MONASH University

Deep Learning for Bioacoustic Recognition on Microcontrollers

by

Md Mohaimenuzzaman

A Thesis Submitted for the Degree of Doctor of Philosophy at
Monash University in 2022

Department of Data Science and Artificial Intelligence,
Faculty of Information Technology

Abstract

Acoustic monitoring is crucial for the conservation and management of ecosystems and their flora and fauna. Traditional animal monitoring is based on passive acoustic recordings that are manually assessed by human experts, which adds a significant workload. To alleviate this burden, Machine Learning (ML)-based solutions for automating this process have been in the spotlight.

This research explores end-to-end Deep Learning (DL) techniques for species recognition using bioacoustic data from the Internet of Things (IoT) devices installed in remote wild areas. The reason is twofold. First, DL-based techniques extract hierarchical features automatically through multiple layers of abstraction. Therefore, it eliminates the very expensive and time-consuming process of handcrafting the features required in Non-Deep Learning (Non-DL)-based techniques. Second, we try to leverage the resounding success of DL in producing state-of-the-art (SOTA) performance for a wide range of applications to acoustic analysis.

Due to the unavailability of power sources and the internet in remote areas, the extremely resource-constrained IoT devices (battery powered, low memory, and low processing power) cannot send the data to the cloud server for recognition. As a result, the recognition activities must be performed locally on IoT devices. However, DL-based techniques are resource and time-intensive. To harness the power of DL in IoT, the DL models must be significantly resource-optimized.

This study focuses on significantly downsizing the DL model architecture without sacrificing accuracy. It develops a compression-friendly Deep Convolutional Neural Network (Deep-CNN) architecture for end-to-end raw audio classification, a structured compression technique, and a generic pipeline that compresses this Deep-CNN architecture to produce a tiny DL model suitable for deployment on a resource-impoverished Microcontroller Unit (MCU). Beyond theorizing, we demonstrate such a deployment practically.

One of the major challenges in biological applications is the availability of labeled real-world data for the training phase. We demonstrate how Active Learning (AL) can be used with our edge models to address this issue. Our approach goes beyond standard AL by incorporating feature extraction into the AL loop, and we show that this results in superior performance.

To demonstrate the relevance of our work for real-world applications, we go beyond standard benchmarks and test the applicability of our methods with real-world data from conservation biology projects.

Dedicated to
my parents - **Abdus Samad** & **Rezia Samad**
my wife - **Swopna Zaman**
and my angels - **Saarah** & **Zaarah Zaman**

Declaration

I declare that this thesis is an original work of my research and contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made.

Md Mohaimenuzzaman
November 18, 2022

Acknowledgements

Let me begin by expressing my gratitude to the Almighty, who created me and is the source of my existence. I shall be eternally grateful to my parents for their unconditional support and sacrifice, which paved the road for me to seek my undergraduate degree in computer science in 2002.

Thank you to my friend Mizanur Rahman and my teacher Prof. Muhammad Abdul Halim Shaikh, for helping me out when I could not afford to buy food during my undergraduate studies. In addition, I would like to express my gratitude to Markentile Bank and others who assisted me in covering the remaining tuition fee after the waiver during my undergraduate studies. It might have seemed insignificant to the rest of you, but for me, it meant a million times more. I would also like to thank my friend-cum-brother, Dr. Abdullah Arafat, for his unwavering support since the moment I walked through the Melbourne airport gate.

It all began 20 years ago and now I am on the verge of completing another significant milestone: my long-awaited Ph.D. This has been possible because of my supervisor, **Prof. Bernd Meyer**, who stepped up and took me under his wing after my initial supervisors left Monash at the end of my first year. I restarted on a brand new Ph.D. project based on applied Deep Learning that I had always wanted to work on. He was a constant source of support, guidance and encouragement that kept my spirits high throughout this long journey of overcoming enormous obstacles and dealing with real-world problems.

Dr. Christoph Bergmeir, my associate supervisor, deserves a special thank you for all of his help and advice throughout this time. He was always there to assist me when I needed guidance or clarification. I also appreciate Julie Holden's assistance with language at the beginning of writing this thesis.

I am grateful to the Australian Government for funding my Ph.D. through the Australian Government Research Training Program (RTP) Scholarship. I would also like to thank the Department of Data Science and AI for providing me with an extra year of funding to compensate for the year I lost in the begining.

I want to express my heartfelt gratitude to Lin Schwartzkopf and Slade Allen-Ankins from James Cook University Townsville for providing real-world data to work with to demonstrate the applicability of the methodologies presented in this research work.

Finally, I thank my wonderful wife Swopna for shouldering all family responsibilities, including caring for our angels Saarah and Zaarah. I am curious about how she manages everything she has been doing since 2010. Thank you so much for doing all of this for me. It would not have been possible without your support.

Contents

Abbreviations

AE	Audio Event
AL	Active Learning
AL-KNC	Active Learning using K-Neighbors Classifier
AL-LogisticReg	Active Learning using Logistic Regression
AL-RidgeC	Active Learning using Ridge Classifier
BaC	Bioacoustic Classification
BaR	Bioacoustic Recognition
BTF	Black-throated Finch
CNN	Convolutional Neural Network
CRNN	Convolutional Recurrent Neural Network
CV	cross validation
DeepFeatAL	Deep Active Feature Learning
Deep-CNN	Deep Convolutional Neural Network
DAN	Deep Acoustic Network
DAL	Deep Active Learning
DeepIcL	Deep Incremental Learning
DL	Deep Learning
DNN	Deep Neural Network
ESC	Environmental Sound Classification
ESC-10	ESC with 10 classes
ESC-50	ESC with 50 classes
ESN	Echo State Network
FCNN	Fully Connected Neural Network
FLOPs	floating point operations
IoT	Internet of Things
iWingBeat	InsectWingBeat

KD	Knowledge Distillation
K-NN	K-Nearest Neighbors
LR	Logistic Regression
MCU	Microcontroller Unit
MCUs	Microcontroller Units
ML	Machine Learning
Non-DL	Non-Deep Learning
NN	Neural Network
RAC	Raw Audio Classification
RF	Random Forest
RNN	Recurrent Neural Network
SDAL	Standard Deep Active Learning
SOTA	state-of-the-art
SED	Sound Event Detection
SVM	Support Vector Machine
TNN	Transformer Neural Networks
TS	Time Series
TSC	Time Series Classification
US8K	UrbanSound8k Dataset
95%CI	95% Confidence Interval
CE	Cross Entropy
KLD	KL Divergence
SGD	Stochastic Gradient Descent
ADAM	Adaptive Momentum Estimation

Glossary

fine-tune Retrain a Deep Neural Network (DNN) model with a small set of new data mixed with/without a similar amount of old data. 65, 67, 69, 71, 72, 80, 86, 96–98

fine-tuning Retrain a DNN model with a small set of new data mixed with/without a similar amount of old data. 24, 45, 65, 71, 72, 75, 86

fixed-freeze Fine-tune all layers except some specific layers. 72, 73

no-freeze Fine-tune all the layers of the network. 72, 73

retrain Retrain a DNN model with the full training set. 96–98

scheduled-freeze Gradually unfreeze layers from the end and fine-tune them. 72, 73

Chapter 1

Introduction

Intelligent sound recognition is one cornerstone of smart Internet of Things (IoT) applications ranging from technical safety [1–3], surveillance and urban monitoring [4–6] to environmental applications [7, 8]. Smart sensors driven by Microcontroller Units (MCUs) are at the core of these applications. In a typical IoT-based monitoring application, the MCUs installed at the edge of the network sense data and send it to the cloud for recognition. Transporting an unprecedented amount of increasingly diverse, high-dimensional, low-quality, and mostly unlabeled [9–13] recorded data from the edge of the network to the central server requires a significant amount of energy, time, and internet bandwidth.

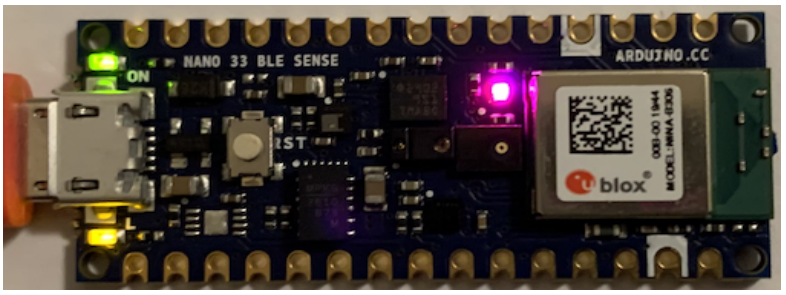
The focus of this study is on acoustic monitoring of animal vocalizations, which is a useful and well-established methodology in biodiversity management [14–17]. Traditionally, animal monitoring has been accomplished through passive acoustic recording and subsequent manual evaluation by human experts. Given how labor-intensive this is, AI-based solutions have recently attracted a lot of attention [14]. However, current AI-based animal monitoring systems require recordings to be uploaded to cloud-based platforms or high-powered desktop machines that provide a significant amount of resources to process AI audio models [16, 18]. Unlike traditional animal monitoring applications, this study focuses on an application where monitoring must take place in remote areas [15], posing some interesting technical challenges for automated audio recognition. The following section goes into detail about this research project.

1.1 Bioacoustic Recognition (BaR) on Microcontrollers

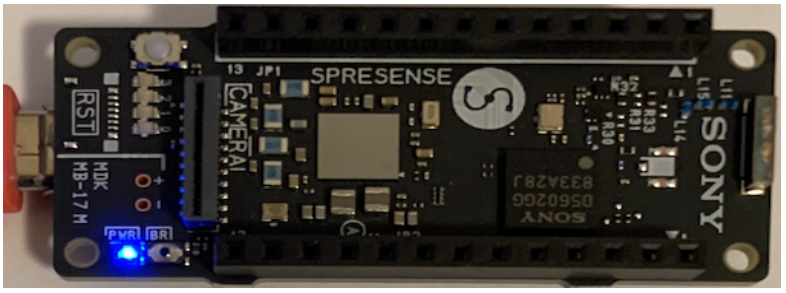
The goal of this research project is to perform end-to-end bioacoustic recognition of data sensed by MCUs installed in remote wild areas. Remote areas are constrained by limited power (typically battery) and insufficient or no network coverage, rendering the upload of recorded audio data impossible. Uplinking data via satellite communication is also not practically viable in the context of this research project, as the battery-powered MCUs possesses extremely little memory and processing power. Continuous long-term

monitoring, which is essential for biodiversity management [15], can thus only become a reality if recognition can take place directly on the monitoring devices in the field. Thus, our objective is to enable acoustic classification 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, as the recognition is performed directly in the field. Specifically, cost is a factor that commonly limits the available compute power and network bandwidth in edge applications.



(A) Arduino Nano 33 BLE Sense: 32-bit ARM Cortex-M4 FPU processor with direct memory access, 64MHz cpu clock, 256KB SRAM, 1MB Flash and a dedicated Bluetooth processor with BLE 5



(B) Sony Spresense Development Board: 32-bit ARM Cortex-M4F processor with direct memory access, 156MHz cpu clock, 20uA@5V power uses, 1.5MB SRAM, 8MB Flash

FIGURE 1.1: IoT Development Boards

MCUs are low-powered resource-constrained devices typically based on system-on-a-chip (SoC) hardware with less than 1MB of RAM and slow clock speeds. GPU support is available on relatively high-powered specialized 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 [19] and the STM32F4 [20], is the 32-bit ARM Cortex M4F CPU, which typically runs at clock speeds below 200 MHz. Examples of such off-the-shelf devices include *Arduino Nano 33 BLE Sense* (see Figure 1.1A) [21] and *Sony Spresense* (see Figure 1.1B) [22].

Both devices are capable of running Deep Neural Network (DNN) models built using Tensorflow [23] and deployed on them using Tensorflow Lite Micro [24].

Since the MCUs of this research project do not have the luxury of sending the sensed data to the central server when needed, if a device fails to classify the sensed data (e.g., data from a new species on which they have not been trained), it must wait for the intermittent internet connection to send the data to the cloud server for further processing. Once the unlabeled data from the devices is received by the cloud server, human experts manually label the data. The Machine Learning (ML) model used for recognition in the embedded device is then retrained in the cloud server and sent to the device, replacing the old model with the retrained one. The problem is depicted in a structured manner in Figure 1.2, as are various paths toward potential solutions and the scope of this research project.

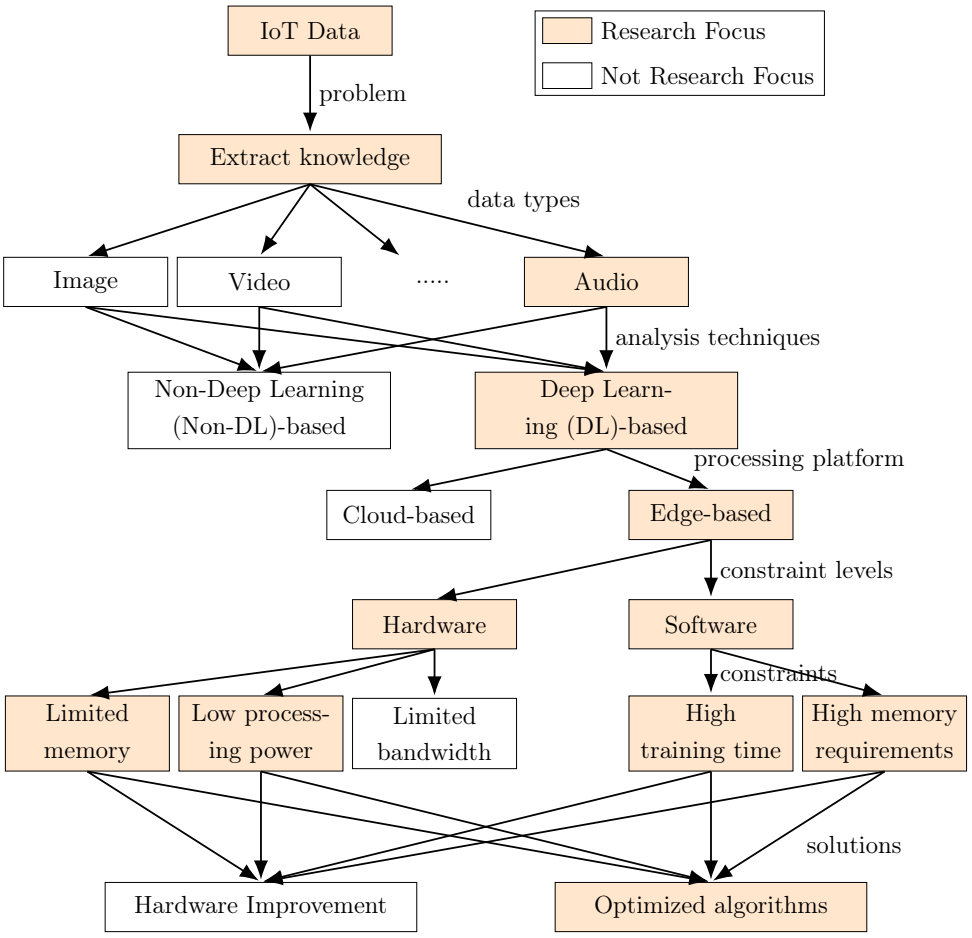


FIGURE 1.2: Breakdown of the research problem and the scope of this research project

1.2 Processing Audio Data

1.2.1 Data Representation

Audio can be represented in the time domain as a wave form of amplitude changes over time (Figure 1.3). If this time series is directly used as an input to the network, we speak of raw audio classification.

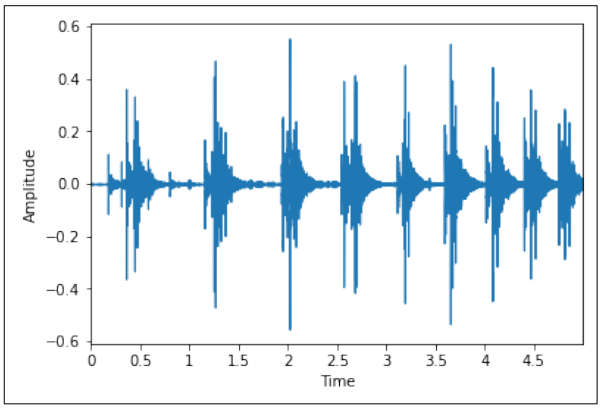
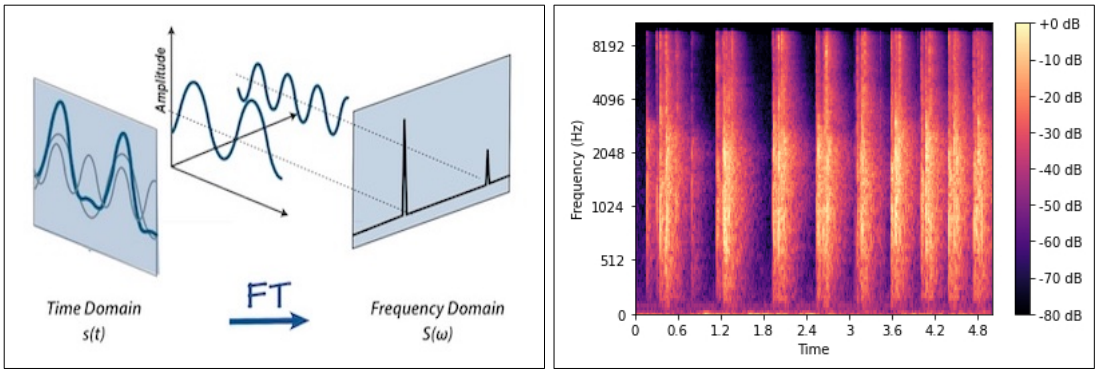


FIGURE 1.3: 1-D representation of audio as wave

An alternative is to use spectrograms. For this, the audio is first transformed into the frequency domain using Fourier transforms [25] of short overlapping windows and presented with respect to time, frequency, and amplitude (Figure 1.4A). The spectrogram captures the intensity of the different frequency component of the signal against time (see Figure 1.4B).



(A) Audio with resprct to time, frequency and amplitude [26]

(B) Spectrogram

FIGURE 1.4: 2-D representation of audio

1.2.2 Feature Extraction

Feature extraction is the process of generating the most representative features from a high-dimensional feature space while retaining the useful information from the input feature space [27, 28]. This simplifies the data by representing each variable as a (linear or non-linear) combination of the original input variable [27, 28].

There are diverse audio feature extraction strategies. A few of the commonly used methods are openSMILE [29] open-source toolkit, acoustic indices [30], spectrogram (e.g., Mel Frequency Cepstral Coefficients (MFCC) including its first and second-order derivatives, Log Power Mel Spectrogram (LPMS), Chromagram, statistics of MFCCs in each audio segment, etc), pretrained DNN models such as VGGish [31], learnt dictionary-based techniques such as Gabor Dictionary [32] and a mixture of hand-crafted features along with automatic features from the time-frequency domain of the audio signal used

in [33]. Many studies, including [34], [35], and [36], use raw audio Time Series (TS) as input to their DNN models and let the models extract the features automatically through various layers.

1.2.3 Feature Processing

Inspired by the tremendous success of DNN in computer vision, researchers have used spectrograms directly as input representations to the standard computer vision techniques [37–40]. Surprisingly, the results obtained thus far have not matched the performance that might have been expected based on the state-of-the-art in visual image processing [38, 41]. When applying DL to any problem, the representation of the input is a key decision.

While spectrograms are visual structures, their properties differ from those of natural images. A pixel of a specific color or similar neighboring pixels in an image may often belong to the same visual object. However, while frequencies move together according to a common sound relationship, a specific frequency or several frequencies do not belong to a single sound [38, 41]. Furthermore, while both axes in images carry spatial information, the axes in spectrograms have different meanings. Sounds are not two-dimensional static objects like images; they genuinely are TS. As a result, the invariances in natural images and spectrograms differ fundamentally. For example, moving a face in any direction does not change the fact that it is the same face. However, shifting frequencies upwards may not only transform an adult's voice into that of a child's voice or something completely different, it may also alter the spatial information of the sound [38].

Hence, this research considers audio classification using raw audio time series, an approach that has proven to be successful with traditional full-precision networks [34–36, 42].

1.3 Machine Learning (ML) Approaches for Time Series Classification (TSC)

Time Series Classification (TSC) problems can be handled using the two major categories of ML-based approaches, namely, distance-based and feature-based approaches [43]. The feature-based approaches work based on the extracted features from the TS. These features can be extracted through various techniques discussed in Section 1.2.2. The features are then used to learn using Non-DL-based approaches as well as DL-based approach that includes various types of DNN algorithms. Figure 1.5 presents the TSC approaches for a concise overview. Chapter 2 provides a detailed overview of the algorithms involved in the above approaches.

This research project requires species recognition and incremental feature learning from audio data on the MCUs at the edge. At the same time, the MCUs have to be comprised of ML algorithms that can function under severely constrained settings. In contrast, the

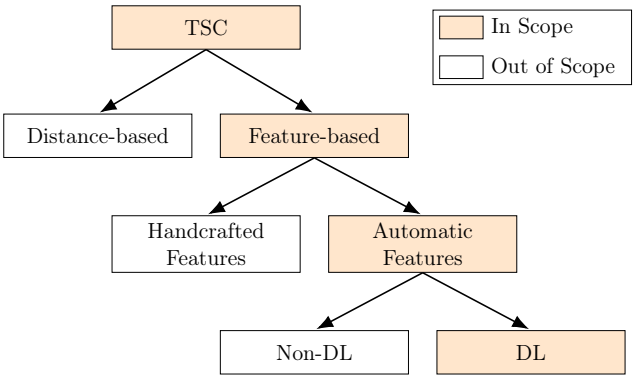


FIGURE 1.5: Time Series Classification (TSC) Approaches

state-of-the-art Non-DL-based algorithms are power-hungry and require excessive runtimes [44, 45]. The current state-of-the-art algorithm for TSC has 37 classifiers having the nearest neighbor as its essential component that needs to scan the whole training dataset before drawing an inference. To avoid this issue, Chen et al. [46] use Bayesian Classifier for their Bioacoustic Classification (BaC) instead of using any state-of-the-art techniques. Their method achieves 98.27% classification accuracy for binary classification. However, the accuracy is significantly compromised with the increase in the number of classes (e.g., 97.31% for 4-class and 79.44% for 10-class). Furthermore, their method requires hand-crafted features. In fact, feature-based Non-DL techniques mostly depend on hand-crafted features that require domain experts for feature engineering, which is very slow and expensive, especially when the data has complex characteristics. Hence, Non-DL-based state-of-the-art machine learning algorithms are either not scalable for large datasets with high dimensional data (e.g., IoT data) [47] or fail to satisfy the analytic needs of this project. Thus, using the state-of-the-art Non-DL-based algorithms for TSC at the edge is computationally infeasible.

Unlike Non-DL-based approaches, DL-based techniques extract hierarchical features automatically through different layers of abstraction. They have produced state-of-the-art results in a variety of application areas, including computer vision and machine listening. Despite being resource and time-intensive (for training), DL-based techniques outperform Non-DL-based methods in terms of accuracy. Thus, the purpose of this study is to explore DL-based approaches. However, there are currently several challenges that form the focus of this research: i) how to find the DL-based methods and architectures that best suit the acoustic recognition task, ii) how to fit the large energy-hungry DL model to resource-constrained embedded devices, and iii) how to label new data and incrementally learn from it.

1.4 Research Objectives

To address the above challenges, this research project aims to find a Microcontroller Unit (MCU)-compatible DL model that conducts the recognition task on the MCUs and can learn incrementally from the future data for improved prediction and adapt to changes in data characteristics for long term sustainability. To achieve these goals, this research has been divided into the following research objectives (ROs):

RO-1: Design a baseline DNN model architecture for fully-resourced computing environments for raw audio classification. This objective constructs a baseline DNN model architecture for fully-resourced computing environment that produces the best possible accuracy for end-to-end raw audio classification. Furthermore, the architecture must be compression-friendly for future optimization.

RO-2: Formulate a DNN architecture compression technique that is suitable to achieve the best-compressed version of the baseline model for the target MCU. This objective conducts a comprehensive study on different state-of-the-art DNN model compression techniques by applying them to various widely used benchmark datasets. It then works on optimizing the baseline model using the best compression technique to derive a tiny MCU-compatible model architecture for the target MCU. Finally, it deploys the tiny model on the MCU and runs the acoustic recognition task on the MCU.

RO-3: Devise the learning technique to be incorporated into the optimized model for incremental learning from new data. Due to the lack of training data to learn existing classes well, learn new classes, and adapt changes for long-term sustainability, the model needs to learn features from the new data continuously. For this, this objective investigates Deep Incremental Learning (DeepIcL) and Deep Active Learning (DAL) techniques and incorporates the most suitable method to the model.

RO-4: Demonstrate the applicability of the outcomes of the former objectives to real-world problems. To achieve the goal of this objective, this research implements the proposed methodologies to solve a real-world bird call identification problem using the MCU-compatible model with data from a conservation biology project that could not previously be analyzed automatically.

1.5 Contributions to Knowledge

The contributions of this research are as follows:

i. This research introduces the first Deep Acoustic Network (DAN) for running inferences on the extremely resource-constrained MCUs. This has been accomplished by

   - introducing a compression-friendly DAN architecture for end-to-end raw audio classification that achieves above state-of-the-art at the time of the study on a widely used 50-class dataset and near-state-of-the-art on other benchmark audio datasets.
   - presenting a hybrid structured compression technique to compress DANs. This is the first structured compression technique built specifically for DANs.
   - conducting a comprehensive study of current network compression techniques, such as structured pruning, Knowledge Distillation (KD), and XNOR-Net, to our proposed hybrid structured compression technique using the same benchmark datasets. It demonstrates that our method produces the best-compressed network.
   - introducing the first-ever generic pipeline that compresses a state-of-the-art DNN architecture to match the constraints of the target MCU and deploys that compressed model on the MCU.

ii. It presents a solution to a previously unsolved practical problem of rare bird call identification by incorporating semi-supervised learning in the minuscule MCU-compatible model. This has been accomplished by

   - introducing a deep Active Learning (AL) framework for bringing the feature extraction process inside the AL loop to refine it after each round of human annotation to improve the quality of the features.
   - demonstrating a successful implementation of the proposed deep AL framework on the minuscule DAN model to incorporate continuous feature learning, which is the first work of its kind.
   - finally, by successfully applying the presented approaches to solving the problem of rare bird call identification.

1.6 Thesis Structure

This thesis has been organized into seven chapters. The remaining chapters are organized as follows:

Chapter 2 presents a detailed literature review of the ML techniques this research project investigates. It presents overviews of various Non-DL-based and DL-based TSC approaches; it then provides a thorough review of the different DNN architecture compression techniques. Finally, it presents the review of continuous learning techniques in DL literature.

Chapter 3 illustrates our investigation and outcomes for RO-1 and a part of RO-2. It presents the systematic design of a DNN architecture for end-to-end raw audio classification for fully-resourced computing environment and a hybrid structured pruning technique to compress the proposed DNN model architecture. It then presents a pipeline for compressing the baseline DNN architecture to derive a MCU-compatible model, deploys and test the model on the MCU.

Extending the comparison of DL model compression techniques presented in Chapter 3, Chapter 4 presents a comprehensive study on various DNN model compression techniques, compares their performance and presents an analysis of their suitability to produce MCU-compatible tiny DNN models according to the problem characteristics.

Chapter 5 presents our work toward achieving RO-3. It first presents a comparative study on different incremental learning techniques followed by the presentation of our proposed deep AL framework and demonstrates incremental learning in the MCU-compatible model.

Chapter 6 demonstrates the implementation of the proposed methodologies by this research project to solve the real-world problem scenario.

Finally, Chapter 7 concludes this thesis with a summary, a list of overall contributions to knowledge, followed by a discussion of limitations and future work.

Chapter 2

Literature Review

2.1 Approaches for Time Series Classification (TSC)

There are two distinct TSC approaches: Non-DL-based and DL-based. The DNN algorithms are covered by the DL-based approach and the rest are covered by the Non-DL-based approach. In general, the state-of-the-art TSC algorithms fall under Non-DL-based approaches. We briefly discus both the TSC approaches in the following sections.

2.1.1 Non-Deep Learning (Non-DL)-based Approaches

Many TSC algorithms have been developed in diverse areas of research. Bagnall et al. [48] grouped TSC algorithms into four categories based on the different types of discriminatory features they work on. Elastic Ensembles (EE) [49], Shapelet Transform (ST) [50], Bag of SFA Symbols (BOSS) [51], and Collection of Transformation Ensembles (COTE) [52] are the state-of-the-art methods in the four categories, respectively. However, the Hierarchical Vote COTE (HIVE-COTE) [53] and Proximity Forest [45] are the two recent publications that also claim to be the new state-of-the-arts. In the following sections, we will present an overview of each method, then go on to the tree-based TSC strategies that are not found in any existing state-of-the-art techniques. Furthermore, we discuss some other ML techniques that are not state-of-the-art techniques but are very common in BaC domain.

2.1.1.1 Nearest Neighbor (NN) and Elastic Ensembles (EE)

The k-Nearest Neighbors [54] approach has been widely used for many years as a supervised ML algorithm for classification and regression. It compares the distance between two series to determine their similarity. This algorithm calculates the distance between the unlabeled data and the entire training set whenever a TS with an unlabeled class is provided for classification. The class of the TS in the k data points closest to the unlabeled data is used as the class for the unlabeled data.

Several distance-measuring methods have been developed to boost classification accuracy. The simplest is the Euclidean distance, which adds up the absolute difference between each point in the two TS being compared. In contrast, Dynamic Time Warping (DTW) [55] is widely adopted as the benchmark for distance measurement. This distinguishes between two comparable TS but does not belong to the same class. It is accomplished by finding the shortest path from a given point t in one TS to points t to $t + w$ in another. Despite being one of the oldest TS classifiers, 1-NN, along with DTW, is still considered the benchmark for classification accuracy [56]. The EE [49] technique is an ensemble of eleven distinct NN classifiers, each using one of eleven different distance measures. Due to its computational complexity of $O(n^2m^3)$[48], this TSC technique is often infeasible when the dataset is large since it calculates the distance between the test data and all the data in the training set at test time [44].

2.1.1.2 Shapelet Transform (ST)

For TSC, shapelet algorithms select the top k shapelets from a given TS. Time series shapelets by [57] initially arrange the shapelets in the closest location in the given TS and use the distance as the splitting criterion in a decision tree. The computational complexity of these algorithms is of order $O(n^2m^4)$ since they have to repeatedly scan the TS to find all the subseries for identifying a class. Furthermore, shapelets embedded in a decision tree do not aid in improving classification accuracy [48].

In contrast, the state-of-the-art shapelet transformation algorithm proposed by Hills et al. [50] achieves the same goal in a single run. In the next step, the shapelets are converted into a list of attributes, with each attribute standing in for one individual shapelet. The value of that shapelet is the shortest distance between the shapelet and the given TS. Although the converted dataset can be used with conventional classification techniques, these algorithms typically cannot handle massive datasets [44].

2.1.1.3 Bag of SFA Symbols (BOSS)

The BOSS [51] is a state-of-the-art TS classifier that belongs to a larger family of algorithms known as *bag of words*. It uses windows over a series to form words. In order to do this, it first employs Discrete Fourier Transformation [25] to truncate the series. Then it uses Multiple Coefficient Binning to locate the breakpoints for discretizing the truncated series. Finally, the discretized series is converted into words using the Symbolic Fourier Approximation. Though it is one of the most accurate state-of-the-art algorithms for TSC, its training time (i.e., $O(nm(n - w))$) is prohibitive for large datasets [48].

2.1.1.4 Collection of Transformation Ensembles (COTE)

As Bagnall et al. [48] point out, among the four state-of-the-art algorithms from various groups, the Collection of Transformation Ensembles (COTE) [52] is the frontrunner. This is an ensemble approach that includes 35 different classifiers, including two of the previously stated state-of-the-art classifiers, EE and Shapelet Transforms. The first 11 of the

35 classifiers are 1-NN models with varying distance measures. COTE transforms the incoming data into three different domains: frequency, rate of change, and shapelets. Each of these datasets is then used to train eight classic ML algorithms: Bayesian Network, Decision Tree, k-Nearest Neighbours, Naive Bayes, Random Forest, Rotation Forest, and Support Vector Machine.

Even though COTE has a lower error rate than the other methods we have looked at, it is not practical to apply when the dataset is large [48]. This has only been tried on smaller datasets found in the UCR repository. Since COTE is bounded by the computationally expensive Shapelet Transform and EE, its complexity is even higher than that of other similar methods [48].

2.1.1.5 The Hierarchical Vote COTE (HIVE-COTE)

Lines et al. [53] extended COTE to introduce the Hierarchical Vote System to COTE (HIVE-COTE) consists of 37 classifiers (the 35 classifiers from COTE plus two new classifiers). HIVE-COTE, by combining a new hierarchical structure with probabilistic voting, significantly outperforms COTE and replaces COTE as the state-of-the-art solution for TSC [53].

However, the authors recognize that HIVE-COTE has become computationally incredibly costly to reach high accuracy. It requires training 37 classifiers and cross-validating each of the hyperparameters for all of the classifiers. More specifically, HIVE-COTE includes COTE, which has shapelet transformation with computational complexity $O(n^2m^4)$ and EE with computational complexity $O(n^2m^3)$. With these factors in mind, it becomes difficult to train HIVE-COTE in some settings [45].

2.1.1.6 Decision Trees

Several research works that use tree-based algorithms for TSC are outlined in [45, 48], however, none of them have reached state-of-the-art. Their performance is generally worse than 1-NN with DTW [48].

Proximity Forest, an exception, developed by Lucas et al. [45], is a decision tree-based technique that integrates distance metrics and generates ensemble models of highly randomized trees. In contrast to the methods previously outlined, Proximity Forest scales massive datasets. When it comes to TS, traditional decision trees perform poorly since their branches divide based on attribute values, but in Proximity Forest, the branches split based on how close they are to the TS, improving both efficiency and accuracy.

2.1.1.7 Approaches Used in Bioacoustics

Many different ML algorithms are used in relatively simple Bioacoustic Recognition (BaR) tasks. Despite the fact that we have already established that we will use DL-based techniques to achieve automatic feature extraction, state-of-the-art performance for the complex task at hand and continuous learning, we will briefly cover some commonly

used methods such as Support Vector Machine (SVM) [58], Random Forest (RF) [59], K-Means [54], DTW [55], Hidden Markov Model (HMM) [60], Gaussian Mixture Models (GMMs), Multivariate Discriminant Analysis (MDA) [61] and Discriminant Function Analysis (DFA) [62] to acknowledge them in this thesis.

SVM-based methods are used by Fagerlund [63], Acevedo et al. [64] for bird call recognition whereas DTW is used by Somervuo et al. [65], Kogan and Margoliash [66]. Several other methods, including adaptive boosting [67] in [68], HMM in [65, 66], RF classifier in [69] and GMMs in [65], are also used for the analysis of species using bird calls.

Acevedo et al. [64] use SVM to analyze amphibian calls. Gradišek et al. [70] use RF classifiers to identify bumblebee species by analyzing their flight buzzing sounds. Clemins et al. [71] use HMM to analyze African elephant calls.

Roch et al. [72] employ GMMs, whereas Lin et al. [73] employ K-Means to analyze whale sounds. For dolphin call recognition, Steiner [74], Deecke and Janik [75], Frasier et al. [76], and [77] use MDA, DWT and GMMs, respectively. In contrast DFA is used for the same purpose by Oswald et al. [78].

Bianco et al. [79] provide a thorough examination of the ML methods used in acoustics. They further iterate that the most recent trends in bioacoustic is the use of DNNs, which reduce classification errors while also making the feature extraction easier.

2.1.2 Deep Learning (DL)-based Approaches

Due to its significant achievements in the fields of computer vision, speech recognition, and natural language processing, DL, also known as DNN, is one of the most active study areas of ML. DNN models have been particularly successful because they can learn hierarchical features on their own, are resilient to the curse of dimensionality, and can be trained in parallel using graphical processing units [44]. In the case of TSC, DNN models not only produce results close to COTE and HIVE-COTE [44], but also significantly surpass NN-DTW (see Section 2.1.1.1). In Figure 2.1, we see a general DNN architecture, where x_t represents uniformly spaced input timestamps.

