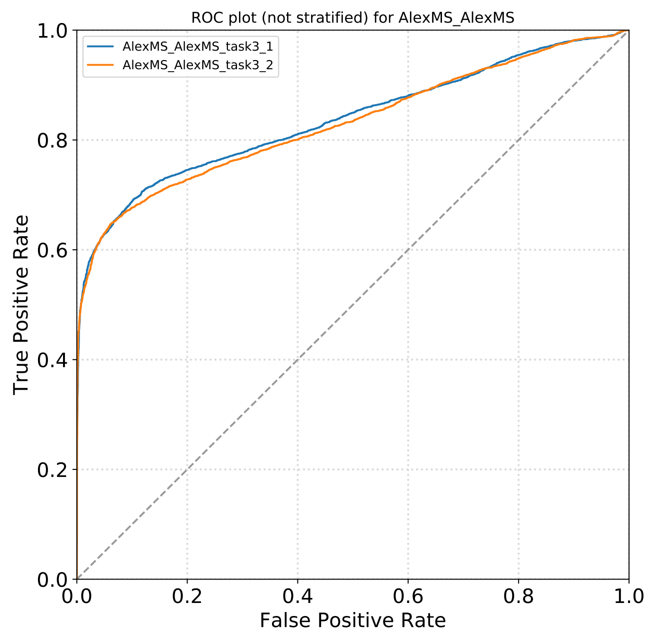

Fig. 1. ROC plots for our C++ implementations of the two systems tested on Bird Audio Challenge data: bulbul (shown as '1'/blue), and our smaller network ('2'/orange). Note that this is calculated across all 3 of the testing datasets aggregated together. The official score (quoted in the text) is a stratified score calculated from each testing set scored separately. Compare Fig 2 in [3].

AUC measure generalises over possible thresholds which the model could be tuned for, which unlike accuracy it remains unaffected by 'unbalanced' test sets [3].

In the second step (Section 4.2), we time how long the device takes to run the prediction algorithm. We do this by measuring the time required to compute the spectrogram and total time to make the prediction (spectrogram plus feed-forward).

4.1. Detector performance

On the validation data used during training, both models achieve an AUC of 94%, with model A requiring double the epochs to reach this, which we attribute to its increased complexity. On the official test data of the Bird Audio Detection challenge 2018, however, it is model B which performs best setting itself apart from model A with an AUC of 82.5%. Model A achieves an AUC of 80.0%. We attribute this 2% increase to the reduced complexity of model B, which features 47.4% fewer parameters, and thus less scope for overfitting.

We implemented both models on the edge device and do not make any further attempts to optimize the detection scores, focusing instead on implementation and performance on the Bela Mini.

4.2. Edge performance

We tested both models on the Bela Mini, measuring computation time of the network and the spectrogram generation programs. Fig 2 plots the empirical distribution of time taken for both models. The collection was carried out by measuring the time (in seconds) the system took to carry out the prediction, from input to prediction output. Although model B had fewer parameters, model A took less time as model A loops over fewer elements in the convolution layers. The mean time difference between both, however, is only 0.146 seconds. It is useful to note, however, that although this is a relatively short time to wait for a prediction, the system needs to compute a 10-second Mel-spectrogram before the prediction phase. This step takes an average of 3.98 seconds under the current implementation and increases total prediction time substantially. However, the spectrogram settings can be changed in the code. For example, reducing the sample rate to 22.05 kHz and the FFT window size to 512 results in an average time of 2.14 seconds to compute the spectrogram.

We also compare the predictions of the model in C++ with the predictions of the original model in Keras for 2,000 files in the test set. The mean absolute error (MAE) between keras predictions and C++ predictions, taken at Keras' precision (1e−7, or float32)[6] is 6.33e−8.

The table below, illustrates the accumulation of small discrepancies in floating-point processing between the two implementations for the 5 instances with the highest MAE:

Table 2: Test set precision of worst 5 cases

File name	Keras output	Model output	Difference
4eb32628-d7a4-448e-91e9.wav	0.68623024	0.68623072	−4.80e−7
286dd5a5-420f-4630-a565.wav	0.34130067	0.34130117	−5.00e−7
f0721db1-8a4e-451d-afd2.wav	0.46779215	0.46779278	−6.30e−7
753cdaaa-1b68-4e89-a079.wav	0.51409733	0.51409799	−6.60e−7
7d47e2a9-d26f-4ab5-9256.wav	0.53456223	0.53456295	−7.20e−7

5. DISCUSSION

In this paper, we implement a sound detection algorithm on the Bela Mini embedded system, highlighting the device's easy to use development environment and ultra-low latency audio processing environment, prioritising audio over other operating system processes. Our work goes beyond previous work on embedded deep learning for bioacoustics [7] in that it works towards an implementation that can work under lower constraints of power, computation, and memory.

We have presented two deep learning models for this embedded Linux system for detecting bird calls in audio recordings. Out of the two models, model B would rank among the top 4 models in the competition, with an AUC score of 82.5%, with both models emphasising Bela as a viable solution for fast prototyping and deployment.

