ACDNet

ACDNet Architecture

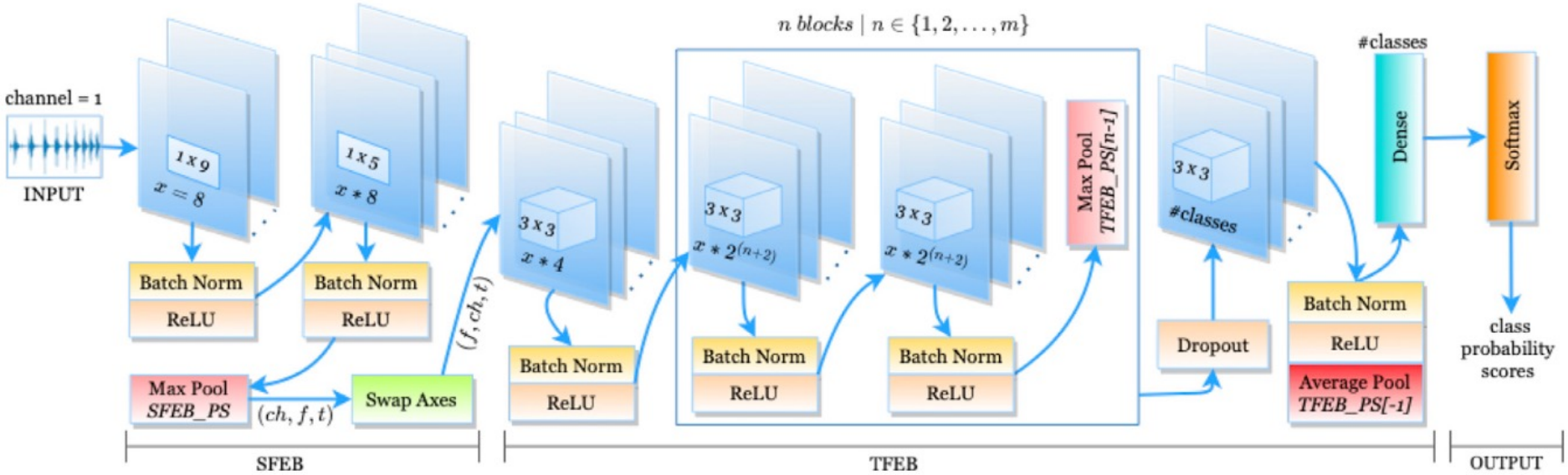

Fig. 1. Visual representation of ACDNet model architecture.

ACDNet Architecture

conv1	(1, 9)	(1, 2)	x	$(x, 1, w = \frac{w-9}{2} + 1)$	SFEB
conv2	(1, 5)	(1, 2)	$x\prime = x * 2^3$	$(x\prime, 1, w = \frac{w-5}{2} + 1)$	
Maxpool1	(1, *SFEB_PS*)	(1, *SFEB_PS*)		$(x\prime, 1, w = \frac{w}{ps})$	
swapaxes				$(1, h = x\prime, w)$	

SFEB架構

$$SFEB\_PS = \frac{w}{((i\_len/sr) * 1000)/10} \quad (1)$$

SFEB_PS

conv3	(3, 3)	(1,1)	$x\prime = x * 2^2$	$(x\prime, h, w)$	TFEB
Maxpool2	*TFEB_PS*[0]	*TFEB_PS*[0]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv4,5	(3, 3)	(1, 1)	$x\prime = x * 2^3$	$(x\prime, h, w)$	
Maxpool3	*TFEB_PS*[1]	*TFEB_PS*[1]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv6,7	(3, 3)	(1, 1)	$x\prime = x * 2^4$	$(x\prime, h, w)$	
Maxpool4	*TFEB_PS*[2]	*TFEB_PS*[2]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv8,9	(3, 3)	(1, 1)	$x\prime = x * 2^5$	$(x\prime, h, w)$	
Maxpool5	*TFEB_PS*[3]	*TFEB_PS*[3]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv10,11	(3, 3)	(1, 1)	$x\prime = x * 2^6$	$(x\prime, h, w)$	
Maxpool6	*TFEB_PS*[4]	*TFEB_PS*[4]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
Dropout (0.2)					
conv12	(1, 1)	(1, 1)	n_cls	(n_cls, h, w)	
Avgpool1	*TFEB_PS*[5]	*TFEB_PS*[5]		$(n\_cls, 1, 1)$	
Flatten				(n_cls)	
Dense1			n_cls	(n_cls)	