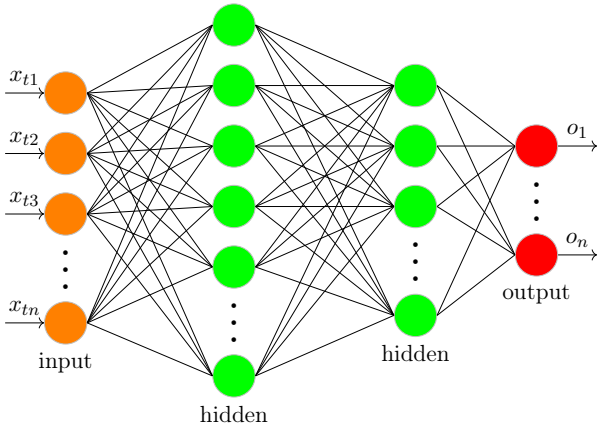


FIGURE 2.1: Fully Connected Neural Network (FCNN)

A DNN is built up of multiple layers, each of which is made up of neurons. The output of a layer is computed by multiplying the associated weights of the layer's neurons by the output of the preceding layer (see Figure 2.1), then applying the non-linearity/activation function $f(x)$ (e.g., ReLu, sigmoid, tanh etc) [80]. This is accomplished through a series of iterations, wherein each iteration comprises of a forward pass (or forward propagation) and a backward pass (i.e., backpropagation [81]). For example, if we have a Neural Network (NN) with input x, layers L, and an activation function f applied to each L, then the forward pass calculation may be expressed as follows:

$$f(x) = f_L(f_{L-1}(f_{L-2}(f_{L-3}, .., f_1(x)))) \tag{2.1}$$

The final step in the forward pass is to compute the loss, which is the difference between the predicted probability and the probability distribution provided in the training dataset. Categorical cross entropy [80], calculated using Equation 2.2, is an example loss function.

$$L(X) = -\sum_{i=1}^{n} y_i \, log \, \hat{y}_i \tag{2.2}$$

where L denotes the loss for classifying the input TS X, y_i is the class i out of n classes and $\hat{y}_i$ is the probability of X belonging to class y_i.

Following the completion of a forward pass, the estimated loss is optimized via backpropagation [81] by adjusting the node weights in each successive layer of the network. The gradients calculated using chain rule partial derivatives of the loss function with respect to those weight parameters determine the level of adjustment.

In the testing phase, the probabilistic classifier is tested with data it has never seen before. In this stage, only a forward pass takes place to determine the output of the final which is then used to make the prediction.

Although many different types of DNNs exist, the scope of this study is limited to the three most extensively used DNN architectures for TSC: Fully Connected Neural Network (FCNN), in which all layers are fully connected layers, Recurrent Neural Network (RNN), in which most of the layers are RNN layers, Convolutional Neural Network (CNN), in which most of the the layers are convolution layers, and the Transformer Neural Networks (TNN), a recent advancement of DNN, in which the blocks consists of attention layers and fully connected layers.

2.1.2.1 Fully Connected Neural Network (FCNN)

FCNN is the simplest and most common DNN architecture, in which every neuron in one layer is connected to every neuron in the following layer. A FCNN has the same architecture as shown in Figure 2.1.

Each neuron in a layer has a weight associated with it, which is updated by forward and backpropagation cycles. During forward propagation, the output of a layer is calculated

as:

$$A_{l_i} = f(w_{l_i} * X + b) \tag{2.3}$$

where A_{l_i} is the activation of i^{th} layer l, X is the input TS, w_{l_i} is the matrix of all weights and b is the bias term (Note: $i \in [1, n]$).

The final layer of a DNN model for TSC is often a discriminative layer with a softmax activation function (equation 2.4). This layer takes the preceding layer's output and uses a probability distribution over the class labels to decide the label for the input TS. The rationale for adopting softmax at the final layer is as follows: the sum of probability distributions is guaranteed to be 1, it is differentiable, and according to Fawaz et al. [44] it enables multinomial logistic regression adaption.

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}} \tag{2.4}$$

However, the issue with adopting FCNN for TSC is that it treats the TS elements in isolation, which causes the association between them to be lost [44].

2.1.2.2 Recurrent Neural Network (RNN)

Another popular DNN architecture is RNN (see Figure 2.2) which is mainly used for TS forecasting. The basic version of RNN is very rarely used for TSC due to the fact that it suffers from the vanishing gradient[1] problem while training long TS and it is designed to predict output for each timestamp of a TS [82]. Furthermore, it is difficult to distribute the training, resulting in a very slow process [83].

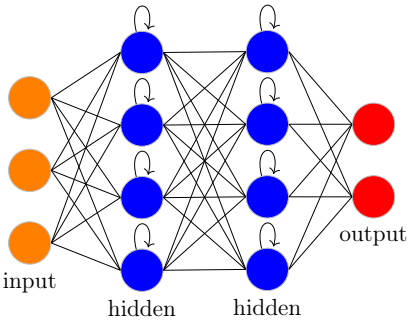


FIGURE 2.2: Recurrent Neural Network (RNN)

To avoid vanishing gradient, different variations of RNN such as long short term memory (LSTM), gated recurrent (GRU) and Echo State Network (ESN) [84] are introduced. A RNN is comprised of randomly initialized hidden layers [44] where the output weights are learned using algorithms like logistic regression. The hidden state can be calculated using equation 2.5 where $I(t)$ is the internal hidden state, W_{in} is the weight matrices for input TS, and $X(t)$ is the vector for input TS. The output is calculated according to equation 2.6.

$$I(t) = f(W_{in}X(t) + WI(t-1)) \mid t \in [1, n] \tag{2.5}$$

[1] when the gradients of a loss function approache to zero making the training of the NN hard

$$\hat{Y}(t) = W_{out} I(t) \tag{2.6}$$

Although LSTM, GRU and ESN solve the vanishing gradient problem of RNN, the lack of parallelization still persists. Furthermore, RNNs do not use all surrounding context when encoding a word.

2.1.2.3 Convolutional Neural Network (CNN)

The TSC researchers have been inspired to adopt CNN for TSC from its resounding success in computer vision, speech recognition, and natural language processing. A CNN consists of an input layer, convolution layers, pooling layers, fully connected layers and an output layer at the end [80]. Figure 2.3 represents a typical CNN architecture where x_t denotes evenly spaced input time stamp.

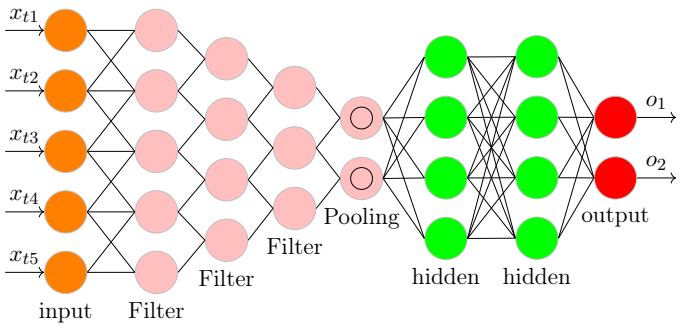


FIGURE 2.3: Deep Convolutional Neural Network (CNN)

A Convolution is the point-wise multiplication of data with the filters (i.e., a matrix that moves over the data, also known as kernels or feature detectors) associated with a specific layer to generate feature maps (also known as activation). See Section 9.1 of *Deep Learning* [80] for a more in-depth understanding of convolution. The advantage of using multiple filters is twofold: it allows you to learn multiple discriminative features across the TS and it requires far fewer model parameters than FCNN. Equation 2.7 represents the general form of convolution on an evenly spaced input time stamp X_t of a TS of length T with filter w of length l [44].

$$C_t = f(w * X_{t-l/2:t+l/2} + b) \,|\, \text{for all } t \in [1, T] \tag{2.7}$$

Following the success of CNN in Alexnet [85], that has 60 million parameters and 650,000 neurons in a total of 8 layers (5 convolution and three fully connected layers followed by a 100-class softmax output layer), research has begun to focus on making deeper models. The success of VGG-16 and GoogleNet strongly support the argument that *the deeper the model, the higher the classification accuracy.* However, this has introduced a number of issues: the model size has got bigger and bigger, training such large number of parameters has become computationally expensive, and vanishing gradient problem during back propagation.

2.1.2.4 Transformer Neural Network (TNN)

TNN is the recent advancement in DNN research. It was proposed by a Google-led team in 2017 in a widely cited paper titled *Attention Is All You Need* [86] for natural language processing (NLP), which addresses the issues with RNNs with the introduction of the multi-headed attention mechanism [87]. The attention mechanism allows the network to concentrate on a specific, small specific but important part of the input sequence. As the TNN is based on the attention mechanism, the recurrent architectures are no longer needed thus the earlier RNN, LSTM and GRU are replaced by the TNN.

Dosovitskiy et al. [88] from Google Brain team presented the first Vision Transformer (ViT) in 2020, influenced ny the NLP TNN. ViT solely employs the encoder component of the original NLP TNN. A FCNN is added before the softmax layer at the end of the encoder network [88]. TNNs for audio are essentially the same as ViT. Audio spectrograms are used as input to ViT instead of real images. Figure 2.4 shows the overall structure of TNN for audio spectrograms. The first implementation of TNN in audio classification called Audio Spectrogram Transformer (AST) [89] was published in 2021.

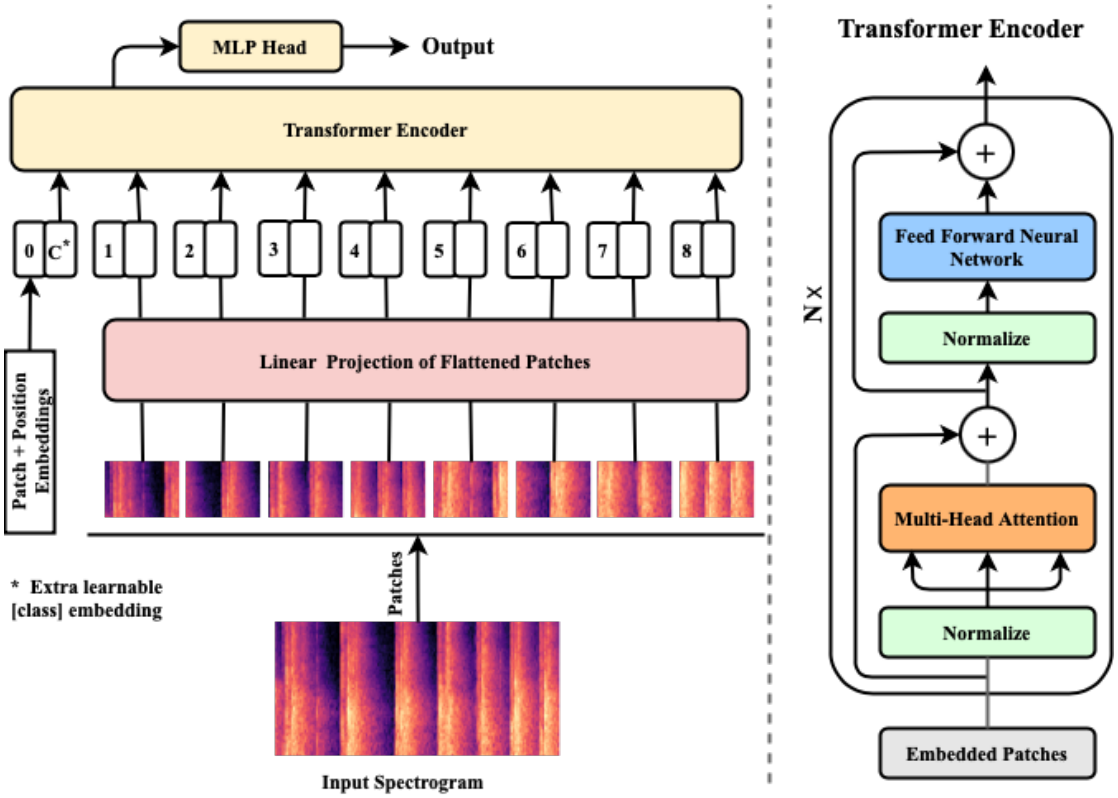


FIGURE 2.4: Transformer Neural Network for Audio (modified version of [88])

According to Figure 2.4, the spectrogram patches are projected onto a lower-dimensional space by the first layer of the audio TNN. The patches are then embedded with their learned position. These embedded patches are subsequently fed into the transformer encoder. The encoder uses attention mechanism that allows audio TNN to include relevant information across the whole image, even in the lowest layers [88].

According to the authors of ViT, it performs well when trained on large datasets (14M-300M images) and then applied on small or medium-sized tasks [88]. Furthermore, the performance of ViT depends on optimization, dataset specific hyperparameters and network depth [88], whereas CNNs are much easier to optimize.

2.2 Deep Model Compression

Model compression is a technique for reducing the size of a model while maintaining performance. This can be done in two ways: either by compressing the data (such as weight and input) of the trained model or by compressing the architecture of the DNN model.

2.2.1 Data Compression

Data compression can be accomplished using various techniques such as *Weight Sharing*, *Huffman Coding* and *Quantization*. These approaches are discussed further below.

Weight Sharing [90] is a process that involves clustering the weights and figuring out the centroids of each cluster. Once the gradients for each cluster are added up, the learning rate is multiplied by the sum, and the sum is then subtracted from the centroids. Figure 2.5 depicts the weight sharing process. In some literature [90, 91], this process is also known as weight sharing by training quantization.

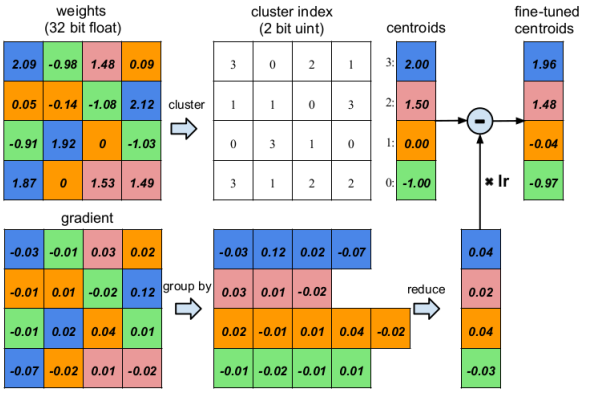


FIGURE 2.5: Weight sharing by training quantization (taken from [90])

Huffman Coding [90, 92] can be used for lossless data compression [92]. It uses a variable-length optimal prefix code to represent data with encoded symbols. The probability of each symbol appearing is converted into a table. Symbols that are used more often are presented with fewer bits in the table [90].

The process of reducing the number of bits used to represent weight is known as **Quantization**. This reduces both the memory requirements (e.g., representing 32 bit floating points to 8 bits) and the computational cost of DNN models. Extensive research has shown that 8-bit integers can be used to represent weights and activations [93]. However, quantization using lower than 8 bits, for example, the binary quantization technique has

shown to affect the accuracy significantly [93]. The built-in quantization techniques in TensorFlow [23] and PyTorch [94] DL frameworks, such as post-training quantization and quantization-aware-training, can be used to quantize the models built in the respective frameworks.

2.2.2 Architecture Compression

There are various techniques such as pruning and KD for generating a smaller DNN network from a large one. Pruning is a technique for removing unimportant connections, neurons, and filters from the original DNN network [95]. In contrast, KD is used to replicate the knowledge of large networks to smaller networks [96] The detailed discussion about these methods are presented below.

2.2.2.1 Pruning

Pruning is an iterative process in which the weights/channels/neurons are ranked in each iteration and the lowest ranked one is removed (see Figure 2.6) [90, 91, 95, 97, 98]. A trained model is pruned in a typical pruning technique, and the weights are then fine-tuned to minimize the loss of accuracy as much as possible. There are two types of pruning: unstructured pruning and structured pruning.

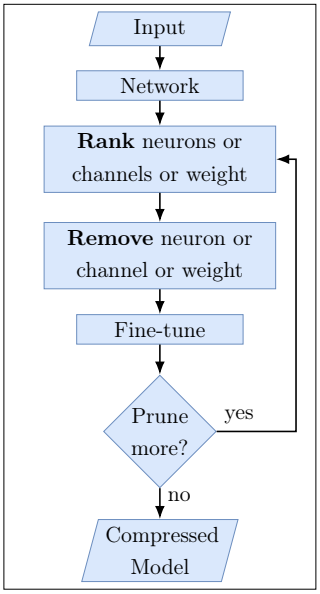


FIGURE 2.6: Iterative pruning steps

In an ***Unstructured pruning*** (i.e., weight pruning) process the unimportant weight values are removed from the weight matrices of different layers in the DNN model by setting them to zeros [90, 99, 100]. Only a few important weights that influence the model's decision making are left in the weight matrices. These matrices with many zeros are referred to as sparse matrices (Figure 2.7 right matrix).

There is a large amount of research work on unstructured pruning. The theoretical compression of the base DNN model provided by this pruning technique is astounding.

Dense Matrix

4	9	1	6	4	1	7	10	2	7
4	2	7	1	7	0	5	6	4	1
5	9	2	1	2	3	1	2	7	0
6	3	1	9	1	6	1	0	8	0
4	10	10	2	2	7	9	5	2	10
6	9	0	0	1	6	9	6	4	2
7	4	5	10	1	2	3	4	7	1
6	2	9	3	6	0	10	9	4	7
5	10	1	5	4	1	2	1	2	6
9	7	7	4	7	6	0	5	9	1

Sparse Matrix

.	9	.	.	.	.	7	10	.	7
.	.	7	.	7	.	.	.	.	.
.	9	.	.	.	.	.	.	7	.
.	.	.	9	.	.	.	.	8	.
.	10	10	.	.	7	9	.	.	10
.	9	.	.	.	.	9	.	.	.
7	.	.	10	.	.	.	.	7	.
.	.	9	.	.	.	10	9	.	7
.	10	.	.	.	.	.	.	.	.
9	7	7	.	7	.	.	.	9	.

FIGURE 2.7: Example dense matrix and sparse matrix

However, it generates sparse matrix models, which require special representation and hardware to perform the sparse computation. A sparse matrix can be represented in two ways: triplet/array representation and linked representation. Figure 2.8 depicts a triplet/array representation. According to the figure, each non-zero value usually requires 3x memory for implementation.

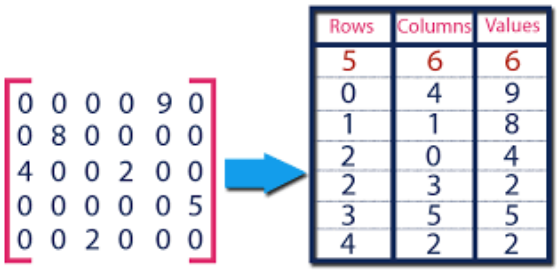


FIGURE 2.8: Triplet/Array Representation of sparse matrix (taken from [101])

Sparse matrix models are not ideal for direct implementation on embedded devices. According to the literature, all such compressed models were deployed on mobile devices or in devices with at least equivalent resources. Oktay et al. [99], for example, deployed their model in the Samsung Galaxy S7 (32GB flash drive and 4GB RAM). Edge-L^3, the only work we found on sound classification at the edge by Kumari et al. [100] at the time of this study, requires 12MB of run-time memory, which prevents it from being deployed in the device that this research project is aimed at. Furthermore, the MCUs lack the dedicated hardware and software support for sparse matrix computations required to capitalize on these theoretical savings [98, 102].

Given the issues with unstructured pruning, there is a tremendous amount of research being done in structured pruning of DNN architectures such as [91, 95, 98, 103–106]. ***Structured pruning***, as opposed to unstructured pruning, always results in dense matrices by removing an entire row or the entire weight matrix (Figure 2.7 left matrix) [91, 95, 98]. Structured pruning in FCNN removes unimportant neurons rather than just the connections. As a result, the entire row of a weight matrix for a layer is removed from the network. In the case of a CNN, a channel/filter is removed during a pruning iteration, removing the entire matrix that represents the filter's weights [91, 95].

The ranking of neurons or channels is determined using various methods such as the sum of absolute kernel weights [103], Taylor expansion criteria [95], particle filtering [98],

group convolution scheme [102], optimal dropout probability [91], lowest gradient loss [104], hybrid method(i.e., combining several of these methods) [105] and pair of filters with the highest correlation coefficient [106].

2.2.2.2 Knowledge Distillation (KD)

Knowledge Distillation (KD) is the process of replicating the knowledge of a large model to a smaller model [96] in order to mimic the behavior of the larger model. A large DNN model or an ensemble model has more learning capacity, which can then be transferred to a smaller model that can run on low-power hardware. This is considered as another type of structured compression technique. One way to achieve this is to perform forward passes on both the models during training, and then compute the cross entropy between their predictions and add it to the loss of the smaller model [107]. This, however, only works in classification tasks. A softmax function is typically employed in the final layer of the model. The softmax is used in the following form:

$$y_i(X|T) = \frac{e^{\frac{z_i(X)}{T}}}{\sum_j e^{\frac{z_j(X)}{T}}} \tag{2.8}$$

where T is temperature, normally which for a standard softmax is set to 1. Furthermore, the model assumptions are sometimes too strict to ensure competitive performance [93]. It also requires valid data to retrain (e.g., a portion of original dataset or a synthetic dataset derived from the feature representation of the teacher model or a transfer dataset) making it difficult to adopt a state-of-the-art trained model and compress it for a specific application.

2.2.3 XNOR-Net

XNOR-Net represents the weights in 1-bit and performs convolutions using bit-wise XNOR operations, thus saving significant amount of memory and computation. In an XNOR-Net [108], all layers except the first and the last are binary layers. The input, activations, and the weights of the binary layers are represented using either +1 or -1 and are stored efficiently with single bits. Figure 2.9 shows the construction of a typical convolution layer and an equivalent binary convolution layer.

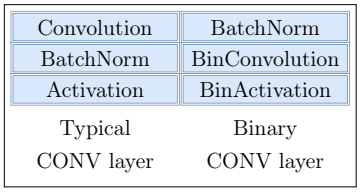


FIGURE 2.9: Typical convolution layer vs binary convolution layer

The convolutions between the matrices (input/activations and weights) are implemented using exclusive-NOR (XNOR) and bit-counting operations. For this, the convolution between two vectors $\in \mathbb{R}^n$ is approximated by the dot products between two vectors

$\in \{-1, +1\}^n$ [108]. Thus, convolution between the input X and weight W can be written as:

$$Z \approx (sign(X) \odot sign(W)) \odot \alpha\beta \tag{2.9}$$

where α and β denote the scaling factors for all the sub tensors in input X and weight W respectively and $\odot$ denotes element-wise multiplication. Due to the binary activations the dot product between $sign(X)$ and $sign(W)$ can be replaced by XNOR and pop-count operations which require extremely little computation. Thus, Equation 2.9 is reduced to:

$$Z \approx (X' \circledast W') \odot \alpha\beta \tag{2.10}$$

where $X' = sign(X)$, $W' = sign(W)$ with all zeros replaced by -1, and $\circledast$ denotes the XNOR and pop-count operations between X' and W'. This process is extremely efficient in terms of memory and energy usage. According to Rastegari et al. [108] XNOR-Net requires 32× less memory and reduces 58× computation requirements. Figure 2.10 provides an example of how the dot product between $sign(X)$ and $sign(W)$ is replaced by XNOR and pop-count operations between X' and W'.

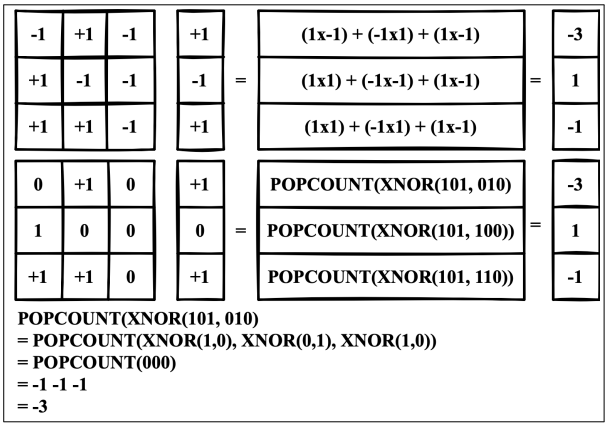


FIGURE 2.10: XNOR and POPCOUNT in XNOR-Net

Like structured pruning-based compression works, almost all the works based on XNOR-Net are proposed for computer vision. A few recent state-of-the-art XNOR-Net models for the benchmark image datasets are: [108] for MNIST, [109] for CIFAR-10 and CIFAR-100 and [110] for ImageNet.

2.3 Incremental Feature Learning

Most real-world applications start with a scarcity of labeled data. A DNN model must constantly learn from incoming data to improve performance or even keep the performance level the same on the ever-changing data characteristics. Waiting until the MCU's battery is replaced before gathering new data to retrain the model would be prohibitively long. It is preferable to recover the stored data as soon as feasible at the start of the project. Several techniques for dealing with these new learning problems include Incremental Feature Learning and Active Feature Learning.

2.3.1 Incremental Learning

Incremental learning is a form of learning that occurs whenever new examples emerge and previously acquired knowledge must be modified in response to the new examples. When a model gains additional knowledge from new data of known classes, this is known as online learning [111]. Class-incremental learning, on the other hand, refers to the process of learning new classes [112].

The model needs to learn from new data without forgetting what it has already learned. For this, the model needs to have access to both old and the new data. When the model does not have access to any old data, it will only be optimized for the new task, and its performance on the old task can drop significantly. This kind of forgetting is called catastrophic forgetting [113]. To solve this problem, we must find the best balance between being rigid (good at old tasks) and flexible (good at new tasks).

Many approaches [114–120] exist in the current literature to address the catastrophic forgetting problem in incremental learning. For example, Rebuffi et al. [114] propose rehearsal learning, which suggests storing some old data to blend with new data during the incremental learning training and fine-tuning process. Shin et al. [115] and Kemker and Kanan [116] proposed storing the class distribution rather than the old data, which a generative model uses to generate old class data on-the-fly.

Another issue with incremental learning is the imbalance between new and old classes. This class imbalance encourages the model to overestimate new classes while underestimating old ones, worsening the catastrophic forgetting. Castro et al. [121] train their model with this class imbalance in mind but fine-tune it by under-sampling the old and new classes to have as many images as possible. Guo et al. [122] use a small linear model and apply it only to the new class logits to handle the overconfidence problem, whereas Wu et al. [123] recommend re-calibrating the model on some old data. A similar solution, but using class statistics, was proposed by Belouadah and Popescu [112].

However, incremental learning requires labeling all the new data used to retrain or fine-tune the models to enhance their knowledge. It does not consider reducing the effort of labeling new data.

2.3.2 Active Learning (AL)

AL is a semi-supervised machine learning technique [124, 125] in which a learning algorithm collaborates with the user to label the data that it wishes to learn from [126, 127]. The basic idea behind the learning concept is that if a ML algorithm is allowed to choose which data to learn from, it may be able to improve its accuracy while using fewer training labels [126, 127]. Typically, the AL algorithms request human annotators to label the data instances for learning. Incorporating humans into the learning process transforms AL in to powerful example of the human-in-the-loop paradigm. This method of iterative

learning method aims to accelerate learning, particularly when a large labeled dataset is unavailable for traditional supervised learning [126, 127].

The essential components of an AL technique are extracting features, selecting samples for human annotation, training a classifier with the annotated data, and labeling the remaining samples with the trained classifier [124, 126–128]. This is an iterative process. The quantity of samples that can be manually annotated depends on the labeling budget for the specific project. Figure 2.11 depicts the process flow of a standard AL process.

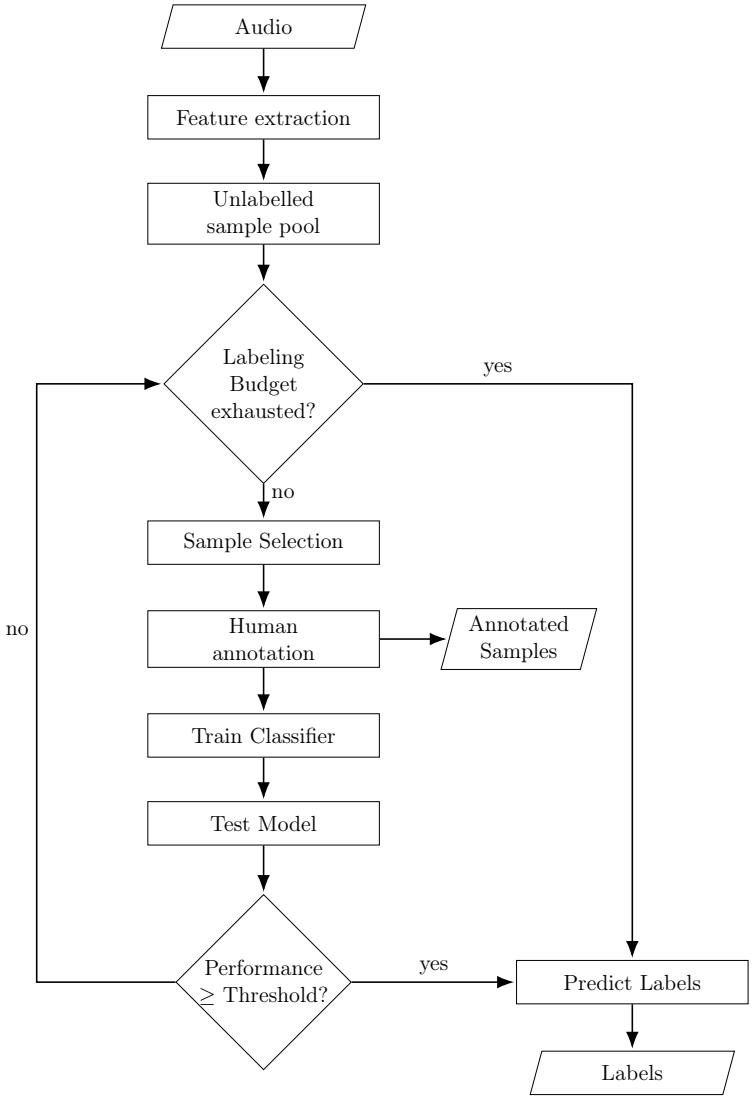


FIGURE 2.11: The detailed architecture of commonly used active learning system

Feature Extraction: Different audio feature extraction strategies are used in various research projects. We have already discussed these methods in Section 1.2.2. The main issue is that the feature extraction process is static and never changes. The question is whether fine-tuning the feature extractor while learning new features improves feature quality. This is, to the best of our knowledge, an unexplored question. We will investigate it in Chapter 5.

Sample Selection: There are a variety of sample selection techniques available for selecting samples for manual annotation. These are also referred to as acquisition functions. Random sample selection, uncertainty sampling [129] (e.g., least confidence, margin of confidence etc), diversity sampling (e.g., medoid-based AL [130], furthest traversal [131], cluster-based outliers, model-based outliers etc), query-by-committee [132] and expected error reduction [133] are the examples of such sample selection techniques.

Human Annotation: The samples selected by the acquisition function are handed over to a human expert for annotation. Rather than asking the human expert to annotate a specific sample, it is more common to present a segment of the audio signal containing the sample for human annotation.

Classifier for Labeling: The samples annotated by human experts are used to train classifiers in order to improve their capacity to predict the labels of the remaining data once the annotation budget has been exhausted. For classification and detection, numerous machine learning and AL algorithms are used. It is difficult to train a DL model to achieve acceptable performance because the manually annotated examples are typically insufficient in number. Many research projects (e.g., [29, 130, 134–137]) employ machine learning algorithms such as RF [59], SVM [58], K-Nearest Neighbors (K-NN) Classifier [54], and Logistic Regression (LR) [138].

2.4 Summary

This chapter covered Non-DL-based and DL-based TSC approaches, as well as their characteristics and applications. It is apparent that the bioacoustic recognition problem at hand demands the use of DL-based approaches. Section 2.1.2 studied different DNN-based approaches and their applicability in various application domains. In Chapter 3, we will dive deep into these approaches in order to devise a deep acoustic model.

The deep model architecture compression approaches (e.g., weight pruning, channel pruning, KD, XNOR-Net, and quantization) covered in depth in Section 2.2 have provided us with a solid background of how a deep model can be compressed to fit on MCUs to run inferences. The later half of Chapter 3 and the entire Chapter 4 explore these methods for practically compressing a deep acoustic model to meet the specifications of the target MCU.

The compressed model will then need to continuously learn new features using the continuous feature learning approaches outlined in Section 2.3. The existing research reviewed there clearly advocates for the use of Active Learning (AL) to solve the problem at hand. Chapters 5 and 6 will show how we can leverage these strategies to learn features on a continual basis.

Chapter 3

Baseline DNN Architecture and its MCU-Compatible Version

This study has been published in ***Pattern Recognition****. The article is titled as* ***Environmental sound classification on the edge: Deep acoustic networks for extremely resource-constrained devices.*** *[42]. The published paper has been significantly reworked for inclusion in the thesis.*

3.1 Introduction

In this chapter, we take steps toward achieving the goals of the first objective (RO-1) of this research project. To accomplish this, we must first define a baseline DNN architecture that is both compression-friendly and capable of producing the highest Raw Audio Classification (RAC) prediction accuracy possible. Then, we seek a suitable structured compression technique and a standard pipeline that begins with the baseline DNN model, compresses it according to the constraints of the target MCU, and then deploys the compressed model onto the MCU.

While many DL-based model compression pipelines/frameworks have been proposed for use in image [139–142] and video analysis [143], none have been proposed for the classification of sounds. There has also been considerable effort put into developing small and efficient CNN models for mobile devices [144, 145], smartphones, CPUs, and GPUs [146–148] that have multiple GBs of RAM, flash drive and GHz of CPU clock with multi core. In contrast, the embedded devices we envision have less than 1MB of SRAM, a few MB of Flash with less than 200MHz CPU clock. Therefore, further radical minimization is required for the aforementioned proposed methods.

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 [149]. Deploying DL models on such MCUs requires aggressive minimization techniques like model compression [90, 95, 150, 151], Knowledge Distillation (KD) [96]

and quantization [90, 107]. Some works have used such techniques for efficient models in computer vision [152, 153], but acoustic classification has not benefited from this yet.

We present an acoustic classification solution for energy-efficient edge devices that achieves close to state-of-the-art performance for raw audio classification on Environmental Sound Classification (ESC) datasets such as ESC with 10 classes (ESC-10) [154], ESC with 50 classes (ESC-50) [154], UrbanSound8k Dataset (US8K) [155], and Audio Event (AE)[156], the benchmarks that are used to assess large, non-resource-constrained networks. Importantly, we do not design the network specifically for these target devices. Rather, we design a compression pipeline that compresses and quantizes a large Deep Convolutional Neural Network (Deep-CNN) network into a network suitable for the target edge devices. Furthermore, the compression method is not constrained by raw audio-based networks only, it can handle a CNN network with any type of input. According to our knowledge, this type of work has not been done before for such a resource-constrained target device.

We first introduce ACDNet, a new, flexible and compression friendly sound classification architecture that, at the time of this study (i.e., in 2019), exceeds state-of-the-art performance on raw audio classification with accuracies of 96.65±0.06% and 87.10±0.02% on ESC-10 and ESC-50 datasets and close to state-of-the-art performance with accuracies of 84.45±0.05% and 92.57±0.05% on US8K and AE datasets, respectively. These performances are also very close to the overall state-of-the-art for the mentioned benchmarks (see Table 3.1). We then compress ACDNet using a novel network-independent compression approach to obtain an extremely small model (Micro-ACDNet). Despite 97.22% size reduction and 97.28% reduction in floating point operations (FLOPs), Micro-ACDNet still achieves classification accuracy of 96.25%, 83.65%, 78.28% and 89.69% on the same datasets, which are significantly higher than human accuracy (i.e., 81.30% for ESC-50) and still very close to the state-of-the-arts.

To classify 1.5s audio sampled at 20kHz, ACDNet uses 4.74M parameters (18.06MB) and requires approximately 544M FLOPs for a single inference. On the other hand, the micro version, Micro-ACDNet has only 0.131M parameters (0.50MB) and requires only 14.82M FLOPs for an inference, which is well within the capabilities of the MCUs we are targeting. We describe a successful deployment of a model for classifying 50 classes on a standard off-the-shelf MCU and, beyond laboratory benchmarks, report successful tests on real-world data (see Section 3.5).

To the best of our knowledge, this is the first time 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 generic conversion pipeline rather than hand-crafting a network for minimal size.

Hence, the contributions of this chapter are as follows: 1) it introduces the first generic pipeline for sound classification using DL in extremely resource-constrained MCUs; 2)

it presents ACDNet, a flexible and compression-friendly state-of-the-art DL model architecture for raw audio classification; 3) instead of hand-crafting a once-off solution, it introduces a novel Deep-CNN compression approach to obtain tiny models for MCUs; and 4) finally, after 8-bit quantization, we deploy Micro-ACDNet on a standard off-the-shelf MCU with successful tests on real-world data.

3.2 Related Work

A wide range of DL models for acoustic classification achieve state-of-the-art performance on different environmental sound classification benchmarks. Table 3.1 displays the state-of-the-art models proposed as of June 2022 for the four datasets discussed (ESC-10, ESC-50, US8K and AE). Unfortunately, the majority of them have not disclosed their model sizes and computation requirements adequately. In reality, none of the most recent state-of-the-art models for the ESC-10, US8K and AE datasets [156–160] have provided information on model sizes and computation requirements. The case is the same for the majority of the state-of-the-art models for ESC-50 [160–163]. This is not only our experience; another publication achieving state-of-the-art performance in the ESC-10&50 dataset by Guzhov et al. [158] has detailed this fact as well. The only few of the latest twenty-seven state-of-the-art models presented in Table 3.1 that have fully/partially detailed their computational requirements are EnvNet-v2 [35], AclNet [36], AST [89], CLAP [164] and HST-AT [165] (see Table 3.6). The high parameter requirement of EnvNet-v2 (101.25M) is partly due to the use of multiple dense layers [35] and AST (87M) [89], CLAP (190M) [164], and HST-AT(31M) [165] are due to the use of many encoder networks, which unfortunately do not lend themselves well to compression [98, 102]. Furthermore, the latter three of the aforementioned four models were released approximately after one and a half year after this research was conducted; therefore, they are not comparable to this work.

