Universitat Rovira i Virgili (URV)

Universitat Oberta de Catalunya (UOC)

MASTER IN COMPUTATIONAL AND MATHEMATICAL ENGINEERING

FINAL MASTER PROJECT (FMP)

Area: Artificial Intelligence

Using Deep Learning for Sound Classification in Citizen Science: A Practical Approach with *Soundless*

Author: David Castelló Tejera

Tutor: Pedro García Lopez

Date of Delivery: September 1st of 2023

Dr./Dra. *Pedro García Lopez*, certifies that the student *David Castelló Tejera* has elaborated the work under his/her direction and he/she authorizes the presentation of this memory for its evaluation.

Director's signature:

GNU Free Documentation License (GNU FDL)

Final Project Sheet

Title:	Using Deep Learning for Sound Classification in Citizen Science: A Practical Approach with Soundless
Author:	David Castelló Tejera
Tutor/a:	Pedro García Lopez
Date:	September of 2023
Program:	Master in Computational and Mathematical Engineering
Area:	Artificial Intelligence
Language:	English
Key words:	Deep learning, Audio Classification, Citizen Science

Dedicatory

Dedico este trabajo a mi hermana por su amor y su apoyo incondicional.

Agradecimiento especial a mis compañeros del San Javier. Su amistad, su felicidad y su calidez han sido la inspiración necesaria para trabajar en este proyecto por las tardes.

Gracias a Carmen por creer en mí. Sin su apoyo, su fuerza, sus consejos, y su dedicación, este trabajo no habría salido adelante.

Gracias a Jesús y a Gonzalo por el soporte técnico y emocional (en proporciones iguales).

Abstract

The field of Deep Learning has experienced tremendous growth in recent years, sparking interest among users and researchers. However, deploying Deep Learning models in real-world projects presents significant technical challenges. This Master's Thesis provides a practical approach to designing and constructing a custom Deep Learning model for audio classification, intended for use within the *Soundless* project—a citizen science platform investigating noise pollution and its impact on human health.

The primary objective is to construct a custom model deployable within the Android application of the *Soundless* project. Different model architectures are explored considering the complexity constraints of deploying Deep Learning models on the edge. The model is built using the TensorFlow framework. Evaluated against the ESC-50 benchmark, the model demonstrates prediction accuracies of over 86%. The model is then integrated into an Android app prototype for testing.

A custom dataset is constructed, termed NBAC, comprising 780 audio samples covering 13 distinct classes. NBAC is designed to be aligned with the acoustic context of the *Soundless* project. The model's performance on NBAC achieves over 90% accuracy.

Further, this work investigates various implementation alternatives for utilizing and enhancing the model in a production environment. A centralized improvement approach is proposed, which entails locally storing labeled feature representations of audio samples and training a classifier. Alternatively, a decentralized improvement approach is formulated using Federated Learning. Both strategies, leveraging the custom-designed models, yield promising outcomes. They not only preserve the anticipated accuracies but also facilitate the desired enhancements.

Keywords: Deep Learning, Audio Classification, Citizen Science, Deployment on Android, ESC-50, TensorFlow, Federated Learning.

Index

List of figures

1. Context and Problem

This Master's Thesis intends to propose a novel solution for the *Soundless* project in the area of audio event recognition for Android devices. While this work is fundamentally a practical approach, profound research needs to be conducted as recent state of the art solutions offer strict limitations regarding complexity and deployment limitations. This section of the Thesis outlines the context of the *Soundless* project and defines the problem that will be tackled in the upcoming sections.

1.1. The Soundless Project

Soundless is a citizen science platform designed to audit the effects of noise pollution on human health [1]. The project is based in the city of Tarragona, where all research is conducted. The primary goal of this project is to identify acoustic disturbances that occur at night and raise awareness about them. It aims to determine whether these disturbances surpass existing regulations and if they pose significant health risks to the public. The *Cloud and Distributed Systems Lab* developed the *Soundless* initiative at the *Universitat Rovira i Virgili*, with co-financing from the *Diputación de Tarragona*.

The research project is motivated by recent studies published by the World Health Organization (WHO), warning about the potential effects of noise on human health. Sound levels exceeding 45 dB during nighttime are linked to insomnia, stress, low productivity, obesity, and other irreversible health problems [2].

As a citizen science platform, the project is possible thanks to the participation of citizen communities such as the FAVT (*Federación de Asociaciones de Vecinos de Tarragona*), along with neighborhood associations of Francolí, Tarragona 2 and a community of neighbors from the Serrallo neighborhood.

A primary concern for the public collaborating with Soundless is the regular, scheduled noises during nighttime in Tarragona's neighborhoods that exceed the regulatory sound-level limits. These noises primarily originate from railways and roads. The scientists at Soundless not only study the impact of these sounds on human health but also use their findings as evidence of poor public policy decisions that require improvement.

1.2. How Soundless Works

Citizens participating in the project (referred to as participants in this document) use the *Soundless* app for Android [3], which captures noise levels through the microphone of the participant's mobile device. Additionally, the app records health data through a bracelet worn by the participant. This information is stored in data points that collect the information across 5-second-long windows. The data is voluntarily sent to Google Cloud servers for daily authentication and storage in the cloud. The collected data is then analyzed using statistical methods to identify the causal and temporal correlation between noise levels and health data. The platform also generates heat maps showing nighttime incidents by zones.

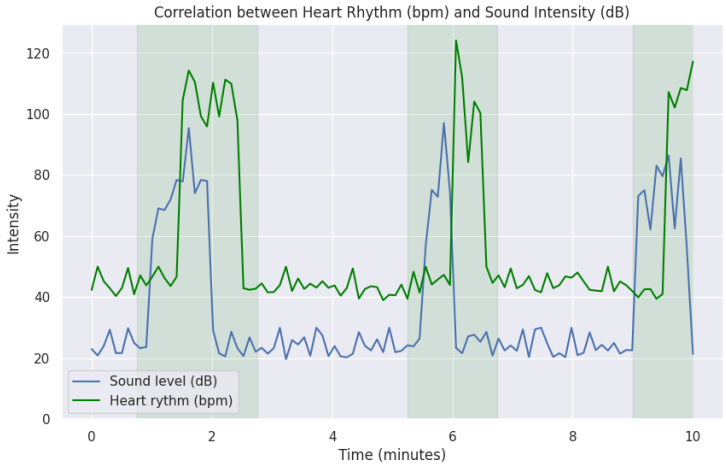

Figure 1: Simulated correlation between heart rhythm and sound intensity

The methodological core of the *Soundless* app is based on the citizen science platform. Citizen science platforms provide tools and resources for individuals to participate in scientific research and address real-world problems [4]. These platforms can include user interface and experience design to make it easier for people to participate.

Researchers, institutions, and governments are increasingly paying more attention to citizen science. It fosters active public participation in scientific research, leading to more ambitious and networked projects, and provides access to extensive data that would otherwise be challenging to collect. Additionally, it serves as a valuable platform to address issues of public interest, thereby strengthening the bridge between science and society [5] [6].

Citizen science enables data collection across a wide geographic range and over extended periods of time without the need for expensive or complex data collection tools. This has been a fundamental feature of the *Soundless* project.

1.3. Aims of the Work

The data points collected at *Soundless* have two dimensions: acoustic data and health data. This work proposes a new feature to expand the depth of the acoustic data captured by the *Soundless* app.

Currently, the research is primarily collecting sound-level data in decibels using microphones on the mobile phones of the participants. This task presents a challenge with average-quality smartphones due to the absence of straightforward methods for acquiring such data [7]. Additionally, sound-level data is shallow in the context of this research, as many different sound events can have very similar sound-level profiles.

This Master's Thesis proposes integrating an audio event classification model in the *Soundless* app to perform on-device audio classification, increasing the depth of the data collected from the participants and overcoming the limitations of sound-level measurement. The model will perform real-time predictions to classify audio recorded within the *Soundless* app and tag each data point with a sound event label.