TFEB架構

by a single average pooling layer. The kernel sizes of the pooling layers are determined by $TFEB\_PS = \{(f(x_h, i), f(x_w, i))_k\}_{k\in\{1,2,\ldots,N\}}$ where x_h and x_w are the height and width of the input to TFEB block, i is the index of the pooling layer, N is the number of pooling layers, and $f(x, i)$ is defined by:

$$f(x, i) = \begin{cases} 2 & \text{if } x > 2 \text{ and } i < N \\ 1 & \text{if } x = 1 \\ \frac{x}{2^{(N-1)}} & \text{if } i = N \end{cases} \quad (2)$$

TFEB_PS

SFEB Maxpool Kernel Size Equation

$$SFEB\_PS = \frac{w}{((i\_len/sr) * 1000)/10}$$

Why does acdnet need to calculate maxpool layer kernel size?
It may because that the following layer‘s input size is fixed, as a result, acdnet need to calculate the kernel size for same input size.

TFEB Pooling Layer Kernel Size Equation

by a single average pooling layer. The kernel sizes of the pooling layers are determined by $TFEB\_PS = \{(f(x_h, i), f(x_w, i))_k\}_{k \in \{1,2,\ldots,N\}}$ where x_h and x_w are the height and width of the input to TFEB block, i is the index of the pooling layer, N is the number of pooling layers, and $f(x, i)$ is defined by:

$$f(x,i) = \begin{cases} 2 & \text{if } x > 2 \text{ and } i < N \\ 1 & \text{if } x = 1 \\ \frac{x}{2^{(N-1)}} & \text{if } i = N \end{cases} \tag{2}$$

Training Audio Data Mix Equation

We follow the data preprocessing, augmentation, and mixing of classes described in EnvNet-v2 [43] to produce training samples. 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

$$p = \frac{1}{1 + 10 * \frac{g1-g2}{20} * \frac{1-r}{r}} \tag{3}$$

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

$$S_{mix} = \frac{ps_1 + (1-p)s_2}{\sqrt{p^2 + (1-p)^2}} \tag{4}$$

Loss Function: Kullback-Leibler Divergence Loss: why?

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. Since we mix up class labels to generate samples, the mini-batch ratio labels should represent the expected class probability distribution. Hence, we use *KLDivLoss* (Kullback-Leibler Divergence Loss) as the loss function instead of cross-entropy loss [43]. We optimize it using back-propagation and

Kullback-Leibler Divergence Loss Function

function optimized is shown in Eq. 5. 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 $\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$.

$$L = \frac{1}{n}\sum_{i=1}^{n}(D_{KL}(y^{(i)}||f_\theta(x^{(i)}))) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_j^{(i)} \log \frac{y_j^{(i)}}{(f_\theta(x^{(i)}))_j} \quad (5)$$

The Equation

BC-Learning?

Loss Function: Kullback-Leibler Divergence Loss - explain

Kullback-Leibler（KL）散度损失是一种用于度量两个概率分布之间差异的方法。在机器学习中，通常用它来衡量模型预测分布与实际分布之间的差距。

假设我们有两个概率分布 P 和 Q，KL散度损失可以通过以下公式表示：

$$D_{KL}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

其中，$P(i)$ 是实际分布中事件 i 发生的概率，$Q(i)$ 是模型预测的事件 i 发生的概率。

KL散度损失的值越小，表示两个分布越相似。当两个分布完全相同时，KL散度损失为零。但需要注意的是，KL散度不是对称的，即 $D_{KL}(P\|Q)$ 与 $D_{KL}(Q\|P)$ 可能不相等。

在深度学习中，KL散度损失通常与其他损失函数一起使用，以帮助模型更好地学习目标分布。

Certainly! Kullback-Leibler (KL) Divergence Loss is a method used to measure the difference between two probability distributions. In machine learning, it is commonly employed to assess the disparity between the predicted distribution of a model and the actual distribution.

Suppose we have two probability distributions, P and Q. The KL Divergence Loss is defined by the following formula:

$$D_{KL}(P\|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Here, $P(i)$ represents the probability of event i occurring in the actual distribution, and $Q(i)$ represents the model's predicted probability of event i occurring.

