Environmental Sound Classification on the Edge: Deep Acoustic Networks for Extremely Resource-Constrained Devices

Md Mohaimenuzzaman Christoph Bergmeir Ian Thomas West

Bernd Meyer
Department of Data Science and AI, Monash University
Australia
md.mohaimen,christoph.bergmeir,ian.west,bernd.meyer@monash.edu

Abstract

Significant efforts are being invested to bring the classification and recognition powers of desktop and cloud systems directly to edge devices. The main challenge for deep learning on the edge is to handle extreme resource constraints (memory, CPU speed and lack of GPU support). We present an edge solution for audio classification that achieves close to state-of-the-art performance on ESC-50, the same benchmark used to assess large, non resource-constrained networks. Importantly, we do not specifically engineer the network for edge devices. Rather, we present a universal pipeline that converts a large deep convolutional neural network (CNN) automatically via compression and quantization into a network suitable for resource-impoverished edge devices. We first introduce a new sound classification architecture, ACDNet, that produces above state-of-the-art accuracy on both ESC-10 and ESC-50 which are 96.75% and 87.05% respectively. We then compress ACDNet using a novel network-independent approach to obtain an extremely small model. Despite 97.22% size reduction and 97.28% reduction in FLOPs, the compressed network still achieves 82.90% accuracy on ESC-50, staying close to the state-of-the-art. Using 8-bit quantization, we deploy ACDNet on standard microcontroller units (MCUs). To the best of our knowledge, this is the first time that a deep network for sound classification of 50 classes has successfully been deployed on an edge device. While this should be of interest in its own right, we believe it to be of particular importance that this has been achieved with a universal conversion pipeline rather than hand-crafting a network for minimal size.

1. Introduction

Intelligent sound recognition is receiving strong interest in a growing number of application areas, from technical safety [10, 19, 53], surveillance and urban monitoring [40] to environmental applications [43, 51].

We are specifically interested in acoustic monitoring of animal vocalisations, a useful and well-established methodology in biodiversity management [8, 26, 4, 25, 27]. Traditionally, animal monitoring is conducted using passive acoustic recording and subsequent manual evaluation by human experts. Since this is extremely labour-intensive, AI-based solutions have recently been at the center of interest [26].

The fact that animal monitoring often has to take place in remote regions [4] poses some interesting technical challenges for automated audio recognition. Almost all suitable state-of-the-art methods for environmental sound classification (ESC) are based on Deep Learning (DL) [38, 44, 47, 56] and consequentially have very high computational requirements. Current AI-based animal monitoring systems typically require recordings to be uploaded to cloud-based platforms or high-powered desktop machines that have sufficient resources to process AI audio models [7, 25]. However, remote areas often have insufficient network coverage or no network coverage at all, rendering the upload of audio data impossible. Continuous long-term monitoring, which is essential for biodiversity management [4], can thus only become a reality if recognition can take place directly on the monitoring devices in the field.

Our goal thus is to make acoustic classification possible on small, embedded edge devices, which generally have very little memory and compute power owing to tight constraints on power consumption.

While the reasons to move the recognition directly onto small edge devices are particularly compelling in the case

of ecological monitoring, similar considerations apply to many other applications of intelligent sound recognition, from industrial safety to consumer devices, for a variety of reasons, including cost and convenience. Specifically cost is a factor that commonly limits the available compute power and network bandwidth in edge applications.

Significant efforts have already been invested into the development of smaller, more efficient CNN models for mobile devices [39, 54], smartphones, CPUs and GPUs [46, 50, 57]. However, smartphones and similar devices still have compute power orders of magnitude above the embedded devices we are targeting and thus much more extreme minimisation is required.

Embedded edge devices use energy efficient microcontroller units (MCUs) and are typically based on system-on-a-chip (SoC) hardware with less than 512kB of RAM and slow clock speeds. GPU support is available on comparatively high-powered specialised edge devices, such as Google's Coral TPU and Nvidia's Jetson, but not on the ultra low-power MCUs we are targeting. The basis of some of the most common energy efficient SoCs, like the Nordic nRF52840 and the STM32F4, is the 32 bit ARM Cortex M4F CPU, typically run at clock speeds below 200 MHz.

To the best of our knowledge, practical acoustic classification has not previously been achieved on such extremely small devices beyond very simple tasks, such as wake-word recognition. Deploying DL models on such MCUs requires aggressive minimisation techniques like model compression [14, 33, 34, 35], knowledge distillation [17] and quantization [14, 37]. Some works, including [11], have used such techniques for efficient models in computer vision, but acoustic classification has not benefitted from this yet.

We present an acoustic classification solution for energy-efficient edge devices that achieves close to state-of-the-art performance on ESC-50 [36], the same benchmark that is used to assess large, non resource-constrained networks. Importantly, we do not specifically design the network for these target devices. Rather, we present a universal pipeline that converts a large deep convolutional neural network (CNN) automatically via compression and quantization into a network suitable for edge devices.

We first introduce a new sound classification architecture, ACDNet, that exceeds current state-of-the-art performance with a classification accuracy of $86.90\% \pm 0.12$. We then compress ACDNet using a novel network-independent compression approach to obtain an extremely small model (ACDNet-20). Despite 97.23% size reduction and 95.55% reduction in FLOPs, the compressed network still achieves 83.71% accuracy on ESC-50, which is significantly higher than human accuracy (81.3%) and still very close to the state of the art.

To classify $1.5s$ audio sampled at 44.1kHz, ACDNet uses 4.74M parameters (18.06MB) and requires approximately 0.57B FLOPs for a single inference. The minimised version, ACDNet-20, has only 0.131M parameters (0.5MB) and requires only 0.025B FLOPs for an inference, which is well within the capabilities of the MCUs we are targeting. We describe a successful deployment on a standard off-the-shelf MCU and, beyond laboratory benchmarks, report successful tests on real-world data.

To the best of our knowledge, this is the first time that a deep network for sound classification of 50 classes has successfully been deployed on an edge device. While this should be of interest in its own right, we believe it to be of particular importance that this has been achieved with a universal conversion pipeline rather than hand-crafting a network for minimal size.