For an indication of how much our detector improves wildlife data logging, we can use our ROC results to indi-

[6]`https://keras.io/backend/`

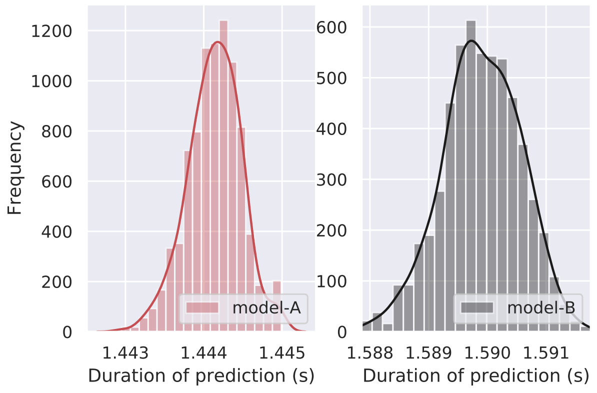

Fig. 2. Recording duration of our implementations on the Bela Mini. Model A varies between 1.44 and 1.45 seconds, while model B varies between 1.59 and 1.60 seconds

cate what would happen in a typical configuration. Choose a true-positive rate of 80%, which implies a false positive rate of around 40% (Figure 1). In a monitoring situation with animals active in 10% of the recorded frames, we thus expect the detector to trigger in 12% of cases. This enables an eight-fold reduction in the amount of data stored to disk, or an eight-fold increase in the time the device can be active before storage is full.

6. DATA ACCESSIBILITY

- Development and testing sets:
  `http://dcase.community/challenge2018/task-bird-audio-detection`
- Spectrogram generation code: `https://github.com/alexandrusoloms/Bela-Spectrogram`
- Feed-forward network code: `https://github.com/alexandrusoloms/KerasToCpp/tree/feature/my-keras-to-cpp`

7. REFERENCES

[1] Alison Johnston, Malcolm Ausden, Andrew M. Dodd, Richard B. Bradbury, Dan E. Chamberlain, Frédéric Jiguet, Chris D. Thomas, Aonghais S. C. P. Cook, Stuart E. Newson, Nancy Ockendon, Mark M. Rehfisch, Staffan Roos, Chris B. Thaxter, Andy Brown, Humphrey Q. P. Crick, Andrew Douse, Rob A. McCall, Helen Pontier, David A. Stroud, Bernard Cadiou, Olivia Crowe, Bernard Deceuninck, Menno Hornman, and James W. Pearce-Higgins, "Observed and predicted effects of climate change on species abundance in protected areas," *Nature Climate Change*, vol. 3, pp. 1055, Nov 2013, Article.

[2] Brown AF Hearn RD Lock L Musgrove AJ Noble DG Stroud DA Eaton MA, Aebischer NJ and Gregory RD (2015), "Birds of Conservation Concern 4: the population status of birds in the United Kingdom, Channel Islands and Isle of Man.," 2015.

[3] D. Stowell, Y. Stylianou, M. Wood, H. Pamuła, and H. Glotin, "Automatic acoustic detection of birds through deep learning: the first Bird Audio Detection challenge," *Methods in Ecology and Evolution*, vol. 16, no. 153, pp. 368–380, Apr. 2019.

[4] Fotios Zantalis, Grigorios Koulouras, Sotiris Karabetsos, and Dionisis Kandris, "A review of machine learning and iot in smart transportation," *Future Internet*, vol. 11, no. 4, 2019.

[5] Liangzhen Lai, Naveen Suda, and Vikas Chandra, "CMSIS-NN: Efficient neural network kernels for ARM Cortex-M CPUs," 2018.

[6] Liangzhen Lai and Naveen Suda, "Rethinking machine learning development and deployment for edge devices," 2018.

[7] Oisin Mac Aodha, Rory Gibb, Kate E. Barlow, Ella Browning, Michael Firman, Robin Freeman, Briana Harder, Libby Kinsey, Gary R. Mead, Stuart E. Newson, Ivan Pandourski, Stuart Parsons, Jon Russ, Abigel Szodoray-Paradi, Farkas Szodoray-Paradi, Elena Tilova, Mark Girolami, Gabriel Brostow, and Kate E. Jones, "Bat detective—deep learning tools for bat acoustic signal detection," *PLOS Computational Biology*, vol. 14, no. 3, pp. 1–19, 03 2018.

[8] Sarab S Sethi, Robert M Ewers, Nick S Jones, David Orme, and Lorenzo Picinali, "Robust, real-time and autonomous monitoring of ecosystems with an open, low-cost, networked device," *Methods in Ecology and Evolution*, vol. 9, no. 12, pp. 2383–2387, 2018.

[9] Andrew P. Hill, Peter Prince, Jake L. Snaddon, C. Patrick Doncaster, and Alex Rogers, "AudioMoth: A low-cost acoustic device for monitoring biodiversity and the environment," *HardwareX*, vol. 6, pp. e00073, 2019.

[10] Andrew McPherson and Victor Zappi, "An environment for submillisecond-latency audio and sensor processing on BeagleBone Black," in *Audio Engineering Society 138th Convention*, 2015.

[11] T. Grill and J. Schlüter, "Two convolutional neural networks for bird detection in audio signals," in *2017 25th European Signal Processing Conference (EUSIPCO)*, Aug 2017, pp. 1764–1768.