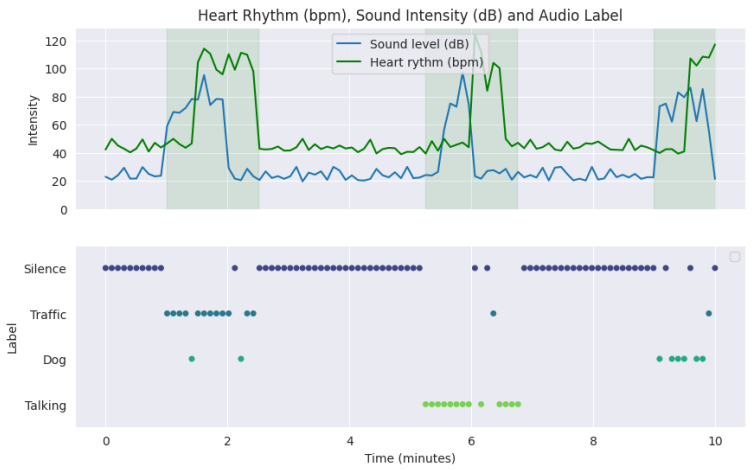

Figure 2: Visualization of the proposed dimensional expansion

The anticipated outcome of this project is a model that can be integrated into the *Soundless* Android app, performing real-time audio classification during nighttime within a bedroom setting. The main contribution of this work is the development of a highly customizable and flexible deploy-ready audio event. The practical approach of this work is grounded on an exhaustive analysis, defining the best working solution to develop a model within the specific constraints of the *Soundless* project.

The objectives established by the *Soundless* team include attaining an accuracy of over 85% and enabling real-time inference on-device to categorize the 5-second-long data points.

The classification will be evaluated on the ESC-50 dataset and the beta version of a dataset manually composed for this project called the *Nocturnal Bedroom Audio Classification* dataset (NBAC).

1.4. Motivation

While this project has broad potential applications and offers opportunities for enhancements to *Soundless*, its primary aim is to enable the *Soundless* app on each participant's device to track specific trains. The Soundless team has agreed to develop a general-purpose audio event classifier, which will enrich the data collected from participants. However, the ultimate goal is to create an audio event classifier capable of distinguishing not just between general sounds like breathing and traffic, or dogs and snoring, but also differentiating among various train sounds. This work is focused on developing a general audio event classifier, always keeping this specific goal in mind, as the solution is designed to be flexible enough to push the boundaries of the audio event classifier's performance.

1.5. State of the Art

Recent advances in deep learning have greatly improved audio event classification. While newer models even outperform humans in tests, mobile devices often use simpler, less capable models. This is due to memory and computational constraints. Speech recognition on mobile devices is ahead because researchers have put strong efforts on it, especially for features like voice-assisted typing, and large datasets are available to train and test the models. However, audio event classification still needs work. Current projects can only

identify a limited range of sounds on mobile devices. During the research stage of this work, there are no signs of similar audio event classification products on the market. State of the art projects such as Google's YAMNet, an TensorFlow model built for mobile phone deployment, can identify over 500 sounds quickly but with low accuracy.

1.6. Scope and Limitations

Many efforts are being put on deploying machine learning models on edge devices. This type of deployment intends to take data storage and processing closer to the end user. The work of this Master's Thesis focuses on deployment in mobile devices. This preference is evident considering the principles of citizen science. All citizens have access to commodity smartphones and can collaborate with city science problems such as *Soundless*.

For this project, *Soundless* requires the model to be deploy-ready for Android applications. The model will be developed with the TensorFlow libraries. The choice of this library is influenced by its good compatibility with Android. Google develops both frameworks and has extensive support and documentation. As of Spetmeber 2023, the *Soundless* app is only supported by Android devices.

The project faces two main limitations: i) computational constraints and ii) data variability.

> i) To maintain privacy, the sound classification must occur on-device; recorded sounds can't be uploaded to the Google Cloud server. This approach also optimizes memory usage on the cloud server. Given the limited computational resources of smartphones, the model must be efficient, compact, and energy-conserving to ensure smooth operation without draining the battery or interfering with other applications.
>
> ii) The model will be operating in environments where multiple sounds often occur simultaneously, acoustic conditions greatly fluctuate, and the quality of sound sensors differ. Therefore, the model must be designed to handle large data variability and perform effectively under highly unpredictable conditions.

1.7. Approach and Methodology

The execution of this project started with the careful selection and preparation of datasets. An established dataset, the ESC-50, was employed as a benchmark due to its high support in the literature and its similarity to the *Soundless* context. A custom dataset, the NBAC, is created to ensure a broad spectrum of audio instances that fit the requirements of the project.

Following the data selection, the project progressed into the modeling phase. Various deep learning models were trained on the Kaggle notebooks platform, leveraging its powerful computational capabilities. This approach facilitated a comparative analysis of the models' performance, enabling the identification of the most effective model(s) for audio classification based on precision and computational complexity.

Once the top-performing models were identified, a machine-learning application to build the custom model with the desired architecture was developed on a local machine using the Windows Subsystem for Linux 2 (WSL2).

The final phase involved integrating the custom-trained model into an Android application prototype. A prototype was modified to integrate the custom model, providing a practical, user-oriented environment for observing and assessing the model's performance.

After the completion of the final product, an exploratory analysis is conducted to investigate possible improvement methodologies to exploit the potential of citizen science on production. The analysis outlines the capabilities of the model, and proposes two different approaches to exploit data from participants and improve the model.

Throughout this project, an iterative approach was maintained - each phase was continually revisited and refined based on the insights and results from the subsequent stages.

1.8. Planning of the Work

Planning for the hardware resources is an integral step in developing a machine-learning project of this scope. To ensure an efficient work environment and guarantee the project's replicability, this section details the allocated hardware resources.

The project employs two distinct work environments to facilitate development and investigation. Kaggle Notebooks serve for testing procedures, while a Windows Personal Computer operates locally the machine learning application designed for the deployment model and creates a simulated Federated Learning environment.

Kaggle Notebooks: This platform delivers ready-to-use virtual environments equipped with hardware acceleration to run Python scripts for machine learning and data science tasks. Kaggle grants 30 hours of free computing time monthly and is pre-installed with all the essential libraries. It is possible to install additional libraries directly onto the virtual machine. Despite these benefits, Kaggle offers limited customization and does not support certain TensorFlow Lite libraries.

Windows Personal Computer with GPU and WSL2: This environment includes a standard laptop equipped with a basic Nvidia GPU (GeForce MX330) to run on the Windows Subsystem for Linux (WSL2). The source code required for the TensorFlow Lite libraries employed in this work is exclusively buildable on a UNIX-based system. Additionally, the Federated Learning environment is created on WSL2.

The project also involves the utilization of three modern smartphones to create the NBAC dataset. The devices used are the Asus Zenphone 8, iPhone 13 PRO, and Xiaomi 13.

The project's developmental process is planned across five stages: Research, Experiments, Deployment, Improvements and Documentation. Each stage is composed of multiple tasks. The diagram in Figure 3 provides a comprehensive overview of the planification followed for this Master's Thesis work. The work has been planned across a 9-month period, from mid-September of 2022 to the end of July of 2023.

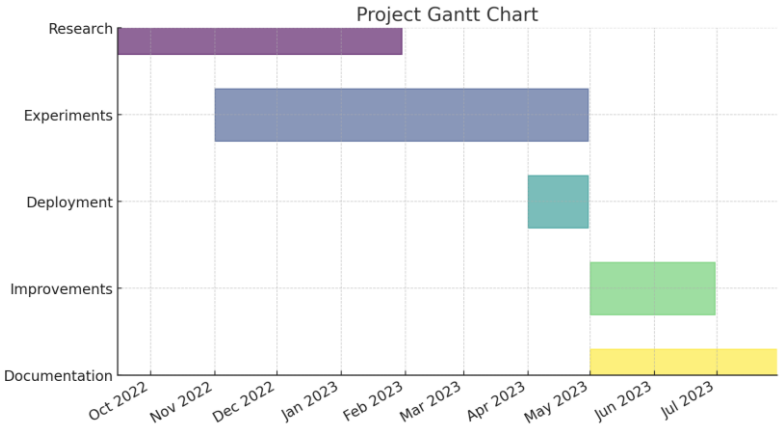

Figure 3: Distribution of the workload

2. Theory

2.1. Fundamentals of Audio Signal Processing

Sound is generated through mechanical vibrations that induce oscillations in the air particles. These oscillations are referred to as sound waves. Regular microphones work by converting sound waves into electric signals. This process involves a thin mechanical sensor called a diaphragm that vibrates when sound waves hit it. This vibration is translated to an electronic device (a coil or a capacitor), creating an electric signal representing the sound wave [8].

During the recording of a sound wave, the sequence of stored electric signals over a period of time is known as a waveform (Figure 4). Each signal is stored as the amplitude of a wave, and the number of electric signals recorded within a one-second time period is known as the frequency [8]. Most mobile devices have a sampling frequency for audio of 44.1 kHz, meaning the number of captured electric signals that represent sound vibrations is 44100 per second.