2. Related Work

A large variety of DL models for acoustic classification achieve state-of-the-art performance on ESC-50 or come close to it. Unfortunately, most of these have not fully reported their model sizes and computation requirements. This includes the two models that currently exhibit the highest performance [38, 56] (see Table 2). EnvNet-v2 [47] and AclNet [18] are, in fact, the only ones of the latest ten proposals that have fully detailed their computational requirements (see Table 3). The high requirements of EnvNet-v2 are partly due to the use of multiple dense layers [47], which, unfortunately, do not lend themselves well to compression [2, 32].

Recent approaches also often resort to multi-channel or multiple parallel block models [38, 44, 56] and to attention mechanisms [56]. While this shows some success in improving performance, it also increases model sizes significantly and thus does not constitute an advantageous starting point for our purposes.

Some recent work in computer vision that has specifically focussed on very small CNN models is highly relevant to our work. MCUNet [30] starts from state-of-the-art image classification models, such as ResNet50 [16] and MobileNetV2 [39], and uses search-based optimisation to find models that fit on MCUs. The authors report >70% accuracy after deploying the final models on MCUs.

The recent Sparse Architecture Search (SpArSe [11]) appears to be a very promising approach. It uses multi-objective search to automatically construct models small enough for MCU deployment. It optimises accuracy, model size, and working memory size by performing a search over pruning, parameters, and model architecture. The latter fundamentally sets it apart from other work. While other work minimises specific target models, SpArSe includes the model architecture in the search space. In this way, SpArSe achieves impressive results with models that achieve high accuracy on a variety of standard vision benchmarks (MNIST, CIFAR10, CUReT , Chars4k) and are small

enough to be deployed on standard MCUs.

Model search can require prohibitively vast amounts of computation time. The feasibility of SpArSe is achieved by constraining the search space to model proposals that are specific morphs of a defined starting point. The success of SpArSe thus relies on the availability of suitable models as starting points. While still improving their performance and size, [11] does in fact start from models which themselves are already constructed to fit on MCUs (Bonsai [22] and ProtoNN [13]). Such starting points are currently not available for audio classification, and, to the best of the authors' knowledge, no comparable work exists for this problem domain.

In the audio domain, the only work that specifically has the reduction of computational requirements as its primary focus is Edge-L^3 [24]. It achieves a model size of 0.814MB by pruning L^3-Net [3], which has an initial size of 18MB. While this approach achieves good theoretical compression ratios and good prediction accuracy, it relies on unstructured compression. This results in sparse matrix models which, unfortunately, do not ideally lend themselves to a direct implementation on embedded devices. MCUs generally lack the dedicated hardware and software support for sparse matrix computations required to capitalise on these theoretical savings [2, 12, 28, 32, 42]. Hence, structured compression techniques that produce dense matrix models promise to be a preferable approach for targeting MCUs.

In the following sections we detail the construction of a simple model that betters current state-of-the-art performance on ESC-50, yet provides a suitable starting point for compression, followed by a discussion of the structured compression methods used to minimise this for MCU deployment.

3. Proposed Base Network

We propose a new model architecture (ACDNet) for acoustic classification that is smaller and more efficient than current state-of-the-art networks, yet exceeds their classification accuracy on the most widely used Environmental Sound Classification dataset ESC-50.

3.1. ACDNet Architecture

Unlike most of the proposed state-of-the-art models, the ACDNet architecture focuses on feature extraction through convolution layers. It has two blocks for feature extraction followed by an output block. The feature extraction blocks are: Spectral Feature Extraction Block (SFEB) and Temporal Feature Extraction Block (TFEB). Table 1 shows the different blocks of the ACDNet architecture. There are 12 convolution layers each followed by a batch normalization layer and a ReLU activation layer. There are 6 max pool layers having the stride equal to the pool size to make sure they are not overlapping as they are in AlexNet [21], one average pool layer and finally one dense layer followed by the softmax output layer.

SFEB consists of two 1-D strided convolutions followed by the first pooling layer. This block extracts spectral audio features from raw audio at a frame rate of 10ms.

The axes of the data produced by SFEB are swapped from (ch, f, t) to (ch′=f, f′=ch, t′=t) and the result is fed as input to TFEB to convolve over both frequency and time for extracting hierarchical temporal features. Thus, this block of the network works like a convolution on images. TFEB consists of convolutions 3-12 in Table 1. The first convolution layer is followed by a pooling layer. After that, the convolution layers (4-11) are stacked similar to VGG-13 [41] (two convolutions followed by a pooling layer). The final convolution layer (*conv-12*) is followed by a single average pooling layer. TFEB finishes with a dense layer having output neurons equal to the number of classes to make sure the size of the output vector of TFEB is always equal to the number of classes. This is crucial for compression of the network in a later stage. The output of the TFEB is fed to a softmax output layer for classification. Table 1 shows the generalized ACDNet architecture.

Layers	Kernel Size	Stride	Filters	Output Shape	Block
Input				(1, 1, $w = i\_len$)	
conv1	(1, 9)	(1, 2)	8	(8, 1, $w = \frac{w-9}{2} + 1$)	SFEB
conv2	(1, 5)	(1, 2)	64	(64, 1, $w = \frac{w-5}{2} + 1$)	
Maxpool1	(1, ps)	(1, ps)		(64, 1, $w = \frac{w}{ps}$)	
swapaxes				(1, 64, w)	
conv3	(3, 3)	(1,1)	32	(32, 64, w)	TFEB
Maxpool2	(2, 2)	(2, 2)		(32, 32, $w = \frac{w}{2}$)	
conv4,5	(3, 3)	(1, 1)	64	(64, 32, w)	
Maxpool3	(2, 2)	(2, 2)		(64, 16, $w = \frac{w}{2}$)	
conv6,7	(3, 3)	(1, 1)	128	(128, 16, w)	
Maxpool4	(2, 2)	(2, 2)		(128, 8, $w = \frac{w}{2}$)	
conv8,9	(3, 3)	(1, 1)	256	(256, 8, w)	
Maxpool5	(2, 2)	(2, 2)		(256, 4, $w = \frac{w}{2}$)	
conv10,11	(3, 3)	(1, 1)	512	(512, 4, w)	
Maxpool6	(2, 2)	(2, 2)		(512, 2, $w = \frac{w}{2}$)	
Dropout (0.2)					
conv12	(1, 1)	(1, 1)	n_cls	(n_cls, 2, 4)	
Avgpool1	(2, 4)	(2, 4)		(n_cls, 1, 1)	
Flatten				(n_cls)	
Dense1			n_cls	(n_cls)	
Softmax				(n_cls)	Output