The smaller the value of KL Divergence Loss, the more similar the two distributions are considered. A KL Divergence Loss of zero indicates that the two distributions are identical. It's important to note that KL Divergence is not symmetric, meaning $D_{KL}(P\|Q)$ may not be equal to $D_{KL}(Q\|P)$.

In deep learning, KL Divergence Loss is often used in conjunction with other loss functions to assist the model in learning the target distribution more effectively.

Why unstructured compression is not suitable for MCUs

Unstructured compression refers to compression techniques that do not take advantage of the specific characteristics or structure of the data being compressed. Instead, they treat the data as an arbitrary sequence of bits and attempt to reduce redundancy without considering the inherent patterns or relationships within the data.

Unstructured compression may not be suitable for microcontrollers (MCUs) for several reasons:

1. **Resource Constraints:** Microcontrollers often have limited resources, including processing power, memory, and storage. Unstructured compression algorithms might be computationally intensive and memory-demanding, making them impractical for use in resource-constrained environments.
2. **Lack of Adaptability:** Unstructured compression methods do not adapt well to different types of data. Microcontroller applications frequently involve specific types of data with known characteristics. Compression algorithms that can exploit the structure of the data can achieve better compression ratios and faster compression/decompression speeds.
3. **Processing Speed:** Unstructured compression may require complex algorithms that could be too slow for real-time applications commonly found in microcontroller-based systems. MCU applications often demand fast response times, making it important to use compression techniques that can operate efficiently within these constraints.
4. **Energy Efficiency:** Many microcontroller applications are battery-powered or have stringent power consumption requirements. Unstructured compression algorithms may consume more energy due to their general-purpose nature, making them less energy-efficient compared to compression methods tailored to the specific data characteristics.
5. **Limited Storage:** Microcontrollers typically have limited storage capacity. Unstructured compression may not be as effective at achieving high compression ratios as structured compression methods that exploit the known patterns and redundancies in the data.

For these reasons, when dealing with microcontrollers, it is often more practical to use compression techniques specifically designed for the types of data prevalent in embedded systems, taking into account the limitations and requirements of the target MCU platform.

Model Compression Techniques suitable for MCUs

Model compression techniques for microcontrollers (MCUs) need to consider the resource constraints of these embedded systems. Here are some model compression techniques that are suitable for MCUs:

1. **Quantization:**
    1. **Description:** Reducing the precision of weights and activations from floating-point to lower bit-width representations (e.g., 8-bit integers).
    2. **Benefits:** Reduces memory footprint and computational requirements.
    3. **Considerations:** Balance between compression ratio and model accuracy.
2. **Pruning:**
    1. **Description:** Removing or setting to zero a portion of the model's weights during training.
    2. **Benefits:** Reduces the number of parameters and computations.
    3. **Considerations:** Pruning criteria and threshold determination.
3. **Knowledge Distillation:**
    1. **Description:** Training a smaller student model to mimic the behavior of a larger teacher model.
    2. **Benefits:** Transfers knowledge from a large model to a smaller one, reducing the size of the model.
    3. **Considerations:** Trade-off between accuracy and model size.
4. **Weight Sharing:**
    1. **Description:** Grouping similar weights and sharing parameters.
    2. **Benefits:** Reduces the number of unique parameters.
    3. **Considerations:** Impact on accuracy and training complexity.

Model Compression Techniques suitable for MCUs (Cont.)

5. **Low-Rank Factorization:**
    - **Description:** Decomposing weight matrices into low-rank approximations.
    - **Benefits:** Reduces the number of parameters and computations.
    - **Considerations:** Balance between compression and loss of expressiveness.
6. **Knowledge Pruning:**
    - **Description:** Pruning the knowledge learned by a model, often through techniques like knowledge distillation.
    - **Benefits:** Reduces model size while preserving important knowledge.
    - **Considerations:** Proper balance to avoid loss of critical information.
7. **Winograd Convolution:**
    - **Description:** Transforming convolutional operations into a more computationally efficient form.
    - **Benefits:** Reduces the number of multiplications in convolutional layers.
    - **Considerations:** Trade-off between computation speedup and increased memory requirements.
8. **Binary and Ternary Networks:**
    - **Description:** Representing weights and activations using binary or ternary values.
    - **Benefits:** Drastically reduces memory and computational requirements.
    - **Considerations:** Balancing compression with accuracy trade-offs.