The work presented in this chapter was completed between the middle of 2019 and the middle of 2020. Table 3.1 shows that all the state-of-the-art approaches until 2021 resort to multi-channel image-like spectrogram-based inputs, multiple parallel blocks or ensembles models and attention mechanisms. While this demonstrates some success in improving performance, it also increases the input and model sizes significantly and thus, does not constitute an advantageous starting point for our purposes. The state-of-the-art list also indicates that no spectrogram-based state-of-the-art model ran on single-channel input in the past. The reason behind this lack of success in inputting audio as gray images (single-channel spectrogram) is that unlike vertical and horizontal axes of images, the axes (time & frequency) of spectrograms are not homogeneous and hence do not carry the same spatial information [5, 38, 41]. Using multiple channels to input different feature sets usually compensates for this problem; however, this merely makes the input size bigger than the available primary memory available in a typical MCUs. On the other hand, inputting audio as raw waveform does not require expensive hand-designed features, allows us to exploit better modeling capability and learning representations [5],

Networks	Year Published	#Channels	Input(s)	Accuracy (%) on Datasets			
				ESC-10	ESC-50	US8K	AE
Human [154]	2015	-	-	95.70	81.30	-	-
Piczak-CNN [166]	2015	Multi	Mel	90.20	64.50	73.70	-
Method B [156]	2016	Multi	Spec	-	-	-	**92.80**
GSTC⊕TEO-GSTC [167]	2017	Multi	TEO, GT	-	81.95	-	-
CNN-CNP [168]	2017	Multi	Spec	-	-	-	85.10
GTSC⊕ConvRBM [169]	2017	Multi	(PE)FBE	-	83.00	-	
FBEs⊕PEFBEs [170]	2017	Multi	FBE	-	84.15	-	-
EnvNet [34]	2017	Single	Raw	88.10	74.10	71.10	-
GoogLENet [171]	2017	Multi	Mel, MFCC, CRP	86.00	73.00	93.00	-
FBEs⊕ConvRBM [169]	2017	Multi	Spec	-	86.50	-	-
EnvNet-v2 [35]	2018	Single	Raw	88.80	81.60	76.60	-
EnvNet-v2 + BC [35]	2018	Single	Raw	91.30	84.70	78.30	-
Kumar-CNN [172]	2018	Multi	Mel	-	83.50	-	-
VGG-CNN+Mixup [173]	2018	Multi	Mel, GT	91.70	83.90	83.70	-
AclNet (WM = 1.5) [36]	2018	Single	Raw	-	85.65	-	-
TSCNN-DS [159]	2018	ENS, Multi	Mel, MFCC, CT	-	-	**97.20**	-
Multi-stream [157]	2019	Multi, ATTN	Spec	94.20	84.00	-	-
CRNN [174]	2019	Multi+ATTN	Spec	-	85.20	-	-
CNN [161]	2020	ENS, Multi	DGT, Mel, GT, CO	-	88.65	-	-
Multi-CNN [162]	2020	Multi	Mel	-	89.50	-	-
WEANET [163]	2020	Multi	Spec	-	94.10	-	-
BNN-GAP8 [175]	2020	Multi	Spec	-	-	-	77.90
ESResNet [158]	2021	Multi	LP	**97.00**	91.50	85.42	-
AST [89]	2021	Single	Spec	-	95.60	-	-
VIP∼ANT [160]	2021	Single	Mel	-	95.70	86.00	-
CLAP [164]	2022	Single	log Mel	-	96.70	87.96	-
HTS-AT [165]	2022	Single	Mel	-	**97.00**	-	-

TABLE 3.1: state-of-the-art models for ESC-10, ESC-50, US8K and AE datasets. **Note: The grayed-out rows represent the DNN models that were published after the work in this chapter was completed (i.e., after 2019)**. **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 Coefficients), PE (Phase-Encoded), Raw (Raw audio wave), Spec (Spectrogram), TEO (Teager's Energy Operator)

yet is suitable for single-channel input for MCUs. Table 3.1 also demonstrates that, prior to the publication of TNN-based models in 2021, all the single-channel input based state-of-the-art models ran on raw audio waveform. Given the facts and circumstances, this study employs DL models that operate on a single-channel raw audio waveform.

Some recent work in computer vision that has specifically focused on very small CNN models is highly relevant to our work. MCUNet [153] starts from state-of-the-art image classification models, such as ResNet50 [176] and MobileNetV2 [144], 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 [152]) appears to be a very promising approach. It uses multi-objective search to automatically construct models small enough for MCU deployment. It optimizes 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 minimizes 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 [152] 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, [152] does, in fact, start from models that themselves are already constructed to fit on MCUs (Bonsai [177] and ProtoNN [178]). 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 before this study that specifically has the reduction of computational requirements as its primary focus is Edge-L^3 [100]. It achieves a model size of 0.814MB by pruning L^3-Net [179], 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 capitalize on these theoretical savings [98, 102]. Hence, structured compression techniques that produce dense matrix models promise to be a preferable approach for targeting MCUs.

In November 2021, a TNN-based work for the ESC task on edge devices was published (almost 1 year after the work in this chapter was submitted for publication). However, in the first version of the work [180], the model was tested on the ESC-50 dataset using a Samsung Galaxy S9 device with 4GB RAM and at least 64GB of memory. It only managed to achieve a low prediction accuracy (69%), making it incomparable to state-of-the-art accuracy. The authors later published a revised version of their work [181], in which they tested their model on a different dataset using a Raspberry Pi Zero W (also known as the Pi Zero) with a 1GHz single-core ARMv6 CPU and 512MB RAM. Both devices used in their works have orders of magnitude more resources than the MCUs targeted in this research (i.e., a MCU having less 1MB of SRAM, a few MB of Flash with less than 200MHz CPU clock).

In the following sections, we detail the construction of a flexible and compression friendly DL model that betters current state-of-the-art performance for raw audio classification on ESC-10, ESC-50, US8K and AE datasets (see Section 3.3.2.1 for details) while also providing a suitable starting point for compression, followed by a discussion of the structured compression methods used to minimize this for MCU deployment.

3.3 Proposed Base Network

We present a new model architecture (ACDNet) for acoustic classification that is smaller, flexible, compression-friendly and more efficient than the state-of-the-art networks at the time of this study, yet produces classification accuracy close to them and exceeds classification accuracy of the state-of-the-art models for raw audio classification on the most widely used ESC datasets ESC-10, ESC-50, US8K and AE.

3.3.1 ACDNet Architecture

Unlike other suggested state-of-the-art models, which use pre-extracted features and multi-channel input (such as hand-crafted features and spectrograms), the ACDNet architecture concentrates solely on feature extraction through convolution layers. All the convolution layers are followed by a batch normalization and a ReLU activation layers. The maxpool layers have strides equal to their pool size to avoid overlapping. The network is fed with raw audio time series (i.e., single-channel input). ACDNet consists of two feature extraction blocks followed by an output block. The feature extraction blocks are Spectral Feature Extraction Block (SFEB) and Temporal Feature Extraction Block (TFEB). Figure 3.1 depicts the layer structure of different blocks of the ACDNet architecture.

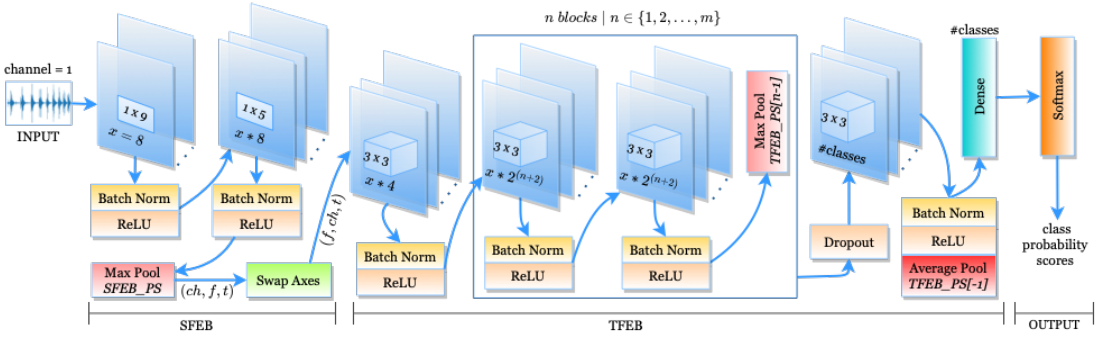


FIGURE 3.1: Visual representation of ACDNet model architecture.

SFEB consists of two 1-D strided convolutions followed by the first pooling layer. This block extracts low level audio features (spectral features) from raw audio through convolutions at a frame rate of 10ms. The output of this block is achieved by downsampling the convolution output using the maxpool with kernel size determined by the following equation:

$$SFEB\_PS = \frac{w}{((i\_len/sr) * 1000)/10} \tag{3.1}$$

where w is the width of the output of previous layer, i_len is the input length and sr is the sampling rate.

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 high level features also known as hierarchical temporal features. TFEB consists of convolutions 3-12 in Table 3.2. The first convolution layer is followed by a

pooling layer. After that, the convolution layers (4-11) are stacked like VGG-13 [182] (two convolutions followed by a pooling layer). The final convolution layer (*conv-12*) is followed 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:

$$f(x,i) = \begin{cases} 2 & \text{if } x > 2 \text{ and } i < N \\ 1 & \text{if } x = 1 \\ \frac{x}{2^{(N-1)}} & \text{if } i = N \end{cases} \tag{3.2}$$

TFEB block ends with an average-pool layer followed by a dense layer. The dense layer has 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 dense layer is crucial for the compression of the network in a later stage. The output of the TFEB is fed to a softmax output layer for classification. Table 3.2 represents the optimal overall ACDNet architecture obtained after considerable analysis, as discussed in the subsequent discussion.

Layers	Kernel Size	Stride	Filters	Output Shape	Block
Input				(1, 1, $w = i\_len$)	
conv1	(1, 9)	(1, 2)	x	(x, 1, $w = \frac{w-9}{2} + 1$)	
conv2	(1, 5)	(1, 2)	$x\prime = x * 2^3$	($x\prime$, 1, $w = \frac{w-5}{2} + 1$)	SFEB
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)	
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]		($x\prime$, $h = \frac{h}{kh}$, $w = \frac{w}{kw}$)	TFEB
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 3.2: 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

While the structure of the first two convolution layers has some similarity with EnvNet-v2 [35] and the other convolution layers with AclNet [36], EnvNet-v2 and AclNet architectures cannot flexibly be adapted to different lengths of audio recorded at different

sampling rates. Furthermore, they are not compression friendly for deriving tiny network models suitable for MCUs with comparable accuracy.

EnvNet-v2 has eight 1D convolutions and two 2D convolutions with 1056 filters, followed by two massive dense layers with 4096 neurons each. These dense layers produce approximately 100M parameters out of 101M parameters of the entire network. Hence, these two layers will be heavily pruned during the compression process, which would heavily impact the network's performance.

On the other hand, AclNet has twelve convolution layers with 3050 filters followed by the softmax output layer. The number of output filters from the last convolution layer always equals to the number of classes. Hence, this last convolution layer cannot be compressed. This might lead to pruning some more important filters from other layers, which may cause further accuracy loss. In addition to that, this may make the network structure pyramid-shaped which is wider towards the end when the number of classes is higher (e.g., 50 or 100). Such wide layers require extra primary memory during inference time that a typical MCU would struggle to provide.

ACDNet architecture is strongly motivated from the perspective of compressibility of the network and adaptability with respect to different audio lengths recorded at different sampling rates (see Table 3.2). It addresses the issue of EnvNet-v2 by adding more convolution layers and a tiny dense layer followed by the softmax output layer to allow the network to adapt to the changes in data shape when model compression is applied. The addition of a tiny dense layer also addresses AclNet's compressibility issue.

3.3.2 Experimental Setup

ACDNet is implemented in PyTorch [94] version 1.7.1 and *Wavio* audio library is used to process the audio files. The full code is available at: https://github.com/mohaimenz/acdnet

3.3.2.1 Datasets

The experiments are conducted on three of the most robust and widely used audio benchmark datasets - Environmental Sound (ESC-10, ESC-50 [154] and UrbanSound8k Dataset (US8K) [155]) and another audio dataset named Audio Event (AE) [156].

ESC-50 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 five splits is available, helping researchers to achieve unbiased comparable results in 5-fold cross validation (CV). ESC-10 is a subset of ESC-50 with ten classes and 400 samples equally distributed over the classes. US8K dataset contains 8732 labeled audio clips (<=4s) of urban sounds from 10 classes recorded at 22.05kHz. The clips are pre-sorted to 10 folds for 10-fold CV to achieve unbiased comparable results. The AE dataset has 5223 samples unevenly distributed over 28 classes recorded at 16kHz.

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 [183]. However, LifeCLEF itself is not suitable as a benchmark for the problem tackled here as this starts from strong labeling, whereas LifeCLEF explores weak labeling. As a real-world test, we evaluate ACDNet on a dataset of frog recordings[1] that is known to be challenging for conventional animal monitoring solutions [184, 185]. It contains 9132 field recordings of 10 different frog species across a variety of locations in Australia sampled at 32kHz.

3.3.2.2 Data Processing and Training Setup

For setting up the output filters of the convolutions of ACDNet, we set $x = 8$ to the $conv1$ layer of the $SFEB$ block (see Table 3.2. The network is trained and tested on 20kHz data with inputs of length 30,225 (approximately 1.51s audio). We downsampled the data to 20kHz in order to reduce input size, model size, and power consumption. We have not observed any difference in performance with audios re-sampled at a lower sampling rate.

We follow the data preprocessing, augmentation, and mixing of classes described in EnvNet-v2 [35] 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

$$p = \frac{1}{1 + 10 * \frac{g1-g2}{20} * \frac{1-r}{r}} \tag{3.3}$$

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

$$S_{mix} = \frac{ps_1 + (1-p)s_2}{\sqrt{p^2 + (1-p)^2}} \tag{3.4}$$

In the testing phase, we pad T/2 zeros to each side of the test input sample and then extract ten windows (each of length 30,225) at a regular interval of T/9 as input for the network. The input data for training and testing are normalized by dividing them by 32,768, 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. Since we mix up class labels to generate samples, the mini-batch ratio labels should represent the expected class probability distribution. Hence, we use *KL Divergence (KLD)* as the loss function instead of Cross Entropy (CE) loss [35].

[1]Data supplied by Lin Schwartzkopf, James Cook University Townsville

We optimize it using backpropagation 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* [186] 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 3.5. 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 $\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}$.

$$L = \frac{1}{n}\sum_{i=1}^{n}(D_{KL}(y^{(i)}||f_\theta(x^{(i)}))) = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_j^{(i)} \log \frac{y_j^{(i)}}{(f_\theta(x^{(i)}))_j} \tag{3.5}$$

3.3.2.3 Training and Testing ACDNet

We conduct 5-fold CV for ESC-10&50, 10-fold CV for US8K and 80-20 train-test split for AE over 10 independent runs.

To establish the statistical significance of comparisons between ACDNet and other state-of-the-art models, we cannot apply typical standard procedures, such as the Friedman test [187] with post-hoc [188] analysis or the ROC curve. This is because the implementations of the comparison models are not available, and we thus have to rely on published data only. Generally, only the mean accuracy is reported (without standard error), and only a few models have been tested on all four datasets. Instead, we calculate the 95% Confidence Interval (95%CI)[2] of classification accuracy of ACDNet for all four datasets to confirm the model's generalizability and reliability.

We run ACDNet 1000 times on the test sets of all four datasets using bootstrap sampling with replacement (Equation 3.7) and then construct the 95%CI (Equation 3.6) of classification accuracy.

$$CI = \mu \pm 1.96 * \frac{\sigma}{\sqrt{N}} \tag{3.6}$$

where μ is the mean accuracy of all the tests, σ is the standard deviation, and N is the number of tests.

$$\begin{aligned} I' &= random.choice(I, replace = True, size = N) \\ X' &= X[I'] \end{aligned} \tag{3.7}$$

where $I = \{0, 1, 2, \ldots, N\}$ and N is the size of the test set X. Table 3.3 presents the CV accuracy and the estimated 95%CI of classification accuracy of ACDNet on the mentioned datasets.

[2]We acknowledge and thank Dr. Akhtar Hossain (Sanofi US) for his assistance with the 95%CI

Datasets	Classification Accuracy (%)	
	CV	95%CI
ESC-10	96.65±0.06	96.73±0.04
ESC-50	87.10±0.02	87.15±0.06
US8K	84.45±0.05	84.34±0.03
AE	92.57±0.05	92.56±0.04

TABLE 3.3: CV Accuracy and estimated 95% CI of classification accuracy of ACDNet on ESC-10, ESC-50, US8K and AE datasets.

Table 3.3 reveals that the CV accuracy of ACDNet and its estimated 95%CI accuracy on various datasets are almost equivalent, if not identical, indicating that the network is well generalized over all the datasets.

Networks	Accuracy (%) on Datasets			
	ESC-10	ESC-50	US8K	AE
EnvNet [34]	88.10	74.10	71.10	-
EnvNet-v2 [35]	88.80	81.60	76.60	-
EnvNet-v2 + BC [35]	91.30	84.70	78.30	-
AclNet [36]	95.75*	85.65	79.17*	90.15*
ACDNet (ours)	**96.65 ± 0.06**	**87.10 ± 0.02**	**84.45 ± 0.05**	**92.57 ± 0.05**

TABLE 3.4: ACDNet in the state-of-the-art leaderboard for raw audio classification on ESC-10&50, US8K and AE datasets. Accuracy values with asterisk (*) are reproduced by us.

We have now added ACDNet to the state-of-the-art classification table for raw audio. ACDNet outperforms state-of-the-art with an overall accuracy of 96.65 ± 0.06% on ESC-10, 87.10 ± 0.02% on ESC-50, 84.45 ± 0.05% on US8K and 92.57 ± 0.05% on AE datasets, as shown in Table 3.4. In addition, when compared to all state-of-the-art models, regardless of the types of inputs they process, ACDNet outperforms state-of-the-art on ESC-50 and was the leader among state-of-the-art models until late 2020, with comparable prediction accuracy on other datasets (see Table 3.5)

For the real-world frog dataset, ACDNet is trained for 1000 epochs. We obtain 5-fold CV accuracy of 88.98% even without data augmentation.

The size and computation need of state-of-the-art models are rarely reported in the literature. To the best of our knowledge of the literature up until 2020, EnvNet-v2 and AclNet are the only two of the top ten models for either of the two datasets (i.e., ESC-50 and US8K) to report parameter count, size, and FLOPs. According to Table 3.6, ACDNet requires 21.39× less memory and 2.98× less FLOPs than EnvNet-v2 and 2.25× less memory and 1.97× less FLOPs than AclNet, respectively.

Table 3.6 also lists the state-of-the-art prediction accuracy of the latest three models (published a year and a half after this study) on ESC-50 are known as AST [89], CLAP [164] and HTS-AT [165]. However, only the model sizes and not the FLOPs were revealed. When compared to them, ACDNet requires 18.4×, 40.1×, and 6.5× less memory, respectively.

Datasets → Networks ↓	Year Published	ESC-10		ESC-50		US8K		AE	
		Acc (%)	Rank	Acc (%)	Rank	Acc (%)	Rank	Acc (%)	Rank
Method B [156]	2016	-	-	-	-	-	-	**92.80**	1
FBEs⊕ConvRBM [169]	2017	-	-	86.50	10	-	-	-	-
EnvNet-v2 + BC [35]	2018	91.30	6	84.70	12	78.30	8	-	-
VGG-CNN+Mixup [173]	2018	91.70	5	83.90	14	83.70	6	-	-
AclNet (WM = 1.5) [36]	2018	95.75*	3	85.65	11	79.17*	7	90.15*	3
TSCNN-DS [159]	2018	-	-	-	-	97.20	1	-	-
Multi-stream+Attn. [157]	2019	94.20	4	84.00	13	-	-	-	-
ACDNet (ours)	**2019**	**96.65**	**2**	**87.10**	**9**	**84.45**	**5**	**92.57**	**2**
CNN [161]	2020	-	-	88.65	8	-	-	-	-
Multi-CNN [162]	2020	-	-	89.50	7	-	-	-	-
WEANET [163]	2020	-	-	94.10	5	-	-	-	-
ESResNet [158]	2021	97.00	1	91.50	6	85.42	4	-	-
AST [89]	2021	-	-	95.60	4	-	-	-	-
VIP∼ANT [160]	2021	-	-	95.70	3	86.00	3	-	-
CLAP [164]	2022	-	-	96.70	2	**87.96**	2	-	-
HTS-AT [165]	2022	-	-	**97.00**	1	-	-	-	-

TABLE 3.5: ACDNet in overall state-of-the-art leaderboard of ESC-10&50, US8K and AE datasets. Column 'Acc' presents model accuracy. Accuracy values with asterisk (*) are reproduced by us. **Note: The grayed-out rows represent the DNN models that were published after the work in this chapter was completed (i.e., after 2019)**.

Networks	Year published	#Filters	Params(M)	Size(MB)	FLOPs(M)
EnvNet-v2 [35]	2017	1056	101.25	$386.25 = 21.4x$	$1620 = 3x$
AclNet [36]	2018	3050	10.63	$40.57 = 2.25x$	$1070 = 2x$
ACDNet (ours)	**2019**	**2074**	**4.74**	$\mathbf{18.06} = x$	$\mathbf{544} = x$
AST [89]	2021	-	87	$331.85 = 18.4x$	-
CLAP [164]	2022	-	190	$724.80 = 40.1x$	-
HTS-AT [165]	2022	-	31	$118.00 = 6.5x$	-

TABLE 3.6: 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. **Note: The grayed-out rows represent the DNN models that were published after the work in this chapter was completed (i.e., after 2019)**.

3.4 Network Compression

As discussed before, unstructured compression is not suitable for MCUs. Therefore, we introduce a new class of structured hybrid compression techniques. We first sparsify the weight matrices of ACDNet using unstructured compression (e.g., L0-Norm) and then apply structured 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 enhances the model performance [189]. Furthermore, many researchers have shown that sparse pruning of neural networks often produces the same or even better accuracy than the base network [90, 190, 191]. 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 Channel Pruning [95], Weight Sharing [90, 192], Huffman Coding [92], Knowledge Distillation [96, 193] and Quantization [90, 194]. 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 investigate the best-established structured pruning-based model compression techniques proposed for computer vision to exploit their benefits and find a suitable compression method for compressing DANs. 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.

Structured pruning, namely channel pruning for CNN is conducted by ranking the channels globally and removing the lowest ranked channels. Let z_l be the feature maps of a network with $z_l \in \mathbb{R}^{h_l \times w_l \times c_l}$ where $h_l \times w_l$ is the dimension and c_l are the channels of layers with $l \in [1, 2, ..., L]$. $z_l^{(i)}$ is an individual feature map with $i \in [1, 2, ..., c_l]$. We denote the layer-wise normalization process by χ which is determined by Equation 3.8, the global ranking of channels by κ and finding the lowest ranked channel by ψ. Thus, the candidate channel c_l to be pruned is determined by Equation 3.9 where c is the channel index of i^{th} layer l.

$$\chi(z_l) = \frac{|z_l^{(i)}|}{\sqrt{\sum |z_l|^2}} \tag{3.8}$$

$$l_i,\ c_l = \psi(\kappa(\chi(z_l))) \tag{3.9}$$

Many ranking criteria exist in the current literature. Few of them are magnitude-based (L2-Norm) ranking [90, 151], Taylor criteria-based ranking [95], and binary index-based ranking in AutoPruner [195]. In this paper, we explore the magnitude-based ranking which is one of the very common techniques and the Taylor criteria-based ranking which is proposed in one of the recent state-of-the-art channel pruning techniques for computer vision applications. On the other hand, our proposed hybrid pruning technique uses a new approach that combines unstructured and structured ranking methods to achieve structured compression.

To make sure the network works flawlessly with the updated output shape of different layers during and after compression, the kernel sizes of the pooling layers are adjusted accordingly. 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 the *Dense1* layer is dynamically adjusted so that it can handle the incoming input data to produce 50 output elements for the softmax output layer. During this process, if the kernel size of a pooling layer becomes (1,1), we remove that layer from the network.

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

3.4.1 Channel Pruning Using Magnitude-based Ranking

In this method, the channels are ranked using their individual sum of absolute weights (L1 Norm) followed by layer-wise L2 normalization. Many approaches remove all the channels having a sum below a predefined threshold [90, 95, 151]. However, we remove the channels 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. An iteration of this pruning process can be expressed as:

$$l_i, c_l = \psi(\kappa(\chi(\sum |z_l^{(i)}|))) \tag{3.10}$$

3.4.2 Channel Pruning Using Taylor Criteria-based Ranking

This is an iterative channel pruning technique recently proposed by Molchanov et al. [95]. In this approach, the channels are ranked using Taylor criteria, and the lowest ranked channel is pruned in a pruning iteration. More specifically, 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 channel 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.

The change in the gradient can be expressed as Equation 3.11 where z_l denotes the feature maps of layer l, Δc is the change in loss denoted by $\frac{\delta c}{\delta z_l}$ and $\Theta_{TE}(l_l)$ denotes the change determined by TE. Equation 3.12 shows how the features maps are updated with the change.

$$\Theta_{TE}(z_l) =\mid \Delta c z_l \mid \tag{3.11}$$

$$z_{T_l} = z_l + \Theta_{TE}(z_l) \tag{3.12}$$

Finally, the pruning candidate c_l for an iteration can be expressed as:

$$i_i, c_l = \psi(\kappa(\chi(\sum |z_{T_l}|))) \tag{3.13}$$

3.4.3 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. Let w be the weights of all the feature maps of all the layers, ψ' is the function that sorts w and returns the all the feature maps after zeroing out $x\%$ weight values where $x \in \{1, 2, \ldots, 99\}$. This process is expressed as:

$$\hat{z} = w[\psi'(w)] \tag{3.14}$$

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. Equation 3.15 is an example of how a channel is selected for pruning using Taylor criteria-based ranking.

$$i_i, c_l = \psi(\kappa(\chi(\sum |\hat{z}_{T_l}|))) \tag{3.15}$$

After pruning a channel, we 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 1: 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

3.4.4 Experimental Results

We use the same experimental setup that has been discussed in Section 3.3.2 for this experimental study. ESC-50 (fold-4) is used to fine-tune the network during the compression process. The performance of the resulting network is cross-validated on all the four datasets and the results are reported.

3.4.4.1 Compression by Structured Pruning

To compare our proposed method, at first, we compress ACDNet using magnitude-based pruning and taylor pruning. Then we experiment with two different combinations of the methods for testing our proposed iterative hybrid pruning approach. The first one consists of L0 Norm followed by channel pruning using magnitude-based ranking and the second one of L0 Norm followed by channel pruning using Taylor criteria-based ranking. Finally, we compare the results.

In this process, we first sparsify ACDNet by zeroing out 85%, 90%, 95% and 98% weights using L0 norm. We opted for the 95% sparsed network since it produces the same classification accuracy as the original ACDNet (see Table 3.7).

% Sparse	Original Accuracy	Final Accuracy
85%	91.00%	90.50%
90%	91.00%	89.25%
95%	91.00%	91.00%
98%	91.00%	88.75%

TABLE 3.7: Sparsifying the Weights in ACDNet.

The sparsified model is further pruned through channel pruning. We prune 80% and 85% channels using all the discussed methods and present the results in Tables 3.8 and 3.9. Models 1-2, 3-4, 5-6 and 7-8 in both the tables are the resulting models of magnitude-based pruning, taylor pruning, and the two combinations of our proposed hybrid pruning respectively. All models are iteratively pruned and fine-tuned. The column *Fine-tuning Accuracy* shows the accuracy obtained at the end of the iterative pruning and fine-tuning process.

From the tables (Table 3.8 and Table 3.9), we observe that pruning and fine-tuning does not recover the loss in accuracy as hoped for. To achieve the best accuracy, we therefore conduct further full re-training of the networks with their existing weights, scratch-training by re-initializing their weights. We use the base network's training settings for re-training and scratch training. The accuracy of these different training processes are reported in *Re-training Accuracy* and *Scratch-training Accuracy* columns respectively. In addition to re-training and scratch-training, we also train the re-initialized fresh networks using knowledge distillation [96], and the result is presented in Table 3.10.

Model No.	Pruning Method	SFEB	TFEB	Parameters			FLOPs		Fine-tuning	CV Accuracy%	
				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 3.8: Models found after 80% channel pruning using magnitude-based ranking, Taylor criteria-based ranking and our hybrid pruning approach.

Model No.	Pruning Method	SFEB	TFEB	Parameters			FLOPs		Fine-Tuned	CV Accuracy%	
				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 3.9: Models found after 85% channel pruning using magnitude-based ranking, Taylor criteria-based ranking and our hybrid pruning approach.

Tables 3.8 and 3.9 show that the compressed networks with scratch-training produce better classification accuracy. To further justify this argument, we check the results obtained by training the models using KD.

3.4.4.2 Compression by Distillation Loss

We use the best performing 80% compressed model (i.e., Model 8 from Table 3.8) as the student network while the trained base ACDNet model acts as the teacher network. The current weights of the compressed network are re-initialized and trained as a new network with distilled loss. We apply three different custom loss functions (see Equation 3.16, 3.17, and 3.18 using CE and KLD loss, where the student output and the teacher output are denoted by SO and TO, respectively. We use $\alpha = 0.1, \beta = 1 - \alpha$ and Entropy (T) $\in [1 - 5]$. Table 3.10 shows the results of this training approach.

$$L1 = CE(SO/T, TO/T) * \beta * T^2 + CE(SO, Target) * \alpha \tag{3.16}$$

$$L2 = KLD(SO/T, TO/T) * \beta * T^2 + CE(SO, Target) * \alpha \tag{3.17}$$

$$L3 = KLD(SO/T, TO/T) * \beta * T^2 + KLD(SO, Target) * \alpha \tag{3.18}$$

Loss	5-Fold CV Accuracy%				
	T=1	T=2	T=3	T=4	T=5
L1	81.45	80.15	78.75	78.45	78.35
L2	81.70	81.90	81.40	81.65	81.95
L3	81.55	**81.95**	81.75	81.25	81.85

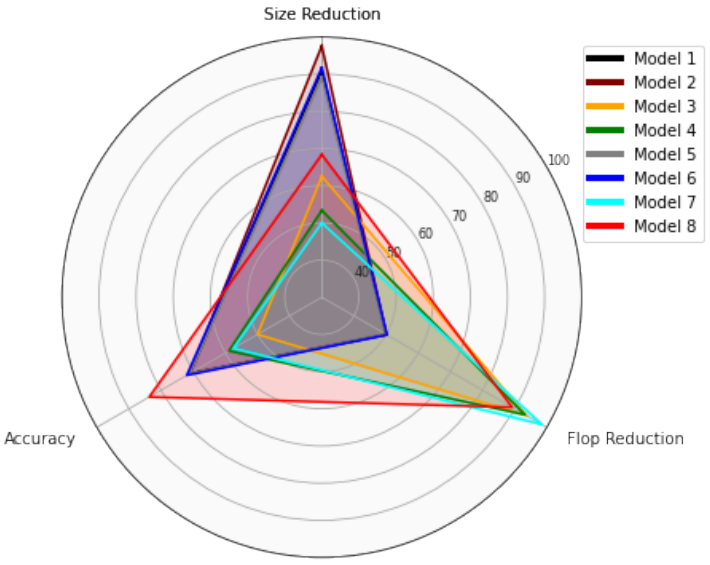
TABLE 3.10: Performance of Micro-ACDNet when trained using Knowledge Distillation

From Tables 3.8, 3.9 and 3.10 it is evident that our proposed hybrid pruning approach including scratch-training outperforms other approaches in terms of size and FLOPs reduction, and prediction accuracy. Note that for all the experiments, we have performed 5-fold CV.

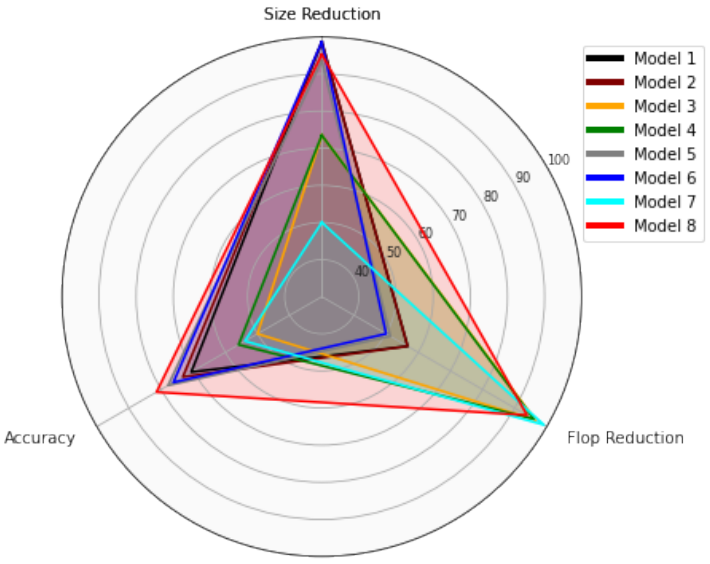
3.4.4.3 Selecting the Compressed Network

We choose the best compressed network depending on three factors: compression, FLOPs reduction and accuracy. Considering the result on all three measurements for 80% pruning shown in Table 3.8 and Figure 3.2A, we select Model 8, obtained by our hybrid pruning technique. For this model, our hybrid pruning technique achieves 97.22% model size reduction, 97.28% FLOPs reduction (Table 3.8) and 83.65% CV accuracy on ESC-50

dataset (Table 3.14). This is still very close to the state of the art and significantly higher than human accuracy (81.30%). We term this network Micro-ACDNet as it only retrains 20% filters/channels and use it henceforth.



(A) 80%



(B) 85%

FIGURE 3.2: (a) Comparison of 80% pruned models from Table 3.8. (b) Comparison of 85% pruned models from Table 3.9. For visualization, the variables are linearly transformed to range [50 - max(variable)].

With 85% pruning, our hybrid technique produces an even smaller version of Model 8 with 312 (15% retained) and only 240kB model size, which we term Macro-ACDNet (Figure 3.2B). We select Micro-ACDNet over this macro version for three reasons. Firstly, Micro-ACDNet 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. Table 3.11 presents the architecture of Micro-ACDNet.

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 3.11: Micro-ACDNet architecture for input length 30225 (approximately 1.51s audio @ 20kHz)

3.4.4.4 Compressing other Networks

Now, we perform the same methodology that is used to derive Micro-ACDNet to find a micro version of AclNet. We do this to verify whether our proposed compression approach generalizes well for other networks. We name the resulting compressed AclNet as Micro-AclNet. The output filter configurations of the 12 convolution layers of Micro-AclNet are 7, 26, 10, 20, 26, 41, 22, 48, 55, 64, 58 and 50. Micro-AclNet also produces comparable classification accuracy on ESC-50 while comparing it with its base network accuracy. Table 3.12 clearly shows that the method works as expected in compressing other networks.

Network	Pruning Method	SFEB	TFEB	Size			FLOPs			Scratch-Training
				Base (MB)	Compressed (MB)	Reduced%	Base (M)	Compressed (M)	Reduced%	Accuracy (%)
ACDNet	Hybrid-Taylor	✓	✓	18.06	0.50	97.22	544.42	14.82	97.28	**83.65**
AclNet	Hybrid-Taylor	✓	✓	40.54	0.50	98.79	1806.57	21.50	98.81	80.05

TABLE 3.12: ACDNet vs AclNet performance after 80% compression

3.4.4.5 Compressed Networks on Different Datasets

Now we test and cross validate Micro-ACDNet and Micro-AclNet on all the four datasets used for this research. Since we have already established that training from scratch works

better and Micro-ACDNet achieves the highest prediction accuracy, we conduct scratch-training for all the networks, 5-fold (ESC-10&50) and 10-fold (US8K) CV over five independent runs. Table 3.13 presents their classification accuracy on different datasets. The results in the table clearly indicate that the compressed models generalize well on all the four datasets. Their classification accuracy is very close to their base model's classification accuracy.

Accuracy (%) → Datasets ↓	Networks			
	ACDNet	AclNet	Micro-ACDNet	Micro-AclNet
ESC-10	96.75	95.75	96.25	94.00
ESC-50	87.10	85.65	83.65	80.05
US8K	84.45	79.17	78.28	75.80
AE	92.57	90.15	89.69	87.51

TABLE 3.13: ACDNet, AclNet , Micro-ACDNet and Micro-AclNet performance on different datasets

3.4.4.6 Comparing Micro-ACDNet with Other Networks

Now that we have the resulting compressed model in our hand, we can compare it with Edge-L^3, the only existing edge audio architecture. Table 3.14 shows our results in comparison to Edge-L^3.