Most modern mobile devices integrate stereo recording to enhance sound quality, aiming to create a wider and more immersive sound field. Stereo sampling involves the use of two microphones that record two separate sound channels. There is no evidence in the literature that using stereo recording improves the accuracy of audio classification.

The classification of audio signals is a field that has been active for decades. Waveforms have singular mathematical properties that allow scientists to extract features from sounds and perform automatic classification [9]. State-of-the-art audio classification models leverage a combination of feature extraction techniques and advanced machine learning models to accurately identify and classify audio signals [10].

Recent classification deep learning models integrate feature extraction with spectrograms. A Spectrogram provides a highly informative two-dimensional representation of an audio signal, showing how the frequencies contained in the signal are distributed with respect to time. This time-frequency representation makes them akin to images, making them well-suited for Convolutional Neural Networks (CNNs), which have demonstrated exceptional performance in image classification tasks. Experiments show that the capacity of CNNs to identify features in images is useful for audio classification using spectrograms [11].

The spectrogram of a waveform can be obtained by segmenting the time-domain signal into equal lengths, applying the Fast Fourier Transform (FFT) to each segment, and graphically representing the spectrum of each segment [12]. In this work, spectrograms are obtained using the Mel Scale. The Mel Scale is designed in a way that equal distances on the scale correspond to what humans perceive as equal differences in pitch [13]. Consequently, Mel spectrograms (Figure 4), which are based on the Mel Scale, are frequently used as the preferred feature for training deep learning algorithms in audio processing [11] [14].

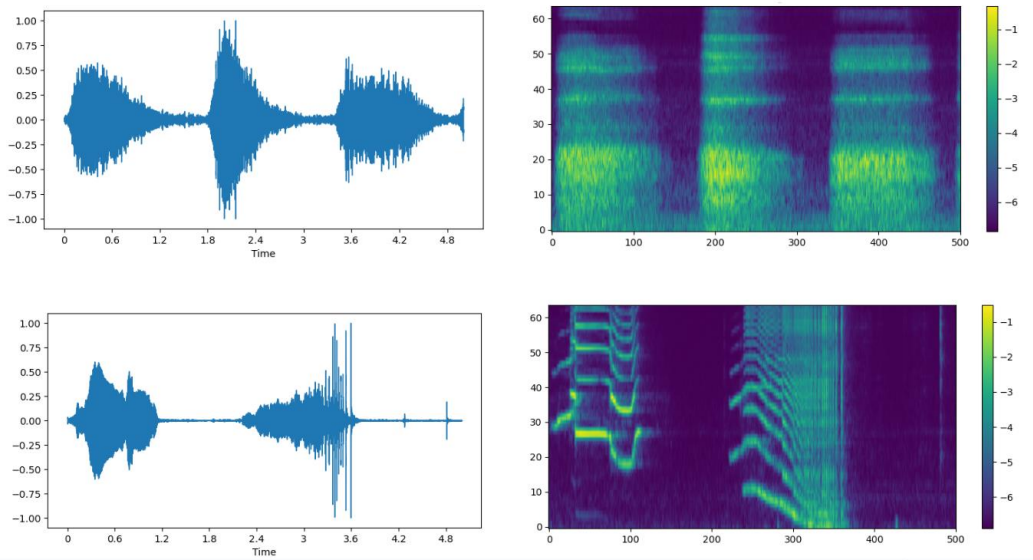

Figure 4: Waveform and spectrogram representation

2.2. Deep Learning for Audio Classification

The model is developed through deep learning, a subfield of machine learning that utilizes neural networks with several layers to progressively extract higher-level features from input data. This methodology allows the model to learn complex patterns and relationships within the data, enabling it to perform tasks ranging from image recognition, natural language processing, or predictive analytics. With the increasing power of computational resources and the abundance of data, deep learning has been at the forefront of many technological advancements and breakthroughs in recent years.

The learning process for the proposed audio classification task uses a training dataset consisting of a large number of labeled audio samples. Each sample is an instance of an

audio clip paired with a corresponding category label, reflecting the type of sound it contains - such as dog barks, music, traffic, etc. The model is trained by iteratively processing these samples and adjusting its internal parameters to minimize the difference between predicted and true labels. This process is known as backpropagation. In addition, a separate validation dataset is used during training to tune hyperparameters and monitor the model's performance to prevent overfitting. Once the model is adequately trained, its generalization ability is further tested on a completely unseen test dataset, providing an unbiased evaluation of its classification accuracy.

2.2.1. Convolutional Neural Networks

Convolutional neural networks (CNNs) are at the core of state-of-the-art approaches to a variety of computer vision tasks. These types of architectures utilize convolution and pooling layers (also referred to as subsampling layers) to extract features from input image-like data. The input is fed into convolutional layers, which conduct feature extraction by moving filters over the input data. This action results in the creation of a feature map. This feature map is subsequently passed to a pooling layer, which performs sub-sampling to reduce the dimensions and size of the feature maps, thereby lessening the computational power required (Figure 5). These sub-sampled feature maps can then continue to be inputted into subsequent convolutional layers, repeating the process [15].

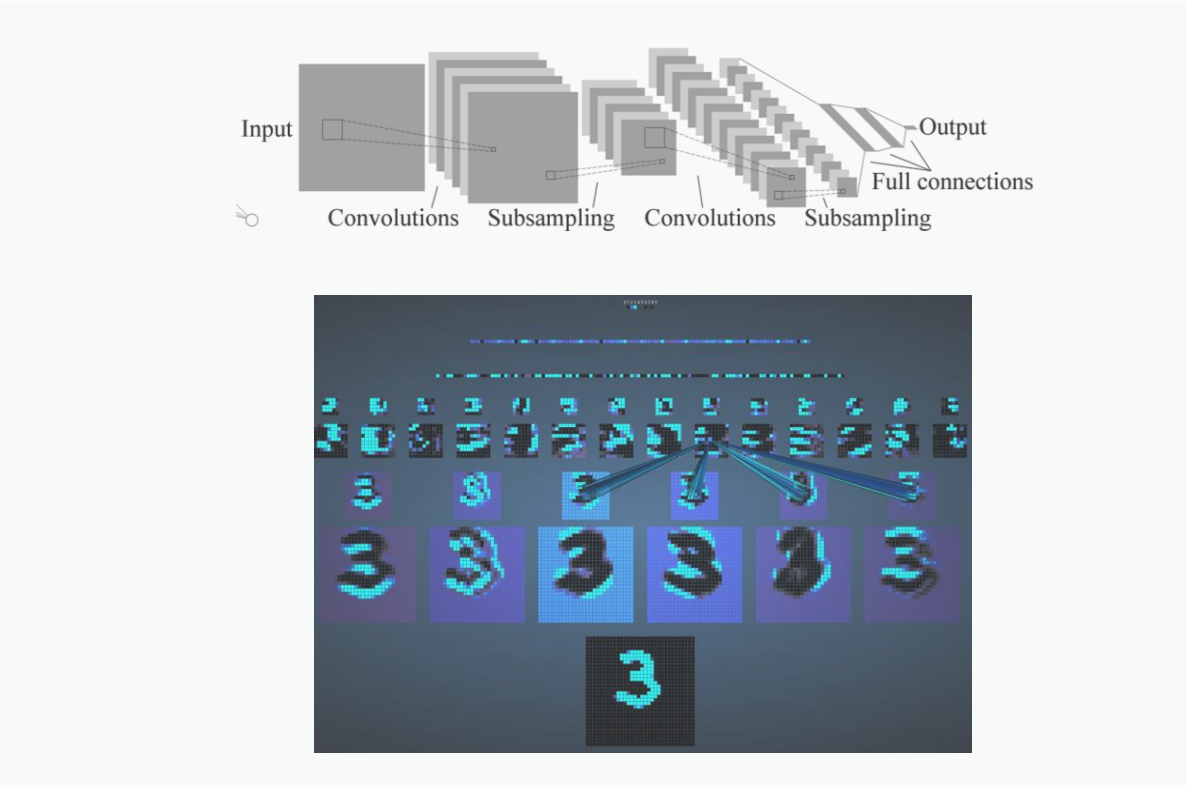

Figure 5: Visualization of the inner layers of a CNN. Extracted from [15] (top) and [16] (bottom).

2.2.2. Depthwise Separable Convolutions

Another type of deep neural networks are Depthwise Separable Convolutions, based on a variation of the standard convolutional layer that reduces the computational cost while maintaining model performance. Depthwise Separable Convolutions consist of two separate layers - a depthwise convolutional layer and a pointwise convolutional layer.