Table 1. ACDNet architecture. Output shape represents (channel, frequency, time), i_len is the input length, n_cls is the number of output classes, sr is the sampling rate in Hz and $ps = \frac{w}{((i\_len/sr)*1000)/10}$

While the structure of the convolution layers has some similarity with EnvNet-v2 [47] and AclNet [18], the ACDNet architecture is strongly motivated from the perspective of feature extraction. It shifts the processing to the initial layers to ensure better feature extraction. We use only one very small dense layer (number of neurons equal to the number of classes) at the end of the TFEB block to allow ACDNet to adapt to the changes in data shape when model compression is applied. In contrast, EnvNet-v2 uses 8 1D convolutions

with different numbers of filters and has 3 larger dense layers, which would make compression difficult. No code is available for AclNet. We re-implemented AclNet but were unable to reproduce the results reported in the original paper.

3.2. Experimental Setup

ACDNet is implemented in PyTorch version 1.3 using the *Wavio* audio library. The full code is available in the supplementary material.

3.2.1 Datasets

The experiments are conducted on one of the most robust and widely used audio benchmark datasets - Environmental Sound (ESC-50 [36]). This dataset contains 2000 samples (5-sec-long audio recordings, sampled at 16kHz and 44.1kHz) which are equally distributed over 50 balanced disjoint classes (40 audio samples for each class). Furthermore, a division of the dataset into 5 splits is available, helping researchers to achieve unbiased comparable results in 5-fold cross validation.

Our work is ultimately aimed at real-world applications for biodiversity and these can be more difficult than standard benchmarks, as evidenced in the LifeCLEF competition [20]. As a real-world test, we evaluate ACDNet on a dataset of frog recordings that is known to be challenging for conventional animal monitoring solutions [5, 6]. It contains 9132 field recordings of 10 different frog species across a variety of locations in Australia sampled at 32kHz.

3.2.2 Data Processing and Training Setup

ACDNet is trained and tested with inputs of length 30,225 (approximately 1.51s audio @ 20kHz). Our data preprocessing, augmentation, and mixing of two classes to produce training samples follow EnvNet-v2 [47]. According to EnvNet-v2, two training sounds belonging to two different classes are randomly picked, padded with T/2 (T = input length) zeros to each side of both the samples and a *T-s* section from both the sounds is randomly cropped. Then, the two cropped samples are mixed using a random ratio. We denote the maximum gains of the cropped samples s_1, s_2 by g_1, g_2, and r is the random ratio between $(0, 1)$. The ratio of the mixed sounds p according to EnvNet-v2, is

$$p = \frac{1}{1 + 10^{\frac{g1-g2}{20}} * \frac{1-r}{r}} \tag{1}$$

Finally, the mixed sound sample S_{mix} for training is determined by

$$S_{mix} = \frac{ps_1 + (1-p)s_2}{\sqrt{p^2 + (1-p)^2}} \tag{2}$$

In the testing phase, we pad T/2 zeros to each side of the test input sample and then extract 10 windows (each of length 30,225) at a regular interval of T/9 as input for the network. All the input data for training and testing are regularized by diving them by 32,768, which is the full range of 16-bit recordings.

The network is trained for 2000 epochs with an initial learning rate of 0.1 along with a learning rate scheduler $\{600, 1200, 1800\}$ decaying at a factor of 10. The first 10 epochs are considered as warm-up epochs and a 0.1 times smaller learning rate is used for these 10 epochs. We use *KLDivLoss* (Kullback-Leibler Divergence Loss) as the loss function and optimize it using back-propagation and Stochastic Gradient Descent (*SGD*) with a *Nestrov momentum* of 0.9, a weight decay of 5e-4, and a batch size of 64. Furthermore, the weights of all convolution and dense layers are initialized using the *he_normal* [15] initialization method. At the end of the training and validation, the best model is used as the final model. The loss function optimized is shown in Equation 3. Here, x is the input mini-batch, $f_\theta(x)$ is the approximation and y is the true distribution of labels for the input data. Furthermore, n is the mini-batch size, m is the number of classes, η is the learning rate, and $\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$.

$$\begin{aligned} L &= \frac{1}{n}\sum_{i=1}^{n}(D_{KL}(y^{(i)}||f_\theta(x^{(i)}))) \\ &= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_j^{(i)} \log \frac{y_j^{(i)}}{(f_\theta(x^{(i)}))_j} \end{aligned} \tag{3}$$

3.2.3 Training and Testing ACDNet

We conduct 5-fold cross validation over 10 independent runs. The results in Table 2 show that ACDNet outperforms the current state of the art with an overall accuracy of 86.90 ± 0.12 .

For the real-world frog dataset, we trained ACDNet for 1000 epochs with *maxpool6* adjusted to pool size (2,1) to handle the different input length. We obtain 5-fold cross validation accuracy of 88.98% even without data augmentation.

Few state-of-the-art models report their size and computation requirements in the literature. To the best of our knowledge, EnvNet-v2 and AclNet are the only two of the top-ten models for ESC-50 to report parameter count, size, and FLOPs. Table 3 shows that ACDNet requires $21.39x$ less memory and $2.8x$ less FLOPs than EnvNet-v2 and $2.25x$ less memory and $1.9x$ less FLOPs than AclNet, respectively.