When selecting a compression technique for MCUs, it's important to consider the specific requirements of the application, such as the available memory, computational power, and the desired balance between model size and accuracy. Additionally, fine-tuning and optimization may be necessary to achieve the best performance for a given MCU platform and application.

L0-Norm unstructured model compression

The idea of sparsifying the weight matrices is inspired by the fact that bringing sparsity into the network reduces the chance of model over-fitting and enhancesthe model performance [73] . Furthermore, many researchers have shown that sparse pruning of neural networks often produces the same or even better accuracy than the base network.

Certainly! The L0-norm is a mathematical concept that represents the number of non-zero elements in a vector or a matrix. In the context of unstructured compression, the L0-norm is used as a regularization term to induce sparsity in a model's parameters. Specifically, it encourages a model to have fewer non-zero weights.

The L0-norm of a vector or matrix is defined as:

$$\|X\|_0 = \text{number of non-zero elements in } X$$

For a vector X, the L0-norm counts the number of non-zero elements in the vector. For a matrix, it counts the total number of non-zero elements across all entries in the matrix.

In the context of unstructured compression using the L0-norm, the idea is to apply sparsity to the weights of a model. By introducing the L0-norm regularization term into the model's training objective, the optimization process is encouraged to produce a solution where many of the weights become exactly zero.

Mathematically, the L0-norm regularization term is added to the standard loss function $J(\theta)$ as follows:

$$J_{\text{total}}(\theta) = J(\theta) + \lambda \|W\|_0$$

Here, W represents the weights of the model, $\|W\|_0$ is the L0-norm of the weights, and λ is a regularization parameter that controls the strength of the sparsity-inducing penalty. The optimization process, during training, seeks to find a set of weights that minimizes this combined objective.

It's worth noting that optimization with L0-norm regularization is computationally challenging because the L0-norm is a non-convex function and finding the exact solution involves combinatorial optimization. In practice, approximations and heuristics are often used to make the optimization more tractable.

In summary, unstructured compression using the L0-norm regularization encourages sparsity in the model's weights, leading to a more compact representation by forcing many weights to be exactly zero. This can be beneficial for reducing memory and computational requirements in certain scenarios.

ACDNet: Model Size Shrinking

1. First, applying unstructured L0-Norm to model.
2. Second, ACDNet adapts Pruning + Quantization techniques.
   - Guess: since the acdnet is essentially a cnn-based model, so the author applying the best established pruning-based model compression techniques proposed for computer vision to acdnet at first.
   - Then, since the quantization does not conflict with other model compression techniques, hence, the author applying 8-bit quantization to acdnet to further reduce the model size.
3. The above is the whole picture of acdnet model compression processes. The author put emphasis on testing different pruning techniques on acdnet.
4. ACDNet mainly adapted channel-pruning, and it uses three approaches to achieve channel-pruning: maganitude-base ranking, taylor criteria-based ranking, and proposed hybrid pruning approach.

Proposed Hybrid Pruning Approach

This pruning technique uses a new approach that combines both the unstructured and structured ranking methods to prune weights and channels from a network. In the first step, it prunes unimportant weights by zeroing them out from the network using unstructured global weight ranking methods, for example L0 Norm. In the second step, the model is further pruned through structured pruning. As we apply structured pruning on the weight-pruned network, we are not producing sparse matrices, which are generally not supported on embedded devices. In this step, the focus is to prune the important weights through channel pruning. The channels are first ranked by using structured channel ranking methods, such as magnitude-based ranking and Taylor criteria-based ranking. Then the lowest ranked channel is removed from the network and retrained the network (we term it as 'fine-tuning') for few epochs to recover the loss. This pruning and fine-tuning is iterated until the model reaches the target size. Then the existing weights of the resulting model is re-initialized and trained as a fresh network (we term it as 'scratch-training') instead of retraining the resulting model for higher classification accuracy (we term this as 're-training'). We present this method in the following form.