In a depthwise convolution, instead of combining all input channels from the previous layers, each channel is convolved with its own set of filters, hence 'depthwise'. This significantly reduces the number of parameters and computational resources needed compared to traditional convolutional layers. Then, in the pointwise convolutional layer, the output of the depthwise convolution is passed through a 1 x 1 convolutional layer - a technique used to alter the depth of the channel. These two stages together form the depthwise separable convolution, which can lead to models that are lighter and faster, but still effective [17] [18].

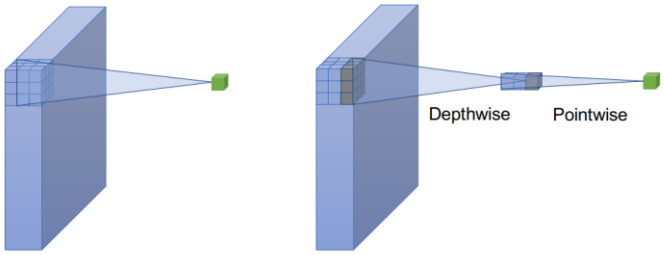

Figure 6: Standard convolution vs depthwise convolution. Extracted from [17]

This makes Depthwise Separable Convolutions particularly useful for deploying models on devices with limited computational resources, such as mobile devices. Initially, this architecture is ideal for building lightweight networks to process RGB images (with 3 input channels). Audio classification with Depthwise Separable Convolutions takes 1 channel image-like spectrograms as input. Although this configuration is not fully exploiting the optimization capacity of these models, using depthwise convolutions in the internal layers, which are multi-channel feature maps, has proven to be efficient [19] [20]. Some authors exploit the full potential of Depthwise Separable Convolutions by creating 3 channel spectrograms with different frequency bands [21].

3. Development of the Solution

This section of the work provides an overall description of the solution process. The main part of the product is the Deep Learning model for audio event classification. The Machine Learning Pipeline in section 3.1. defines the steps to prepare the data, create, train, and evaluate the model. The deployment of the trained model for inference is done on an Android device, for that purpose, section 3.2. outlines the necessary steps to run inference on a prototype app using TensorFlow Lite, which provides an easy-to-use framework to deploy TensorFlow models on edge devices. The solution requires carefully selected data. Section 3.3. explains the datasets that will be used across this work.

Section 4 offers a thorough look at how the audio event classification models are made and tested.

3.1. Machine Learning Pipeline

A machine learning pipeline is a term used to describe a series of steps involved in building a model that learns from data. The pipeline includes different stages that define the flow of data. This section describes the machine learning pipeline of this project.

First, audio pre-processing is applied to segment, and sample audio files. Adding normalization is a common technique in data pre-processing, however, as described in section 3.1.2, normalization is applied downstream in the LLF block. Data augmentation, such as time stretching, time shifting, and adding noise, is then used to increase the diversity of the waveforms. Low-level features are extracted by converting the audio into Mel spectrograms, which highlight perceptually significant frequency bands. Additional data augmentation techniques can be applied to expand the spectrograms as well. High-level features or embeddings are obtained using a deep learning model that processes the Mel spectrograms. These features are then used to make predictions about the classes of the waveforms using a trained classifier. Lastly, the performance of the classification model is evaluated using metrics like accuracy, the number of total parameters, or the number of multiply-accumulate (MAC) operations, and adjustments are made to optimize the model based on these results.

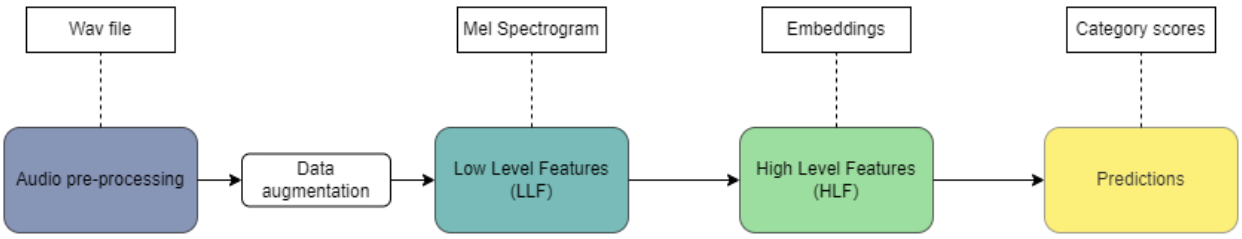

Figure 7: Visualization of the machine learning pipeline

This structure is typical in state-of-the-art models, which feed the spectral features to a neural network to extract the embeddings [22] – [25]. Other innovative approaches incorporate raw waveform classification by constructing trainable Low-Level Feature CNN blocks within the model that emulate spectral feature extractors [26] – [28].

3.1.1. Audio Pre-processing

Handling media files requires a careful procedure, factoring in aspects such as file format, characteristics, and size. The datasets used in this project (refer to section 3.3) include audio files that are 5 seconds long with a sampling frequency of 44.1 kHz in mono channel.

To introduce the data into the machine-learning pipeline, the audio files are reconfigured to a frequency of 16Hz. This frequency reduction is commonly employed in the field, consistently demonstrating good classification results. Importantly, it also reduces the computational memory requirement, making the process more efficient.

Normalization is a common preprocessing step for neural networks, ensuring that the input values are within a scale that the model can efficiently process. Thus, to guarantee consistency data must be normalized before the training process. The low-level feature extractor will work with waveforms in the range of -1 to 1. Normalization has been integrated using the util from Librosa [29] or manually based on the maximum and minimum values of the waveforms.

Regarding audio length, the solutions developed in this work explore two different configurations:

al5s: Classifying audios with 5-second audio length
al2s: Classifying audios with 2-second audio length

The aim of these alternate configurations is to understand the optimal method for high-level feature extraction - whether it is through shorter time frames that exhibit high variability or longer ones that pull out more generalized features.

The datasets comprise 5-second audio snippets. To obtain the 2-second time frame set up, 2-second windows are slid across the original snippets with a hop size of 0.5 seconds. This setup essentially acts as a form of data augmentation. As a result, from a 5-second snippet, 7 files are generated, totaling a duration of 14 seconds.

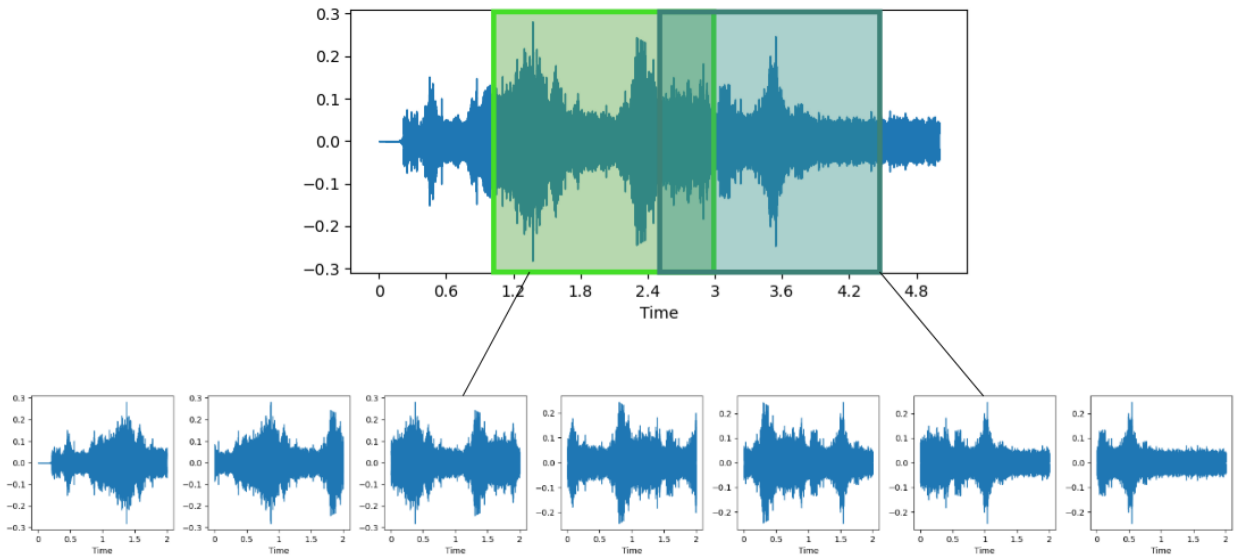

Figure 8: Splitting audio fragments

3.1.2. Data Augmentation

Using appropriate data augmentation techniques is crucial in machine learning as it helps address data scarcity and improve model performance by enhancing the size and quality of training datasets. In the context of acoustic machine learning, data augmentation techniques can cover environmental variability, thereby improving the model's performance in different acoustic settings. By providing more diverse training data, the model is forced to learn more general features rather than memorizing the training data [30]. Furthermore, data augmentation helps mitigate overfitting, a common problem where a model performs well on training data but poorly on unseen data. Data augmentation is only applied to the training partition, as applying it to the whole dataset could imply having highly similar data in the training, validation, and testing partitions.