Networks	#Filters	Params (M)	Size (MB)	FLOPs (B)	Accuracy (%)
ACDNet	2074	4.74	18.06	0.54	87.10 ± 0.02
Edge-L^3 [100]	1920	0.213	0.814	-	73.75
Micro-ACDNet	415	0.131	0.50	0.0148	83.65

TABLE 3.14: Parameters, size and computation requirements for ACDNet and Micro-ACDNet for approximately 1.51s audio @ 20kHz

3.4.4.7 Experimental Findings

Tables 3.8 and 3.9 provide the experimental results for the different pruning approaches tested on ACDNet. In Table 3.8 we see that our hybrid approach (Models 5-8) produces the best prediction accuracy and FLOPs 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.

According to Molchanov et al. [95], *pruning and fine-tuning* produces comparable prediction accuracy. Our experiments did not yield comparable prediction accuracy after pruning and fine-tuning. The experimental results (Table 3.8 and 3.9) indicate that the final compressed network requires full *re-training* or *scratch-training* to produce comparable prediction accuracy. This finding is supported by previous research work by Crowley et al. [196] and Liu et al. [197]. They also advocate that pruning or compression should be considered as an efficient DL model architecture search method.

3.5 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 [198], which is not suitable for fast, frequent write access and is used to hold the (fixed) model parameters and (2) SRAM [198], 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 [22] which provides high-quality audio input and processing on-board. We have selected this unit for these reasons.

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 of SRAM for intermediate calculations and 15kB of SRAM for input buffering, for a total of 318kB of SRAM, which is well below the target. This can be further reduced to below 200kB using a hand-optimized implementation as detailed below. While we have not yet taken this last step in a physical deployment, the implication is that Micro-ACDNet 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 Micro-ACDNet requires only 500kB of model storage (Table 3.14).

We picked TensorFlow Lite Micro over the other platform-independent DNN software frameworks for small-device deployment (e.g., PyTorch Mobile, TensorFlow Lite Micro) because it is the most frequently used and the only one that allows us to implement[3] ACDNet directly. ACDNet's *transpose* layer is not currently supported by PyTorch Mobile.

To achieve the required model size, we need to quantize Micro-ACDNet. This study is not concerned with quantization itself, so that we simply apply the quantization methods directly available in TensorFlow Lite Micro [24]. Quantization, unfortunately, reduces the model accuracy noticeably and the micro version of TensorFlow Lite does not seem to provide the best results here. We used 8-bit post-training quantization which reduces the accuracy to 71.00%. This could be due to a bug in the post-training quantization tool in TensorFlow Lite at the time of this study [199, 200]. To confirm that better results are possible, we also tested alternative frameworks. We found that PyTorch

[3]We acknowledge and thank Ian West for his support in the implementation phase of ACDNet in TensorFlow Lite Micro.

achieved the highest accuracy with 81.50% for an 8-bit quantized model of equivalent size (see Table 3.15). While our actual deployment uses TensorFlow Lite Micro 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.

Network	Library	Quantized Size	Quantized Accuracy (%)
Micro-ACDNet	PyTorch	157kB	81.50%
Micro-ACDNet	TF Lite Micro	153kB	71.00%

TABLE 3.15: Prediction accuracy on ESC-50 after quantization

The required working memory of 303kB SRAM could be reduced further. The current requirement results from essentially computing tensors layer by layer, keeping one intermediate layer in memory at a time. This can be optimized 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.

3.6 Conclusions and Future Work

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 all the four datasets ESC-10, ESC-50, US8K and AE benchmarks for raw audio classification and compressed this for MCU deployment. While the limitations of the programming environment at the time of this study restricted the accuracy of our current test deployment on a physical MCU, we have conclusively demonstrated that 81.50% accuracy is achievable on such a resource-impoverished device, close to 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.

Hence, the contributions of this chapter are as follows:

- it introduces the first generic pipeline for sound classification using DL in extremely resource-constrained MCUs.
- it presents ACDNet, a flexible and compression-friendly state-of-the-art DL model architecture for raw audio classification.
- instead of hand-crafting a once-off solution, it introduces a novel Deep-CNN compression approach to obtain tiny models for MCUs.
- finally, after 8-bit quantization, we deploy Micro-ACDNet on a standard off-the-shelf MCU with successful tests on real-world data.

Some next steps immediately arise. Firstly, it is likely that the performance can be improved by using quantization-aware training and pruning. Secondly, now that we have Micro-ACDNet as a suitable starting point for its optimizations, it would be instructive to try the SpArSe [152] approach for further optimizations.

We believe it to be of particular importance that we have constructed and used a generic 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.

Chapter 4

Study on Deep Architecture Compression Techniques

*This study has been published in **IEEE Access**. The article is titled as **Pruning vs XNOR-Net: A Comprehensive Study of Deep Learning for Audio Classification on Edge-devices** [201]. The published paper has been significantly reworked for inclusion in the thesis.*

4.1 Introduction

This chapter aims to work toward the second objective (RO-2) of this research project. The primary goal is to conduct a comprehensive empirical study on various DNN architecture compression techniques proposed for various application domains and to determine the best method to compress the baseline DNN model.

Chapter 3 has introduced ACDNet, a compression-friendly baseline DNN-based acoustic network architecture, a hybrid structured pruning technique, and a pipeline for compressing the baseline architecture and deploying it on the MCU device. In addition, it compares various structured and unstructured DNN pruning techniques and concludes that the proposed hybrid pruning technique is the best DNN architecture pruning technique. Following that, a comparison of the proposed hybrid structured compression technique and KD is presented, demonstrating that the proposed hybrid compression technique produces the best-compressed model while retaining the highest prediction accuracy.

Expanding on the previous chapter, this chapter compares and contrasts the proposed hybrid structured compression technique with the recent XNOR-Net [108]-based compression technique for model compression.

There are many state-of-the-art XNOR-Net models for different computer vision tasks, such as Rastegari et al. [108] for MNIST, Wang [109] for CIFAR-10, and Bulat et al.

[110] for CIFAR-100 and ImageNet datasets. However, for audio classification, the only work we are aware of (at the time of this study) is presented by Cerutti et al. [175]. This study uses XNOR-Net in combination with spectrogram-based input to effectively perform audio classification via image classification. As the literature on full-precision audio networks shows, this approach does not usually deliver the best results for difficult classification problems with conventional CNN architectures[38, 41] (see Section 1.2.3 for further details). To the best of our knowledge, the current literature has not yet considered XNOR-Net for raw audio classification.

Most of the work on XNOR has been performed for computer vision. The leaderboards of benchmark image datasets show considerable differences in the classification accuracy of state-of-the-art full precision nets and XNOR-Nets (see Table 4.9 for the CIFAR-100 and ImageNet datasets). Furthermore, models produced by XNOR-Net generally yield a up to 32× reduction in memory requirements and up to 58× reduction in computation cost compared to their full-precision counterparts [108]. This may not result in sufficient reduction for MCUs. The memory requirements of XNOR-Net-based DL models producing comparable accuracy [108–110] typically reach several MBs, while typical MCUs offer only 128KB to 1MB of memory.

In this study, we seek to understand the comparison of XNOR model minimization with pruning-and-quantization approaches and the potential of XNOR in the context of audio classification. We present XNOR-Nets for raw audio classification followed by a comprehensive study that compares this approach with traditional model compression techniques (pruning-and-quantization). Our extensive experimental study reveals that XNOR-Net may be preferred for scenarios comprising a small number of classes (e.g., 10 classes) along with extremely tight resource constraints, specifically where computation speed is concerned. In contrast, for complex scenarios with a larger number of classes, pruning-and-quantization-based compression techniques would still be the choice because they produce sufficiently small models for off-the-shelf MCUs that have higher performance in terms of accuracy.

Experiments show that, when two models are generated by pruning-and-quantization and XNOR-Net for the same memory constraint, XNOR-Net requires $\approx 8\times$ less computation while both produce comparable classification accuracy for small problem sizes (number of classes). While XNOR-Nets require extremely little computation compared to their pruning-and-quantization based counterparts, their classification performance on datasets degrades rapidly with an increasing number of classes. The performance loss of pruning-and-quantization based models for the same audio benchmarks is much more graceful, so that this approach still appears to be preferable for problems with larger class numbers.

To harden the above findings, we have conducted a similar study on image classification datasets and analyzed this in the context of current state-of-the-art full precision and

XNOR-Net leaderboards for various image benchmarks. The behavior is found to be consistent in both the audio and the image domains.

Thus, the contribution of the study of this chapter is twofold: 1) it presents the first application of XNOR-Net for raw audio classification as a benchmark for future research. 2) It presents the first comprehensive empirical comparison of pruning, quantization, and XNOR-Net-based model compression techniques and derives guidelines for when to use pruning, quantization, and XNOR-Net, respectively.

4.2 Related Work

We have discussed structured DNN architecture compression techniques and KD-based DNN model compression techniques in the previous chapter (see Section 3.4). In this section, we discuss XNOR-Net, before moving on to the comparative analysis.

Like structured pruning-based compression works, almost all the works based on XNOR-Net are proposed for computer vision. The current state-of-the-art XNOR models for the benchmark image datasets are: [108] for MNIST, [109] for CIFAR-10 and CIFAR-100 and [110] for ImageNet. In contrast, [175] proposed the only XNOR-Net-based work (at the time of the study) for audio classification. However, that work uses spectrograms as the input to the model, which is similar to image classification using XNOR-Net. According to the discussion presented in Section 1.2.3, using raw audio as input is the appropriate method for audio classification in MCUs. According to our knowledge, there is no such XNOR-Net based work present in the current literature. For detailed information and XNORNet, please see Chapter 2 Section 2.2.3.

In the following sections, we compare our hybrid structured compression technique to XNOR-Net. In addition, we used a few other networks from the computer vision domain to see if our findings hold true for image classification as well.

4.3 Experimental Details

All experiments are conducted using Python version 3.7.4, and the GPU versions of Torch 1.8.1. All experimental codes are available at: https://github.com/mohaimenz/pruning-vs-xnor.

4.3.1 Datasets

For audio classification experiments, we use the same four datasets (ESC-10&50, US8K and AE) as we did in Chapter 3. The details of these datasets are provided in Chapter 3 Section 3.3.2.1. In addition, we have created more subsets of ESC-50 and AE datasets for our study where the classes are chosen randomly without replacement and kept them fixed for reproducibility. We use these class incremental subsets (e.g., S_{10} subset has 10 classes while S_{20} has 20 classes) to evaluate XNOR-Net's performance on datasets with a small to large number of classes. The subsets of the ESC-50 datasets are as follows:

S_{10} = The original ESC-10 dataset
$S_{20} = S_{10} \cup \{5, 31, 18, 27, 48, 8, 15, 45, 25, 34\}$
$S_{30} = S_{20} \cup \{3, 14, 23, 36, 43, 7, 22, 28, 30, 49\}$
$S_{40} = S_{30} \cup \{6, 9, 16, 17, 24, 29, 32, 35, 37, 44\}$
S_{50} = The original ESC-50 dataset

The subsets of the AE dataset are as follows:
$S_{10} = \{0, 2, 5, 9, 11, 12, 17, 21, 25, 26\}$
$S_{20} = S_{10} \cup \{1, 4, 7, 8, 14, 15, 19, 20, 23, 27\}$
S_{28} = The whole AudioEvent dataset

For the image classification experiments, we use the CIFAR-10 and CIFAR-100 benchmark image datasets. The CIFAR-10 dataset contains 60,000 image samples equally distributed over 10 classes. 50,000 of the samples are used for training and 10,000 for testing. CIFAR-100 has 100 classes, and 60,000 samples are equally distributed across the classes. For each class, there are 500 training samples and 100 test samples. We create the subsets of CIFAR-100 using Equation 4.2. The subsets are defined as follows:

$$S_{x_i} = S_{x_{i-1}} \cup \{\frac{x}{10} + m\} \tag{4.1}$$

where $x \in \{10, 20, \ldots, \lfloor n \rfloor\} \mid n \in [1, 100]$ and $m \in [1, 10]$. This gives us the following subsets such as:

S_{10} = The whole CIFAR-10 dataset.
$s_{10} = \{0, 10, 20, 30, 40, 50, 60, 70, 80, 90\}$
$S_{20} = s_{10} \cup \{1, 11, 21, 31, 41, 51, 61, 71, 81, 91\}$
$S_{30} = S_{20} \cup \{2, 12, 22, 32, 42, 52, 62, 72, 82, 92\}$
...
...
$S_{90} = S_{80} \cup \{9, 19, 29, 39, 49, 59, 69, 70, 89, 99\}$
S_{100} = The whole CIFAR-100 dataset.

4.3.2 Data Preprocessing

We apply the same data preprocessing and training for the audio datasets described in Chapter 3 Section 3.3.2. For the image datasets, we use random cropping, horizontal flipping, and rotation available in the Transforms module of the PyTorch TorchVision library. All implementations of the data augmentation procedures are available in our GitHub repository.

4.3.3 Models and Hyperparameters

The model configuration is available in the GitHub repository, and the hyperparameters for all the experiments conducted on the audio and image datasets are listed in Table 4.1.

Datasets→	Audio Datasets		Image Datasets	
Networks→	ACDNet, AclNet & their derivatives		RESNET-18	
Hyperparams↓	Full Precision	XNOR	Full Precision	XNOR
Input shape (ch, h, w)	(1, 1, 30225)		(3, 32, 32)	
Loss function	KLD		CE	
Optimizer	SGD	Adaptive Momentum Estimation (ADAM)	SGD	ADAM
Weight decay	5e-4	1e-4	5e-4	1e-4
Momentum	0.9	-	0.9	-
Initial LR	0.1	0.001	0.1	0.001
Epochs	2000 (ESC-10,...50), 1000 (UrbanSound8k) & 1500 (AudioEvent)		400	
LR Scheduler	(0.3, 0.6, 0.9)	CosAnLR	CosAnLR	
Warmup epochs	10	-	10	-
LR decay	$0.1x$	-	-	
Batch size	64			

TABLE 4.1: Hyperparameter settings for experiments conducted on Audio Datasets (ESC10, ...,50, UrbanSound8k and AudioEvent) and Image datasets (CIFAR-10,...,100). In the LR Scheduler row, CosAnLR stands for CosineAnnealingLR.

4.4 Analysis: Pruning vs XNOR-Net

In this section, we compare pruning-and-quantization techniques with XNOR-Net for raw audio classification. Our analysis through extensive experiments on different standard benchmark datasets for audio classification shows that XNOR is more effective for relatively simple problems and very tight constraints on the computational resources; however, the higher classification accuracy achieved by pruning-and-quantization outweighs the benefit of XNOR-Net for problems with complex data characteristics, large number of classes, etc.

In addition to ACDNet and its derivatives (e.g., Micro-ACDNet), we use another recent state-of-the-art DNN network for raw audio classification called AclNet[36] and build its derivatives equivalent to the derivatives of ACDNet using pruning-and-quantization as well as the XNOR-Net technique. We test these models on the same standard benchmark sound classification datasets and on their subsets for an in-depth analysis of the effect of increasing the number of classes. We create subsets of a dataset S_n with n classes as:

$$S_x \subseteq S_n \mid x \in \{10, 20, \ldots, \lfloor n \rfloor\} \tag{4.2}$$

We measure classification accuracy through 5-fold cross validation. The experimental details are provided in Section 4.3.

4.4.1 Pruning-and-Quantization

We have used the hybrid structured pruning technique described in Chapter 3 Section 3.4.3 to derive Micro-ACDNet by pruning 80% of the channels from ACDNet. Micro-ACDNet is then quantized using a post-training 8-bit quantization technique. We refer to this quantized model as QMicro-ACDNet. The memory and computation requirements of QMicro-ACDNet make it suitable for current off-the-shelf MCUs (see Table 4.2). We follow the same procedure to derive the respective derivatives of AclNet.

Models	Params (M)		Size (KB)		FLOPs (M)	
	ACDNet	AclNet	ACDNet	AclNet	ACDNet	AclNet
Baseline	4.74	10.63	18498.00	41512.96	544.00	1806.57
Micro	0.13	0.13	501.00	502.00	14.82	21.50
QMicro	0.13	0.13	133.12	128.50	14.82	21.50

TABLE 4.2: Size and computation requirements for ACDNet, AclNet including their Micro and QMicro versions

Micro-ACDNet and QMicro-ACDNet require $36x$ and $144x$ less memory, respectively, than ACDNet that requires 18.06MB to store its 4.74M parameters. Furthermore, both smaller versions require $37x$ less floating-point operations (FLOPs) compared to the base model. For AclNet, the memory reductions for the same derivatives are $86x$ and $324x$ with $84x$ fewer FLOPs. We note that the QMicro version requires the same number of FLOPs as the Micro version but only at quarter precision. If a full 32-bit floating-point unit (FPU) is used, this does not make a practical difference, but it allows us to exploit significant speed gains using 16-bit FPUs, and 16-bit FPU modes on full-precision FPUs, and software emulations when no hardware FPU is available.

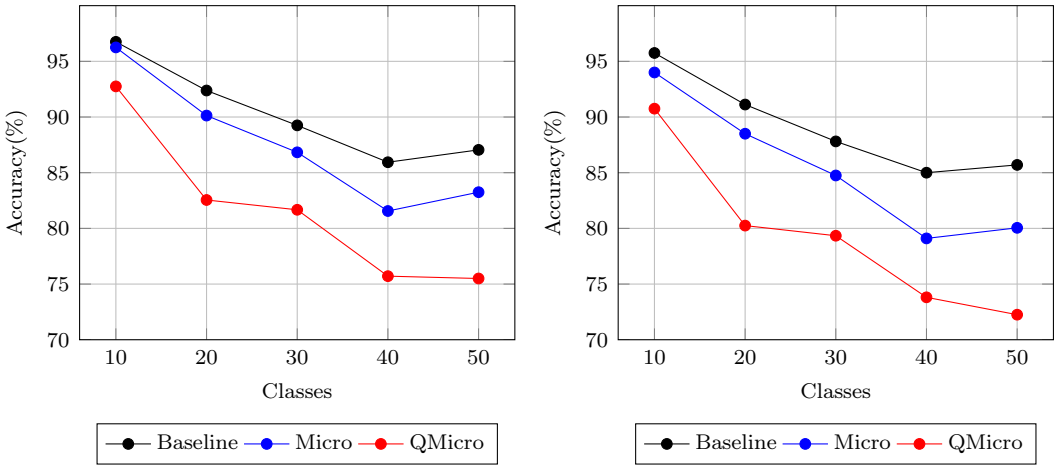
Although the smaller models require fewer resources, they lose classification accuracy because they have less capacity to learn. According to [42], Micro-ACDNet has 80% less capacity than ACDNet and QMicro-ACDNet is a quarter precision version (8-bit) of Micro-ACDNet. Table 4.3 and Figure 4.1 present a comparison of the accuracy achieved by the three versions of both the baseline networks on ESC-10,...,50 datasets (derived using Equation 4.2. For further details, see Section 4.3.1). The table and the figure show that all the three versions of both the networks produce state-of-the-art and near state-of-the-art accuracy on ESC-10, however, the accuracy drops continuously with an increase in the number of classes, as may be expected.

#Cls.	ACDNet			AclNet		
	Baseline	Micro	QMicro	Baseline	Micro	QMicro
10	96.75	96.25	92.75	95.75	94.00	90.75
20	92.38	90.13	82.55	91.12	88.50	80.25
30	89.25	86.83	81.67	87.81	84.75	79.34
40	85.94	81.56	75.71	85.00	79.10	73.81
50	87.05	83.25	75.50	85.70	80.05	72.25

TABLE 4.3: Prediction accuracy (%) of Micro versions of ACDNet and AclNet including their quantized versions (QMicro) on ESC-10,...,50 datasets. The column headers '#Cls' stands for 'No. of Classes' and 'Baseline' stands for the base model (i.e., ACDNet or AclNet).

The base ACDNet achieves an accuracy of 96.75% and 87.05% on ESC-10 and ESC-50, respectively, whereas Micro-ACDNet produces 96.25% and 83.25% on ESC-10 and ESC-50, respectively. The quantized version experiences a larger drop in accuracy as the number of classes increases, starting at 92.75% for ESC-10 and ending with 75.50% for ESC-50.

For AclNet and its different smaller versions, the trend is similar if not the same. Figure 4.1 clearly shows these trends.



(A) ACDNet (B) AclNet

FIGURE 4.1: Comparison of prediction accuracy between the baseline models (i.e., ACDNet and AclNet) and the micro versions including their quantized versions (QMicro) on ESC-10,...,50 datasets.

Models	Accuracy (%)	
	ACDNet	AclNet
Baseline	84.45	79.17
Micro	78.28	75.80
QMicro	70.93	67.47

TABLE 4.4: Prediction accuracy (%) of Micro versions of ACDNet and AclNet including their quantized versions (QMicro) on UrbanSound8k dataset. Here, 'Baseline' stands for the base model (i.e., ACDNet or AclNet)

The scenario is not different when we apply the same networks on the UrbanSound8k dataset. Table 4.4 shows that QMicro-ACDNet has lost almost 13.5% accuracy compared to the base network. For QMicro-AclNet, the loss in accuracy is 11.7% compared to the base network. We note that specialized quantization targeted to a particular model can potentially reduce the loss of the accuracy that the models experience during the quantization process. However, improving any technique used to demonstrate the process is beyond the scope of the study of this chapter and while a change in quantization techniques may shift the results, the general trends are expected to remain the same.

4.4.2 XNOR-Net

We have applied XNOR-Net technique described in Section 2.2.3 to derive the XNOR-Net versions of ACDNet (i.e., XACDNet) and AclNet (i.e., XAclNet). The layer architecture of these XNOR-Versions are identical to the binary convolution layer architecture shown is Figure 2.9. Table 4.5 lists the memory and the computation requirements of the full precision networks and their XNOR counterparts. The table shows that the memory required by the XACDNet is 578KB and XAclNet is 1300KB. These figures are still too much for typical MCUs offering less than 1 MB of RAM, since the parameter

spaces alone already leave hardly any room for the actual computations. Hence, we need smaller versions of the original networks whose resource requirements do not exceed the resources available in the MCUs. To compare the network performance, memory, and computation requirements with QMicro-ACDNet, we create Mini-ACDNet such that its XNOR-Net version has similar requirements to QMicro-ACDNet (see Tables 4.2 and 4.5). To derive Mini-ACDNet, we use the same technique as that used to derive Micro-ACDNet, summarized above and fully detailed in [42]. The same procedure is used to derive Mini-AclNet from AclNet.

Model	Params (M)		Size (KB)		FLOPs (M)	
	ACDNet	AclNet	ACDNet	AclNet	ACDNet	AclNet
Baseline	4.74	10.63	18498	41512.96	544	1806.57
XBaseline	4.74	10.63	578.00	1300.48	8.88	30.02
Mini	1.05	1.05	4096.00	4096.00	112.00	119.00
XMini	1.05	1.05	128.5	133.12	2.79	3.20

TABLE 4.5: Size and computation requirements for the baseline(i.e., ACDNet or AclNet), Mini and their XNOR-Net versions XBaseline and XMini respectively.

The sizes of the XNOR-Net versions of ACDNet and AclNet are calculated according to [108]. The number of binary operations required for the networks is calculated according to [202]. We express the computation (Binary operation + FLOPS) required for an XNOR network as FLOPs for simplicity. If a and b are the FLOPs required for the first and the last full precision layers of the XNOR network and x is the total number of FLOPs of the full precision version of the network, then the calculation of FLOPs of an XNOR network can be expressed as $FLOPs = a + (x - (a + b))/64 + b$.

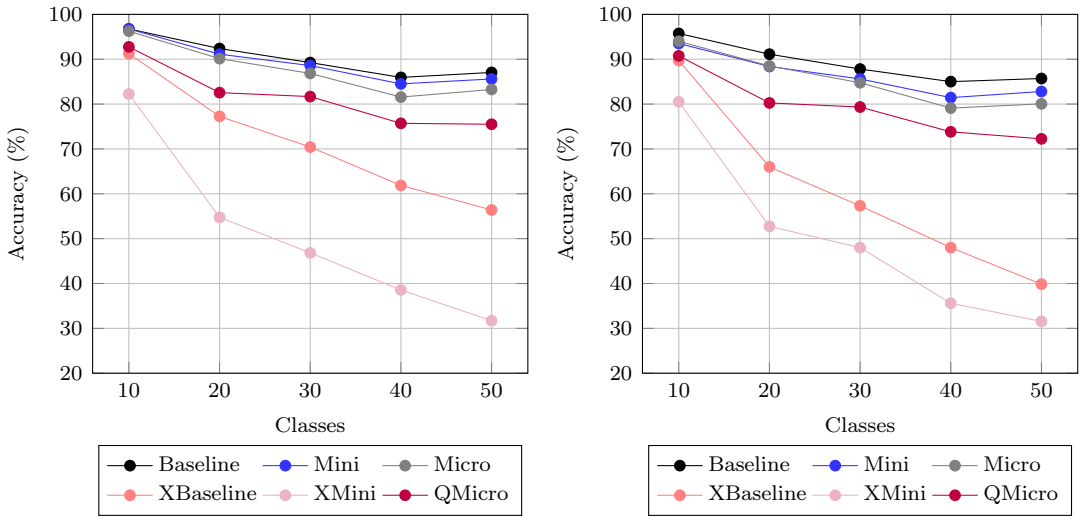
Table 4.5 also shows that XMini versions are extremely small (128.5KB and 133.12KB) and require considerably less computation. This clearly allows the network to be deployed on the current off-the-shelf MCUs. Furthermore, from Table 4.6, we can see that the XNOR networks produce reasonable accuracy for a smaller number of classes (ESC-10 with 10 classes). However, as the number of classes increases, the network performance of the XNOR versions decreases rapidly.

#Cls.	ACDNet				AclNet			
	Baseline	XBaseline	Mini	XMini	Base	XBase	Mini	XMini
10	96.75	91.25	96.75	82.25	95.75	89.70	93.50	80.50
20	92.38	77.25	91.12	54.75	91.12	66.00	88.38	52.75
30	89.25	70.42	88.58	46.83	87.81	57.33	85.6	48.00
40	85.94	61.87	84.50	38.56	85.00	48.00	81.44	35.60
50	87.05	56.40	85.60	31.70	85.70	39.85	82.80	31.55

TABLE 4.6: Prediction accuracy (%) of Baselines and Mini versions including their XNOR-Net versions on ESC-10,...,50 datasets. The column header '#Cls' stands for 'No. of Classes' and 'Baseline' stands for the base network (i.e., ACDNet or AclNet)

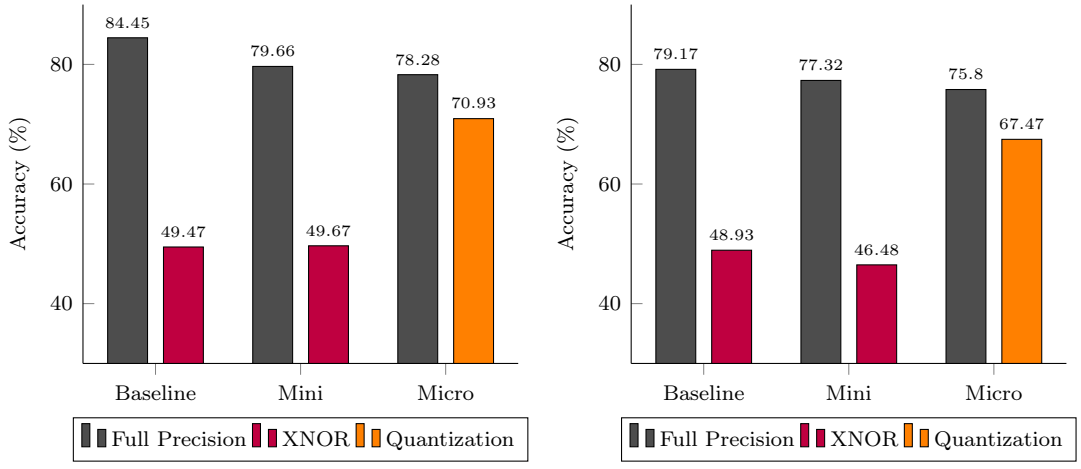
Figure 4.2 shows the performance graph of the Baseline models (i.e., ACDNet and AclNet) and their mini, pruning-and-quantization and XNOR counterparts. The line

at the bottom (XMini) shows how extremely the smallest XNOR-Net is affected when the number of classes increases, while QMicro manages degrade much more gracefully. For all the XNOR-Net networks (XBaseline and XMini versions of the Baselines), the slope is much steeper than that for the other compression methods.



(A) ACDNet (B) AclNet

FIGURE 4.2: Comparison of prediction accuracy between Baseline, Mini including their XNOR-Net versions and Micro including its quantized version (QMicro) on ESC-10,...,50 datasets. The Baseline network is AcdNet or AclNet. XMini and QMicro versions have approximately the same memory requirements.



(A) ACDNet (B) AclNet

FIGURE 4.3: Comparison of prediction accuracy between Baseline, Mini including their XNOR-Net versions and Micro including its quantized version (QMicro) on the UrbanSound8k dataset. The Baseline network is ACDNet or AclNet. XMini and QMicro versions have approximately the same memory requirements.

Figure 4.3 shows the performance of the same networks on the UrbanSound8k dataset, another widely used audio benchmark. From the graph, we observe that the results for UrbanSound8k confirm our findings for ESC-10,...,50.

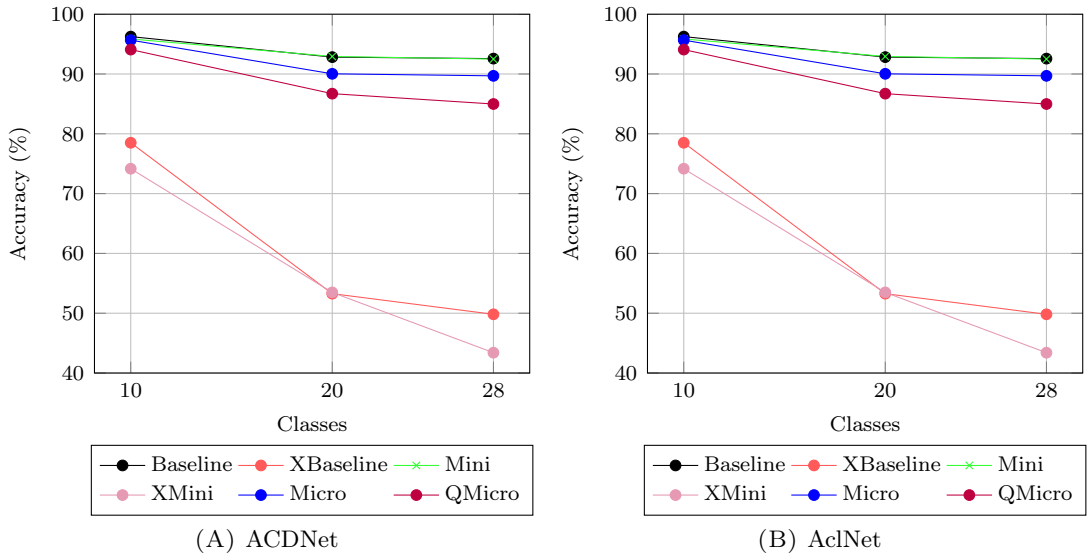
In summary, we observe that the memory requirements of QMicro and XMini versions are essentially the same. XMini-ACDNet requires $7.97x$ fewer FLOPs than QMicro-ACDNet; on the other hand, XMini-AclNet requires $6.72x$ fewer FLOPs that QMicro-AclNet. However, the XNOR-based models do not reach comparable classification accuracies when the number of classes is larger. Although the pruning-and-quantization-based QMicro-ACDNet also sees a loss in accuracy, the accuracy is still reasonable for larger problems given its minuscule model size. This observation is consistent with both the baseline models and their derivatives.

4.4.3 Extended Analysis

4.4.3.1 Comparing with Existing Work

To the best of our knowledge, Cerutti et al. [175] have presented the only XNOR network for audio classification so far, termed BNN-GAP8. They have used a different benchmark, namely, the less commonly used AudioEvent dataset [156]. We cannot test BNN-GAP8 on the most widely used benchmarks, ESC50 and Urbansound-8k, since BNN-GAP8 has not been described in sufficient detail in [175] to make it reproducible.

To facilitate a direct comparison, we extend our analysis to this dataset, which has also been used in a several other studies [156, 168]. These results are consistent with what we have seen above for the most widely used standard benchmarks. Table 4.7 and Figure 4.4 show the performance of our base nets on the AudioEvent dataset and its smaller subsets.



(A) ACDNet (B) AclNet

FIGURE 4.4: Comparison of prediction accuracy between the Baseline (ACDNet or AclNet), Mini including their XNOR-Net versions and Micro including its quantized version (QMicro) on AudioEvent datasets and its subsets (10, 20 and 28 classes). XMini and QMicro versions have approximately the same memory requirements.

#Cls.→	10		20		28	
Models↓	ACDNet	AclNet	ACDNet	AclNet	ACDNet	AclNet
Baseline	96.25	94.05	92.82	90.46	92.57	90.15
XBaseline	78.05	77.08	53.27	50.84	49.83	41.34
Mini	95.86	93.45	92.93	90.82	92.49	89.99
XMini	74.16	73.33	53.48	50.09	43.40	44.61
Micro	95.66	94.06	90.03	88.35	89.69	87.51
QMicro	94.08	92.48	86.71	84.28	84.98	81.57

TABLE 4.7: Prediction accuracy (%) of the Baseline (ACDNet or AclNet), Mini including their XNOR-Net versions and Micro including its quantized version (QMicro) on AudioEvent-10,20,28 datasets. The column header '#Cls' stands for 'No. of Classes'. XMini and QMicro versions have approximately the same memory requirements.

However, the larger resource requirements of our base networks do not allow us a direct and fair comparison to the BNN-GAP8 network presented in [175]. Therefore, we derive two additional smaller networks (QNano-ACDNet and XGAP8-ACDNet) with memory requirements equivalent to BNN-GAP8. QNano-ACDNet is a 8-bit quantized version of the associated full-precision network Nano-ACDNet that is derived by pruning the channels from ACDNet. XGAP8-ACDNet is an XNOR network derived from the full precision network GAP8-ACDNet. GAP8-ACDNet is also derived by pruning channels from ACDNet. The equivalent derivatives are also derived from AclNet. The new models are constructed using the same procedures as described above. The tests of these networks on the AudioEvent dataset used in [175] are summarized in Table 4.8.

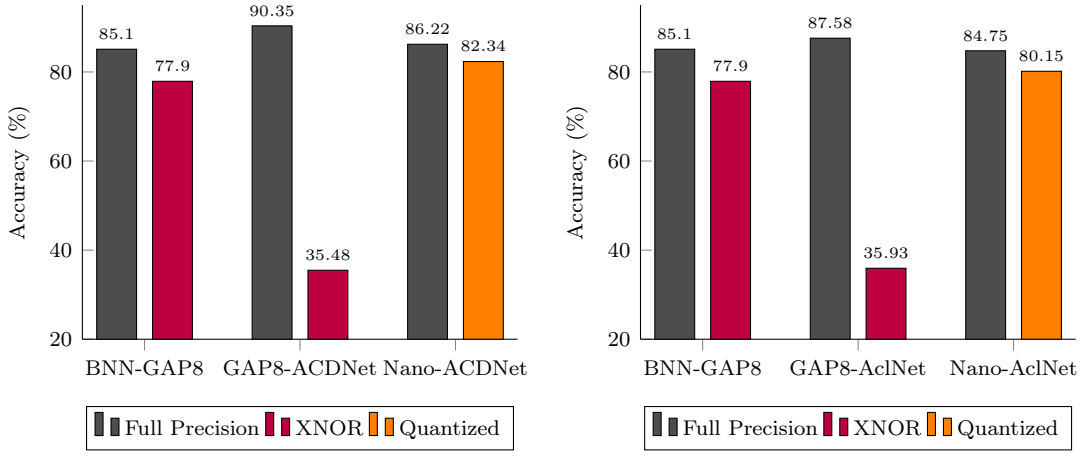
Models	Accuracy (%)		Memory (KB)		FLOPs (M)	
CNN-CNP [168]	85.1		1766		2478	
BNN-GAP8 [175]	77.9		58		1768	
	ACDNet	AclNet	ACDNet	AclNet	ACDNet	AclNet
Nano	86.22	84.75	245	245	10.54	11.37
QNano	82.34	80.15	61	61	10.54	11.37
GAP8	90.35	87.57	1960	1960	40.51	54.02
XGAP8	35.48	35.93	61	61	1.60	2.18

TABLE 4.8: Prediction accuracy (%) of BNN-GAP8, Nano and GAP8 versions of both the baselines including the XNOR-Net for GAP8 (i.e., XGAP8) and the quantized version of Nano (i.e., QNano) on AudioEvent (28 classes) dataset

BNN-GAP8 shows good performance but the quantized networks QNano-ACDNet and QNano-AclNet clearly outperforms it for equivalent storage requirements. The computational requirements for BNN-GAP8 are given in MACs rather than FLOPs in [175]. Using the same re-scaling as detailed above, we arrive at 1768M FLOPs for BNN-GAP8. The quantized QNano networks compare very favorably, using only 10.54M FLOPs for the ACDNet version and 11.37M FLOPs for the AclNet version. In a nutshell, the pruned-and-quantized QNano versions deliver better performance than BNN-GAP8 XNOR-Net with smaller resource requirements.