Method: Hybrid Pruning

Input: Trained Model, Target (size/percentage)
Output: Compressed Model
Data: Training Set, Test Set
Step 1:
- Sparsify the trained model

Step 2:
while *model_size* > *Target* **do**
- A full epoch forward pass
- Calculate the effect in gradient
- Update activation
- Layer-wise normalization
- Global ranking of channels
- Prune lowest ranked channel
- Fine-tune for 2 epochs

Step 3:
- Re-initialize the weights
- Scratch-training: Train the model from scratch

Findings

4.4.6. Experimental findings

Tables 8 and 9 provide the experimental results for the different pruning approaches tested on ACDNet. In Table 8 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).

Pruning channels 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 fine-tuning* (each iteration consists of pruning one channel and retraining the resulting network for two epochs) is not enough to achieve comparable prediction accuracy, which is contrary to earlier findings in the literature, e.g., in Molchanov et al. [32]. The experimental results (Table 8 and 9) show that the final compressed network requires full *re-training* or *scratch-training* to produce comparable prediction accuracy.

Table 8

Table 8
Models found after 80% channel pruning using magnitude-based ranking, Taylor criteria-based ranking and our hybrid pruning approach.

Model	Pruning Method	SFEB	TFEB	Parameters			FLOPs		Fine-tuning	CV Accuracy%	
No.				Count(M)	Size(MB)	Reduced%	Count(M)	Reduced%	Accuracy%	Re-training	Scratch-training
1	Magnitude		✓	0.119	0.45	**97.49**	36.94	93.22	15.25	81.80	82.30
2	Magnitude	✓	✓	0.119	0.45	97.57	36.94	93.22	15.25	82.45	82.30
3	Taylor		✓	0.135	0.52	97.15	11.03	97.97	18.00	77.45	80.05
4	Taylor	✓	✓	0.140	0.53	97.04	11.40	97.91	6.50	78.40	81.00
5	Hybrid-Magnitude		✓	0.120	0.46	97.47	36.96	93.21	12.00	81.85	82.30
6	Hybrid-Magnitude	✓	✓	0.118	0.45	97.50	36.81	93.24	12.50	82.10	82.40
7	Hybrid-Taylor		✓	0.142	0.54	97.00	8.22	**98.49**	23.75	80.30	80.85
8	Hybrid-Taylor	✓	✓	0.131	0.50	97.22	14.82	97.28	48.50	81.30	**83.65**

Table 9

Table 9
Models found after 85% channel pruning using magnitude-based ranking, Taylor criteria-based ranking and our hybrid pruning approach.

Model	Pruning Method	SFEB	TFEB	Parameters			FLOPs		Fine-Tuned	CV Accuracy%	
No.				Count(M)	Size(MB)	Reduced%	Count(M)	Reduced%	Accuracy%	Re-Trained	Scratch-Training
1	Magnitude		✓	0.065	0.25	**98.63**	22.61	95.85	3.00	79.65	79.65
2	Magnitude	✓	✓	0.065	0.25	**98.63**	22.61	95.85	3.00	79.20	80.15
3	Taylor		✓	0.072	0.27	98.48	6.09	98.88	13.25	72.95	76.60
4	Taylor	✓	✓	0.072	0.28	98.48	6.31	98.84	9.50	73.80	77.50
5	Hybrid-Magnitude		✓	0.066	0.25	98.60	24.82	95.44	10.75	78.85	80.85
6	Hybrid-Magnitude	✓	✓	0.065	0.25	**98.63**	25.39	95.34	6.75	79.90	80.60
7	Hybrid-Taylor		✓	0.079	0.30	98.34	5.01	**99.08**	15.50	74.10	77.20
8	Hybrid-Taylor	✓	✓	0.066	0.25	98.61	10.54	98.68	37.00	76.90	**81.40**

Table 2 and Table 11

Table 2
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.