Data augmentation of the waveforms has been applied in all the experiments. Proven effective techniques include noise injection, time shifting, and time stretching [30].

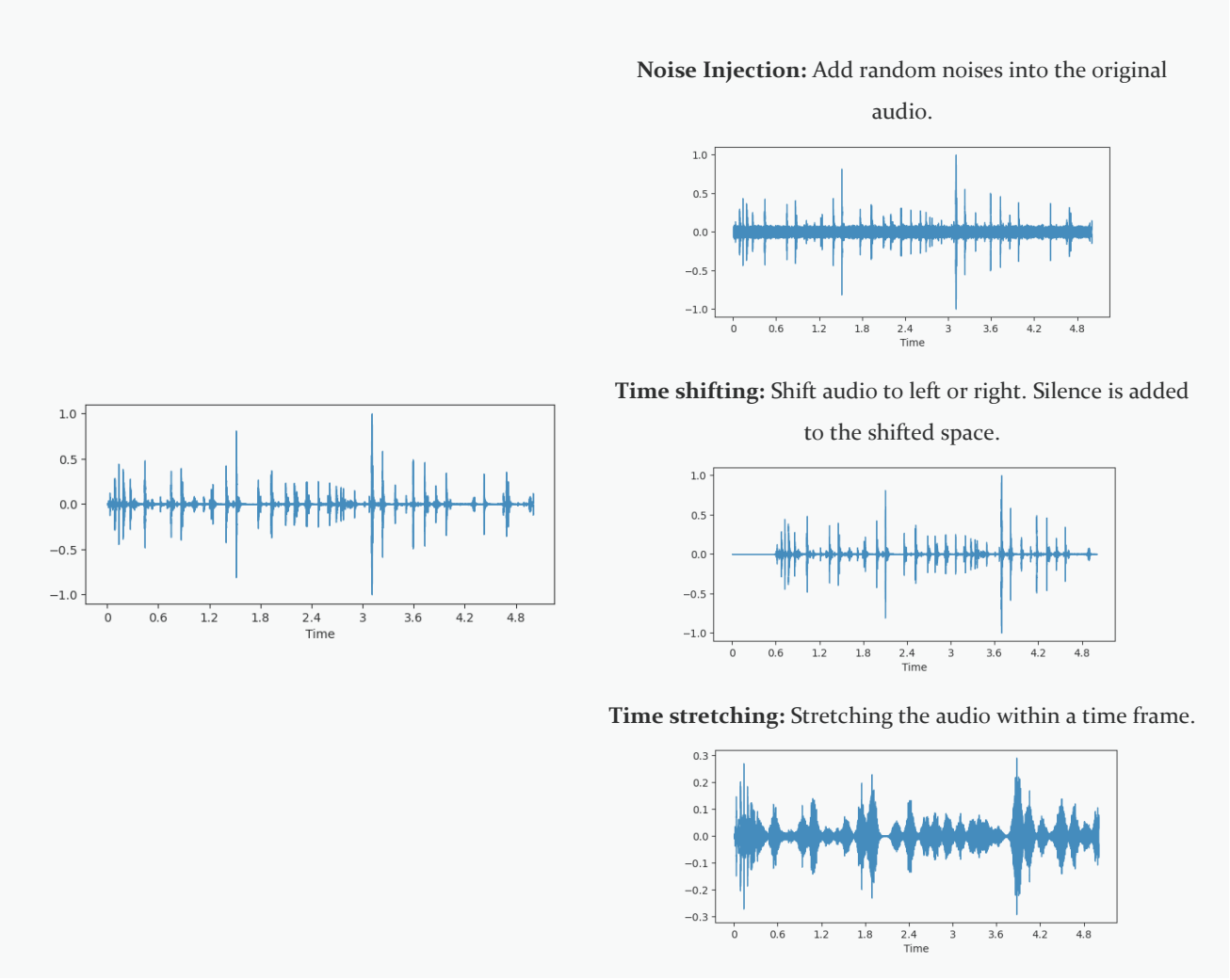

Figure 9: Waveform data augmentation techniques

Additionally, a second data augmentation step can be applied to the spectrograms (Figure 10). This data augmentation procedure is only applied in models that use the Low-Level Feature extractor outside the deep learning model (view section 4 for further details). To achieve this, time and frequency masks are applied to the spectrograms. The chosen augmentations are designed to enhance the network's resilience against time-direction deformations, partial frequency information loss, and partial loss of small segments in the input [31]. An illustration of this augmentation strategy can be seen in Figure 11.

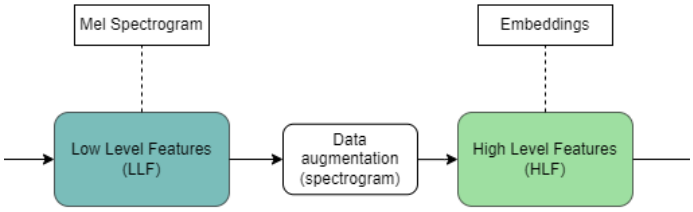

Figure 10: Spectrogram augmentation block

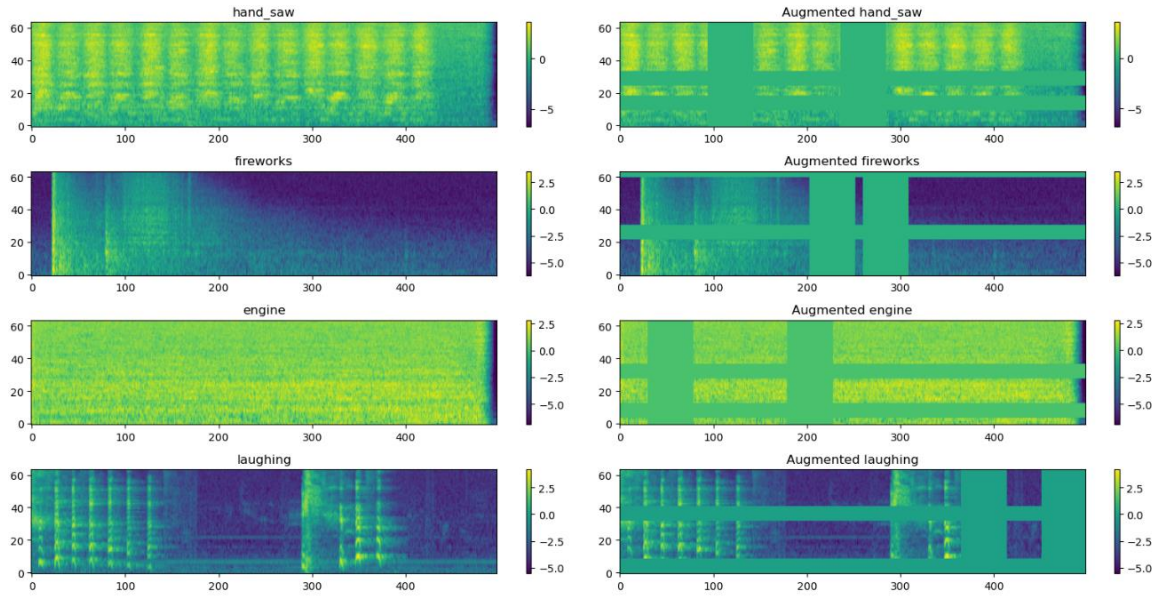

Figure 11: Spectrogram data augmentation techniques

Other advanced data augmentation techniques are used by mixing sounds [30]. The examination of more sophisticated data augmentation methods is planned for future research.

3.1.3. Low-Level Features

As discussed in section 2.1, Mel Spectrograms provide excellent acoustic features for Deep Learning purposes. Thus, the Low-Level Feature extractor employed in this work is a Mel Spectrogram generator. Librosa, an audio processing library for Python, and TensorFlow, the deep learning framework, provide functions to extract the Mel Spectrogram of a waveform [32] [33].

Both options are explored to compute the Mel Spectrograms (view Annex 1). Librosa provides a simple and effective framework to compute the Mel Spectrograms. However, it

is not possible to integrate Librosa's framework into the Deep Learning models (view section 4.2), therefore a custom spectrogram generator based on TensorFlow is developed.