4. Network Compression

As discussed before, unstructured compression is not suitable for MCUs. Therefore, we propose a new class of structured hybrid compression techniques. We first sparsify the

Networks	Accuracy(%)	
	ESC-10	ESC-50
Human[36]	95.70	81.30
EnvNet-v2[47]	88.80	81.6 ± 0.2
GSTC⊕TEO-GSTC [1]	-	81.95
GTSC⊕ConvRBM-BANK [38]	-	83.00
Kumar-CNN [23]	-	83.50
CNN+Augment+Mixup [55]	91.70	83.90
Multi-stream+Attention [29]	94.20	84.00
FBEs⊕PEFBEs [45]	-	84.15
EnvNet-v2[47] + Strong Augment	91.3	84.9 ± 0.2
AclNet (Width = 1.5)[18]	-	85.65
CRNN + Attention [56]	-	85.8 ± 0.6
FBEs⊕ConvRBM-BANK[38]	-	86.50
ACDNet (proposed)	**96.75**	$\mathbf{87.05 \pm 0.02}$

Table 2. State-of-the-art models for ESN-10 and ESC-50 dataset

Networks	#Filters	Params(M)	Size(MB)	FLOPs(B)
EnvNet-v2[47]	1056	101.25	386.25 = $21.39x$	1.62 = $2.8x$
AclNet[18]	3050	10.63	40.57 = $2.25x$	1.07 = $1.9x$
ACDNet (proposed)	2074	4.74	18.06 = x	0.54 = x

Table 3. Parameters, size, and computation requirements for current state-of-the-art models on the ESC-50 dataset. Here, x denotes the size and FLOPs of ACDNet, for the Size and FLOPs column, respectively.

weight matrices of ACDNet using unstructured compression and then apply structured compression. This allows us to focus the structured compression on the important weights.

There are different techniques by which structured compression can be achieved, such as Filter Pruning [34], Weight Sharing [14, 52], Huffman Coding [48], Knowledge Distillation [17] and Quantization [14, 49]. Though many works in the literature use these techniques in computer vision, they have hardly ever been used for audio tasks. Due to this lack of guidance from the literature, we use the most well-established pruning-based model compression techniques proposed for computer vision to compress ACDNet. Furthermore, since quantization does not conflict with other compression techniques, we use quantization of the compressed model to further reduce its size before deploying it on the embedded device.

We use two pure structured pruning techniques, namely magnitude-based pruning and Taylor pruning [34], and based on those we propose a novel hybrid pruning technique, where unstructured pruning is followed by structured pruning. Magnitude-based pruning, also known as L1 Norm-based pruning, is a common type of filter pruning approach. Taylor pruning is one of the recent state-of-the-art filter pruning techniques for computer vision applications. On the other hand, the hybrid pruning technique is a combination of structured and unstructured pruning.

To make sure the networks work without errors during and after compression, *conv2* is forced to retain at least 32 filters, as otherwise the downsampling through max pool and average pool layers would fail. The kernel size of *Avgpool1* has been adjusted to 1x4 from 2x4 during the pruning process when *Maxpool6* produces an output of shape (channels, height=1, width) after pruning, whereas the uncompressed network would produce outputs of shape (channels=512, height=2, width=4). Furthermore, after flattening the *Avgpool1* output, when the flattened vector has elements less than the number of classes (i.e., 50), the number of input neurons of *Dense1* is dynamically adjusted so that it can handle the incoming input data to produce 50 output elements for the softmax output layer.

During the compression process, we remove 80% and 85% of filters in two runs, respectively, from the original network. These amounts are selected to obtain a model small enough for our target MCUs.

4.1. Magnitude-based Pruning

In this method, the filters are ranked using their individual sum of absolute weights (L1 Norm) followed by layer-wise L2 normalization. Many approaches remove all the filters having a sum below a predefined threshold [35, 14, 34]. However, we remove the filters having the lowest sum iteratively (one at a time) and retrain the network to recover the loss until the network reaches the target size required to be fitted in the MCU. Let F be the filters of a network, i be the layer index and j be the filter index. Then, an iteration of this pruning process is defined by

$$Fmag = \sum(|F_j|) \tag{4}$$

$$Fmag_n = \frac{|Fmag_{ij}|}{\sqrt{\sum |Fmag_i|^2}} \tag{5}$$

$$PruningCandidate = argmin(Fmag_n) \tag{6}$$

The compressed models (Models 1-2 in Table 4 and 5) show that pruning and re-training does not recover the loss in accuracy as hoped for (see Fine-tuned accuracy column). To achieve the best accuracy, we therefore retrain the network from scratch.

4.2. Taylor Pruning

This is an iterative filter pruning technique recently proposed by Molchanov et al. [34]. In this approach, the filters are ranked and the lowest ranked filter is pruned in a pruning iteration. The ranking of filters is calculated by conducting a forward pass of the trained network for the whole dataset and observing the change to the cost function. The least affected filter after layer-wise L2 normalization is ranked highest to be pruned. The iterative pruning and retraining process continues until the model reaches the target size.

Model No.	Pruning Method	SFEB	TFEB	#Params Left	Size (MB)	Size Reduced	#FLOPs	FLOPs Reduced	Fine-Tuned Accuracy	Re-trained Accuracy	Scratch-training Accuracy
1	Magnitude		✓	0.098M	0.37MB	97.94%	0.098B	82.87%	3.75%	84.75%	85.00%
2	Magnitude	✓	✓	0.115M	0.44MB	97.57%	0.054B	90.46%	4.25%	83.50%	83.00%
3	Taylor		✓	0.109M	0.41MB	97.70%	0.067B	88.29%	12.75%	85.00%	84.50%
4	Taylor	✓	✓	0.147M	0.56MB	96.90%	0.036B	93.73%	4.50%	84.00%	81.75%
5	Hybrid (L0 → Magnitude)		✓	0.099M	0.38MB	97.90%	0.102B	82.18%	3.25%	85.25%	85.00%
6	Hybrid (L0 → Magnitude)	✓	✓	0.118M	0.45MB	97.50%	0.056B	90.25%	4.00%	83.00%	83.00%
7	Hybrid (L0 → Taylor)		✓	0.114M	0.43MB	97.60%	0.065B	88.63%	40.75%	85.50%	84.75%
8	**Hybrid (L0 → Taylor)**	✓	✓	**0.131M**	**0.50MB**	**97.23%**	**0.025B**	**95.55%**	**14.25%**	**82.75%**	**85.25%**

Table 4. Models found after 80% filter pruning using magnitude-based pruning, Taylor pruning and our hybrid pruning approach.