Layers	Kernel Size	Stride	Filters	Output Shape	Block
Input				$(1, 1, w = i\_len)$	
conv1	(1, 9)	(1, 2)	x	$(x, 1, w = \frac{w-9}{2} + 1)$	SFEB
conv2	(1, 5)	(1, 2)	$x\prime = x * 2^3$	$(x\prime, 1, w = \frac{w-5}{2} + 1)$	
Maxpool1	(1, *SFEB_PS*)	(1, *SFEB_PS*)		$(x\prime, 1, w = \frac{w}{ps})$	
swapaxes				$(1, h = x\prime, w)$	
conv3	(3, 3)	(1,1)	$x\prime = x * 2^2$	$(x\prime, h, w)$	TFEB
Maxpool2	*TFEB_PS*[0]	*TFEB_PS*[0]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv4,5	(3, 3)	(1, 1)	$x\prime = x * 2^3$	$(x\prime, h, w)$	
Maxpool3	*TFEB_PS*[1]	*TFEB_PS*[1]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv6,7	(3, 3)	(1, 1)	$x\prime = x * 2^4$	$(x\prime, h, w)$	
Maxpool4	*TFEB_PS*[2]	*TFEB_PS*[2]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv8,9	(3, 3)	(1, 1)	$x\prime = x * 2^5$	$(x\prime, h, w)$	
Maxpool5	*TFEB_PS*[3]	*TFEB_PS*[3]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
conv10,11	(3, 3)	(1, 1)	$x\prime = x * 2^6$	$(x\prime, h, w)$	
Maxpool6	*TFEB_PS*[4]	*TFEB_PS*[4]		$(x\prime, h = \frac{h}{kh}, w = \frac{w}{kw})$	
Dropout (0.2)					
conv12	(1, 1)	(1, 1)	*n_cls*	(n_cls, h, w)	
Avgpool1	*TFEB_PS*[5]	*TFEB_PS*[5]		$(n\_cls, 1, 1)$	
Flatten				(n_cls)	
Dense1			*n_cls*	(n_cls)	
Softmax				(n_cls)	Output

Table 11
Micro-ACDNet architecture for input length 30,225 (approximately 1.51s audio @ 20kHz).

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 1

Table 1
SOTA models for ESC-10, ESC-50, US8K and AE datasets sorted by year of publication. **Abbreviations**: ATTN (Attention), CO (Cochleagram), CRP (Cross Recurrence Plot), CT (Chromagram), DGT (The Discrete Gabor Transform), ENS (Ensemble Model), FBE (FilterBank Energies), GT (GammaTone), LP (Log-Power Spectrogram), Mel (Mel Spectrogram), MFCC (Mel-Frequency Cepstral Coeficients), PE (Phase-Encoded), Raw (Raw audio wave), Spec (Spectrogram), TEO (Teagerfis Energy Operator).

			Accuracy (%) on Datasets			
Networks	#Channels	Input(s)	ESC-10	ESC-50	US8K	AE
Human [38]	-	-	95.70	81.30	-	-
Piczak-CNN [47]	Multi	Mel	90.20	64.50	73.70	-
GSTC⊕TEO-GSTC [48]	Multi	TEO, GT	-	81.95	-	-
GTSC⊕ConvRBM [40]	Multi	(PE)FBE	-	83.00	-	
FBEs⊕PEFBEs [49]	Multi	FBE	-	84.15	-	-
EnvNet [50]	Single	Raw	88.10	74.10	71.10	-
EnvNet-v2 [43]	Single	Raw	88.80	81.60	76.60	-
EnvNet-v2 + BC [43]	Single	Raw	91.30	84.70	78.30	-
GoogLENet [51]	Multi	Mel, MFCC, CRP	86.00	73.00	93.00	-
Kumar-CNN [52]	Multi	Mel	-	83.50	-	-
VGG-CNN+Mixup [53]	Multi	Mel, GT	91.70	83.90	83.70	-
AclNet (WM = 1.5) [44]	Single	Raw	-	85.65	-	-
TSCNN-DS [15]	ENS, Multi	Mel, MFCC, CT	-	-	**97.20**	-
Multi-stream [54]	Multi, ATTN	Spec	94.20	84.00	-	-
CRNN [55]	Multi+ATTN	Spec	-	85.20	-	-
FBEs⊕ConvRBM [40]	Multi	Spec	-	86.50	-	-
ESResNet [14]	Multi	LP	**97.00**	91.50	85.42	-
Multi-CNN [41]	Multi	Mel	-	89.50	-	-
WEANET [13]	Multi	Spec	-	**94.10**	-	-
CNN [42]	ENS, Multi	DGT, Mel, GT, CO	-	88.65	-	-
BNN-GAP8 [56]	Multi	Spec	-	-	-	77.90
CNN-CNP [57]	Multi	Spec	-	-	-	85.10
Method B [16]	Multi	Spec	-	-	-	**92.80**