The custom Mel Spectrogram generator is built with the parameters employed in the YAMNet model [25]. The generator process follows the steps indicated below.

Loading and Normalization: The audio input is first loaded and cast to a float32 data type. The data is then normalized to be in the range (-1, 1). The original waveform signal is scaled so that its minimum value is mapped to -1, and its maximum value is mapped to 1. This normalization step will allow the custom Mel spectrogram generator to accept the raw waveform, without the need of using Librosa's normalization util in the pre-processing stage.

Short-Time Fourier Transform (STFT): Next, the Short-Time Fourier Transform (STFT) is applied to the audio signal. The STFT is a method used to analyse the frequency content of a signal over time. It divides the signal into short frames, each of size 25 milliseconds. These frames overlap by 10 milliseconds. The size of 25 milliseconds is a common choice as it is long enough to capture relevant information but short enough to assume that the audio signal is stationary. The overlap of 15 milliseconds ensures that no information is missed between frames. The Fourier transform of each frame is computed and the Hann window function is used to reduce the side lobes of the spectrum.

Spectrogram: A spectrogram is a visual representation of the spectrum of frequencies in a signal as it varies with time. Here, the spectrogram is obtained by taking the magnitude (absolute value) of the STFT. This results in a 2D representation, where the x-axis represents time, the y-axis represents frequency, and the color represents the magnitude of frequencies at each point in time.

Mel Spectrogram: The Mel spectrogram is a version of the spectrogram where the frequencies are converted to the Mel scale. The Mel scale approximates the human ear's response to different frequencies. To do this, a Mel filter bank is calculated (a set of triangular filters) with the `linear_to_mel_weight_matrix` function. This filter bank uses 64 Mel bins, with a lower frequency of 125 Hz and an upper frequency of 7500 Hz. The filter bank is then applied to the spectrogram using a tensor dot product, which effectively maps the spectrogram to the Mel scale.

Log Mel Spectrogram: Finally, a log Mel spectrogram is computed by taking the logarithm of the Mel spectrogram. This is done to approximate the human ear's response to loudness (which is logarithmic, not linear). An offset (0.001) is added before taking the logarithm to avoid taking the logarithm of zero. This small value ensures that the logarithm is defined for all values.

3.1.4. Model Selection

The core of an audio classification problem is using an effective deep learning architecture. This section of the document explains the rationale behind the model selection.

The architecture of a deep learning model refers to the structure of interconnected layers, which include input and output layers, and nodes (also referred to as neurons). The connections between nodes are weighted and adjusted during training according to an objective function, which in this case is minimizing the difference between predicted scores and true scores. The mathematical parameters that define the nature of these connections are called 'weights', which are randomly initialized at the beginning of the training process.

Two custom architectures are trained from scratch. These architectures have been designed to achieve acceptable performance with the simplest possible solution. This is a good practice in machine learning workflows to understand the necessities of the problem and avoid over engineering. These custom architectures employ standard convolutional layers and depthwise separable convolutional layers.

Using existing working solutions is a common practice in machine learning. For this purpose, the MobileNetV1 architecture has been tested, an efficient deep learning architecture developed at Google primarily for edge and mobile vision applications. It uses depthwise separable convolutions to significantly reduce the model's parameters and computational cost while maintaining accuracy. Due to its smaller size and lower computational demands, MobileNetV1 is ideal for deployment on devices with limited resources or applications requiring low latency [34].

Transfer learning can enhance the training process. Instead of randomly initializing the weights of the model at the beginning of the training process, a pre-trained model with pre-defined weights can be used. These pre-trained models have been trained on extensive, high-quality datasets and are able to extract sophisticated high-level features.

Authors prove that even using transfer learning for audio classification with models pre-trained on image datasets is effective [11]. Nonetheless, the best-performing models on audio classification tasks using the ESC-50 benchmark, among others, are models pre-trained on large audio datasets. State-of-the-art models are pre-trained on the AudioSet dataset [11] [25] [26]. This dataset comprises 2,084,320 human-labeled 10-second audio snippets extracted from YouTube videos [35].

Access to large datasets such as AudioSet for model training remains a challenge, as high computational memory is required. For this purpose, transfer learning is tested using YAMNET, which uses the MobileNetV1 architecture trained on the AudioSet dataset. The YAMNET model has the capacity to predict 521 unique audio events [25].

3.1.5. Performance Evaluation

Performance is assessed according to prediction accuracy and computational complexity.

Prediction accuracy, measured as the ratio of correct predictions to the total number of predictions, is a simple and effective method to measure accuracy in balanced datasets. During the training phase, this accuracy is assessed on both the training and validation datasets. After training, the model's performance is evaluated on a separate test dataset with unseen audio fragments to measure its predictive accuracy.

To study the computational complexity reference is drawn from the annual DCASE competition. The objective of the competition is to create simple scene classification models that operate on various portable devices. Among the several rules set by the competition, the principal limitation pertains to the complexity of the model, capped at 128k parameters and a maximum of 30 MMACs (million multiply-accumulate operations) per inference [36]. While these constraints serve as a benchmark, the ultimate aim of this project is to develop a model that predicts accurately and performs efficiently on a mobile phone.

3.2. Inference

3.2.1. TensorFlow Lite

TensorFlow Lite is a lightweight machine learning solution designed by Google for mobile and edge devices [37]. It enables developers to run TensorFlow models on-device, providing a low-latency, privacy-preserving, and efficient solution for embedding machine learning capabilities into applications.

The workflow for TensorFlow Lite involves three main steps: creating a model in TensorFlow, converting the TensorFlow model into a TensorFlow Lite format using the TensorFlow Lite converter, and then deploying that model on a mobile or edge device.

The TensorFlow Lite converter reduces the size of the model and optimizes it for mobile and embedded devices by removing unnecessary information from the TensorFlow graph, slightly reducing precision, and using a more compact file format. The converter supports a variety of optimizations, including quantization, a technique that reduces the numerical precision of the model's weights and, optionally, activations. This process shrinks the model size and speeds up computation, typically without a significant impact on accuracy. There are various types of quantization available in TensorFlow Lite, including dynamic range quantization, full integer quantization, and float16 quantization.

3.2.2. Running Inference on Android

Running on-device inference follows four important stages. TensorFlow [38] provides a good description of the process in the documentation.

Loading the Model: First, the TensorFlow Lite model, which has been converted and optimized from its original TensorFlow version, is loaded into memory on the Android device. The model typically resides in the assets folder of the Android app and is loaded using the `MappedByteBuffer` utility in Android. This loading step consumes RAM, with the amount depending on the size of the model. A smaller model will require less RAM and load faster, providing a better user experience, particularly on lower-end devices.

Processing Audio Input: The Android device captures real-time audio from the microphone, which is then converted into a format suitable for the model. This step requires CPU resources and potentially additional RAM to temporarily store the audio data and its processed form.

Running Inference: The processed audio data is then fed into the TensorFlow Lite model for inference. This step also uses CPU resources and may use additional RAM to store the intermediate outputs of the layers of the model as the data propagates through the network.

Interpreting the Output: The output of the model, which is a vector with prediction scores over the possible classes, is then interpreted to yield the final classification result. This involves picking the class with the highest score or applying some other decision rule.

3.2.3. Android application

The Android application employed in this work is a prototype based on the following repository:

https://github.com/tensorflow/examples/tree/master/lite/examples/audio_classification/android

This Android project is a plug-and-play application that will load the TensorFlow Lite model in the assets folder and perform on-device inference. To run the application with a custom model view Annex VI.

3.3. Datasets

3.3.1. ESC-50 Dataset

The ESC-50 dataset is the primary benchmarking reference for this project. Using a benchmark supported by the literature is important to effectively evaluate and compare the models.

ESC-50 is an organized compilation of 2000 labeled environmental audio clips that serve as a suitable benchmark for environmental sound classification methods. This dataset comprises 50 semantic categories, each containing 40 examples, with every recording duration of 5 seconds. The domain of this dataset is comparable to the *Soundless* project. The four main categories of ESC-50 include natural soundscapes and water sounds, human non-speech sounds, interior and domestic sounds, and exterior and urban noises [10].

3.3.2 NBAC Dataset

The *Nocturnal Bedroom Audio Classification* (NBAC) dataset is currently in a beta phase, as it allows for potential expansions with the addition of more classes, as well as improvements in the variability of recording contexts.

Primarily developed for *Soundless*, the dataset represents a significant element in the proposed machine learning pipeline and demonstrates the project's potential.

The NBAC dataset currently comprises 13 distinct audio event classes that are typically experienced in a bedroom during night hours. The samples in class `trains` are recordings of trains that pass during nighttime in Tarragona directly collected by the *Soundless* team with their smartphones.