Model No.	Pruning Method	SFEB	TFEB	#Params Left	Size (MB)	Size Reduced	#FLOPs	FLOPs Reduced	Fine-Tuned Accuracy	Re-trained Accuracy	Scratch-training Accuracy
1	Magnitude		✓	0.061M	0.23MB	98.71%	0.042B	92.64%	1.50%	79.25%	82.00%
2	Magnitude	✓	✓	0.062M	0.24MB	98.70%	0.042B	92.62%	2.00%	81.50%	80.25%
3	Taylor		✓	0.050M	0.19MB	98.94%	0.058B	89.90%	40.25%	80.25%	82.00%
4	Taylor	✓	✓	0.081M	0.31MB	98.29%	0.026B	95.38%	35.75%	82.75%	81.25%
5	Hybrid (L0 → Magnitude)		✓	0.064M	0.25MB	98.64%	0.043B	92.50%	30.00%	82.00%	82.00%
6	Hybrid (L0 → Magnitude)	✓	✓	0.063M	0.24MB	98.67%	0.044B	92.36%	3.75%	80.75%	81.25%
7	Hybrid (L0 → Taylor)		✓	0.052M	0.20MB	98.90%	0.057B	89.98%	45.25%	77.75%	80.00%
8	Hybrid (L0 → Taylor)	✓	✓	0.063M	0.24MB	98.66%	0.025B	95.70%	32.25%	77.75%	81.25%

Table 5. Models found after 85% filter pruning using magnitude-based pruning, Taylor pruning and our hybrid pruning approach.

An iteration of this pruning process is defined by

$$A = A + \overline{(A * G)} \tag{7}$$

$$A_n = \frac{|A_{ij}|}{\sqrt{\sum |A_i|^2}} \tag{8}$$

$$PruningCandidate = argmin(A_n) \tag{9}$$

Here, A are the activations of the layers of the network, G is the gradient, i is the layer index and j is the filter index.

As for the magnitude-based pruning, we observe from Tables 4 and 5, Models 3-4, that we need to re-train or train the pruned network from scratch to achieve comparable accuracy.

4.3. Proposed Hybrid Pruning Approach

We test two iterative hybrid approaches. The first one consists of L0 Norm followed by magnitude-based pruning and the second one of L0 Norm followed by Taylor pruning. The pruning and retraining is iterated until the model reaches the target size. In this process, 95% of the weights are pruned using L0 norm and then the model is further pruned using structured pruning techniques. As we apply structured pruning on the weight-pruned network, we are not producing sparse matrices, which are generally not supported on embedded devices.

4.4. Experimental Findings

Tables 4 and 5 provide the experimental results for the different pruning approaches tested on ACDNet. In Table 4 we see that our hybrid approach (Models 5-8) produces the best prediction accuracy and FLOP reduction for a model size reduction that fits the target specifications (Model 8).

Note that for the purpose of compression method comparison we have only worked with Fold 2 data. Our pilot experiments indicated minimal variance across folds. We fully evaluate the final chosen model in Table 7 using 5-fold cross validation.

All models are iteratively pruned and retrained. The column *Fine-Tuned Accuracy* shows the accuracy obtained at the end of the iterative pruning and retraining process. Since the fine-tuned accuracy is very low, we conduct further full retraining using the base network's training settings and the accuracy is reported in *Re-trained Accuracy*. We also rebuild the network from scratch using the compressed network's architectural settings and train the network as a brand new network. The prediction accuracy is reported in column *Scratch-training Accuracy*.

Pruning filters from initial convolution layers leads to more size and FLOPs reduction and causes little accuracy loss, thus making it a suitable approach for extremely resource-constrained MCUs.

From the experiments, we observe that *pruning and retraining* is not enough to achieve comparable prediction accuracy, which is contrary to earlier findings in the literature, e.g., in Molchanov et al. [34]. The experimental results (Table 4 and 5) show that we require separate full retraining or building and training the compressed network from scratch to achieve good prediction accuracy.

Our experiments furthermore suggest, supported by earlier work by Crowley et al. [9] and Liu et al. [31], that pruning or compression should be considered as an efficient DL model architecture search method.

4.5. Selecting the Compressed Network

We choose the best compressed network depending on three factors: compression, FLOP reduction and accuracy. Con-

sidering the result on all three measurements for 80% pruning shown in Table 4 and Figure 1, we select Model 8, obtained by our hybrid pruning technique. For this model, our hybrid pruning technique achieves 97.22% model size reduction and 97.28% FLOP reduction (Table 4), retaining 82.90% accuracy for ESC-50 (Table 7). This is still very close to the state of the art and significantly higher than human accuracy (81.3%). We term this net ACDNet-20 for "20% of filters retained".

With 85% pruning, our hybrid technique produces an even smaller version of Model 8 with 312 filters (15% retained) and only 240kB model size, which we term ACDNet-15 (Figure 2). We select ACDNet-20 over this for three reasons. Firstly, ACDNet-20 delivers significantly higher accuracy. Secondly, its model size of approximately 500kB already fits into typical MCU flash sizes and can be further decreased fourfold by 8-bit quantization to a size below the memory limits of even small microcontrollers. Thirdly, both models require approximately the same amount of FLOPs.