The performance of BNN-GAP8 on the 28-class problem is much better than that of the the XNOR networks derived from ACDNet and AclNet (i.e., XGAP8-ACDNet and XGAP8-AclNet). The results for the latter are consistent with the performance drop established for the slightly larger XMini-ACDNet on ESC30 (ESC50 reduced to 30 classes, cf. Table 4.6).

How can BNN-GAP8 achieve such good performance on a 28-class problem? The likely reason is that BNN-GAP8 benefits from using spectrograms as input to the network. This means that in the BNN-GAP8 implementation much of the necessary full precision computation required, for which a binary net is not suitable, is encapsulated outside of the network in the conversion of raw audio to spectrograms. The other XNOR networks in this comparison are deprived of this possibility because they perform end-to-end classification for raw audio input and thus must perform all required computations within the network.



(A) ACDNet (B) AclNet

FIGURE 4.5: Comparison of prediction accuracy between BNN-GAP8, GAP8-ACDNet and Nano-ACDNet on AudioEvent dataset for 28 classes. BNN-GAP8 runs on spectrogram, in contrast, GAP8-ACDNet and Nano-ACDNet run on raw audio. GAP8-ACDNet has approximately the same resource requirements of BNN-GAP8. QNano-ACDNet has equivalent memory requirement of the XNOR versions of BNN-GAP8 and XGAP8-ACDNet

From a pragmatic perspective, using spectrograms may be a useful approach for handling simple audio classification problems on MCUs that feature a DSP with hardware support for FFT computation, which is the basis of generating spectrograms. However, as has been shown in Table 4.8 and Figure 4.5, even where such support exists, forgoing it and using end-to-end classification with a pruned-and-quantized network instead still offers performance and resource benefits over the XNOR network. Furthermore, from the broader literature on DNN audio classification, it is evident that purely spectrogram-based classification does not always allow us to achieve state-of-the-art performance with conventional CNN architectures and that features learned from raw audio or multi-channel features are preferable for more difficult benchmarks [89, 161–163, 169].

4.4.3.2 Analysis with Image Datasets

Most of the work on XNOR nets so far has taken place in the image domain. A comparison with this body of work is instructive. We summarize the state-of-the-art for the most widely used datasets in Table 4.9. The accuracy achieved by XNOR nets is impressive (second last column), but it has to be noted that these nets are significantly larger than our target size. None of these models fit on the relevant MCUs with the single exception of the one for MNIST. This is a very simple dataset with only 10 classes.

Datasets (#cls.)	Full Precision		XNOR	
	Accuracy	Size (MB)	Accuracy	Size (MB)
MNIST (10)	99.87 [203]	5.8	99.23 [108]	0.10
CIFAR-10 (10)	99.50 [88]	2411	88.74 [109]	1.3
CIFAR-100 (100)	96.08 [204]	-	77.80 [110]	7.8
Imagenet (1000)	90.45 [205]	7030	71.20 [110]	7.8

TABLE 4.9: State-of-the-art accuracy (%) for various image datasets. '#Cls' stands for 'number of Classes'.

To verify whether our own results for audio classification are consistent with the image domain, we conducted additional experiments on image classification using XNOR. We have used the XNOR version of RESNET-18 [176] to classify the widely used benchmark image datasets CIFAR-10 and CIFAR-100. We have created variably sized subsets of CIFAR-100 to determine whether the trend of the loss in accuracy is similar to audio classification. The experimental details are provided in Section 4.3. Table 4.10 and Figure 4.6 summarize the results. The sizes of the full precision models are between 42.63MB and 42.80MB and the computation requirements between 95.17M FLOPs and 95.21M FLOPs. For the XNOR-Net version, the model size is approximately 1.34MB using 2.06M FLOPs. The results of this experiment are consistent with those for the audio domain, and it is clearly visible that a similar performance drop for XNOR occurs as the number of classes increases.

Datasets	Full Precision	XNOR
	Accuracy (%)	Accuracy (%)
CIFAR-10	93.29	80.24
CIFAR-20	85.70	69.45
CIFAR-30	80.87	60.90
CIFAR-40	76.67	57.00
CIFAR-50	75.18	54.36
CIFAR-60	72.28	49.77
CIFAR-70	72.70	51.20
CIFAR-80	71.81	49.51
CIFAR-90	71.61	50.22
CIFAR-100	71.49	49.56

TABLE 4.10: RESNET-18 and its XNOR-Net version on CIFAR-10,20,....,100 datasets.

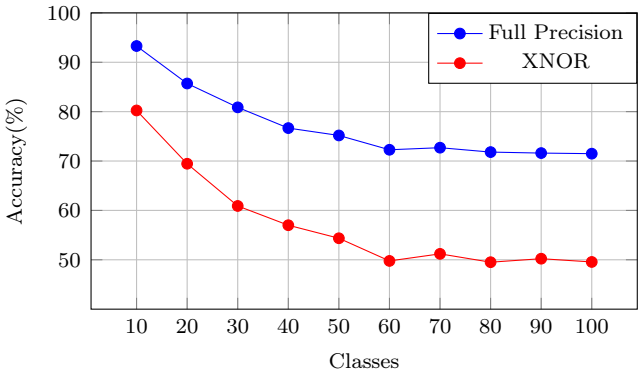


FIGURE 4.6: Prediction accuracy of RESNET-18 and its XNOR-Net version on CIFAR-10,20,....,100 datasets

4.5 Conclusion

This research presents the first XNOR-Net study for the end-to-end raw audio classification. Our comprehensive experimental analysis shows that XNOR-Net can produce models with reasonable performance that are sufficiently small to fit the target architectures for small problem sizes, as measured by the number of classes. For our test scenarios with small problem sizes, XNOR networks require significantly less computation than comparably sized models generated by pruning-and-quantization alone. Our experiments indicates that, for relatively simple problems, the performance of XNOR-Nets on MCUs can be further increased by using spectrograms as input to the net [108, 109]. This enables us to capitalize on special DSP hardware so that the full-precision computations required to handle raw audio can be confined outside of the net in the spectrogram generation.

The picture changes as the problem complexity increases and more classes need to be distinguished. Our analysis shows that XNOR-Nets suffer from significant drops in classification accuracy for datasets with many classes. This degradation of performance is much faster than that of their pruning-and-quantization counterparts. Although this performance degradation can be mitigated by growing network size while still allowing the network to run on CPUs, its ability to fit in the MCUs is eliminated because XNOR-Net can only reduce memory requirements by a maximum of 32-fold when compared to its full-precision counterpart. As a result, other model compression techniques are required to find a model that is no more than 32 times larger than the target size before applying XNOR.

In contrast, comparably sized models generated by pruning-and-quantization show significantly better performance while still satisfying the constraints of the target architectures. The advantages of spectrogram-based network input can no longer be exploited with complex data characteristics as feature-learning from raw audio or multi-channel input is required to reach state-of-the-art performance. This renders pruning-and-quantization

the preferred approach from a certain problem complexity unless computation speed rather than model size dominates the decision.

Furthermore, to the best of our knowledge, there is no off-the-shelf computation kernel for XNOR-Nets yet. Hence, it is difficult to realize the theoretical advantage of XNOR-Net on existing add-multiplication-based hardware and that custom hardware is required to achieve the full benefit of faster computation [206]. However, we hope that such support is not too far away.

In summary, the contributions of this chapter are as follows:

- it presents the first application of XNOR-Net for raw audio classification as a benchmark for future research.
- It presents the first comprehensive empirical comparison of pruning, quantization, and XNOR-Net-based model compression techniques and provides guidelines for when to use each.

Chapter 5

Deep Active Learning of Audio Features

5.1 Introduction

In earlier chapters, the methodologies required to fit Deep Neural Network (DNN) models to edge devices have been presented. The next goal is to design a practical application for sound classification based on these discoveries.

Any prediction task requires the training of the predictive models with labeled data. With a DNN-based model, the quantity of labeled data required increases considerably. This poses two obstacles to building a well-trained DNN model with acceptable performance: in our application domain, the first is the lack of sufficient labeled data at the outset to prepare a well trained DNN model, and the second is labeling the infrequent target data from terabytes of recordings. Manual labeling is a very labor-intensive process in typical bioacoustics applications.

The recent success of using DNN-based methods has largely been limited to application domains with significant volumes of labeled data available for training the models [128]. According to Ren et al. [127], for a DNN model to be able to extract high-quality hierarchical features, a long training period with a huge amount of data is required to optimize the enormous number of parameters associated with it. Often it is not practical to expect a sufficient amount of high-quality labeled data at the beginning of a project. In a continuous audio streaming environment where data is recorded 24 hours a day, seven days a week, resulting in terabytes of accessible data, meaningful target data may appear only occasionally, yet all the data need to be screened for labeling. Labeling a significant quantity of this uncommon data manually in order to train a DNN model is therefore exceedingly time-consuming and costly.

To address the lack of labeled data, a process that can select a small number of the most informative samples from a large amount of unlabeled data is required. The DNN model

can then be trained on the limited labeled samples. One of the learning strategies that can address both the aforementioned concerns with little effort is Active Learning (AL), a semi-supervised machine learning technique [124]. This iterative learning method is intended to speed up learning, especially when a large labeled dataset is unavailable for traditional supervised learning [126–128]. AL algorithms are comprised of intelligent acquisition functions that select the most informative samples and assign them to human annotators [128]. A robust classifier can be trained on these manually labeled informative samples with minimal labeling effort using specialized incremental training techniques used in AL [124, 128].

In the initial stage of an AL process, audio features are extracted using a variety of strategies that have been developed and used by many different researchers, including Ash et al. [128], Coleman et al. [134], Hilasaca et al. [135], and [33, 124, 127, 130, 133, 136, 137, 137, 207–211]. In a Deep Active Learning (DAL) procedure, the output of a fixed pretrained DNN model without the output layer (and sometimes without a few additional layers) is typically used as the feature set. The acquisition function analyzes these features, identifies the most informative samples, and requests that they be manually labeled. The DNN model's output layer is then added back to the network and fine-tuned, or a separate classifier is selected and trained iteratively on these labeled samples. The feature extractor, on the other hand, is never fine-tuned with labeled samples in order to improve the quality of the extracted features [33, 124, 127, 128, 130, 133–137, 137, 207–211].

We propose that fine-tuning the audio feature extraction level in the active learning process will improve performance and save further labeling effort. This has previously not been explored. To accomplish this, we propose a Deep Active Feature Learning (DeepFeatAL) framework that incorporates the feature extractor into the active learning loop and fine-tunes it with each iteration of human annotation to enhance its ability to extract better features.

Our method contributes in the following ways: i) it demonstrates that incorporating the feature extractor and fine-tuning it in each iteration of AL improves feature learning and the fine-tuned model learns faster and requires 14.28%, 66.67%, and 50% less labeling effort than the Standard Deep Active Learning (SDAL) process for ESC-50, US8K, and iWingBeat datasets, respectively; ii) finally, it demonstrates that these advantages can be incorporated into Microcontroller Unit (MCU)-compatible tiny DNN architectures such as Micro-ACDNet. It reduces the labeling budget for the three benchmark datasets by 50%, 66.7%, and 61%, respectively

The subsequent sections of this chapter are structured as follows: Section 5.2 presents a review on the current state of AL literature and outlines the areas for further research. Section 5.3 introduces the proposed DeepFeatAL framework, followed by Section 5.5's comparison of Deep Incremental Learning (DeepIcL), Standard Deep Active Learning

(SDAL), and DeepFeatAL in Section 5.5. The experimental details are described in Section 5.4, and Section 5.6 concludes the chapter.

5.2 Current Literature

In this section we summarize the current AL literature and outline the areas for further investigation.

The majority of contemporary literature has been devoted to event classification. Han et al. [124] employ the open-source openSMILE [29] toolkit to extract audio features from the FindSound [212] dataset, the least confidence sampling strategy for sample selection, and SVM with linear kernels to train and classify the audio events. The same feature extraction toolkit and classifier are utilized by Qian et al. [207] in the AL system for the classification of bird sounds. They select samples using a random selection technique for human annotation and continue annotating until the budget runs out or the performance of the classifier is not adequate.

Shuyang et al. [130] employ MFCC and its first and second-order derivatives to extract features from audio data. They further use the statistics of MFCCs in each segment: minimum, maximum, median, mean, variance, skewness, kurtosis, median and variance of the first and second-order derivatives in [208]. They cluster the data using k-medoid clustering and annotated the medoids of each cluster as local representatives. The label is then spread throughout the cluster. Once the budget for annotation is exhausted, SVM and RF are trained to predict the labels in the former and the later studies, respectively. If there is a discrepancy between propagated and predicted labels, the labels are submitted to humans for correction.

Coleman et al. [134] use MFCC, LPMS, and Chromagram for audio feature extraction from ESC with 50 classes (ESC-50) [154] dataset, train SVM with a small amount of annotated data for prediction of the labels of other samples, and use the least confidence scores to select samples for further human annotation. Hilasaca et al. [135] also extract features from spectrograms for soundscape ecology data, cluster the data using k-medoid clustering, and select samples from clusters using random, medoid (samples closest to the cluster centroid), contour (samples furthest from the centroid), and their combinations for human annotation. They use Random Forest (RF) classifier to predict the labels for the rest of the data.

Kholghi et al. [136] use acoustic indices for feature extraction, k-means clustering, and hierarchical clustering algorithms to cluster the data, and randomly listen to sounds from each cluster for human annotation. They train RF using the annotated data and use it to predict the labels.

Ji et al. [210] and Qin et al. [209] use Gabor Dictionary [32] and a learned dictionary for their for their AL based audio classification task on UrbanSound8k Dataset (US8K) and

ESC-50 datasets. Both use k-medoid clustering to cluster the data and manually label the medoid of each cluster and propagate the labels to the clusters.

Ash et al. [128] and Shi et al. [33] retrain the model in every iteration of human annotation and at the end of the learning process, they use the final trained model for labeling. Ash et al. [128] adopts diverse gradient embeddings and k-means++ [213] seeding algorithm in their acquisition function (i.e., BADGE) to take predictive uncertainty and sample diversity into consideration. Ash et al. [128] have tested their proposed method on image data only, whereas Shi et al. [33] have used extremely low-frequency (i.e., 1kHz) data and manually constructed features.

Unlike the previous studies, Wang et al. [137] extracted features from the sonic sensor data using a pretrained VGGish audio model [31]. They selected samples using uncertainty sampling technique (i.e., least confidence score) and trained the RF classifier to label the data at the end of human annotation.

In contrast to audio event classification, Kim and Pardo [214, 215] utilize MFCC to extract acoustic features from DCASE2015 [216] dataset for audio event detection. Then, they calculate the distance among the samples using the nearest neighbor algorithm, rank the samples nearest to the previously annotated sample as high, and choose them for further annotation by a human expert. Shuyang et al. [211] employ spectrogram of TUT Rare Sound Events 2017 [217] and TAU Spatial Sound Events 2019 [218] datasets [211] and change point detection to identify segments from the spectrograms that have events. The segments are then clustered using k-medoid clustering, and the medoid of each cluster is annotated. The samples for human annotation are picked using the mismatch-first-farthest traversal approach suggested in their earlier study [208]. In order to detect and classify the rest of the samples, they train a DNN model architecture presented in [219].

According to the above discussion, k-medoid clustering [220] and the Farthest Traversal [131] appear to be the most popular sample selection techniques. However, due to their computational complexity, they are not scalable for large amounts of data.

Crucially, all the proposed AL processes rely on fixed feature sets that do not change over the course of the AL life cycle. When a pretrained model is used to extract features, it is never refined. It has never been investigated whether the fixed feature set is sufficiently flexible to allow the model's behavior to be accommodated to time-varying data features. We intend to investigate this issue in this study.

This AL project endeavors something fundamentally different from the aforementioned works. This project, unlike its predecessors, aims to incorporate the feature extractor into the active learning loop so that it can be fine-tuned in each iteration. Furthermore, it aims to learn raw audio-based features using automatic feature extraction techniques. Chapters 1, 3, and 4 have established that raw audio is the optimal way for end-to-end

audio event classification and detection. To the best of our knowledge, there has been no study conducted specifically on raw audio.

5.3 Proposed Active Learning Framework (DeepFeatAL)

The proposed Deep Active Feature Learning (DeepFeatAL) system refines feature extraction and classification from raw audio. It aims to improve feature extraction quality and recognition accuracy while requiring minimal data annotation effort. Figure 5.1 depicts the detailed construction of the DeepFeatAL.

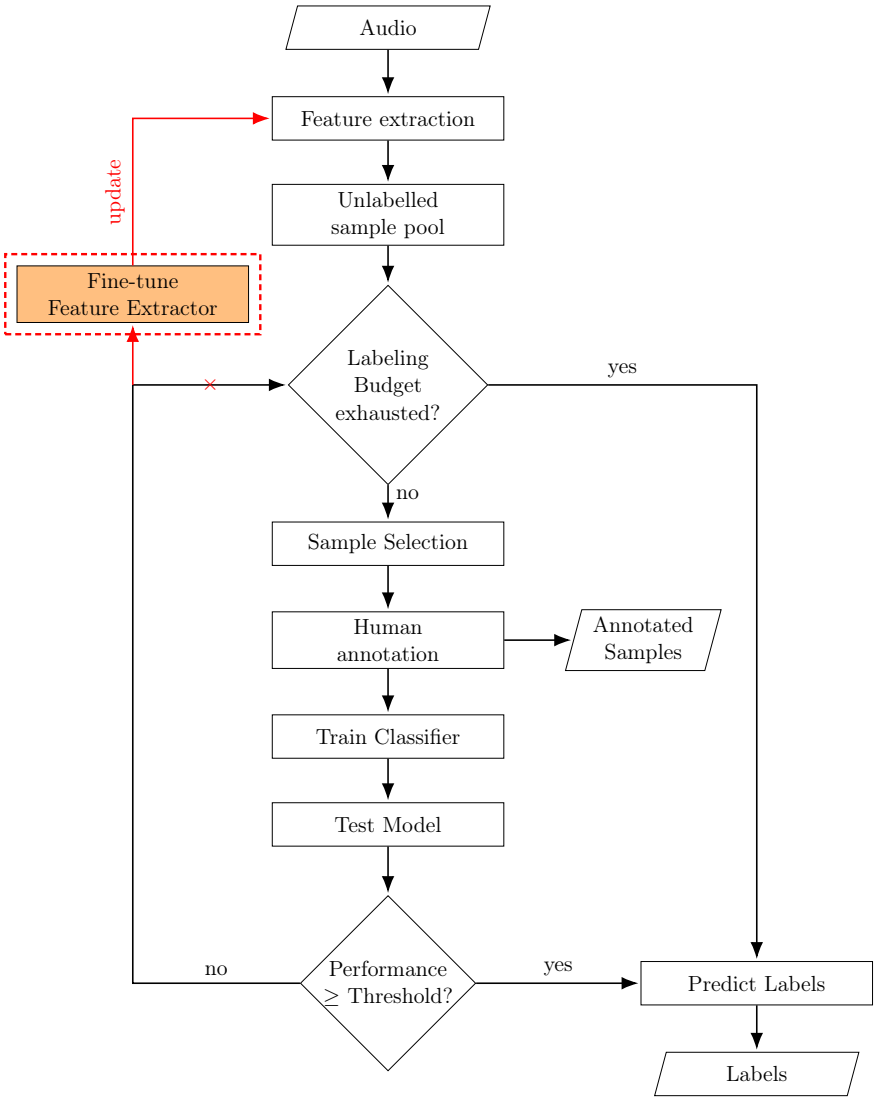


FIGURE 5.1: The detailed architecture of proposed DeepFeatAL

A pretrained DNN model is employed as a feature extractor. The output layer is removed from the pretrained model to accomplish this. The unlabeled samples are placed in the unlabeled sample pool with the extracted features. We adopt the BADGE [128] sample selection technique in our acquisition function, which combines diverse gradient embeddings and k-means++ [213] seeding algorithm to take predictive uncertainty and sample diversity into consideration to identify the most challenging samples.

The feature extractor is fine-tuned in each iteration of human annotation (simulated) on the annotated examples to improve feature embeddings. The previously extracted features are then replaced with new features extracted from the fine-tuned model. This iterative process is repeated until the labeling budget is drained or a desirable level of classification accuracy is attained. We then retrain the classifier(s) using the extracted features from the feature extractor. We present our proposed system in Algorithm 2.

Algorithm 2: DeepFeatAL

Input: DNN Model (M), Labeling Budget (B), Fine-tune Epochs (E), Performance Threshold (P_{thres}), No. of Classes (C)
Output: Labelled Data (L)
Data: Labeled Set (S_l), Validation set (S_v), Test Set (S_d), Unlabeled Pool (S_u)

$M_{found} \longleftarrow FALSE$
$N \longleftarrow$ no. of samples to be selected
$AF \longleftarrow$ Acquisition Function
$HA \longleftarrow$ Human Annotation
$l_c \longleftarrow GetLayerCount(M)$
`/* last convolution index */`
$lconv \longleftarrow l_c - 2$
`/* Replace the filters of last convolution and train */`
$M \longleftarrow M.layers[lconv].filters \longleftarrow 5 * C)$
$M \longleftarrow Train(M)$
while $B > len(S_l)$ **and** M_{found} **is** $FALSE$ **do**
    `/* Keep layers up to the last convolution */`
    $M_f \longleftarrow M - M[0 : lconv]$
    $S \longleftarrow AF(M_f(N))$
    $S_a \longleftarrow HA(S)$
    $S_u \longleftarrow S_u - S_a$
    $S_l \longleftarrow S_l \cup S_a$
    $S_t \longleftarrow S_a \cup S_t$[Random $len(S_a)$ samples]
    $val \longleftarrow 0.0$
    foreach e **in** E **do**
        $M' \longleftarrow$ Fine-tune M on S_t by minimizing KLD loss
        $val' \longleftarrow M'(S_v)$
        if $val' > val$ **then**
            $val \longleftarrow val'$
            $M \longleftarrow M'$
    $R_{es} \longleftarrow M(S_d)$
    if $R_{es} \geq P_{thres}$ **then**
        $M_{found} \longleftarrow TRUE$
$L \longleftarrow M(S_u) \cup S_l$
return L

5.4 Experimental Details

All experiments were conducted using Python version 3.7.4 and the PyTorch 1.8.1.

5.4.1 Datasets

Experiments were conducted on three widely used audio benchmark datasets: ESC-50 [154], US8K [155] and InsectWingBeat (iWingBeat) [46]. The first two datasets are described in detail in Chapter 3 Section 3.3.2. The iWingBeat dataset includes 50,000 labeled audio clips (1s each) from 10 classes of insects, with 5,000 examples per class. We resampled all audio samples from all datasets to 20kHz.

5.4.2 Splitting the Datasets for DeepIcL, Standard AL and DeepFeatAL

We began by separating the datasets into two parts. One is used for training, validation, and testing, while the rest are placed in an unlabeled data pool. For the initial task (i.e., training, validation and testing) we keep 50%, 50% and 35% data instances from every class of the ESC-50, US8K and iWingBeat datasets, respectively. We separated the data in this way (i.e., class-wise) to ensure that the unlabeled samples have the same statistical distribution.

We then create training sets, validation sets and test sets with 40%, 20% and 40% of the reserved data (for training, validation and testing) of ESC-50 dataset, 40%, 20% and 40% of US8K dataset and 57%, 28.5% and 28.5% of iWingBeat dataset, respectively.

Finally, we move the remaining data to the unlabeled data pool. Experiments in this chapter are based on the assumption that agents offer highly trustworthy annotations, as the system's reliability would suffer if we supplied it with unreliable annotations.

5.4.3 Data Preprocessing

For ESC-50 and US8K, we follow the procedure described in Chapter 3. However, for the iWingBeat dataset, we use input samples of length 20,000, or 1s audio at 20kHz since the recordings are only 1s long. We used the data augmentation techniques described in Chapters 3 and 4 to augment the training sets of ESC-50 and US8K datasets. However, we do not use the mixup of two classes for the training set augmentation and 10-crops of test data.

5.4.4 Hyperparameter Settings for Initial Training

ACDNet was trained for 600 epochs with a learning rate scheduler 0.3, 0.6, 0.9 with the same learning settings we have described in Chapter 3. At the end of training and validation, we use the best model for DeepIcL, SDAL and DeepFeatAL.

5.4.5 Learning Settings for DeepIcl, Standard AL and DeepFeatAL

We use seven, fifteen, and twenty iterations of human annotation (simulated) on the ESC-50, US8K, and iWingBeat datasets, respectively. In each iteration, the simulated annotation system is requested to provide the labels of 100 specific unlabeled samples (determined by the acquisition function) for the first two datasets and 500 unlabeled samples for the third dataset. In each iteration of the simulated labeling, we fine-tune the model for 100 epochs for each dataset during DeepIcL and DeepFeatAL; however, we add the same number of old training data to the newly labeled data to prevent the network from forgetting its prior knowledge. During this stage of learning with new data, we employ a significantly lower learning rate (i.e., 0.001), a new learning rate scheduler of [15, 60, 90] without any warm-ups and a batch size of 16.

5.5 Comparative Analysis

In this section, we compare DeepFeatAL's performance to that of DeepIcL and the existing SDAL.

This analysis is presented using three standard benchmark datasets: ESC-50 [221], US8K [155] and iWingBeat [46]. A preliminary study was conducted to determine the fine-tuning strategy. Our experiments (see Section 5.5.1) show that fine-tuning the entire model produces the best results.

For DeepIcL, the samples are randomly selected for annotation and then standard incremental learning technique is used. We have implemented the process from scratch. In contrast, for the SDAL and DeepFeatAL experiments, samples are selected using the BADGE [128] sample selection technique. The implementation of BADGE is retrieved from [222] and used in our AL implementation. In each case, ACDNet was used as the feature extractor. We fine-tuned the feature extractor during DeepFeatAL, which was not the case with DeepIcL and SDAL.

We begin by contrasting the performance of DeepIcL and SDAL. We used three different classifiers from scikit-learn [223] for the SDAL experiments: KNeighborsClassifier, LogisticRegression and RidgeClassifier. We call them Active Learning using Logistic Regression (AL-LogisticReg), Active Learning using K-Neighbors Classifier (AL-KNC) and Active Learning using Ridge Classifier (AL-RidgeC), in that order. After analyzing, we select the best method for each dataset and compare its performance to DeepFeatAL's performance on each of the three datasets.

We use the full-sized ACDNet model for the first set of experimental analysis. Our analysis shows that for ESC-50 and US8K, AL-RidgeC and AL-LogisticReg outperforms DeepIcL, respectively, while for iWingBeat, DeepIcL outperforms SDAL. We then compared these methods to our proposed DeepFeatAL and discovered that DeepFeatAL performs significantly better. DeepFeatAL requires approximately 14.28%, 66.67%, and

50% less labeling effort than other approaches on the ESC-50, US8K, and iWingBeat datasets, respectively.

We repeated the same experiments with Micro-ACDNet and obtained the same results. DeepFeatAL outperforms all other methods for all the three datasets, requiring approximately 42.85%, 66.67%, and 60% less labeling effort than other methods, respectively.

This proves the hypothesis we set out to investigate, namely that including feature extraction in the AL loop improves performance and helps to save labeling effort.

5.5.1 NoFreeze vs. Freeze Layers for Incremental Learning

Three distinct strategies were used with ACDNet to fine-tune the feature extractor in this preliminary analysis for incremental learning. The first step is to allow all ACDNet layers to fine-tune their weight (no-freeze). The second allows only the last three layers to fine-tune (fixed-freeze), while the third unfreezes the last three layers for a few epochs, then two more layers for another few epochs, and finally two more layers (scheduled-freeze). This research was carried out using the ESC-50 [221] dataset. Our experiments show that fine-tuning the entire network yields the best results (see Figure 5.2).

Figure 5.2 generated from Table 5.1 shows that the model's performance was significantly improved through the fine-tuning process when the no-freeze policy was followed. As a result, for the remainder of this chapter, we will refer to this task as fine-tuning.

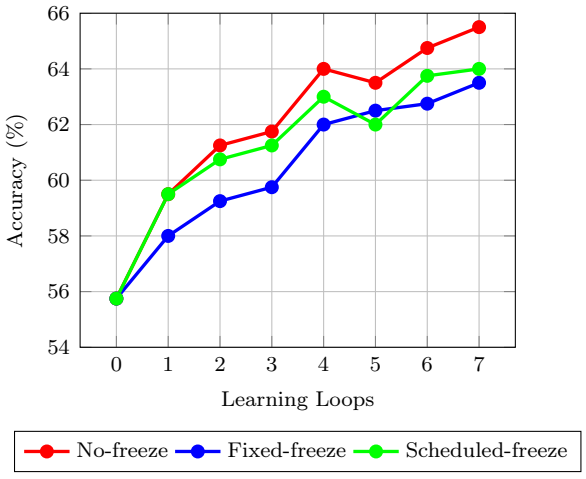


FIGURE 5.2: ACDNet in different training settings for incremental Learning. The first row indicates the performance of the model before incremental learning.

The outcomes of all three steps of fine-tuning are provided in Table 5.1. In learning loops 1-7, the model was fine-tuned with 100 new data instances.

Learning Loops	Prediction Accuracy (%)		
	No-freeze	Fixed-freeze	Scheduled-freeze
0	55.75	55.75	55.75
1	59.50	58.00	59.50
2	61.25	59.25	60.75
3	61.75	59.75	61.25
4	64.00	62.00	63.00
5	63.50	62.50	62.00
6	64.75	62.75	63.75
7	**65.50**	63.50	64.00

TABLE 5.1: ACDNet in different training settings for incremental Learning. The first row indicates the performance of the model before incremental learning.

5.5.2 Analysis with Full-sized ACDNet Model

This section presents the detailed analysis for DNN models run on high performance computing systems that can accommodate very large models. At first we compare DeepIcL with SDAL and pick the best performing method and then compare the performance of the best method to the performance of DeepFeatAL. We represent accuracy by means of 95% Confidence Interval (95%CI). For DeepIcL, we select samples at random from the unlabeled pool and tune ACDNet. The method resembles Algorithm 2 excluding the acquisition function.

Note that we simulate the "human annotation" process by withholding the labels of data placed in the unlabeled pool and delivering them on demand, just as a human expert would. In the remainder of this thesis, the term "human annotation" refers to the simulated process.

5.5.2.1 Deep Incremental Learning (DeepIcL) vs Standard Deep Active Learning (SDAL) with ACDNet

This section investigates whether using randomly selected samples or samples that the classifier finds particularly difficult improves learning. We present the analyses on the three datasets in the following order: ESC-50, US8K, and iWingBeat.

Figure 5.3, which is generated from Table 5.2, shows that AL-RidgeC achieves the highest prediction accuracy achieved by DeepIcL (i.e., 65.43%) in only 4 iterations on the ESC-50 dataset, saving ≈43% of the labeling budget. At the end of seven human labeling iterations, AL-RidgeC achieved the highest accuracy of 67.94%. The difference in prediction accuracy ensures that AL aids classifiers in learning particularly difficult samples, allowing them to produce better classification performance. We will compare the performance of AL-RidgeC to that of our proposed DeepFeatAL in Section 5.5.2.2.

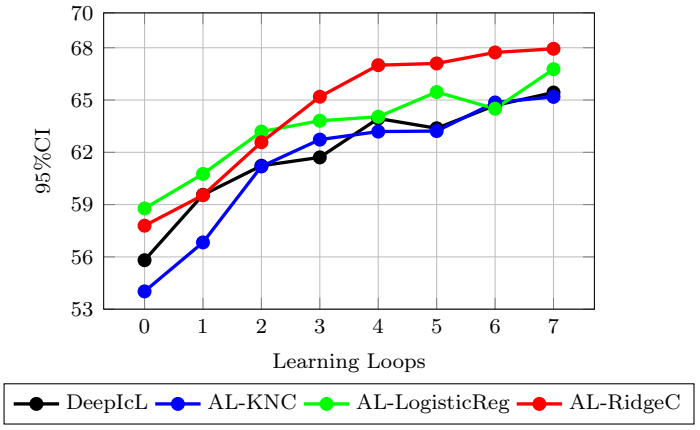


FIGURE 5.3: DeepIcL vs SDAL using ACDNet on ESC-50 dataset

Learning Loops	Accuracy(95%CI)			
	DeepIcL	AL-KNC	AL-LogisticReg	AL-RidgeC
0	55.81 ± 0.15	54.02 ± 0.15	58.78 ± 0.15	57.79 ± 0.16
1	59.56 ± 0.15	56.83 ± 0.14	60.76 ± 0.16	59.54 ± 0.14
2	61.23 ± 0.15	61.19 ± 0.15	63.19 ± 0.15	62.58 ± 0.15
3	61.71 ± 0.15	62.73 ± 0.15	63.81 ± 0.16	65.19 ± 0.14
4	63.94 ± 0.15	63.19 ± 0.14	64.04 ± 0.15	**67.00 ± 0.15**
5	63.38 ± 0.15	63.22 ± 0.15	**65.46 ± 0.15**	67.10 ± 0.15
6	64.69 ± 0.15	64.86 ± 0.14	64.50 ± 0.15	67.73 ± 0.15
7	**65.43 ± 0.15**	65.18 ± 0.15	66.77 ± 0.15	67.94 ± 0.14

TABLE 5.2: DeepIcL vs SDAL using ACDNet on ESC-50 dataset

The same comparative analysis is performed on the US8K dataset. Figure 5.4 populated from Table 5.3 demonstrates that AL-KNC and AL-LogisticReg learn to predict the data significantly better than DeepIcL. Furthermore, after only 2 iterations, AL-LogisticReg achieves the highest prediction accuracy achieved by DeepIcL (i.e., 88.19%), saving nearly 86.67% of the labeling budget. As US8K has accumulated more data, we have completed fifteen iterations of human labeling for incremental learning for ACDNet and the SDAL process. Since AL-LogisticReg outperforms AL-KNC most of the time, we will compare its performance to our proposed DeepFeatAL performance in Section 5.5.2.2.

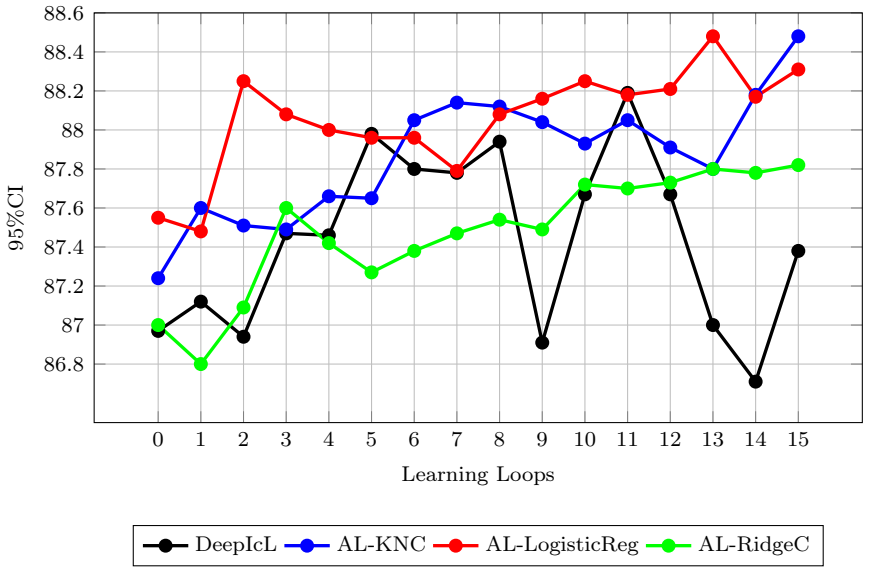


FIGURE 5.4: DeepIcL vs SDAL on US8K dataset

Learning Loops	Accuracy (95%CI)			
	DeepIcL	AL-KNC	AL-LogisticReg	AL-RidgeC
0	86.97 ± 0.06	87.24 ± 0.05	87.55 ± 0.05	87.00 ± 0.05
1	87.12 ± 0.05	87.60 ± 0.05	87.48 ± 0.05	86.80 ± 0.05
2	86.94 ± 0.04	87.51 ± 0.05	**88.25 ± 0.04**	87.09 ± 0.05
3	87.47 ± 0.06	87.49 ± 0.05	88.08 ± 0.04	87.60 ± 0.04
4	87.46 ± 0.05	87.66 ± 0.05	88.00 ± 0.05	87.42 ± 0.05
5	**87.98 ± 0.04**	87.65 ± 0.04	87.96 ± 0.05	87.27 ± 0.06
6	87.80 ± 0.05	**88.05 ± 0.05**	87.96 ± 0.05	87.38 ± 0.05
7	87.78 ± 0.06	88.14 ± 0.05	87.79 ± 0.05	87.47 ± 0.04
8	87.94 ± 0.05	88.12 ± 0.04	88.08 ± 0.04	87.54 ± 0.05
9	86.91 ± 0.05	88.04 ± 0.05	88.16 ± 0.05	87.49 ± 0.05
10	87.67 ± 0.05	87.93 ± 0.05	88.25 ± 0.04	87.72 ± 0.04
11	88.19 ± 0.06	88.05 ± 0.05	88.18 ± 0.05	87.70 ± 0.06
12	87.67 ± 0.05	87.91 ± 0.05	88.21 ± 0.05	87.73 ± 0.05
13	87.00 ± 0.04	87.80 ± 0.05	88.48 ± 0.06	87.80 ± 0.05
14	86.71 ± 0.06	88.18 ± 0.05	88.17 ± 0.05	87.78 ± 0.05
15	87.38 ± 0.04	88.48 ± 0.05	88.31 ± 0.05	87.82 ± 0.05

TABLE 5.3: DeepIcL vs SDAL on US8K dataset

The third dataset on which the same comparative analysis has been performed is iWingBeat. Figure 5.5, which is populated from Table 5.4, reveals that DeepIcL has the highest prediction accuracy. This scenario is clearly different from the previous two datasets. DeepIcL achieves 68.27% at the end of human annotation (simulated), which is higher than the accuracy of SDAL models (65.87%, 67.17% and 66.28%). Notably, the performance of the initial model and the model obtained after human annotations (simulated) and fine-tuning do not differ significantly. Since this dataset contains more labeled samples than the previous two, we ran twenty human annotation (simulated) iterations. The performance of models using different learning techniques is shown in Table 5.4. We will compare the performance of DeepIcL with that of our proposed DeepFeatAL.