Data samples for each class have been acquired using a variety of mobile devices, including the Asus Zenphone 8, iPhone 13 PRO, and Xiaomi 13. All audio files are converted into wav format, with Pulse Code Modulation (PCM) format, featuring a mono channel with a 44100 Hz sample rate and a 16-bit sample size. The conversion was performed using AVS Audio Converter 10.3.

For ease of identification and classification, the recorded audio files are segmented into 5-second intervals and stored in subfolders corresponding to their labels. While the optimal

recording environment is within a bedroom, some traffic-related recordings have been captured outside this setting for diversity.

To enhance the dataset's comprehensiveness, specific audio samples have been borrowed from the esc-50 dataset, as denoted by the -esc50 extension in the file names. Similarly, files with the -synthetic extension represent audio fragments generated from recordings of digitally produced sounds.

4. Experimental Setup: Model Development

This chapter of the document explains how the controlled experiments were set up. The experiments focus on developing and testing the models for the task, performing audio classification on both datasets, ESC-50 and NBAC. The experiments are classified in three main categories:

- Architectures with Mel Spectrogram as input (4.1)
- Compact architectures (4.2)
- Transfer learning with YAMNET (4.3)

Each experiment is conducted using two audio configurations: 5-second-long audios (al5s) and 2-second-long audios (al2s).

4.1. Architectures with Mel Spectrogram as Input

Refer to Annex II for the notebook with the code of this experiment. This part of the experiment explores simple deep-learning architectures built from scratch. The aim is to understand the complexity of the problem. The simplicity of this solution is defined by two factors:

i) The LLF block is separated from the HLF block: This deep learning model will only accept the Mel Spectrogram as input, with mono channel images of size (498, 64, 1) for al5s configuration, and (198, 64, 1) for al2s configuration. This shape is determined by the sampling configuration of the Short-Time Fourier Transform, where each frame covers 10ms of the audio (refer to section 3.1.3). The input shape of both audio configurations is consistent across all experiments in this project.

ii) Its structure only uses three convolutional layers: The Mel spectrograms are generated in a previous step, using the custom generator explained in section 3.1.3. Writing the python script to generate the spectrograms is relatively simple and different libraries can be employed. A function to generate the Mel spectrogram is applied to the waveform data before entering the model. Using this previous step also allows the integration of two data augmentations processes. Waveforms are augmented before generating the Mel spectrogram, and spectrograms are augmented before entering the HLF (refer to Figure 10).

The spectrograms are then fed to the HLF to begin the training process. The HLF structure of this chapter (displayed in Figure 13) is evaluated using normal convolutions and depthwise separable convolutions.

After the input layer, there are three convolutional layers that will map the high-level features of the spectrogram. These convolutional layers are responsible for mapping and detecting the high-level features present in the spectrogram. Each node in these layers utilizes a rectified linear unit (ReLU) as its activation function, which determines the output of the node based on its input.

After these convolutional layers, a process called batch normalization is implemented. This step aims to accelerate the training process and enhance its stability. It does so by normalizing the output of each layer, adjusting it to have a mean of zero and a standard deviation of one, thereby standardizing the outputs across the layers.

Following batch normalization, the network employs a 2D max pooling layer. This layer serves to downsample or reduce the dimensionality of the feature maps produced by the convolutional layers. This layer saves valuable computer memory without losing significant feature information.

After the HLF, an additional sequential layer will generate the embeddings. This layer uses one convolution to map the features with 256 filters. After this convolution, a Dropout layer will randomly turn off 20% of the neurons during training to prevent overfitting. In the end, a Global Average Pooling layer reduces each feature map to a single average value, thereby minimizing computational cost and further avoiding overfitting. This block outputs a vector representation of the features of shape 256.

After the Embeddings block, the classifier is connected. The classifier is composed of a Dense layer and an Activation layer. The Dense layer, with a size equal to the number of classes to predict, is a fully connected layer where each output node represents a potential class for classification. The Activation layer applies the `softmax` function to the Dense layer's output, converting raw scores into probabilities that sum to 1, thus facilitating the interpretation of the output as the probability distribution over the classes. This setup is particularly useful in multi-class classification tasks, as it allows the model to predict the most probable class for a given input.

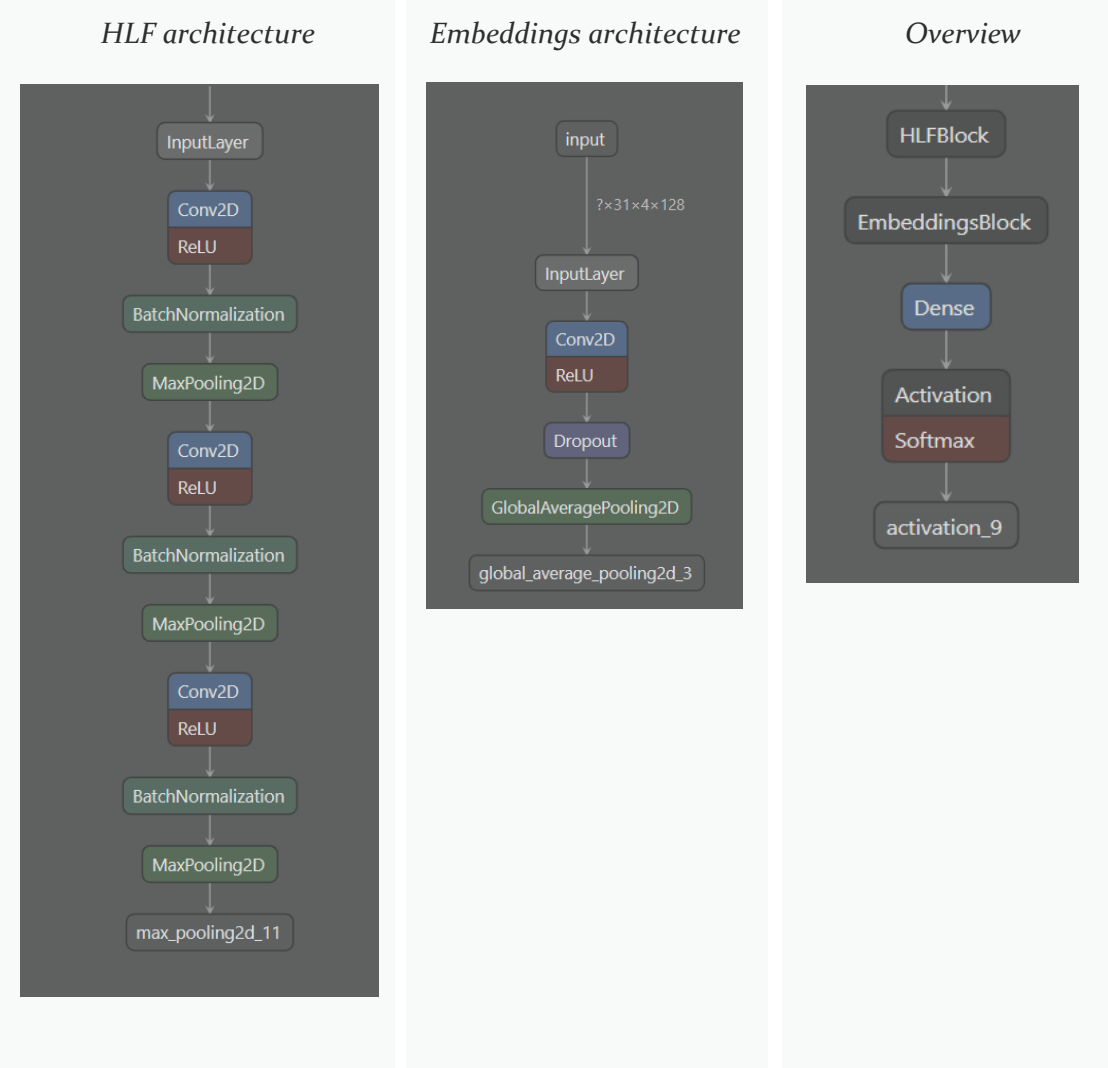

Figure 13: Standard CNN architecture

4.2. Compact Architectures

One of the main constraints in developing machine learning solutions for mobile devices is the difficulty of on-device data pre-processing. Generating the Mel Spectrograms on-device is a challenging task, considering the high number of dependencies and complex mathematical operations. It was quickly found that building an Android app that leverages the solution proposed in section 4.1 is not possible.