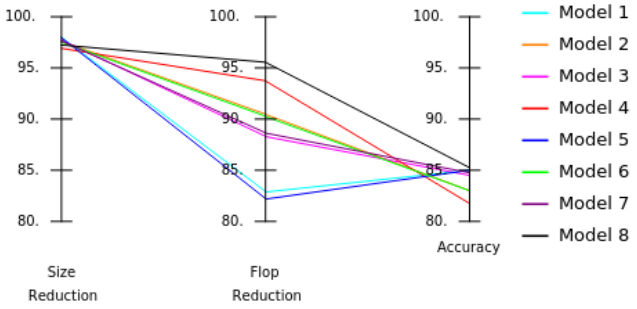

Figure 1. Comparsion of 80% pruned models from Table 4

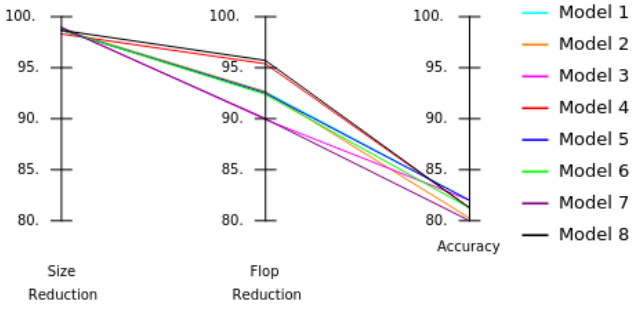

Figure 2. Comparsion of 85% pruned models from Table 5

4.6. Training and Testing ACDNet-20

Since our results indicate that training from scratch works better than retraining the compressed network, we rebuild ACDNet-20 using the remaining filters and the updated kernel size of the average pooling layer. We train this network from scratch with the original ACDNet training settings, conducting 5-fold cross validation over five independent runs. Table 7 shows our results in comparsion to Edge-L^3 as another edge audio architecture.

Layers	Kernel Size	Stride	Filters	Output Shape
Input				(1, 1, 30225)
conv1	(1, 9)	(1, 2)	7	(7, 1, 15109)
conv2	(1, 5)	(1, 2)	20	(20, 1, 7553)
Maxpool1	(1, 50)	(1, 50)		(20, 1, 151)
swapaxes				(1, 20, 151)
conv3	(3, 3)	(1, 1)	10	(10, 32, 151)
Maxpool2	(2, 2)	(2, 2)		(10, 16, 75)
conv4	(3, 3)	(1, 1)	14	(14, 16, 75)
conv5	(3, 3)	(1, 1)	22	(22, 16, 75)
Maxpool3	(2, 2)	(2, 2)		(22, 8, 37)
conv6	(3, 3)	(1, 1)	31	(31, 8, 37)
conv7	(3, 3)	(1, 1)	35	(35, 8, 37)
Maxpool4	(2, 2)	(2, 2)		(35, 4, 18)
conv8	(3, 3)	(1, 1)	41	(41, 4, 18)
conv9	(3, 3)	(1, 1)	51	(51, 4, 18)
Maxpool5	(2, 2)	(2, 2)		(51, 2, 9)
conv10	(3, 3)	(1, 1)	67	(67, 2, 9)
conv11	(3, 3)	(1, 1)	69	(69, 2, 9)
Maxpool6	(2, 2)	(2, 2)		(69, 1, 4)
Dropout (0.2)				
conv12	(1, 1)	(1, 1)	48	(48, 1, 4)
Avgpool1	(1, 4)	(1, 4)		(48, 1, 1)
Flatten				(48)
Dense1				(50)
Softmax				(50)

Table 6. ACDNet-20 architecture for input length 66650 (approximately 1.51s audio @ 20kHz)

Networks	#Filters	Params (M)	Size (MB)	FLOPs (B)	Accuracy (%)
ACDNet	2074	4.74	18.06	0.54	87.05 ± 0.02
Edge-L^3 [24]	1920	0.213	0.814	-	73.75
ACDNet-20	415	0.131	0.5	0.0148	82.90

Table 7. Parameters, size and computation requirements for ACDNet and ACDNet-20 for approximately 1.51s audio @ 20kHz

5. Deployment in MCU (Edge-AI Device)

Our network has been fully deployed on an off-the-shelf MCU. The most limiting factor for MCU deployment is memory requirements. It is important to distinguish between two memory types: (1) Flash memory, which is not suitable for fast, frequent write access and is used to hold the (fixed) model parameters and (2) SRAM, which is used to store inputs and the results of intermediate calculations, i.e. activation values. As mentioned above, typical parameters for widely used highly power-efficient MCUs are less than 1MB of Flash and less than 512kB of SRAM.

Most off-the-shelf MCUs have either no audio capability or very low quality audio on-board. One could, in principle, add additional audio hardware to achieve better quality. However, this would increase power consumption. An exception is the Sony Spresense which provides high-quality audio input and processing on-board. We have selected this unit for these reasons.

We downsampled the data to 20kHz and trained and tested the model with this data. This is motivated by further

reducing power consumption and justified by the fact that little information is contained in the test data at frequencies above 10kHz. A test bears this out: when we trained and tested ACDNet with 44.1kHz data, the accuracy slightly dropped to 86.90 ± 0.12.

While the Spresense is a highly power-efficient device, it has somewhat more generous specifications than the most common MCUs, providing 1.5MB SRAM. It is thus important to note that we are not actually making use of this additional memory. Our final deployed model requires 303kB SRAM for intermediate calculations, which is well below the target. This can be further reduced to below 200kB using a hand-optimised implementation as detailed below. While we have not yet taken this last step in a physical deployment, the implication is that ACDNet-20 can fit on even smaller MCUs, such as those based on the extremely popular Nordic nRF52840 SoC, which offers 256kB SRAM and 1MB Flash. Flash memory is not a limiting constraint, since ACDNet-20 requires only 500kB of model storage (Table 7).

Unfortunately, none of the platform-independent DNN software frameworks for small-device deployment (*i.e.* PyTorch Mobile, Tensorflow Lite) allows us to implement ACDNet directly. This is because ACDNet contains a *transpose* layer, which is not yet supported in these frameworks. We thus had to extend a framework with this new layer type. Among the available options we chose Tensorflow Lite because it is arguably the most widely used such framework and because it offers the best support for such extensions.

To achieve the required model size, we need to quantize ACDNet. The present paper is not concerned with quantization itself, so that we simply apply the quantization methods directly available in Tensorflow. Quantization, unfortunately, reduces the model accuracy noticeably and Tensorflow does not seem to provide the best results here. We used 8-bit post training quantization which reduces the accuracy to 70.75%. To confirm that better results are possible, we also tested alternative frameworks. We found that Pytorch achieved the highest accuracy with 81.5% for an 8-bit quantized model of equivalent size (see Table 8). While our actual deployment uses Tensorflow Lite for implementation-related reasons, we conclude that an alternative version of ACDNet that achieves 81.5% accuracy (i.e. above human performance) can be deployed on a standard MCU of the same size.