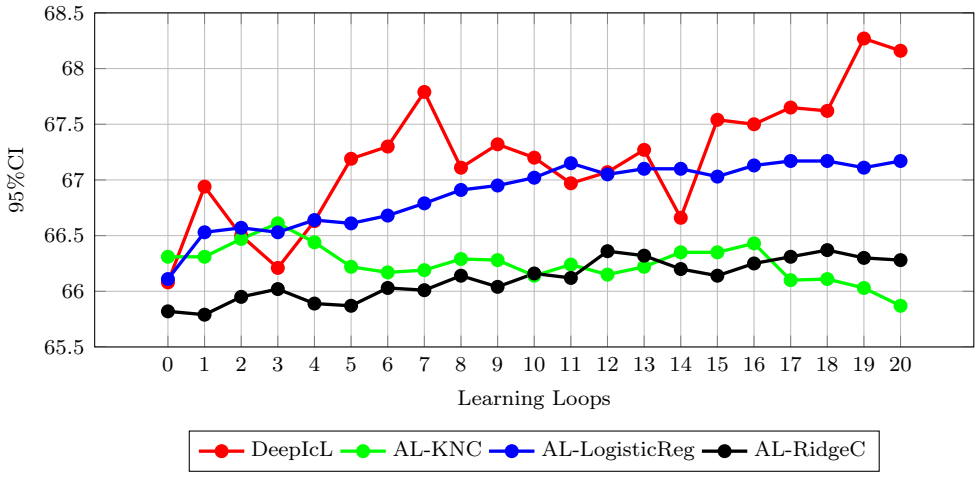


FIGURE 5.5: DeepIcL vs SDAL on iWingBeat dataset

Learning Loops	Accuracy (95%CI)			
	DeepIcL	AL-KNC	AL-LogisticReg	AL-RidgeC
0	66.08 ± 0.04	66.31 ± 0.04	66.11 ± 0.04	65.82 ± 0.04
1	66.94 ± 0.04	66.31 ± 0.04	66.53 ± 0.04	65.79 ± 0.04
2	66.50 ± 0.04	66.47 ± 0.04	66.57 ± 0.04	65.95 ± 0.04
3	66.21 ± 0.04	66.61 ± 0.04	66.53 ± 0.04	66.02 ± 0.04
4	66.63 ± 0.04	66.44 ± 0.04	66.64 ± 0.04	65.89 ± 0.04
5	67.19 ± 0.04	66.22 ± 0.04	66.61 ± 0.04	65.87 ± 0.04
6	67.30 ± 0.04	66.17 ± 0.04	66.68 ± 0.04	66.03 ± 0.04
7	67.79 ± 0.04	66.19 ± 0.04	66.79 ± 0.04	66.01 ± 0.04
8	67.11 ± 0.04	66.29 ± 0.04	66.91 ± 0.04	66.14 ± 0.04
9	67.32 ± 0.04	66.28 ± 0.04	66.95 ± 0.04	66.04 ± 0.04
10	67.20 ± 0.04	66.14 ± 0.04	67.02 ± 0.04	66.16 ± 0.04
11	67.97 ± 0.04	66.24 ± 0.04	67.15 ± 0.04	66.12 ± 0.04
12	67.07 ± 0.04	66.15 ± 0.04	67.05 ± 0.04	66.36 ± 0.04
13	67.27 ± 0.04	66.22 ± 0.04	67.10 ± 0.04	66.32 ± 0.04
14	66.66 ± 0.04	66.35 ± 0.04	67.10 ± 0.04	66.20 ± 0.04
15	67.54 ± 0.04	66.35 ± 0.04	67.03 ± 0.04	66.14 ± 0.04
16	67.50 ± 0.04	66.43 ± 0.04	67.13 ± 0.04	66.25 ± 0.04
17	67.65 ± 0.04	66.10 ± 0.04	67.17 ± 0.04	66.31 ± 0.04
18	67.62 ± 0.04	66.11 ± 0.04	67.17 ± 0.04	66.37 ± 0.04
19	**68.27 ± 0.04**	66.03 ± 0.04	67.11 ± 0.04	66.30 ± 0.04
20	68.16 ± 0.04	65.87 ± 0.04	67.17 ± 0.04	66.28 ± 0.04

TABLE 5.4: DeepIcL vs SDAL on iWingBeat dataset

5.5.2.2 DeepFeatAL vs Others with ACDNet

We have three best methods for three datasets from the previous section. We now compare their performance to that of DeepFeatAL. Table 5.5 summarizes the three best methods (detailed in Table 5.2, Table 5.3, and Table 5.4) and the performance of DeepFeatAL. We follow the same testing regime as in the previous sections. In the case of DeepFeatAL, we have used ACDNet as the model for learning and classification as

opposed to SDAL. The main difference between previous methods and DeepFeatAL is that it is calibrated with the annotated samples and the feature extractor is updated.

This comparative study demonstrates that DeepFeatAL significantly outperforms the best-performing methods of the preceding section. On ESC-50, US8K, and iWingBeat datasets, DeepFeatAL requires approximately 14.28%, 66.67%, and 50% less labeling effort than its counterparts, respectively.

Furthermore, our significance test shows that the performance of the model produced by DeepFeatAL is statistically significant. We used the statistical significance test described in [44] to determine the statistical significance of the performance of each model. We begin by rejecting the null hypothesis using the Friedman test [187]. Then, we conduct a pairwise post-hoc analysis in accordance with Benavoli et al. [224], utilizing the Wilcoxon signed-rank test [188] and Holm's alpha (5%) correction [225, 226]. We employ Demšar [227]'s Critical Difference (CD) diagram as a graphical representation. A thick horizontal line in the CD diagram indicates that the accuracy of a group of classifiers is insignificant.

Learning	95%CI for ESC-50		95%CI for US8K		95%CI for iWingBeat	
Loops	AL-RidgeC	DeepFeatAL	AL-LogisticReg	DeepFeatAL	DeepIcL	DeepFeatAL
0	57.79 ± 0.16	55.81 ± 0.16	87.55 ± 0.05	86.97 ± 0.06	66.08 ± 0.04	66.08 ± 0.04
1	59.54 ± 0.14	63.76 ± 0.14	87.48 ± 0.05	85.93 ± 0.05	66.94 ± 0.04	66.59 ± 0.04
2	62.58 ± 0.15	64.15 ± 0.15	88.25 ± 0.04	86.82 ± 0.05	66.50 ± 0.04	66.99 ± 0.04
3	65.19 ± 0.14	65.00 ± 0.16	88.08 ± 0.04	87.15 ± 0.06	66.21 ± 0.04	67.57 ± 0.04
4	67.00 ± 0.15	65.94 ± 0.14	88.00 ± 0.05	87.09 ± 0.05	66.63 ± 0.04	67.13 ± 0.04
5	67.10 ± 0.15	67.28 ± 0.15	87.96 ± 0.05	**88.59 ± 0.06**	67.19 ± 0.04	67.40 ± 0.04
6	67.73 ± 0.15	**68.74 ± 0.14**	87.96 ± 0.05	88.20 ± 0.05	67.30 ± 0.04	67.76 ± 0.04
7	**67.94 ± 0.14**	70.02 ± 0.12	87.79 ± 0.05	88.94 ± 0.06	67.79 ± 0.04	67.66 ± 0.04
8	-	-	88.08 ± 0.04	88.62 ± 0.05	67.11 ± 0.04	68.10 ± 0.04
9	-	-	88.16 ± 0.05	88.60 ± 0.05	67.32 ± 0.04	68.03 ± 0.04
10	-	-	88.25 ± 0.04	88.70 ± 0.06	67.20 ± 0.04	**68.86 ± 0.04**
11	-	-	88.18 ± 0.05	88.75 ± 0.06	67.97 ± 0.04	68.95 ± 0.04
12	-	-	88.21 ± 0.05	88.80 ± 0.05	67.07 ± 0.04	68.90 ± 0.04
13	-	-	**88.48 ± 0.06**	89.01 ± 0.06	67.27 ± 0.04	69.05 ± 0.04
14	-	-	88.17 ± 0.05	89.20 ± 0.06	66.66 ± 0.04	69.29 ± 0.04
15	-	-	88.31 ± 0.05	89.45 ± 0.05	67.54 ± 0.04	69.32 ± 0.04
16	-	-	-	-	67.50 ± 0.04	69.76 ± 0.04
17	-	-	-	-	67.65 ± 0.04	70.12 ± 0.04
18	-	-	-	-	67.62 ± 0.04	70.04 ± 0.04
19	-	-	-	-	**68.27 ± 0.04**	70.18 ± 0.04
20	-	-	-	-	68.16 ± 0.04	70.13 ± 0.04

Table 5.5: DeepIcL vs SDAL vs DeepFeatAL on All Datasets

Figure 5.6 compares the performance of AL-RidgeC and DeepFeatAL on the ESC-50 dataset. At the end of the AL loops, DeepFeatAL achieves the highest prediction accuracy. Its counterpart achieves better accuracy only after the fourth iteration. DeepFeatAL performs better in the remaining six cases. Additionally, the SDAL technique's highest accuracy has been achieved in six iterations of DeepFeatAL, saving 14.28% of the labeling budget.

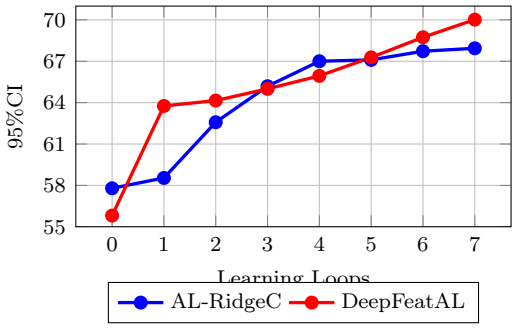


FIGURE 5.6: AL-RidgeC vs DeepFeatAL on ESC-50

Figure 5.7 displays the statistical significance of the final models obtained after completing all human iterations for DeepIcL, SDAL, and DeepFeatAL on the ESC-50 dataset. According to the figure, all the models perform statistically significantly. It demonstrates that the model produced by DeepFeatAL outperforms other models produced by other methods and ranks first.

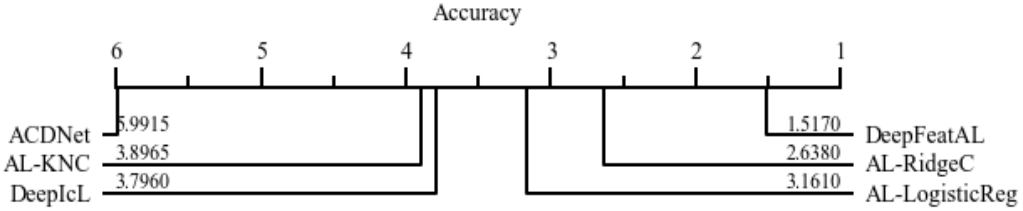


FIGURE 5.7: CD diagram for statistical significance test of performance of different learning methods on ESC-50 dataset.

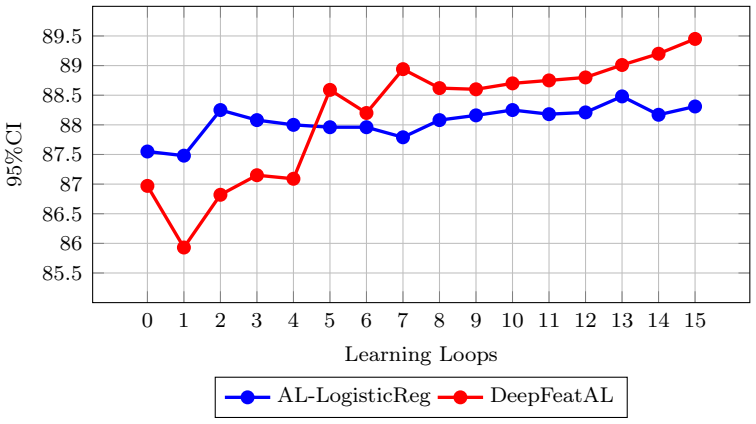


FIGURE 5.8: AL-LogisticReg vs DeepFeatAL on US8K

Figure 5.8 presents the same comparison between performance of AL-LogisticReg and DeepFeatAL on the US8K dataset. It clearly shows that at the end of the AL loops, DeepFeatAL achieves the best prediction accuracy. The highest accuracy achieved by AL-LogisticReg was 88.48% after iteration thirteen, whereas DeepFeatAL reached higher accuracy after only five iterations (88.59%); thus, saving the labeling effort by 66.67%.

Figure 5.9 depicts the statistical significance of the final models obtained upon completion of all human iterations for all applied methods DeepIcL, SDAL, and DeepFeatAL on the US8K dataset. According to the figure, all the models perform statistically significantly.

It demonstrates that the model produced by DeepFeatAL outperforms other models produced by other methods and ranks first.

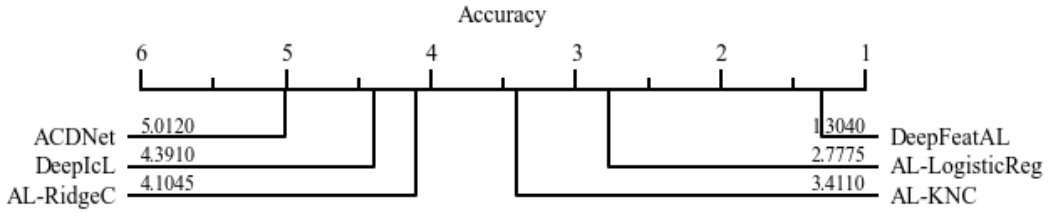


FIGURE 5.9: CD diagram for statistical significance test of performance of different learning methods on US8K dataset.

Figure 5.10 compares the performance of DeepIcL, AL-LogisticReg and DeepFeatAL with respect to the iWingBeat dataset. It demonstrates that at the end of the AL loops, DeepFeatAL significantly outperforms its competitors. The highest accuracy achieved by DeepIcL was 68.27%, whereas DeepFeatAL achieved 68.86% after only ten iterations. Thus, it saves 50% of the labeling effort.

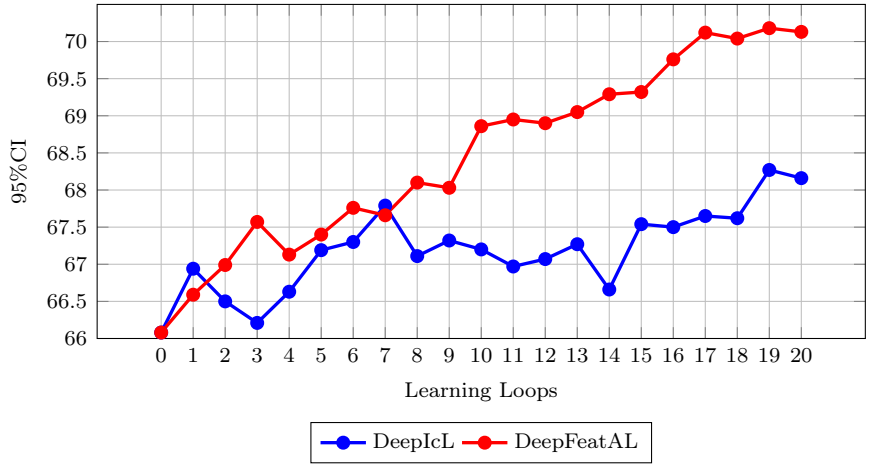


FIGURE 5.10: DeepIcL vs DeepFeatAL on iWingBeat dataset

Figure 5.11 depicts the statistical significance of the final models obtained following the completion of all human iterations for all the applied methods DeepIcL, SDAL and DeepFeatAL on the iWingBeat dataset. According to the figure, all the models perform statistically significantly. It demonstrates that the model produced by DeepFeatAL outperforms other models produced by other methods and ranks first.

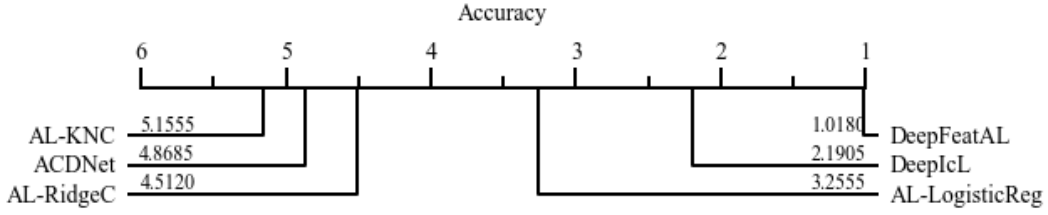


FIGURE 5.11: CD diagram for statistical significance test of different learning methods on iWingBeat dataset.

5.5.3 Analysis with Edge-based Micro-ACDNet Model

We have demonstrated the advantage of DeepFeatAL in the previous section. We aim to carry out the same experiments on the same fully-resourced computing environment in order to train Micro-ACDNet, which will then be deployed on MCUs and run inference on on them. The goal is to see if the advantage can be extended to the MCU-compatible models. Based on the experimental analysis we have performed in the following sections, DeepFeatAL significantly outperforms all other methods for all the three datasets, requiring approximately 42.85%, 66.67%, and 60% less labeling effort than other methods, respectively.

5.5.3.1 Deep Incremental Learning (DeepIcL) vs Standard Active Learning (SDAL) with Micro-ACDNet

Following the same methodology as in Section 5.5.2, we first compare the performance of DeepIcL and SDAL on the three datasets and then select the best model for each dataset to compare with the performance of the fine-tuned micro-model produced by our proposed DeepFeatAL.

Figure 5.12 generated from the data presented in Table 5.6 demonstrates that AL-LogisticReg and AL-RidgeC outperform DeepIcL after the second iteration. The highest accuracy produced by DeepIcL is $57.70 \pm 0.15\%$, whereas AL-LogisticReg achieves this accuracy after only three iterations; thus, saving more than 47% of the labeling budget. This difference in performance ensures that active learning assists classifiers in mastering particularly difficult samples, ultimately resulting in improved classification performance. These results are consistent with the results for the full-size network.

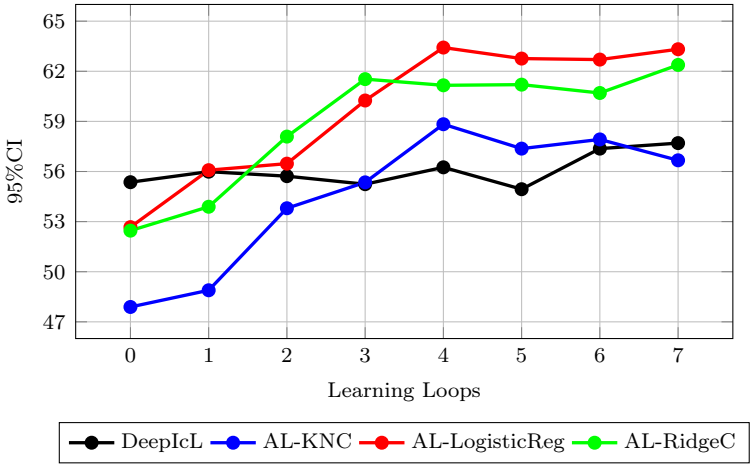


FIGURE 5.12: Edge-based DeepIcL vs SDAL on ESC-50 dataset

Table 5.6 displays the 95%CI for each model in the ESC-50 dataset. After seven iterations of human annotation (simulated), the prediction accuracy of DeepIcL is $57.70 \pm 0.15\%$, whereas AL-KNC, AL-LogisticReg and AL-RidgeC are $56.67 \pm 0.15\%$, $63.32 \pm 0.15\%$ and $62.38 \pm 0.14\%$, respectively. Consequently, AL-LogisticReg provides the highest accuracy for this dataset.

Learning Loops	Accuracy(95%CI)			
	DeepIcL	AL-KNC	AL-LogisticReg	AL-RidgeC
0	55.36 ± 0.14	47.89 ± 0.15	52.67 ± 0.15	52.46 ± 0.16
1	55.99 ± 0.15	48.89 ± 0.14	56.08 ± 0.16	53.89 ± 0.14
2	55.72 ± 0.15	53.80 ± 0.15	56.47 ± 0.15	**58.09 ± 0.15**
3	55.24 ± 0.15	55.35 ± 0.15	**60.25 ± 0.16**	61.53 ± 0.14
4	56.25 ± 0.15	**58.83 ± 0.14**	60.25 ± 0.15	61.16 ± 0.15
5	54.94 ± 0.15	57.37 ± 0.15	62.76 ± 0.15	61.20 ± 0.15
6	57.37 ± 0.15	57.92 ± 0.14	62.70 ± 0.15	60.70 ± 0.15
7	**57.70 ± 0.15**	56.67 ± 0.15	63.32 ± 0.15	62.38 ± 0.14

TABLE 5.6: Edge-based DeepIcL vs SDAL on ESC-50 dataset

Same comparative analysis was carried out on the US8K dataset. Figure 5.13 demonstrates that during incremental learning, DeepIcL produces the highest prediction accuracy of 83.13%. In contrast, AL-LogisticReg, achieves this level of accuracy after only nine rounds of human labeling, thus, saving 40% of the labeling budget. Table 5.7 displays 95%CI for all the models. After fifteen iterations of human annotation (simulated), DeepIcL achieves a prediction accuracy of 82.63 ± 0.06%, whereas AL-KNC, AL-LogisticReg and AL-RidgeC achieve 83.15±0.06%, 83.27±0.06% and 82.10±0.06%, respectively. As a result, AL-LogisticReg produces the highest accuracy for this dataset as well.

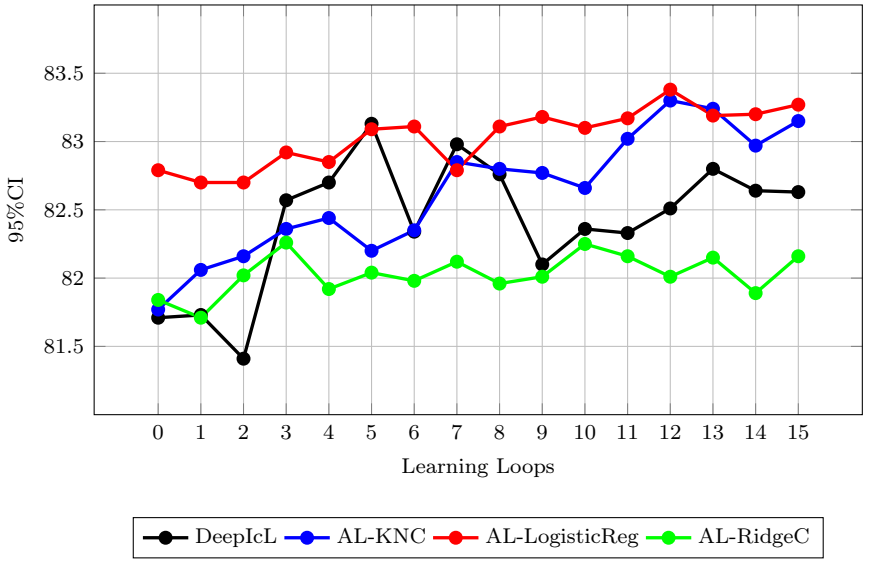


FIGURE 5.13: Edge-based DeepIcL vs SDAL on US8K dataset

Learning Loops	Accuracy (95%CI)			
	DeepIcL	AL-KNC	AL-LogisticReg	AL-RidgeC
0	81.71 ± 0.06	81.77 ± 0.06	82.79 ± 0.06	81.84 ± 0.06
1	81.73 ± 0.06	82.06 ± 0.06	82.70 ± 0.06	81.71 ± 0.06
2	81.41 ± 0.06	82.16 ± 0.06	82.70 ± 0.06	82.02 ± 0.05
3	82.57 ± 0.06	82.36 ± 0.06	82.92 ± 0.06	82.26 ± 0.06
4	82.70 ± 0.06	82.44 ± 0.06	82.85 ± 0.06	81.92 ± 0.06
5	**83.13 ± 0.06**	82.20 ± 0.06	83.09 ± 0.06	82.04 ± 0.06
6	82.34 ± 0.06	82.35 ± 0.06	83.11 ± 0.06	81.98 ± 0.06
7	82.98 ± 0.06	82.85 ± 0.06	82.79 ± 0.06	82.12 ± 0.06
8	82.76 ± 0.06	82.80 ± 0.06	83.11 ± 0.06	81.96 ± 0.06
9	82.10 ± 0.06	82.77 ± 0.06	**83.18 ± 0.06**	82.01 ± 0.06
10	82.36 ± 0.06	82.66 ± 0.06	83.10 ± 0.06	82.25 ± 0.06
11	82.33 ± 0.06	83.02 ± 0.06	83.17 ± 0.06	82.16 ± 0.06
12	82.51 ± 0.06	**83.30 ± 0.06**	83.38 ± 0.06	82.01 ± 0.06
13	82.80 ± 0.06	83.24 ± 0.06	83.19 ± 0.06	82.15 ± 0.06
14	82.64 ± 0.06	82.97 ± 0.06	83.20 ± 0.06	81.89 ± 0.06
15	82.63 ± 0.06	83.15 ± 0.06	83.27 ± 0.06	82.16 ± 0.06

TABLE 5.7: Edge-based DeepIcL vs SDAL on US8K dataset

For iWingBeat dataset, our analysis shows that DeepIcL produces the highest prediction accuracy across the majority of iterations (see Figure 5.14). We observed a similar result in previous section as well.

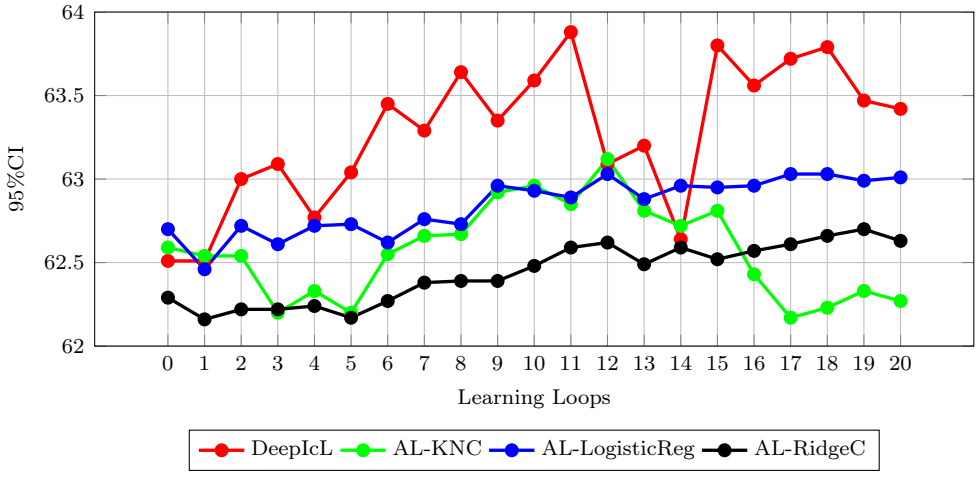


FIGURE 5.14: Edge-based DeepIcL vs SDAL on iWingBeat Dataset

According to Table 5.8, DeepIcL produces a prediction accuracy of $63.42 \pm 0.04\%$ after twenty iterations of human annotation (simulated), whereas AL-KNC, AL-LogisticReg and AL-RidgeC produce $62.27 \pm 0.04\%$, $63.01 \pm 0.04\%$ and $62.63 \pm 0.04\%$, respectively. Therefore, unlike other datasets, DeepIcL provides the highest prediction accuracy for this dataset.

Learning Loops	Accuracy (95%CI)			
	DeepIcL	AL-KNC	AL-LogisticReg	AL-RidgeC
0	62.51 ± 0.04	62.59 ± 0.04	62.70 ± 0.04	62.29 ± 0.04
1	62.51 ± 0.05	62.54 ± 0.04	62.46 ± 0.04	62.16 ± 0.04
2	63.00 ± 0.05	62.54 ± 0.04	62.72 ± 0.04	62.22 ± 0.04
3	63.09 ± 0.04	62.20 ± 0.04	62.61 ± 0.04	62.22 ± 0.04
4	62.77 ± 0.04	62.33 ± 0.04	62.72 ± 0.04	62.24 ± 0.04
5	63.04 ± 0.04	62.20 ± 0.04	62.73 ± 0.04	62.17 ± 0.04
6	63.45 ± 0.05	62.55 ± 0.04	62.62 ± 0.04	62.27 ± 0.04
7	63.29 ± 0.04	62.66 ± 0.04	62.76 ± 0.04	62.38 ± 0.04
8	63.64 ± 0.04	62.67 ± 0.04	62.73 ± 0.04	62.39 ± 0.04
9	63.35 ± 0.04	62.92 ± 0.04	62.96 ± 0.04	62.39 ± 0.04
10	63.59 ± 0.04	62.96 ± 0.04	62.93 ± 0.04	62.48 ± 0.04
11	**63.88 ± 0.04**	62.85 ± 0.04	62.89 ± 0.04	62.59 ± 0.04
12	63.09 ± 0.05	63.12 ± 0.04	63.03 ± 0.04	62.62 ± 0.04
13	63.20 ± 0.05	62.81 ± 0.04	62.88 ± 0.04	62.49 ± 0.04
14	62.64 ± 0.05	62.72 ± 0.04	62.96 ± 0.04	62.59 ± 0.04
15	63.80 ± 0.04	62.81 ± 0.04	62.95 ± 0.04	62.52 ± 0.04
16	63.56 ± 0.04	62.43 ± 0.04	62.96 ± 0.04	62.57 ± 0.04
17	63.72 ± 0.04	62.17 ± 0.04	63.03 ± 0.04	62.61 ± 0.04
18	63.79 ± 0.04	62.23 ± 0.04	62.96 ± 0.04	62.66 ± 0.04
19	63.47 ± 0.04	62.33 ± 0.04	62.99 ± 0.04	62.70 ± 0.04
20	63.42 ± 0.04	62.27 ± 0.04	63.01 ± 0.04	62.63 ± 0.04

TABLE 5.8: Edge-based DeepIcL vs SDAL on iWingBeat dataset

5.5.3.2 DeepFeatAL vs Others with Micro-ACDNet

In this section we compare the performance of the three best methods (detailed in Tables 5.6, 5.7 and 5.8) and the performance of DeepFeatAL using the Micro-ACDNet model.

Learning Loops	95%CI for ESC-50		95%CI for US8K		95%CI for iWingBeat	
	AL-LogisticReg	DeepFeatAL	AL-LogisticReg	DeepFeatAL	DeepIcL	DeepFeatAL
0	52.67 ± 0.15	55.36 ± 0.14	82.79 ± 0.06	81.73 ± 0.05	62.51 ± 0.04	62.51 ± 0.04
1	56.08 ± 0.15	58.31 ± 0.15	82.70 ± 0.06	81.97 ± 0.05	62.51 ± 0.05	61.80 ± 0.04
2	56.47 ± 0.15	60.05 ± 0.15	82.70 ± 0.06	81.95 ± 0.05	63.00 ± 0.05	62.73 ± 0.04
3	60.25 ± 0.15	60.76 ± 0.15	82.92 ± 0.06	82.88 ± 0.06	63.09 ± 0.04	62.36 ± 0.04
4	**63.42 ± 0.15**	**63.61 ± 0.15**	82.85 ± 0.06	83.03 ± 0.05	62.77 ± 0.04	63.24 ± 0.04
5	62.76 ± 0.15	63.25 ± 0.15	83.09 ± 0.06	**83.56 ± 0.05**	63.04 ± 0.04	63.09 ± 0.04
6	62.70 ± 0.15	64.01 ± 0.15	83.11 ± 0.06	83.56 ± 0.05	63.45 ± 0.05	63.37 ± 0.04
7	63.32 ± 0.15	65.2 ± 0.15	82.79 ± 0.06	83.15 ± 0.05	63.29 ± 0.04	63.50 ± 0.04
8	-	-	83.11 ± 0.04	83.13 ± 0.05	63.64 ± 0.04	**63.88 ± 0.04**
9	-	-	83.18 ± 0.06	83.46 ± 0.05	63.35 ± 0.04	64.19 ± 0.04
10	-	-	83.10 ± 0.06	83.41 ± 0.05	63.59 ± 0.04	64.35 ± 0.04
11	-	-	83.17 ± 0.06	83.48 ± 0.05	**63.88 ± 0.04**	64.27 ± 0.04
12	-	-	**83.38 ± 0.06**	84.35 ± 0.05	63.09 ± 0.05	64.35 ± 0.04
13	-	-	83.19 ± 0.06	84.03 ± 0.05	63.20 ± 0.05	64.59 ± 0.04
14	-	-	83.20 ± 0.06	83.99 ± 0.05	62.64 ± 0.04	64.37 ± 0.04
15	-	-	83.27 ± 0.06	84.12 ± 0.05	63.80 ± 0.04	64.39 ± 0.04
16	-	-	-	-	63.56 ± 0.04	64.89 ± 0.04
17	-	-	-	-	63.72 ± 0.04	64.81 ± 0.04
18	-	-	-	-	63.79 ± 0.04	64.98 ± 0.04
19	-	-	-	-	63.47 ± 0.04	64.99 ± 0.04
20	-	-	-	-	63.42 ± 0.04	64.85 ± 0.04

TABLE 5.9: Edge-based DeepIcL vs SDAL vs DeepFeatAL on all the datasets

Table 5.9 and Figure 5.15 illustrates that the model produced by DeepFeatAL consistently outperforms SDAL (i.e., AL-LogisticReg) by a significant margin. In fact, the model from DeepFeatAL achieved the highest accuracy produced by AL-LogisticReg (i.e., 63.42%) in five iterations. Thus, the labeling effort is reduced by 42.85%.

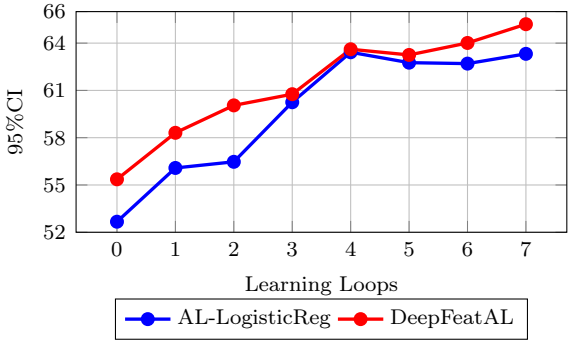


FIGURE 5.15: Edge-based AL-LogisticReg vs DeepFeatAL on ESC-50 dataset

Figure 5.16 depicts the statistical significance of the final models obtained after seven iterations of human annotations (simulated) on the ESC-50 dataset. According to the figure, all the models perform statistically significantly. It demonstrates that the model produced by DeepFeatAL outperforms other models produced by other methods and ranks first.

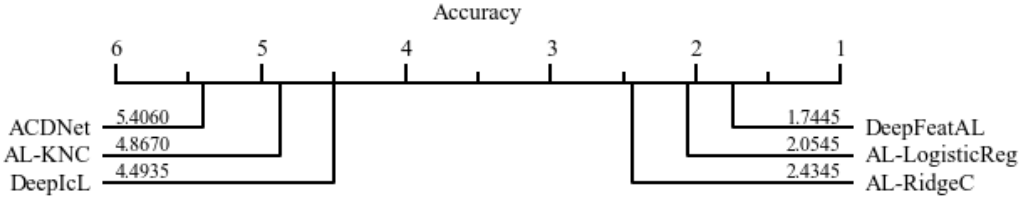


FIGURE 5.16: CD diagram for statistical significance test of performance of different learning methods on ESC-50 dataset.

The same comparison is shown in Figure 5.17 between the performance of AL-LogisticReg and DeepFeatAL on the US8K dataset. It is evident that during the AL process, DeepFeatAL always produces the most accurate predictions. In fact, the highest level of accuracy achieved by AL-LogisticReg is 83.38%, whereas DeepFeatAL achieved a higher level of accuracy after only five iterations (83.56%). Thus, the labeling effort is reduced by 66.67%.