The required working memory of 303kB SRAM could be reduced further. The current requirement results form essentially computing tensors layer by layer, keeping one intermediate layer in memory at a time. This can be optimised by re-grouping computations. The bottleneck for working memory is *conv2*. However, this is immediately followed by a non-overlapping max pooling layer. Therefore, each instance of *Maxpool2* can be immediately computed and the corresponding slice of output from *conv2* can immediately be discarded after this. In this way, at any point of time, the most we need to keep in memory is the output of *conv1*, a 110 units wide segment of *conv2* and the complete output of *Maxpool1*. This modification would allow us to reduce the working memory bottleneck to 141,636 bytes. While this may be detrimental to GPU speed-ups, it does not impact on an MCU implementation. However, this cannot be done in Tensorflow and would require a manual implementation.

Network	Library	Quantized Size	Quantized Accuracy (%)
ACDNet-20	Pytorch	157kB	81.50%
ACDNet-20	TF Lite	153kB	70.75%

Table 8. Prediction accuracy after quantization

6. Future Work and Conclusions

We have presented the first implementation of a 50 class audio classifier that achieves high accuracy, yet is small enough to fit on MCUs commonly used in energy-efficient internet-of-things devices. We constructed a full size network that sets a new standard for both the ESC-10 and ESC-50 benchmarks and compressed this for MCU deployment. While limitations of the programming environment have restricted the accuracy of our current test deployment on a physical MCU, we have conclusively shown that 81.5% accuracy is achievable on such a resource-impoverished device, close to the state-of-the-art and above human performance. This brings our goal of continuous, autonomous animal monitoring into immediate reach and should open new horizons for many other audio applications on the internet-of-things.

Our deployment of a DNN originally not designed for MCUs has shown that structured pruning achieves the best results for this task. It has also highlighted the fact that machine learning on the edge, still very much cutting edge, is not yet sufficiently supported by standard frameworks. We particularly expect that future frameworks will provide improved quantization support.

Some next steps immediately arise. Firstly, it is likely that the performance can be improved by using quantization-aware training and pruning. Secondly, we would like to try the SpArSe approach for further optimisations now that we have developed ACDNet-20 as a suitable starting point for its optimisations.

We believe it to be of particular importance that we have constructed and used a universal pipeline to derive an MCU implementation from a standard DNN. This opens up the same opportunities for a wide range of applications and we are confident that we will be able to transfer our approach to other domains.

References

[1] Dharmesh M Agrawal, Hardik B Sailor, Meet H Soni, and Hemant A Patil. Novel teo-based gammatone features for environmental sound classification. In *2017 25th European Signal Processing Conference (EUSIPCO)*, pages 1809–1813. IEEE, 2017. 5

[2] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017. 2, 3

[3] Relja Arandjelovic and Andrew Zisserman. Look, listen and learn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 609–617, 2017. 3

[4] Cathleen Balantic and Therese Donovan. Dynamic wildlife occupancy models using automated acoustic monitoring data. *Ecological Applications*, 29(3):1–14, 2019. 1

[5] Sheryn Brodie, Slade Allen-Ankins, Michael Towsey, Paul Roe, and Lin Schwarzkopf. Automated species identification of frog choruses in environmental recordings using acoustic indices. *Ecological Indicators*, 119:106852, 2020. 4

[6] Sheryn Brodie, Kiyomi Yasumiba, Michael Towsey, Paul Roe, and Lin Schwarzkopf. Acoustic monitoring reveals year-round calling by invasive toads in tropical australia. *Bioacoustics*, pages 1–17, 2020. 4

[7] Alexander Brown, Saurabh Garg, and James Montgomery. AcoustiCloud: A cloud-based system for managing large-scale bioacoustics processing. *Environmental Modelling & Software*, 131:104778, 2020. 1

[8] Zuzana Burivalova, Edward T. Game, and Rhett A. Butler. The sound of a tropical forest. *Science*, 363(6422):28–29, 2019. 1

[9] Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. Pruning neural networks: is it time to nip it in the bud? In *NIPS 2018 Workshop CDNNRIA*, 2018. 6

[10] Matthias Auf der Mauer, Tristan Behrens, Mahdi Derakhshanmanesh, Christopher Hansen, and Stefan Muderack. Applying sound-based analysis at porsche production: Towards predictive maintenance of production machines using deep learning and internet-of-things technology. In *Digitalization Cases*, pages 79–97. Springer, 2019. 1

[11] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers. In *Advances in Neural Information Processing Systems*, pages 4977–4989, 2019. 2, 3

[12] Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1586–1595, 2018. 3

[13] Chirag Gupta, Arun Sai Suggala, Ankit Goyal, Harsha Vardhan Simhadri, Bhargavi Paranjape, Ashish Kumar, Saurabh Goyal, Raghavendra Udupa, Manik Varma, and Prateek Jain. ProtoNN: Compressed and accurate kNN for resource-scarce devices. In *34th International Conference on Machine Learning*, pages 1331–1340, 2017. 3

[14] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations (ICLR)*, 2016. 2, 5

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 4

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778, 2016. 2

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015. 2, 5

[18] Jonathan J Huang and Juan Jose Alvarado Leanos. Aclnet: efficient end-to-end audio classification cnn. *arXiv preprint arXiv:1811.06669*, 2018. 2, 3, 5

[19] Feng Jia, Yaguo Lei, Liang Guo, Jing Lin, and Saibo Xing. A neural network constructed by deep learning technique and its application to intelligent fault diagnosis of machines. *Neurocomputing*, 272:619–628, 2018. 1

[20] Alexis Joly, Hervé Goëau, Stefan Kahl, Benjamin Deneu, Maximillien Servajean, Elijah Cole, Lukáš Picek, Rafael Ruiz De Castaneda, Isabelle Bolon, Andrew Durso, et al. Overview of lifeclef 2020: A system-oriented evaluation of automated species identification and species distribution prediction. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 342–363. Springer, 2020. 4

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3

[22] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In *International Conference on Machine Learning*, pages 1935–1944, 2017. 3

[23] Anurag Kumar, Maksim Khadkevich, and Christian Fügen. Knowledge transfer from weakly labeled audio using convolutional neural network for sound events and scenes. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 326–330. IEEE, 2018. 5