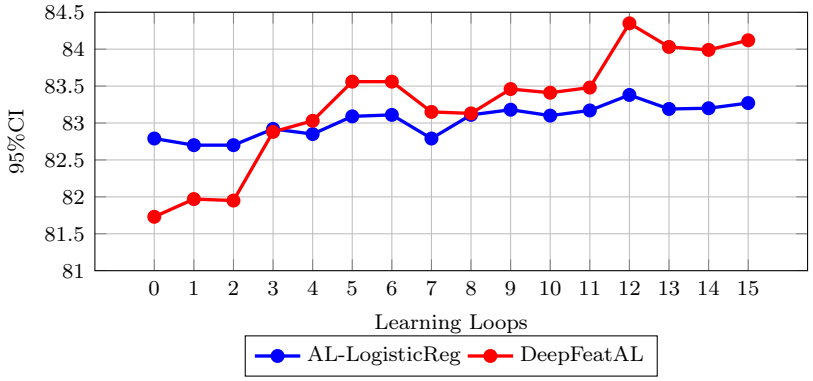


FIGURE 5.17: Edge-based AL-LogisticReg vs DeepFeatAL on US8K dataset

Figure 5.18 depicts the statistical significance of the final models after fifteen iterations of human annotations (simulated). According to the figure, the performance of AL-KNC and AL-LogisticReg is insignificant. Having said that, the figure demonstrates that the model produced by DeepFeatAL outperforms other models produced by other methods and ranks first.

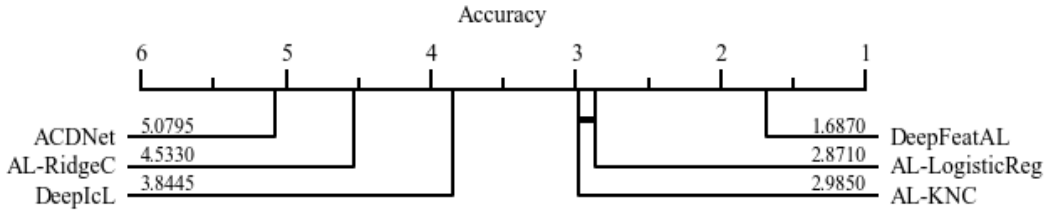


FIGURE 5.18: CD diagram for statistical significance test of performance of different learning methods on US8K dataset.

Figure 5.19 presents the same performance comparison between DeepIcL and DeepFeatAL on the iWingBeat dataset. It clearly reveals that during human annotation (simulated) iterations, DeepFeatAL almost always produces the highest prediction accuracy. The highest accuracy achieved by DeepIcL was achieved by DeepFeatAL after only eight iterations. Consequently, the labeling effort is reduced by 60%.

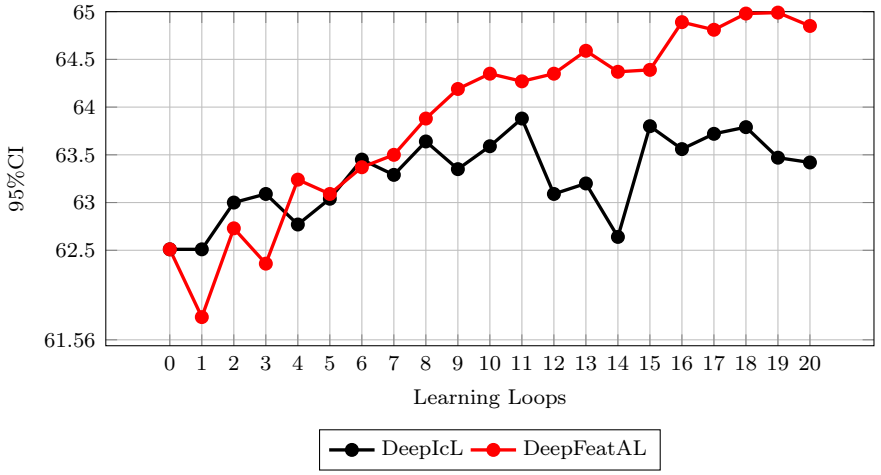


FIGURE 5.19: Edge-based DeepIcL vs DeepFeatAL on iWingBeat dataset

Figure 5.20 displays the statistical significance of the final models after the completion of all iterations of human annotations (simulated). According to the figure, all the models perform statistically significantly. It demonstrates that the model produced by DeepFeatAL outperforms other models produced by other methods and ranks first.

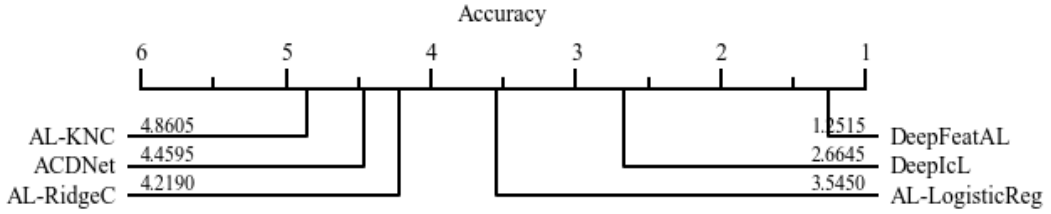


FIGURE 5.20: CD diagram for statistical significance test of performance of different learning methods on iWingBeat dataset.

5.6 Conclusion

This chapter set out to show that integrating the feature extraction level in the active learning process, which had not been done before for audio recognition, improves on the existing advantages of Standard Deep Active Learning (SDAL).

Our extensive experimental investigation on three widely used standard benchmark datasets (ESC with 50 classes (ESC-50), UrbanSound8k Dataset (US8K), and InsectWingBeat (iWingBeat)) shows that fine-tuning the feature extractor in the Active Learning (AL) loop always improves the performance. The statistical significance analysis in Section 5.5.2 demonstrates that the improvement is statistically significant.

Thus, the following are the contributions of this study:

- it shows that integrating and fine-tuning the feature extractor into the AL loop and enhances the quality of the features.
- the fine-tuned model learns faster and saves 14.28%, 66.67%, and 50% labeling effort for the three standard benchmark datasets, respectively.
- the method also works seamlessly with Micro-ACDNet, reducing the labeling budget for the three benchmark datasets by 42.85%, 66.67%, and 61%, respectively.

To the best of our knowledge, this is the first study to consider fine-tuning the feature extractor following each iteration of human annotation (simulated) during an AL process. In addition, this is the first time AL has been investigated for raw audio. We are confident that these findings will broaden the applicability of AL on edge devices. Future research could investigate distributed AL, which would enable Deep Active Feature Learning (DeepFeatAL) directly on edge devices rather than the cloud server.

Chapter 6

Application: Identifying Black-throated Finch Calls

6.1 Introduction

In this chapter, we apply the Deep Learning (DL) techniques we have introduced before to a real-world scenario[1]. The primary objective of this application is to identify the calls of a bird called Black-throated Finch (BTF) (scientifically known as Poephila cincta cincta) [228] in remote wild areas. We are particularly interested in the Southern Black-throated Finch (BTF), which is an endangered subspecies found in the Australian states of New South Wales and Queensland [228].

The identification of the BTF calls is currently done manually. We would like to automate this identification process. Due to the lack of power and internet connection in the remote wild areas, only battery-powered, extremely resource-constrained hardware devices (e.g., Microcontroller Unit (MCU)) can be used. Consequently, we need a model that can be easily deployed on such MCUs and can perform instant classification.

We can approach this BTF call recognition problem as a classification or event detection problem. These two types of recognition tasks are fundamentally different. When it comes to classification problems, the system's job is to determine whether or not an audio recording contains a BTF call [229]. However, when it is handled as a detection problem, the system must recognize the presence of a call and determine the location of the call in the audio recording (i.e., onset and offset) [229–231]. In a real-world scenario, sound events frequently overlap with other sound events [229]. As a result, determining whether a specific call is present is relatively easier than determining the position of the particular sound that overlaps with other sound events [229, 230]. That is why event detection is well established to be a more complex problem than classification [231]. Having a well-performing model trained on a balanced dataset is relatively easy. However, when applied

[1] Data supplied by Lin Schwartzkopf and Slade Allen-Ankins (James Cook University Townsville) and humanly labeled under their supervision.

to detect events on real-world continuous streaming environment, the scenario changes because the target events are less frequent than those that are not. As a result, even with a low false positive (FP) rate, the total number of FPs increases dramatically [231].

For example, a model trained on a balanced dataset is tested on 2000 1s long audio instances. Assume that 800 of the test data instances are positive and 1200 are negative. Let us further assume that the trained model has a 95% accuracy on the test data with a 2% false positive rate (i.e., only 24 FPs). Let us now apply the model to classify events over the course of 24h in a continuous streaming scenario. 24h of data should yield 864,000 1s instances using a 0.1s sliding window. Furthermore, these instances are extremely class imbalanced. In this case, the trained model is expected to generate at least 17,280 FPs, if not significantly more, requiring approximately 4.8h of manual verification, which is 60% of a workday to verify the recordings of one day. This would be extremely expensive.

In the succeeding sections of this chapter, we provide an overview of the related works in Section 6.2, followed by Section 6.3 which provides the details of the data available to build a predictive model. We treat this problem as a classification problem in Section 6.4 and as an event detection problem in Section 6.5 employing a fully-resourced computing model, an MCU-compatible model and present the study in detail. Section 6.6 illustrates how the Micro-Model can gradually learn from new data, and Section 6.7 concludes the chapter.

6.2 Related Works

In this section, we provide an overview of the current literature on audio recognition in continuous recording environments.

In Chapters 3, 4 and 5, we have covered audio event classification works in depth, including the most recent state-of-the-art techniques [89, 162–164, 232]. Furthermore, Chapter 5 Section 5.2 reveals that most of the works in the current literature are devoted to event classification. We present an overview of the recent development on audio event detection here.

A Sound Event Detection (SED) technique has been presented by Hayashi et al. [233] using log Mel-filterbank features with a bidirectional LSTM recurrent neural network. Following that, the prediction output is post-processed using median filtering, gap filling for a shorter sequence of predictions, and finally, removing events shorter than a predetermined threshold. These preprocessing and post-processing steps significantly enhance the performance of the proposed SED system. Miyazaki et al. [234] employ the same post-processing techniques for their transformer-based SED, including a prediction score threshold. The sound event is considered active when the prediction probability exceeds

the threshold. Kong et al. [235] also proposed a Convolutional Neural Network (CNN)-Transformer-based audio tagging with events that includes a prediction score threshold, but the threshold is optimized according to the performance metrics.

Wolters et al. [236], on the other hand, offer a proposal-based few-shot SED model that integrates attention into window-level and proposal-based localization techniques. Cao et al. [237] present a two-stage polyphonic SED and localization technique that uses log mel features in conjunction with intensity vector and GCC characteristics for event identification. Similar to Adavanne et al. [230], this method trains SED model before applying the learned feature layers to the direction of arrival estimation. Another SED and localization strategy for DCASE2019[218] dataset is presented by Kapka and Lewandowski [238]. They estimate the number of active sources, the direction of arrival of a single source, the direction of arrival of the second source, and a multi-label classification job using an ensemble of four Convolutional Recurrent Neural Network (CRNN) SELDnet-like single output models. Kao et al. [239] uses a modified Faster-RCNN network with multi-task loss to optimize the classification and localization of audio events simultaneously. A similar approach is suggested by Xu et al. [219] where they used gated CRNN including a recommendation of using a temporal attention-based localization strategy for locating and tagging audio events. Similar to other attention-based works, Adavanne et al. [229] proposed their audio event detection using a Fully Connected Neural Network (FCNN) and a CNN with attention.

Phan et al. [240] uses regression for event detection and localization, as opposed to the classification-based method described previously. The audio signals are decomposed into multiple interleaved superframes, and a specific category is learned by a random regression tree. Later on Phan et al. [231] demonstrated multiple methods of event detection, such as using RF classifier [59] and Regression Forest [240] where they segmented the audio signals and used them as input. Adavanne et al. [230] performs multi-label classification task on each time frame, producing temporal activity for all sound event classes. Using multi-output regression as the second output, they localize the events by predicting the direction of arrival for each sound event class. The phase and magnitude components are used to extract the features from the spectrograms produced for each audio channel. In their subsequent work, Phan et al. [241] used CRNN with self-attention for their multiple event detection and localization problem. They used logmel magnitude spectrogram as inputs to their two networks, each with sigmoid/tanh activation to their output units to perform the regression.

From the overview of current work, we find that preprocessing and post-processing are essential for SED regardless of the quality of the predictive model. Additionally, we were unable to locate any recent work that has used raw audio data. To the best of our knowledge, there is also no SED approach described for MCU-compatible DL models.

Therefore, the objective of this chapter is to implement these preprocessing and post-processing approaches for SED utilizing fully-resourced computing and MCU-compatible DL models.

6.3 Dataset Description

The dataset consists of 49 files containing 98h (i.e., ≈2h each) of continuous audio recordings from remote wild areas. These are unrefined field recordings captured at 44.1kHz and 16-bit resolution using identical microphones and recorders from several distant fields. The record level remained constant throughout. The non-overlapping BTF calls with varied amplitudes are manually annotated by human experts in order to develop a Deep Neural Network (DNN) model for the automated labeling of future recordings. There are 2907 labeled BTF calls. The gaps between BTF calls are labeled as *other calls*. Other calls, including other bird calls and environmental sounds are classified as Background Sounds (BGS). The duration of the BTF calls ranges from 0.6s to 0.672s.

6.4 Classify Black-throated Finch (BTF) Calls

In this section, we present the audio classification process used to classify Black-throated Finch (BTF) calls from the audio records. The forthcoming subsections present the classification process.

6.4.1 Data Preprocessing

We used the *librosa* python library for data preprocessing. The audio files are resampled at a rate of 20kHz. We used the first 49h of recording to build a balanced dataset containing 8265 BTF calls (80% overlapping samples while using 0.1x (i.e., 0.0672s) sliding window of 1653 actual BTF calls) and 8265 BGS. We do a 80-20 split of this dataset to create a training set and validation set. The splits produce 6,612 BTF calls and 6,612 BGS for training, and 1,653 BTF calls and 1,653 BGS for validation.

The remaining 49h of recording are used for testing. We used the same sliding window to create the training and the test dataset. In the test set, we considered the 80% overlapping calls of an actual BTF call to be a BTF call (1254 distinct BTF calls, totaling 6,270 80% overlapping calls). The rest of the samples have been labeled as BGS. This yields a test dataset of 2,678,571 overlapping samples, each of which is 0.672s long.

For all the samples, the input length is 13440 (0.672s * 20000). Finally, we divided all the data by 32768.0 to normalize them. The rationale for this is to scale the values extracted from 16-bit recordings to the range of -1 to 1.

6.4.2 Training the Model

We trained ACDNet for 1000 epochs with a learning rate scheduler of 0.3, 0.6, 0.9 with the same learning settings we have described in Chapter 3. After training and

cross-validation, the model achieves a prediction accuracy of 89.11%, with a FP rate of 11.57% and a FN rate of 10.2%. In the next section, we used the trained ACDNet model to predict the labels for the samples in the test set.

6.4.3 Prediction, Post-processing and Results

For rolling recognition, we applied the trained model to the test set (49h of continuous recordings). Four criteria were used during the post-processing stage. The first one is 80% overlapping samples that are predicted to be true (Algorithm 3). And the rest are a sequence of two, three, and four samples (Algorithm 4) predicted to be true.

Algorithm 3: Find Percentage-based Overlapping Predictions

Input: Predictions $PREDS$, Gound Truth GT, Overlap Percentage OLP, Stride ST, Input Length (sec) IL

Output: True Positives TP, False Positives FP

```
TP ⟵ [ ], FP ⟵ [ ], OL ⟵ IL − (IL ∗ OLP/100)
foreach P in PREDS do
    P_START ⟵ P ∗ ST, P_FOUND ⟵ FALSE
    foreach G in GT do
        if |P_START − G.START| ≤ OL then
            TP.append(P), P_FOUND ⟵ TRUE
    if not P_FOUND then
        FP.append(P)
return TP, FP
```

Algorithm 4: Find Sequential Overlapping Predictions

Input: Predictions $PREDS$, Gound Truth GT, Stride ST, Input Length (sec) IL, Sequence Length SL

Output: True Positives TP, False Positives FP

```
1  TP ⟵ [ ], FP ⟵ [ ]
2  foreach P in PREDS do
3      SEQ ⟵ [ ]
4      if len(SEQ)>0 then
5          if P is SEQ[−1] + 1 then
6              SEQ.append(P)
7          else
8              if len(SEQ) ≥ SL then
9                  OL ⟵ [ ]
10                 foreach S in SEQ do
11                     S_START ⟵ S ∗ ST
12                     foreach G in GT do
13                         if |S_START − G.START| ≤ IL then
14                             OL.append(S_START − G.START)
15                 if len(OL) > 0 then
16                     TP.append(SEQ[OL.index(max(OL))])
17                 else
18                     FP.append(SEQ[OL.index(max(OL))])
19             SEQ ⟵ [ ]
20     else
21         SEQ.append(P)
22 return TP, FP
```

Table 6.1 shows the outcome of this post-processing. The findings in the *Accuracy* column, which indicate the percentage of correctly classified samples, are quite impressive. The figures in the *Recall* column, on the other hand, are not that impressive. In addition, the *FN* rates are extremely high. Although the *FP* rate decreases as you progress to the last rows, the other parameters, such as *Recall* and *FN* rates, increase significantly. These findings show that, despite a high accuracy rate, we have unacceptable *FP* and *FN* rates for a real-world application. The impressive accuracy figure is due to the fact that the test set contains only 0.23% BTF calls and the remaining 99.77% are BGS.

Criteria	TP	FP	TN	FN	Accuracy	Recall	FP Rate	FN Rate
Single 80% Overlap	3,920	205,982	2,466,319	2,350	92.22%	62.52%	7.71%	37.48%
Sequence of 2	3,725	126,087	2,546,214	2,545	95.20%	59.41%	4.72%	40.59%
Sequence of 3	2,950	69,118	2,603,183	3,320	97.20%	47.05%	2.59%	52.95%
Sequence of 4	1,635	26,942	2,645,359	4,635	98.82%	26.08%	1.01%	73.92%

TABLE 6.1: Classification of 49h continuous audio recording

As a result, we need to consider alternative approaches to solving this problem better. Instead of considering this recognition problem as a classification problem, we will consider it an event detection problem in the next section.

6.5 Detect Black-throated Finch (BTF) Presence using Classification

We now treat this Black-throated Finch (BTF) call recognition problem as an event detection problem using the classification technique. The goal is to find out if any Black-throated Finch (BTF) exists in a specific location. As a result, determining the presence of just one call out of many should suffice. The FP rate was the limiting factor in the classification approach shown in Section 6.4. Thus, using this detection by classification approach, we hope to detect the presence of a Black-throated Finch (BTF) call in a recording with very few FPs.

6.5.1 Data Preprocessing

We used *librosa* to generate a mel spectrogram from the audio. We used $sr = 20000Hz$, $n\_fft = 4096$, $hop\_length = 320$ and $n\_mels = 128$ to generate the mel spectrogram. When we look at the mel spectrogram data, we see that the majority of the BTF calls are in the 37 to 100 mel bin range. As a result, we zero out 1-36 and 101-128 mel bins to isolate the BTF calls. The mel spectrogram is then converted back to audio using the $inverse.mel\_to\_audio$ function of *librosa*.

Our review of the actual BTF calls revealed that the length of the calls ranges from 0.6s to 0.672s. As a result, we sliced the 0.6s audio records with a 0.1s sliding window to create overlapping samples for rolling recognition. BTF calls were defined as samples with 50% overlap from the start of an actual BTF call. The slices are all 0.6s long. As a result, the input length became 12000 (i.e., 0.6s x 20000Hz).

We created a training set from the first 80h (40 audio files) of recording. The training set contains 10349 overlapping BTF calls extracted from a total of 2602 actual BTF calls. In addition, the training set includes 400,000 BGS samples (10,000 samples from each file). We have also augmented the BTF samples, making the total number of BTF call samples 20,000. During augmentation, we used *adding noise, time-stretching, pitch-shifting, and time-shifting*. We padded zeros at the end of a sample if its length becomes less than 12,000.

The next 6h (3 audio files) of the recordings are used to create a validation set. We extracted 787 overlapping BTF calls from 198 actual BTF calls in the training set. In addition, the validation set includes 30,000 BGS samples (10,000 samples from each file).

The remaining audio files (6 audio files) are used to create a test set. For each of the six audio files, there is a subset in the test set. Each subset has approximately ≈72,000

overlapping samples. There are 426 overlapping BTF calls in the six files (107 actual calls). The details of the training, validation and test sets are presented in Table 6.2.

Datasets	#Files	#Samples	Actual BTF Calls	Overlapping BTF Samples	BGS Samples	Augmented BTF Samples
Training set	40	410,349	2,602	10,349	4000,000	9,651
Validation set	3	30,787	198	787	30,000	-
Test set	6	432,000	107	426	431,576	-

TABLE 6.2: Training set, validation set and test set details

Finally, we normalize the datasets using $S_i = \frac{S_i}{Max(S_{ij})}$ where S is a sample, i is the index of the sample and $j \in \{0, 1, \ldots, m\}$ where m is the number of features in S_i.

6.5.2 Training the Model

We trained both ACDNet and Micro-ACDNet models using the training settings we used in the classification approach (Section 6.4.2). However, we changed the loss function to *CategoricalCrossEntropyLoss* for this binary classification problem.

During training, all the BTF call samples (i.e., 20,000 BTF samples) and $4x$ randomly selected BGS from 400,000 BGS samples are chosen from the training set for every epoch. For validation we used all the BTF call samples from the validation set along with $2x$ BGS samples randomly selected from the 30,000 BGS samples of the validation set.

After training and validation, we used the $F1$ measure to select the best fully-resourced computing model and the validation accuracy to select the MCU-compatible model for rolling recognition on the test set.

6.5.3 Prediction, Post-processing[2] and Results

We applied the trained models to detect the presence of BTF calls in six audio files (2h each). As the objective is to determine if a site has BTF, the 2h audio record is divided into 5s chunks and tagged with a 1 if any of its frames is a true BTF call and a 0 otherwise. After post-processing, we employ the same strategy for predicting labels. The ultimate accuracy is then determined by comparing the genuine labels and anticipated labels of the 5s chunks. Since we are solely interested in identifying the presence of BTF at a site, we believe this is a reasonable way for estimating the overall accuracy.

Post-processing the ACDNet output yields 21 TPs and 8 FPs. That is, only 29 five-second audio segments, which is 2m 25s out of 12h, are handed to human experts for further verification. In the case of Micro-ACDNet, the post-processing yields 16 TPs and 14 FPs. Despite having fewer TPs and more FPs, it does not miss any file with BTF call. Only 30 five-second audio segments, which is 2.5m out of 12h, are handed to human experts for further verification. Table 6.3 and Table 6.4 present the results achieved from the above process.

[2]We acknowledge and thank Sazzad Hossain for his support in optimizing this post-processing task.

Test File No.	#5s Segments	#BTF Call Segments	TP	FP	FN	TN
1	1440	7	5	0	2	1433
2	1440	16	2	1	14	1423
3	1440	12	3	1	9	1427
4	1440	23	6	2	17	1415
5	1440	0	0	1	0	1439
6	1440	13	5	3	8	1424
Total	8,640	71	21	8	50	8,561

TABLE 6.3: Results after post-processing of the predictions from ACDNet

We first applied post-processing techniques such as *Median Filtering* [233, 234] with a predetermined filter span, *Filling the Gaps* between the true and false prediction with a predetermined number [233] and then we *Remove events* [233] that have a shorter duration. However, *Median Filtering* and *Filling the Gaps* techniques did not perform well in this application. As a result, we only use *Removal of short events* during the post-processing stage.

At the beginning of the post-processing, we used a threshold of 0.9 for a sample to be considered as a true prediction. After that, we remove the events having shorter duration by zeroing out all the true predictions except those instances where there are $\geq$4 consecutive true predictions. We chose $\geq$4 as our preprocessing allows a maximum of 4 to 5 overlapping samples to be BTF call samples. Finally, we sliced the records into five-second segments and marked the segment as an accurate prediction if it had at least one set of $\geq$4 consecutive true predictions.

Test File No.	#5s Segments	#BTF Call Segments	TP	FP	FN	TN
1	1440	7	4	0	3	1433
2	1440	16	2	3	14	1421
3	1440	12	1	1	11	1427
4	1440	23	4	4	19	1413
5	1440	0	0	1	0	1439
6	1440	13	5	5	8	1422
Total	8,640	71	16	14	55	8,555

TABLE 6.4: Results after post-processing of the predictions from Micro-ACDNet

Thus, we can say that detecting the presence of BTF calls in MCU is now possible because Micro-ACDNet can clearly do the job and can be easily deployed to an off-the-shelf MCU, as demonstrated in Chapter 3. The immediate next step is to apply Active Learning (AL) to Micro-ACDNet in order to improve its predictive power. This possibility is investigated in the following section.

6.6 Active Learning on Micro-ACDNet

We initially trained the Micro-ACDNet with only the first 17 audio records ($\approx$34h which is only 42.5% of the data used to train the base ACDNet model). Then we run AL

for 10 iterations, each with 1,000 newly labeled samples (1h 40m). Finally, we ran the fine-tuned model on the same set of tests. Micro-ACDNet now generates nearly the same number of TPs as its parent model (i.e., ACDNet trained on 80h of recordings) with roughly twice the number of FPs. Although the FPs are higher, they are still manageable because only 36 five-second audio segments, or 3m out of 12h of recordings, are sent to a human expert for verification. Thus, AL saves 44h 20m of human effort for manually labeling.

The training set includes 10,000 overlapping BTF calls (2,492 from 626 actual calls, the rest augmented from the 2,492) and 170,000 BGS. We used the same validation and test sets for training, Active Learning (AL), and testing. We also kept the same hyperparameter settings, except we did not use the learning rate scheduler and instead we used a much lower learning rate of 1e-5. Table 6.5 displays the test results after post-processing (i.e., *removal of short event* that creates negligible computation overhead for a sample) the prediction, which shows some performance degradation. However, we consider it normal as the model was only trained on less than half of the training data.

Test File No.	#5s Segments	#BTF Call Segments	TP	FP	FN	TN
1	1440	7	2	2	5	1431
2	1440	16	2	3	14	1421
3	1440	12	1	0	11	1428
4	1440	23	6	3	17	1414
5	1440	0	0	1	0	1439
6	1440	13	4	6	9	1421
Total	8,640	71	15	15	56	8,554

TABLE 6.5: Results after post-processing of the predictions from Micro-ACDNet trained on only 34h (i.e., 17 files) of audio recording.

We now apply our proposed Deep Active Feature Learning (DeepFeatAL) to the trained Micro-ACDNet model. We follow the same procedure described in the previous chapter during the DeepFeatAL process. However, this dataset is extremely imbalanced, and as we train the model with data that has BTF and a BGS ratio of 1:4, rather than fine-tuning, we retrain the model during active learning on the entire training set mixed with the newly annotated 1,000 samples. Retraining takes 50 epochs and has a much lower learning rate (i.e., 1e-5). To see if AL improves the model, we run a second set of experiments in which we retrain the model using the same training set (i.e., without the annotated data) and the same learning rate as in active learning.

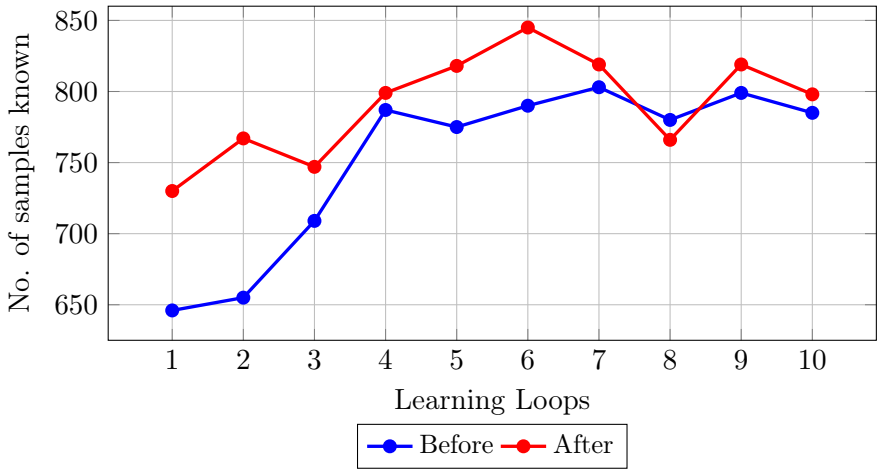


FIGURE 6.1: No. of samples known before and after retraining with the 1,000 newly annotated samples

Figure 6.1 depicts the number of samples recognized by the model before and after fine-tuning with annotated samples. The model was first tested on the annotated samples before being fine-tuned on them. Once the model had been fine-tuned using the annotated samples, it was tested again to see if the model had learned any of those samples. Except in iteration 8, the model could recognize more samples, as shown in the figure. The data for this figure was obtained from Table 6.7.

The prediction accuracy of the fine-tuned model through active learning was then compared to the model that was simply retrained on only the initial training set (i.e., without the annotated samples). Figure 6.2 presents a comparison of prediction accuracy in terms of precision. The graph shows that the fine-tuned model through AL always outperforms its counterpart. This figure was created using data from Table 6.7.

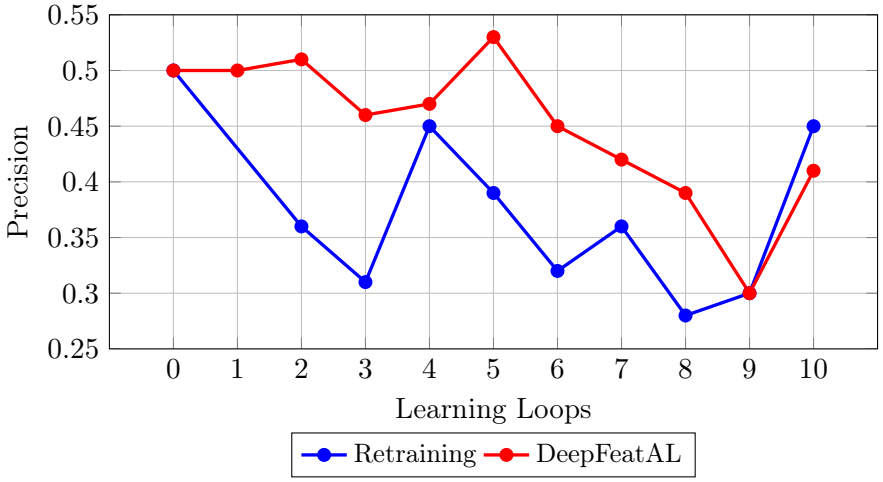


FIGURE 6.2: Retraining vs DeepFeatAL performance

The DeepFeatAL model and the retrained model perform comparably to the naked eye. However, Figure 6.1, Figure 6.2 and Table 6.7 reveal that DeepFeatAL model learns the

newly exposed samples better, has higher detection precision, and a better ratio of TPs to FPs than its counterpart.

In terms of accuracy, Micro-ACDNet trained and fine-tuned on only 35h of recordings (achieved by DeepFeatAL, see Table 6.6) now exceeds Micro-ACDNet trained on 80h of recordings (see Table 6.4). Furthermore, it now produces nearly as many TPs as its parent model (i.e., ACDNet trained on 80h of recordings, see Table 6.4 vs Table 6.3). Despite the fact that the number of FPs is almost doubled, they are manageable because only 36 five-second (i.e., 19 TPs + 17 FPs) audio segments (i.e., 3m) are handed over to the human expert for verification out of 12 hours of recordings. Table 6.6 presents the DeepFeatAL model's detailed results.

Test File No.	#5s Segments	#BTF Call Segments	TP	FP	FN	TN
1	1440	7	4	1	3	1432
2	1440	16	3	4	13	1420
3	1440	12	1	2	11	1426
4	1440	23	7	4	17	1412
5	1440	0	0	1	0	1439
6	1440	13	4	5	9	1422
Total	8,640	71	19	17	53	8,551

TABLE 6.6: Results after post-processing of the predictions from Micro-ACDNet

Learning Loops	#5s Segments	#BTF Call Segments	retraining			DeepFeatAL				
									1,000 Annotated Samples	
			TP	FP	Precision	TP	FP	Precision	Known Before retraining	Known After retraining
0	8,640	71	15	15	0.50	15	15	0.50	-	-
1	8,640	71	19	22	0.46	21	21	0.50	646	730
2	8,640	71	21	38	0.36	19	18	0.51	655	767
3	8,640	71	21	47	0.31	17	20	0.46	709	747
4	8,640	71	19	23	0.45	21	24	0.47	787	799
5	8,640	71	22	34	0.39	19	17	0.53	775	818
6	8,640	71	23	49	0.32	14	17	0.45	790	845
7	8,640	71	22	39	0.36	16	22	0.42	803	819
8	8,640	71	22	57	0.28	13	20	0.39	780	766
9	8,640	71	23	53	0.30	18	42	0.30	799	819
10	8,640	71	18	22	0.45	14	20	0.41	785	798

TABLE 6.7: Results after post-processing of the predictions from Micro-ACDNet

6.7 Conclusion

The goal of this project is to detect the presence of BTF calls in recordings from various locations. Despite the fact that the classification approach (Section 6.4) produces impressive accuracy figures, these figures reflect all unwanted events. Since the target events are extremely rare, the classification method produces a large number of false positives.

Our solution, on the other hand, which uses the presence detection by classification described in Section 6.5, can correctly determine whether a 2-hour audio recording contains any BTF call events with a manageable FR rate. It also shows that post-processing of the prediction result is an important part of presence detection in continuous audio streams.

The work presented in this chapter demonstrates that by using the MCU-compatible Micro-ACDNet, it is now possible to detect the presence of BTF in remote areas. Although the detection rate for some specific sites may be low (for example, test file 3 in Table 6.6), it still saves a significant amount of human labeling effort. While manual labeling took 98 hours to label all the recordings to determine which site has BTFs, our approach requires only 35h 40m of human effort, saving 54h 20m of manual processing time. Furthermore, the trained model produces data equivalent to only 3m for manual verification out of 12h of data.

For manual verification of the TPs and FPs, the data needs to be stored on an external storage device and accessed on a regular basis. Given that the trained model generates data of approximately 3m in length to be validated by a human after 12 hours of observation, for seven days, it would require around 42m of data to be saved on the external storage device. Because the s≈48mB of storage (i.e., (42*60*20,000)/ 1024^2) of storage space.

Chapter 7

Conclusion

The purpose of this research was to develop suitable methodologies for applying Deep Learning (DL) for Bioacoustic Recognition (BaR) on low-power Internet of Things (IoT) devices (i.e., Microcontroller Units (MCUs)) deployed in remote wild areas. In achieving this goal, we had to overcome a significant number of challenges associated with using DL on IoT devices.

The first main challenge was to figure out how to fit the power- and resource-intensive state-of-the-art Deep Neural Network (DNN) model architecture onto a device with limited resources while maintaining comparable prediction accuracy. We developed a compression-friendly state-of-the-art DNN model architecture for acoustic classification using raw audio data and an associate compression pipeline to drastically downsize the state-of-the-art model architecture. This resulted in state-of-the-art models with resource requirements within the limits of the relevant MCUs, on which we successfully deployed our models.

The second main challenge was the fact that bioacoustic applications are typically label poor; in other words, we need to be able to work with as little labeling as possible. This problem was addressed by introducing a new Deep Active Learning (DAL) technique.

Finally, we demonstrated the applicability of our method to real-world problems in the context of an applied conservation study. It is worth mentioning that the total memory required to fit the input, load the dynamic run-time library, compute the activation of a specific layer and store the activation to be used as the input to the next layer must be less than the amount of SRAM available in the Microcontroller Unit (MCU). If not, the model will crash during execution.

Thus, the findings of this research have expanded our understanding of BaR and DL for MCUs. DL in MCU is now a reality. Rather than handcrafting a DNN model architecture for a specific acoustic job, the optimal approach is to start with a robust model architecture capable of producing very high performance and then optimize it

using deep architecture compression techniques. The goal of designing a DNN model should be to produce a compression-friendly network architecture by keeping network layers thin and tall. In addition, the network should have a minimum number of fully connected layers, as this hinders efficient compression. When compressing the DNN architecture, keep in mind that the compression should be structured because only models with structured architecture are supported by the existing hardware design of the MCUs. Finally, we should avoid multi-channel input to reduce the amount of RAM necessary to buffer the input. This opens up the same opportunities for a wide range of applications, and we are confident that we can transfer our approach to other domains. There are significant research opportunities to extend our work and explore new avenues moving forward.

Future research could involve removing the transpose layer from ACDNet and transforming it into a complete 1D Convolutional Neural Network (CNN)-based state-of-the-art Deep Acoustic Network (DAN) model architecture. This would greatly simplify the MCU implementation of the Micro-ACDNet. The second research opportunity could be to investigate XNOR-Net further to determine why it performs poorly on large-class datasets. Developing a generic computation kernel for XNOR-Net to run on MCUs would also constitute a substantial contribution to knowledge. The third option would be to implement audio Transformer Neural Networks (TNN) on MCUs, as TNNs are currently producing spectacular results for audio classification.

As most real-world acoustic data contain multiple labels, we would approach this problem as a multi-label classification problem if we were to restart this project from scratch. We would also include some sort of attention mechanisms to allow the model to focus on the particular target label in the multi-label data. In addition to that, we would possibly include some biological features as well. In the future, when more powerful MCUs will be available, researchers will likely want to combine the outputs of a DAN model with a vision-based model for the final classification to achieve improved performance.

Apart from the above, if we take a bird's-eye view of the problem, we have only solved a minuscule portion of it. The question of how to deal with the unknown data once the system has been deployed in such a resource-constrained context remains unanswered. Furthermore, we will certainly require some form of compression mechanism to save energy and storage space during the early phase of the project because the anticipated volume of unknown data is considerably higher at this stage of the project than at later stages. This is yet to be determined. Since wireless connections are impractical in the context of this research, how data will be collected from the device is a major challenge to tackle. Given that real-world acoustic data has various labels, how could the performance be improved using this weakly labeled data is an interesting problem to handle. Furthermore, at some point, incremental feature learning should occur in edge devices. Distributed deep model training could be one possibility, but it needs to be

realized. We are also not aware of any additional issues that may arise following the physical deployment.

In conclusion, we emphasize the significance of interdisciplinary collaboration in order to more effectively solve this bioacoustic recognition problem. Experts in conservation management, ecology, and conservation biology are primarily responsible for making vast quantities of diverse real-world data available to the public, enabling numerous researchers to use real-world data to solve real-world problems. The development of predictive models is the responsibility of specialists in signal processing and machine learning. Lastly, energy and communication experts must be involved because IoT-based solutions rely heavily on energy and communication infrastructures.

References

[1] 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.

[2] 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.

[3] 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.

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

[5] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing*, 13(2):206–219, 2019.

[6] Antonio Greco, Antonio Roberto, Alessia Saggese, and Mario Vento. DENet: A deep architecture for audio surveillance applications. *Neural Computing and Applications*, pages 1–12, 2021.

[7] 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.

[8] 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.

[9] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[10] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

[11] Feng Chen, Pan Deng, Jiafu Wan, Daqiang Zhang, Athanasios V Vasilakos, and Xiaohui Rong. Data mining for the internet of things: literature review and challenges. *International Journal of Distributed Sensor Networks*, 11(8):431047, 2015.

[12] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data*, 2(1):1, 2015.

[13] Natalia Vassilieva. Sense making in an IoT world: Sensor data analysis with deep learning. In *Proceedings of the GPU Technology Conference, 2016*. Hewlett Packard Enterprise, 2016.

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

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

[16] 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.

[17] 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, Tolulope 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.

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

[19] nrf52840. https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840, 2021.

[20] Microcontrollers & Microprocessors. https://www.st.com/en/microcontrollers-microprocessors.html, 2021.

[21] Arduino Nano 33 BLE Sense. https://store.arduino.cc/usa/nano-33-ble-sense, 2021.

[22] Product specifications - spresense - sony developer world. https://developer.sony.com/develop/spresense/specifications, 2018.

[23] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: a system for large-scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[24] Robert David, Jared Duke, Advait Jain, Vijay Janapa Reddi, Nat Jeffries, Jian Li, Nick Kreeger, Ian Nappier, Meghna Natraj, Tiezhen Wang, et al. Tensorflow Lite Micro: Embedded machine learning for TinyML systems. *Proceedings of Machine Learning and Systems*, 3:800–811, 2021.

[25] Ronald Newbold Bracewell and Ronald N Bracewell. *The Fourier transform and its applications*, volume 31999. McGraw-Hill New York, 1986.

[26] Kartik Chaudhary. Understanding audio data, fourier transform, FFT, spectrogram and speech recognition. https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520, Jun 2021.

[27] Isabelle Guyon, Steve Gunn, Masoud Nikravesh, and Lofti A Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

[28] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*, pages 372–378. IEEE, 2014.

[29] Florian Eyben, Martin Wöllmer, and Björn Schuller. OpenSmile: the munich versatile and fast open-source audio feature extractor. In *Proceedings of the 18th International Conference on Multimedia, 2010*, pages 1459–1462, 2010.

[30] Michael W Towsey. The calculation of acoustic indices to characterise acoustic recordings of the environment. http://eprints.qut.edu.au/53710/, 2012.

[31] Shawn Hershey, Sourish Chaudhuri, Daniel PW Ellis, Jort F Gemmeke, Aren Jansen, R Channing Moore, Manoj Plakal, Devin Platt, Rif A Saurous, Bryan Seybold, et al. CNN architectures for large-scale audio classification. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing, ICASSP 2017*, pages 131–135. IEEE, 2017.

[32] Jens Schröder, Jorn Anemuller, and Stefan Goetze. Classification of human cough signals using spectro-temporal Gabor filterbank features. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016*, pages 6455–6459. IEEE, 2016.

[33] Haotian Shi, Haoren Wang, Chengjin Qin, Liqun Zhao, and Chengliang Liu. An incremental learning system for atrial fibrillation detection based on transfer learning and active learning. *Computer methods and programs in biomedicine*, 187:105219, 2020.

[34] Yuji Tokozume and Tatsuya Harada. Learning environmental sounds with end-to-end convolutional neural network. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017*, pages 2721–2725. IEEE, 2017.

[35] Yuji Tokozume, Yoshitaka Ushiku, and Tatsuya Harada. Learning from between-class examples for deep sound recognition. In *Proceedings of the 6th International Conference on Learning Representations, ICLR 2018*, page https://openreview.net/forum?id=B1Gi6LeRZ. OpenReview.net, 2018.

[36] Jonathan J Huang and Juan Jose Alvarado Leanos. AclNet: efficient end-to-end audio classification CNN. *arXiv preprint arXiv:1811.06669*, 2018.

[37] Prateek Verma and Julius O Smith. Neural style transfer for audio spectograms. *arXiv preprint arXiv:1801.01589*, 2018.

[38] L Wyse. Audio spectrogram representations for processing with convolutional neural networks. In *Proceedings of the First International Conference on Deep Learning and Music, 2017*, pages 37–41, 2017.

[39] Muhammad Irfan, Zheng Jiangbin, Shahid Ali, Muhammad Iqbal, Zafar Masood, and Umar Hamid. Deepship: An underwater acoustic benchmark dataset and a separable convolution based autoencoder for classification. *Expert Systems with Applications*, 183: 115270, 2021.

[40] Akon O Ekpezu, Isaac Wiafe, Ferdinand Katsriku, and Winfred Yaokumah. Using deep learning for acoustic event classification: The case of natural disasters. *The Journal of the Acoustical Society of America*, 149(4):2926–2935, 2021.

[41] Daniel Rothmann. What's wrong with spectrograms and CNNs for audio processing? https://towardsdatascience.com/whats-wrong-with-spectrograms-and-cnns-for-audio-processing-311377d7ccd, Mar 2018.

[42] Md Mohaimenuzzaman, Christoph Bergmeir, Ian West, and Bernd Meyer. Environmental sound classification on the edge: A pipeline for deep acoustic networks on extremely resource-constrained devices. *Pattern Recognition*, 133:109025, 2023.

[43] Gian Antonio Susto, Angelo Cenedese, and Matteo Terzi. Time-series classification methods: Review and applications to power systems data. *Big data application in power systems*, pages 179–220, 2018.

[44] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, pages 1–47, 2019.

[45] Benjamin Lucas, Ahmed Shifaz, Charlotte Pelletier, Lachlan O'Neill, Nayyar Zaidi, Bart Goethals, François Petitjean, and Geoffrey I Webb. Proximity Forest: An effective and scalable distance-based classifier for time series. *Data Mining and Knowledge Discovery*, pages 1–29, 2018.

[46] Yanping Chen, Adena Why, Gustavo Batista, Agenor Mafra-Neto, and Eamonn Keogh. Flying insect classification with inexpensive sensors. *Journal of insect behavior*, 27(5): 657–677, 2014.

[47] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials*, 2018.

[48] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[49] Jason Lines and Anthony Bagnall. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery*, 29(3):565–592, 2015.

[50] Jon Hills, Jason Lines, Edgaras Baranauskas, James Mapp, and Anthony Bagnall. Classification of time series by shapelet transformation. *Data Mining and Knowledge Discovery*, 28(4):851–881, 2014.

[51] Patrick Schäfer. The BOSS is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530, 2015.

[52] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with COTE: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015.

[53] Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 12(5):52, 2018.

[54] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[55] Diego F Silva, Rafael Giusti, Eamonn Keogh, and Gustavo EAPA Batista. Speeding up similarity search under dynamic time warping by pruning unpromising alignments. *Data Mining and Knowledge Discovery*, pages 1–29, 2018.

[56] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[57] Lexiang Ye and Eamonn Keogh. Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *Data mining and knowledge discovery*, 22 (1-2):149–182, 2011.

[58] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3): 273–297, 1995.

[59] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[60] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966.

[61] Fred H Borgen and Mark J Seling. Uses of discriminant analysis following MANOVA: Multivariate statistics for multivariate purposes. *Journal of Applied Psychology*, 63(6): 689, 1978.

[62] Brian Everitt and Sophia Rabe-Hesketh. Discriminant function analysis. In *Analyzing Medical Data Using S-PLUS*, pages 417–437. Springer, 2001.

[63] Seppo Fagerlund. Bird species recognition using support vector machines. *EURASIP Journal on Advances in Signal Processing*, 2007:1–8, 2007.

[64] Miguel A Acevedo, Carlos J Corrada-Bravo, Héctor Corrada-Bravo, Luis J Villanueva-Rivera, and T Mitchell Aide. Automated classification of bird and amphibian calls using machine learning: A comparison of methods. *Ecological Informatics*, 4(4):206–214, 2009.

[65] Panu Somervuo, Aki Harma, and Seppo Fagerlund. Parametric representations of bird sounds for automatic species recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(6):2252–2263, 2006.

[66] Joseph A Kogan and Daniel Margoliash. Automated recognition of bird song elements from continuous recordings using dynamic time warping and hidden Markov models: A comparative study. *The Journal of the Acoustical Society of America*, 103(4):2185–2196, 1998.

[67] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[68] Carel ten Cate, Robert Lachlan, and Willem Zuidema. Analyzing the structure of bird vocalizations and language. finding common ground. Johan J. Bolhuis & Martin Everaert (toim.). *Birdsong, speech, and language. Exploring the evolution of mind and brain*, pages 243–260, 2013.

[69] Dan Stowell and Mark D Plumbley. Automatic large-scale classification of bird sounds is strongly improved by unsupervised feature learning. *PeerJ*, 2:e488, 2014.

[70] Anton Gradišek, Gašper Slapničar, Jure Šorn, Mitja Luštrek, Matjaž Gams, and Janez Grad. Predicting species identity of bumblebees through analysis of flight buzzing sounds. *Bioacoustics*, 26(1):63–76, 2017.

[71] Patrick J Clemins, Michael T Johnson, Kirsten M Leong, and Anne Savage. Automatic classification and speaker identification of african elephant (loxodonta africana) vocalizations. *The Journal of the Acoustical Society of America*, 117(2):956–963, 2005.

[72] Marie A Roch, Melissa S Soldevilla, Jessica C Burtenshaw, E Elizabeth Henderson, and John A Hildebrand. Gaussian mixture model classification of odontocetes in the southern California Bight and the Gulf of California. *The Journal of the Acoustical Society of America*, 121(3):1737–1748, 2007.

[73] Tzu-Hao Lin, Hsin-Yi Yu, Chi-Fang Chen, and Lien-Siang Chou. Passive acoustic monitoring of the temporal variability of odontocete tonal sounds from a long-term marine observatory. *PLoS One*, 10(4):e0123943, 2015.

[74] William W Steiner. Species-specific differences in pure tonal whistle vocalizations of five western north atlantic dolphin species. *Behavioral Ecology and Sociobiology*, 9(4):241–246, 1981.

[75] Volker B Deecke and Vincent M Janik. Automated categorization of bioacoustic signals: avoiding perceptual pitfalls. *The Journal of the Acoustical Society of America*, 119(1): 645–653, 2006.

[76] Kaitlin E Frasier, E Elizabeth Henderson, Hannah R Bassett, and Marie A Roch. Automated identification and clustering of subunits within delphinid vocalizations. *Marine Mammal Science*, 32(3):911–930, 2016.

[77] Marie A Roch, Holger Klinck, Simone Baumann-Pickering, David K Mellinger, Simon Qui, Melissa S Soldevilla, and John A Hildebrand. Classification of echolocation clicks from odontocetes in the Southern California Bight. *The Journal of the Acoustical Society of America*, 129(1):467–475, 2011.

[78] Julie N Oswald, Jay Barlow, and Thomas F Norris. Acoustic identification of nine delphinid species in the eastern tropical pacific ocean. *Marine mammal science*, 19(1):20–037, 2003.

[79] Michael J Bianco, Peter Gerstoft, James Traer, Emma Ozanich, Marie A Roch, Sharon Gannot, and Charles-Alban Deledalle. Machine learning in acoustics: Theory and applications. *The Journal of the Acoustical Society of America*, 146(5):3590–3628, 2019.

[80] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[81] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[82] Martin Längkvist, Lars Karlsson, and Amy Loutfi. A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, 42:11–24, 2014.

[83] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *Proceedings of the International conference on machine learning, ICML 2013*, pages 1310–1318, 2013.

[84] Claudio Gallicchio and Alessio Micheli. Deep Echo State Network (DeepESN): A brief survey. *arXiv preprint arXiv:1712.04323*, 2017.

[85] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings of the Advances in neural information processing systems, NIPS 2012*, pages 1097–1105, 2012.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[87] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[88] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations, ICLR 2020*, 2020.

[89] Yuan Gong, Yu-An Chung, and James Glass. AST: Audio spectrogram transformer. *arXiv preprint arXiv:2104.01778*, 2021.

[90] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *Proceedings of the 4th International Conference on Learning Representations, ICLR 2016*, page https://arxiv.org/abs/1510.00149. OpenReview.net, 2016.

[91] Shuochao Yao, Yiran Zhao, Aston Zhang, Lu Su, and Tarek Abdelzaher. DeepIoT: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems, 2017*, page 4. ACM, 2017.

[92] Jan Van Leeuwen. On the construction of Huffman Trees. In *Proceedings of the 3rd International Colloquium on Automata, Languages and Programming, ICALP 1976*, pages 382–410, 1976.

[93] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[94] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[95] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. In *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017*, page https://openreview.net/forum?id=SJGCiw5gl. OpenReview.net, 2017.

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

[97] Shuochao Yao, Yiran Zhao, Aston Zhang, Shaohan Hu, Huajie Shao, Chao Zhang, Lu Su, and Tarek Abdelzaher. Deep learning for the Internet of Things. *Computer*, 51(5):32–41, 2018.

[98] 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.

[99] Deniz Oktay, Johannes Ballé, Saurabh Singh, and Abhinav Shrivastava. Scalable model compression by entropy penalized reparameterization. *arXiv preprint arXiv:1906.06624*, 2019.

[100] Sangeeta Kumari, Dhrubojyoti Roy, Mark Cartwright, Juan Pablo Bello, and Anish Arora. EdgeL^3: Compressing L^3-Net for mote scale urban noise monitoring. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019*, pages 877–884. IEEE, 2019.

[101] Rajinikanth. Data Structures. http://www.btechsmartclass.com/data_structures/sparse-matrix.html, 2020.

[102] 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*, 41(10):2525–2538, 2019.

[103] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets. In *Proceedings of the 5th International Conference on Learning Representations, ICLR 2017*, page https://openreview.net/forum?id=rJqFGTslg. OpenReview.net, 2017.

[104] Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and Fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

[105] Yiwu Yao, Weiqiang Yang, and Haoqi Zhu. Creating lightweight object detectors with model compression for deployment on edge devices. *arXiv preprint arXiv:1905.01787*, 2019.

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

[107] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. In *Proceedings of the 6th International Conference on Learning Representations, ICLR 2018*, page https://openreview.net/forum?id=S1XolQbRW. OpenReview.net, 2018.

[108] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proceedings of the European conference on computer vision, ECCV 2016*, pages 525–542. Springer, 2016.

[109] Cong Wang. Pytorch XNOR-Net: XNOR-Net, with binary gemm and binary conv2d kernels, support both CPU and GPU. https://github.com/cooooorn/Pytorch-XNOR-Net, 2019.

[110] Adrian Bulat, Brais Martinez, and Georgios Tzimiropoulos. High-Capacity expert binary networks. In *Proceedings of the 9th International Conference on Learning Representations, ICLR 2021*, page https://openreview.net/forum?id=MxaY4FzOTa. OpenReview.net, 2021.

[111] Tyler L Hayes, Kushal Kafle, Robik Shrestha, Manoj Acharya, and Christopher Kanan. Remind your neural network to prevent catastrophic forgetting. In *European Conference on Computer Vision*, pages 466–483. Springer, 2020.

[112] Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, pages 583–592, 2019.

[113] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[114] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. iCaRL: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2001–2010, 2017.

[115] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

[116] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. *arXiv preprint arXiv:1711.10563*, 2017.

[117] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2017.

[118] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[119] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

[120] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv preprint arXiv:1708.01547*, 2017.

[121] Francisco M Castro, Manuel J Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *Proceedings of the European conference on computer vision (ECCV)*, pages 233–248, 2018.

[122] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[123] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, pages 374–382, 2019.

[124] Wenjing Han, Eduardo Coutinho, Huabin Ruan, Haifeng Li, Björn Schuller, Xiaojie Yu, and Xuan Zhu. Semi-supervised active learning for sound classification in hybrid learning environments. *PloS one*, 11(9):e0162075, 2016.

[125] Siyu Huang, Tianyang Wang, Haoyi Xiong, Jun Huan, and Dejing Dou. Semi-supervised active learning with temporal output discrepancy. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3447–3456, 2021.

[126] Burr Settles. Active learning. *Synthesis lectures on artificial intelligence and machine learning*, 6(1):1–114, 2012.

[127] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B Gupta, Xiaojiang Chen, and Xin Wang. A survey of deep active learning. *ACM Computing Surveys (CSUR)*, 54(9):1–40, 2021.

[128] Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In *International Conference on Learning Representations, ICLR 2019*, 2019.

[129] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1994*, pages 3–12. ACM/Springer, 1994.

[130] Zhao Shuyang, Toni Heittola, and Tuomas Virtanen. Active learning for sound event classification by clustering unlabeled data. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2017*, pages 751–755. IEEE, 2017.

[131] Sugato Basu, Arindam Banerjee, and Raymond J Mooney. Active semi-supervision for pairwise constrained clustering. In *Proceedings of the 4th SIAM International Conference on Data Mining, 2004*, pages 333–344. SIAM, 2004.

[132] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the 5th annual workshop on Computational learning theory, 1992*, pages 287–294, 1992.

[133] Nicholas Roy and Andrew McCallum. Toward optimal active learning through Monte Carlo estimation of error reduction. *ICML, Williamstown*, 2:441–448, 2001.

[134] William Coleman, Charlie Cullen, Ming Yan, and Sarah Jane Delany. Active learning for auditory hierarchy. In *Proceedings of the International Cross-Domain Conference for Machine Learning and Knowledge Extraction, 2020*, pages 365–384. Springer, 2020.

[135] Liz Huancapaza Hilasaca, Milton Cezar Ribeiro, and Rosane Minghim. Visual active learning for labeling: A case for soundscape ecology data. *Information*, 12(7):265, 2021.

[136] Mahnoosh Kholghi, Yvonne Phillips, Michael Towsey, Laurianne Sitbon, and Paul Roe. Active learning for classifying long-duration audio recordings of the environment. *Methods in Ecology and Evolution*, 9(9):1948–1958, 2018.

[137] Yu Wang, Ana Elisa Mendez Mendez, Mark Cartwright, and Juan Pablo Bello. Active learning for efficient audio annotation and classification with a large amount of unlabeled data. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing, ICASSP 2019*, pages 880–884. IEEE, 2019.

[138] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2):215–232, 1958.

[139] Jinghua Zhang, Chen Li, Sergey Kosov, Marcin Grzegorzek, Kimiaki Shirahama, Tao Jiang, Changhao Sun, Zihan Li, and Hong Li. LCU-Net: A novel low-cost U-Net for environmental microorganism image segmentation. *Pattern Recognition*, 115:107885, 2021.

[140] Łukasz Rączkowski, Marcin Możejko, Joanna Zambonelli, and Ewa Szczurek. ARA: Accurate, reliable and active histopathological image classification framework with Bayesian deep learning. *Scientific reports*, 9(1):1–12, 2019.

[141] Yanming Guo, Yu Liu, Erwin M Bakker, Yuanhao Guo, and Michael S Lew. CNN-RNN: A large-scale hierarchical image classification framework. *Multimedia tools and applications*, 77(8):10251–10271, 2018.

[142] Qian Chen, Keren Fu, Ze Liu, Geng Chen, Hongwei Du, Bensheng Qiu, and Ling Shao. EF-Net: A novel enhancement and fusion network for RGB-D saliency detection. *Pattern Recognition*, 112:107740, 2021.

[143] Hongyuan Yu, Yan Huang, Lihong Pi, Chengquan Zhang, Xuan Li, and Liang Wang. End-to-end video text detection with online tracking. *Pattern Recognition*, 113:107791, 2021.

[144] 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 2018*, pages 4510–4520, 2018.

[145] 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 2018*, pages 6848–6856, 2018.

[146] 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 2019*, pages 2820–2828, 2019.

[147] 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 2019*, pages 10734–10742, 2019.

[148] 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 2018*, pages 8697–8710, 2018.

[149] Christin Jose, Yuriy Mishchenko, Thibaud Senechal, Anish Shah, Alex Escott, and Shiv Vitaladevuni. Accurate detection of wake word start and end using a CNN. *arXiv preprint arXiv:2008.03790*, 2020.

[150] 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 *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2019*, pages 1–2. IEEE, 2019.

[151] Oyebade Oyedotun, Djamila Aouada, and Bjorn Ottersten. Structured compression of deep neural networks with debiased elastic group LASSO. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision, 2020*, pages 2277–2286, 2020.

[152] Igor Fedorov, Ryan P Adams, Matthew Mattina, and Paul Whatmough. Sparse: Sparse architecture search for CNNs on resource-constrained microcontrollers. In *Proceedings of the Advances in Neural Information Processing Systems, NeurIPS 2019*, pages 4977–4989, 2019.

[153] 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.

[154] Karol J. Piczak. ESC: Dataset for environmental sound classification. In *Proceedings of the 23rd Annual ACM Conference on Multimedia, 2015*, pages 1015–1018. ACM Press, 2015. ISBN 978-1-4503-3459-4. doi: 10.1145/2733373.2806390. URL http://dl.acm.org/citation.cfm?doid=2733373.2806390.

[155] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia, 2014*, pages 1041–1044. ACM, 2014.

[156] Naoya Takahashi, Michael Gygli, Beat Pfister, and Luc Van Gool. Deep convolutional neural networks and data augmentation for acoustic event detection. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, volume 8, pages 2982–2986, 2016.

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

[158] Andrey Guzhov, Federico Raue, Jörn Hees, and Andreas Dengel. Esresnet: Environmental sound classification based on visual domain models. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4933–4940. IEEE, 2021.

[159] Shaobo Li, Yong Yao, Jie Hu, Guokai Liu, Xuemei Yao, and Jianjun Hu. An ensemble stacked convolutional neural network model for environmental event sound recognition. *Applied Sciences*, 8(7):1152, 2018.

[160] Yanpeng Zhao, Jack Hessel, Youngjae Yu, Ximing Lu, Rowan Zellers, and Yejin Choi. Connecting the dots between audio and text without parallel data through visual knowledge transfer. *arXiv preprint arXiv:2112.08995*, 2021.

[161] Loris Nanni, Gianluca Maguolo, Sheryl Brahnam, and Michelangelo Paci. An ensemble of convolutional neural networks for audio classification. *Applied Sciences*, 11(13):5796, 2021.

[162] Jaehun Kim. Urban sound tagging using multi-channel audio feature with convolutional neural networks. *Proceedings of the Detection and Classification of Acoustic Scenes and Events, 2020*, page http://dcase.community/documents/challenge2020/technical_reports/DCASE2020_JHKim_21_t5.pdf, 2020.

[163] Anurag Kumar and Vamsi Ithapu. A sequential self teaching approach for improving generalization in sound event recognition. In *Proceedings of the International Conference on Machine Learning, ICML 2020*, pages 5447–5457. PMLR, 2020.

[164] Benjamin Elizalde, Soham Deshmukh, Mahmoud Al Ismail, and Huaming Wang. CLAP: Learning audio concepts from natural language supervision. *arXiv preprint arXiv:2206.04769*, 2022.

[165] Ke Chen, Xingjian Du, Bilei Zhu, Zejun Ma, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. HTS-AT: A hierarchical token-semantic audio transformer for sound classification and detection. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2022*, pages 646–650. IEEE, 2022.

[166] Karol J Piczak. Environmental sound classification with convolutional neural networks. In *Proceedings of the 25th IEEE International Workshop on Machine Learning for Signal Processing, MLSP 2015*, pages 1–6. IEEE, 2015.

[167] Dharmesh M Agrawal, Hardik B Sailor, Meet H Soni, and Hemant A Patil. Novel TEO-based Gammatone features for environmental sound classification. In *Proceedings of the 25th European Signal Processing Conference, EUSIPCO 2017*, pages 1809–1813. IEEE, 2017.

[168] Matthias Meyer, Lukas Cavigelli, and Lothar Thiele. Efficient convolutional neural network for audio event detection. *arXiv preprint arXiv:1709.09888*, 2017.

[169] Hardik B Sailor, Dharmesh M Agrawal, and Hemant A Patil. Unsupervised filterbank learning using Convolutional Restricted Boltzmann Machine for environmental sound classification. In *Proceedings of the 18th Annual Conference of the International Speech Communication Association, Interspeech 2017*, pages 3107–3111, 2017.

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

[171] Venkatesh Boddapati, Andrej Petef, Jim Rasmusson, and Lars Lundberg. Classifying environmental sounds using image recognition networks. *Procedia computer science*, 112: 2048–2056, 2017.

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

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

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

[175] Gianmarco Cerutti, Renzo Andri, Lukas Cavigelli, Elisabetta Farella, Michele Magno, and Luca Benini. Sound event detection with binary neural networks on tightly power-constrained IoT devices. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics, ISLPED 2020*, pages 19–24, 2020.

[176] 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 2016*, pages 770–778, 2016.

[177] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 KB RAM for the internet of things. In *Proceedings of the International Conference on Machine Learning, ICML 2017*, pages 1935–1944, 2017.

[178] 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 *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, pages 1331–1340, 2017.

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

[180] David Elliott, Carlos E Otero, Steven Wyatt, and Evan Martino. Tiny transformers for environmental sound classification at the edge. *arXiv preprint arXiv:2103.12157*, 2021.

[181] Steven Wyatt, David Elliott, Akshay Aravamudan, Carlos E Otero, Luis D Otero, Georgios C Anagnostopoulos, Anthony O Smith, Adrian M Peter, Wesley Jones, Steven Leung, et al. Environmental sound classification with tiny transformers in noisy edge environments. In *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*, pages 309–314. IEEE, Nov 2021.

[182] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015*, page http://arxiv.org/abs/1409.1556. OpenReview.net, 2015.

[183] 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 *Proceedings of the International Conference of the Cross-Language Evaluation Forum for European Languages, 2020*, pages 342–363. Springer, 2020.

[184] 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.

[185] 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. doi: 10.1080/09524622.2019.1705183.

[186] 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, ICCV 2015*, pages 1026–1034, 2015.

[187] Milton Friedman. A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, 11(1):86–92, 1940.

[188] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in statistics*, pages 196–202. Springer, 1992.

[189] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[190] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[191] Alexander Kozlov, Ivan Lazarevich, Vasily Shamporov, Nikolay Lyalyushkin, and Yury Gorbachev. Neural network compression framework for fast model inference. *arXiv preprint arXiv:2002.08679*, 2020.

[192] 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 2017*, pages 5687–5695, 2017.

[193] Ting-Bing Xu, Peipei Yang, Xu-Yao Zhang, and Cheng-Lin Liu. LightweightNet: Toward fast and lightweight convolutional neural networks via architecture distillation. *Pattern Recognition*, 88:272–284, 2019.

[194] 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 2019*, pages 8612–8620, 2019.

[195] Jian-Hao Luo and Jianxin Wu. AutoPruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, 107:107461, 2020.

[196] Elliot J Crowley, Jack Turner, Amos Storkey, and Michael O'Boyle. Pruning neural networks: Is it time to nip it in the bud? In *Proceedings of the NIPS workshop on Compact Deep Neural Networks with industrial application, 2018*, page https://openreview.net/pdf?id=r1lbgwFj5m. OpenReview.net, 2018.

[197] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *Proceedings of the 7th International Conference on Learning Representations, ICLR 2019*, page https://openreview.net/forum?id=rJlnB3C5Ym. OpenReview.net, 2019.

[198] Memory. https://www.arduino.cc/en/Tutorial/Foundations/Memory, Feb 2018.

[199] Tensorflow. Large bias values for inception V4 tflite model · ISSUE #33735 · Tensorflow/tensorflow. https://github.com/tensorflow/tensorflow/issues/33735, Oct 2019.

[200] The accuracy dropped 50% after covert a pre-trained TF model to quantized tflite. https://groups.google.com/a/tensorflow.org/g/tflite/c/K7sehHQasw8?pli=1, Mar 2020.

[201] Md Mohaimenuzzaman, Christoph Bergmeir, and Bernd Meyer. Pruning vs XNOR-Net: A comprehensive study of deep learning for audio classification on edge-devices. *IEEE Access*, 10:6696–6707, 2022.

[202] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-Real Net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm. In *Proceedings of the European conference on computer vision, ECCV 2018*, pages 722–737, 2018.

[203] Adam Byerly, Tatiana Kalganova, and Ian Dear. No routing needed between capsules. *arXiv preprint arXiv:2001.09136v6*, 2021.

[204] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning, ICML 2019*, pages 6105–6114. PMLR, 2019.

[205] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. *arXiv preprint arXiv:2106.04560*, 2021.

[206] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition, CVPR 2018*, pages 2704–2713, 2018.

[207] Kun Qian, Zixing Zhang, Alice Baird, and Björn Schuller. Active learning for bird sound classification via a kernel-based extreme learning machine. *The Journal of the Acoustical Society of America*, 142(4):1796–1804, 2017.

[208] Zhao Shuyang, Toni Heittola, and Tuomas Virtanen. An active learning method using clustering and committee-based sample selection for sound event classification. In *Proceedings of the 16th International Workshop on Acoustic Signal Enhancement, IWAENC 2018*, pages 116–120. IEEE, 2018.

[209] Xiao Qin, Wanting Ji, Ruili Wang, and ChangAn Yuan. Learnt dictionary based active learning method for environmental sound event tagging. *Multimedia Tools and Applications*, 78(20):29493–29508, 2019.

[210] Wanting Ji, Ruili Wang, and Junbo Ma. Dictionary-based active learning for sound event classification. *Multimedia tools and applications*, 78(3):3831–3842, 2019.

[211] Zhao Shuyang, Toni Heittola, and Tuomas Virtanen. Active learning for sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2895–2905, 2020.

[212] FindSounds. Sound types. https://findsounds.com/types.html, 2017.

[213] Sergei Vassilvitskii and David Arthur. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2006.

[214] Bongjun Kim and Bryan Pardo. I-SED: An interactive sound event detector. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces, 2017*, pages 553–557, 2017.

[215] Bongjun Kim and Bryan Pardo. A human-in-the-loop system for sound event detection and annotation. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 8(2):1–23, 2018.

[216] Dan Stowell, Dimitrios Giannoulis, Emmanouil Benetos, Mathieu Lagrange, and Mark D Plumbley. Detection and classification of acoustic scenes and events. *IEEE Transactions on Multimedia*, 17(10):1733–1746, 2015.

[217] Annamaria Mesaros, Toni Heittola, Aleksandr Diment, Benjamin Elizalde, Ankit Shah, Emmanuel Vincent, Bhiksha Raj, and Tuomas Virtanen. DCASE 2017 challenge setup: Tasks, datasets and baseline system. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events Workshop, DCASE 2017*, 2017.

[218] Nicolas Turpault, Romain Serizel, Ankit Shah, and Justin Salamon. Sound event detection in domestic environments with weakly labeled data and soundscape synthesis. In *Proceedings of the Detection and Classification of Acoustic Scenes and Events Workshop, DCASE 2019*, page 253, 2019.

[219] Yong Xu, Qiuqiang Kong, Wenwu Wang, and Mark D Plumbley. Large-scale weakly supervised audio classification using gated convolutional neural network. In *Proceedings of the IEEE international conference on acoustics, speech and signal processing. ICASSP 2018*, pages 121–125. IEEE, 2018.

[220] Hae-Sang Park and Chi-Hyuck Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

[221] Karol Piczak. ESC-50: Dataset for environmental sound classification. https://github.com/karolpiczak/ESC-50, 2017.

[222] Jordan Ash. Jordanash/Badge: An implementation of the badge batch active learning algorithm. https://github.com/JordanAsh/badge, 2020.

[223] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[224] Alessio Benavoli, Giorgio Corani, and Francesca Mangili. Should we really use post-hoc tests based on mean-ranks? *The Journal of Machine Learning Research*, 17(1):152–161, 2016.

[225] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[226] Salvador Garcia and Francisco Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of machine learning research*, 9(12), 2008.

[227] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

[228] DAWE. Southern black-throated finch (poephila cincta cincta). https://www.awe.gov.au/environment/biodiversity/threatened/assessments/poephila-cincta-cincta-2005, Feb 2005.

[229] Sharath Adavanne, Haytham M Fayek, and Vladimir Tourbabin. Sound event classification and detection with weakly labeled data. In *Acoustic Scenes and Events 2019 Workshop (DCASE2019)*, page 15, 2019.

[230] Sharath Adavanne, Archontis Politis, Joonas Nikunen, and Tuomas Virtanen. Sound event localization and detection of overlapping sources using convolutional recurrent neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 13(1):34–48, 2018.

[231] Huy Phan, Philipp Koch, Fabrice Katzberg, Marco Maass, Radoslaw Mazur, Ian McLoughlin, and Alfred Mertins. What makes audio event detection harder than classification? In *Proceedings of the 25th European signal processing conference, EUSIPCO 2017*, pages 2739–2743. IEEE, 2017.

[232] Paulo Lopez-Meyer, Juan A del Hoyo Ontiveros, Hong Lu, and Georg Stemmer. Efficient end-to-end audio embeddings generation for audio classification on target applications. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 601–605. IEEE, 2021.

[233] Tomoki Hayashi, Shinji Watanabe, Tomoki Toda, Takaaki Hori, Jonathan Le Roux, and Kazuya Takeda. Duration-controlled LSTM for polyphonic sound event detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(11):2059–2070, 2017.

[234] Koichi Miyazaki, Tatsuya Komatsu, Tomoki Hayashi, Shinji Watanabe, Tomoki Toda, and Kazuya Takeda. Weakly-supervised sound event detection with self-attention. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 66–70. IEEE, 2020.

[235] Qiuqiang Kong, Yong Xu, Wenwu Wang, and Mark D Plumbley. Sound event detection of weakly labelled data with CNN-transformer and automatic threshold optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2450–2460, 2020.

[236] Piper Wolters, Chris Daw, Brian Hutchinson, and Lauren Phillips. Proposal-based few-shot sound event detection for speech and environmental sounds with perceivers. *arXiv preprint arXiv:2107.13616*, 2021.

[237] Yin Cao, Turab Iqbal, Qiuqiang Kong, M Galindo, Wenwu Wang, and M Plumbley. Two-stage sound event localization and detection using intensity vector and generalized cross-correlation. *Tech. report of Detection and Classification of Acoustic Scenes and Events 2019 (DCASE) Challange*, 2019.

[238] Sławomir Kapka and Mateusz Lewandowski. Sound source detection, localization and classification using consecutive ensemble of crnn models. *arXiv preprint arXiv:1908.00766*, 2019.

[239] Chieh-Chi Kao, Weiran Wang, Ming Sun, and Chao Wang. R-CRNN: Region-based convolutional recurrent neural network for audio event detection. *arXiv preprint arXiv:1808.06627*, 2018.

[240] Huy Phan, Marco Maaß, Radoslaw Mazur, and Alfred Mertins. Random regression forests for acoustic event detection and classification. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(1):20–31, 2014.

[241] Huy Phan, Lam Pham, Philipp Koch, Ngoc QK Duong, Ian McLoughlin, and Alfred Mertins. Audio event detection and localization with multitask regression network. In *Proceedings of the Acoustic Scenes and Events 2020 Workshop (DCASE2020)*, pages 2–4, 2020.