[24] Sangeeta Kumari, Dhrubojyoti Roy, Mark Cartwright, Juan Pablo Bello, and Anish Arora. Edgelˆ 3: Compressing lˆ 3-net for mote scale urban noise monitoring. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 877–884. IEEE, 2019. 3, 7

[25] Rama Rao Kvsn, James Montgomery, Saurabh Garg, and Michael Charleston. Bioacoustics Data Analysis – A Taxonomy, Survey and Open Challenges. *IEEE Access*, 8:57684–57708, 2020. 1

[26] Roberta Kwok. AI empowers conservation biology. *Nature*, 657(7746):133—134, 2019. 1

[27] Eric R Larson, Brittney M Graham, Rafael Achury, Jaime J Coon, Melissa K Daniels, Daniel K Gambrell, Kacie L Jonasen, Gregory D King, Nicholas LaRacuente, Tollope IN Perrin Stowe, Emily M Reed, Christopher J Rice, Selina A Ruzi, Margaret W Thairu, Jared C Wilson, and Andrew V Suarez. From eDNA to citizen science: emerging tools for the early detection of invasive species. *Frontiers in Ecology and the Environment*, page 2162, 2020. 1

[28] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations (ICLR)*, 2017. 3

[29] Xinyu Li, Venkata Chebiyyam, and Katrin Kirchhoff. Multi-stream network with temporal attention for environmental sound classification. *Proc. Interspeech 2019*, pages 3604–3608, 2019. 5

[30] Ji Lin, Wei-Ming Chen, Yujun Lin, John Cohn, Chuang Gan, and Song Han. Mcunet: Tiny deep learning on iot devices. *arXiv preprint arXiv:2007.10319*, 2020. 2

[31] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations (ICLR)*, 2018. 6

[32] Jian-Hao Luo, Hao Zhang, Hong-Yu Zhou, Chen-Wei Xie, Jianxin Wu, and Weiyao Lin. Thinet: pruning cnn filters for a thinner net. *IEEE transactions on pattern analysis and machine intelligence*, 2018. 2, 3

[33] Xiaolong Ma, Geng Yuan, Sheng Lin, Zhengang Li, Hao Sun, and Yanzhi Wang. Resnet can be pruned 60×: Introducing network purification and unused path removal (p-rm) after weight pruning. In *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pages 1–2. IEEE, 2019. 2

[34] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *5th International Conference on Learning Representations (ICLR)*, 2017. 2, 5, 6

[35] Oyebade Oyedotun, Djamila Aouada, and Bjorn Ottersten. Structured compression of deep neural networks with debiased elastic group lasso. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 2277–2286, 2020. 2, 5

[36] Karol J. Piczak. ESC: Dataset for Environmental Sound Classification. In *Proceedings of the 23rd Annual ACM Conference on Multimedia*, pages 1015–1018. ACM Press, 2015. 2, 4, 5

[37] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *International Conference on Learning Representations (ICLR)*, 2018. 2

[38] Hardik B Sailor, Dharmesh M Agrawal, and Hemant A Patil. Unsupervised filterbank learning using convolutional restricted boltzmann machine for environmental sound classification. In *INTERSPEECH*, pages 3107–3111, 2017. 1, 2, 5

[39] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 4510–4520, 2018. 2

[40] Roneel V Sharan and Tom J Moir. An overview of applications and advancements in automatic sound recognition. *Neurocomputing*, 200:22–34, 2016. 1

[41] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015. 3

[42] Pravendra Singh, Vinay Kumar Verma, Piyush Rai, and Vinay Namboodiri. Leveraging filter correlations for deep model compression. In *The IEEE Winter Conference on Applications of Computer Vision*, pages 835–844, 2020. 3

[43] Dan Stowell, Tereza Petrusková, Martin Šálek, and Pavel Linhart. Automatic acoustic identification of individuals in multiple species: improving identification across recording conditions. *Journal of the Royal Society Interface*, 16(153):20180940, 2019. 1

[44] Yu Su, Ke Zhang, Jingyu Wang, and Kurosh Madani. Environment sound classification using a two-stream cnn based on decision-level fusion. *Sensors*, 19(7):1733, 2019. 1, 2

[45] Rishabh N Tak, Dharmesh M Agrawal, and Hemant A Patil. Novel phase encoded mel filterbank energies for environmental sound classification. In *International Conference on Pattern Recognition and Machine Intelligence*, pages 317–325. Springer, 2017. 5

[46] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2820–2828, 2019. 2

[47] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Learning from between-class examples for deep sound recognition. In *International Conference on Learning Representations (ICLR)*, 2018. 1, 2, 3, 4, 5

[48] Jan Van Leeuwen. On the construction of huffman trees. In *ICALP*, pages 382–410, 1976. 5

[49] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 8612–8620, 2019. 5

[50] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10734–10742, 2019. 2

[51] Xiao Yan, Hemin Zhang, Desheng Li, Daifu Wu, Shiqiang Zhou, Mengmeng Sun, Haiping Hu, Xiaoqiang Liu, Shijie Mou, Shengshan He, et al. Acoustic recordings provide detailed information regarding the behavior of cryptic wildlife to support conservation translocations. *Scientific reports*, 9(1):1–11, 2019. 1

[52] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze. Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5687–5695, 2017. 5

[53] Huitaek Yun, Hanjun Kim, Eunseob Kim, and Martin BG Jun. Development of internal sound sensor using stethoscope and its applications for machine monitoring. *Procedia Manufacturing*, 48:1072–1078, 2020. 1

[54] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 6848–6856, 2018. 2

[55] Zhichao Zhang, Shugong Xu, Shan Cao, and Shunqing Zhang. Deep convolutional neural network with mixup for environmental sound classification. In *Chinese Conference on Pattern Recognition and Computer Vision (PRCV)*, pages 356–367. Springer, 2018. 5

[56] Zhichao Zhang, Shugong Xu, Shunqing Zhang, Tianhao Qiao, and Shan Cao. Learning attentive representations for environmental sound classification. *IEEE Access*, 7:130327–130339, 2019. 1, 2, 5

[57] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 8697–8710, 2018. 2