To tackle this issue, compact architectures are designed (the notebook with the code of this experiment is found in Annex III). These architectures integrate the LLF block within the deep learning model. A layer is built to process the input waveform inside the network and generate the Mel Spectrogram (view Figure 13). Then the spectrogram is fed to the HLF block to extract the embeddings and make the predictions.

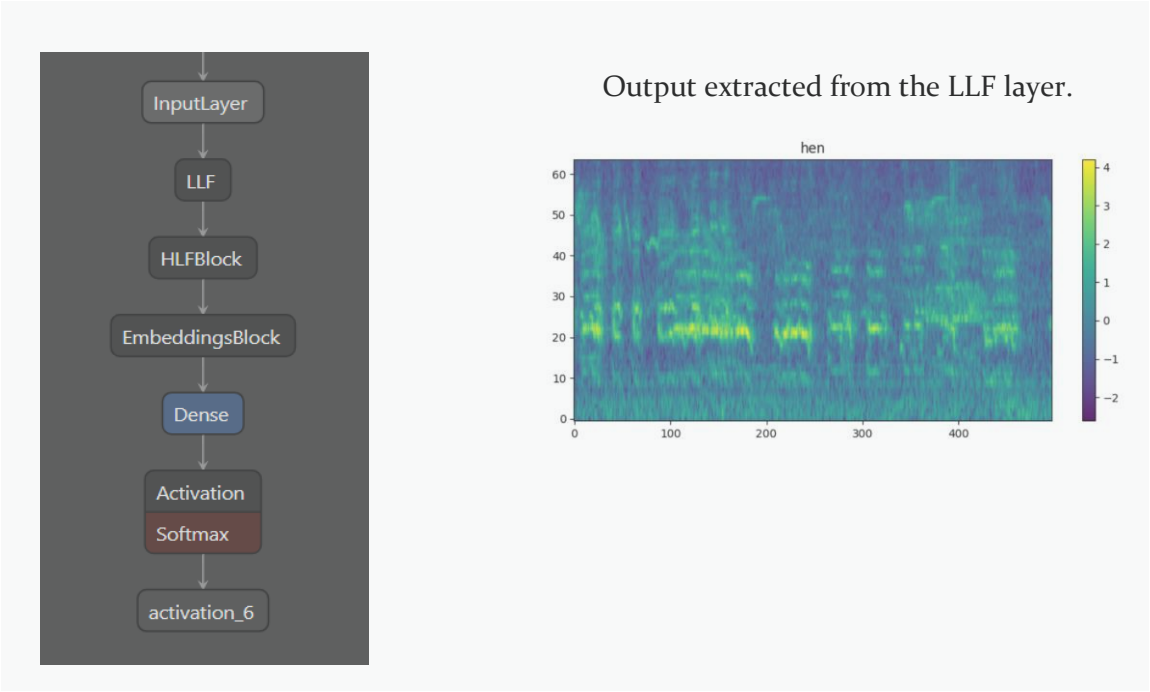

Figure 14: Output extracted from the LLF layer

The main drawback of this solution is that data augmentation cannot be applied to the spectrograms, therefore data augmentation is only applied directly to the waveforms.

4.2.1. Custom Architectures

This experiment involves using the same architectures as the ones explored in section 4.1. The LLF block is integrated into the model, allowing it to accept raw waveforms as input. As discussed in section 5, this modification does not seem to affect performance.

Although the number of trainable parameters in the model will remain the same, this model might increase latency, as an increased number of operations occur inside the network.

4.2.2. MobileNetV1

MobileNetV1 [34] is a type of efficient model specifically designed for mobile and embedded vision applications. It's built on a streamlined architecture that employs depth-wise separable convolutions.

Two key elements of this architecture are the model shrinking hyperparameters: the width multiplier and the resolution multiplier. The width multiplier uniformly reduces the

thickness of the network at each layer, while the resolution multiplier downsizes both the input image and the internal representation at every layer.

Originally, MobileNetV1 was designed for computer vision tasks, processing RGB images with an input shape of (224, 224, 3). The model can be configured to accept different input shapes as long as three input channels are maintained. In this experiment, MobileNetV1 has been adapted to process spectrograms of audio fragments. These spectrograms have shapes of either (498, 64, 3) for 5-second audio fragments or (198, 64, 3) for 2-second audio fragments. This was accomplished by replicating the single channel of the spectrogram three times.

The MobileNetV1 architecture replaces both the HLF and the Embeddings block. The LLF layer is modified to generate a 3-channel spectrogram that is fed to the MobileNetV1 block.

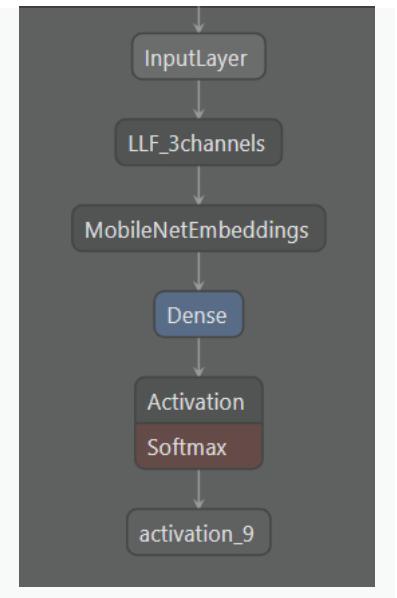

Figure 15: Compact architecture with MobileNetV1

MobileNetV1, compared to the previously discussed models, is a significantly more complex model. Training such large models from scratch can be challenging, often requiring extensive datasets and prolonged training periods. In this case, the model was trained in two ways: from scratch with random weight initialization, and through transfer learning using weights pre-trained on ImageNet.

ImageNet is a vast visual database, specifically designed for computer vision research [39]. It contains over 14 million images, each hand-annotated to denote the objects they depict.

Research indicates that transfer learning from images to spectrograms is effective, as the feature maps generated for successful image recognition can also be beneficial for spectral images [11].

To optimize performance during the transfer learning process, certain adjustments were made. Specifically, the first 20 layers of the MobileNetV1 architecture were frozen, retaining the ImageNet weights. This allowed the final layers to be tuned during training on the custom dataset.

4.3. Transfer Learning with YAMNet

YAMNet [20] is a model developed by Google that trains a model based on the MobileNetV1 architecture on AudioSet. This dataset has over 2 million 10 second audio fragments and covers 632 classes. The classes differ in number of samples and label quality; therefore, it is a highly unbalanced dataset. YAMNet trains on 521 hand-picked classes, demonstrating a good ability to distinguish between different audio classes, as indicated by a balanced average d-prime of 2.318. Testing the model out of the box will provide impressive results.

However, its precision in detecting audio events (balanced mean average precision `mAP` of 0.306) and its ability to correctly rank labels (balanced average label weighted label ranking average `lwlrap` of 0.393) was moderate. This suggests that while the model is fairly effective at classifying audio, it is far from perfect, particularly in its precision and label ranking abilities.

The key distinctions between AudioSet and ESC-50 involve the number of classes and the total duration of audio per class. AudioSet contains a wide range of classes and a longer duration of audio per class. In contrast, ESC-50 consists of 50 classes, each with a total of 200 seconds of audio. Models developed using the AudioSet dataset can effectively learn and extrapolate general features from the comprehensive dataset, which can be utilized to resolve simpler tasks. This approach was successfully employed by Lopez-Meyer et. al [26], whose AemNet model, initially trained on AudioSet, achieved a 92.32% performance rate on the ESC-50 dataset through the application of transfer learning. With fewer convolutional layers than YAMNet and a trainable LLF block, AemNet's architecture is quite similar.

4.3.1. Understanding YAMNet

The experiment described in this section aims to utilize the high-level feature extraction capabilities of YAMNet to categorize the audio events in the ESC-50 and NBAC datasets. More specifically, the model in this experiment will extract 1024-D embeddings from the raw waveforms, that are later fed to a classifier layer. Find the details of the experiment in Annex IV.

As outlined in Section 3.1.1, the primary goal of this project is to determine the ideal audio duration setup using classifiers for two separate audio durations: 2 seconds and 5 seconds. Both short-term and long-term features present unique pros and cons. To balance these, YAMNet employs a hybrid approach, where the model is trained on audio segments of 0.96 seconds. These windows are then slid across the audio chunks with 50% overlap, equivalent to a 0.48-second hop. Every window generates a vector with prediction scores. As indicated in the documentation [40], these scores are aggregated using the mean average to enable ranking.

YAMNet uses a 16000 Hz sampling frequency. The expected waveform length is 15600, or 0.975 seconds. The model will not accept shorter audio fragments.

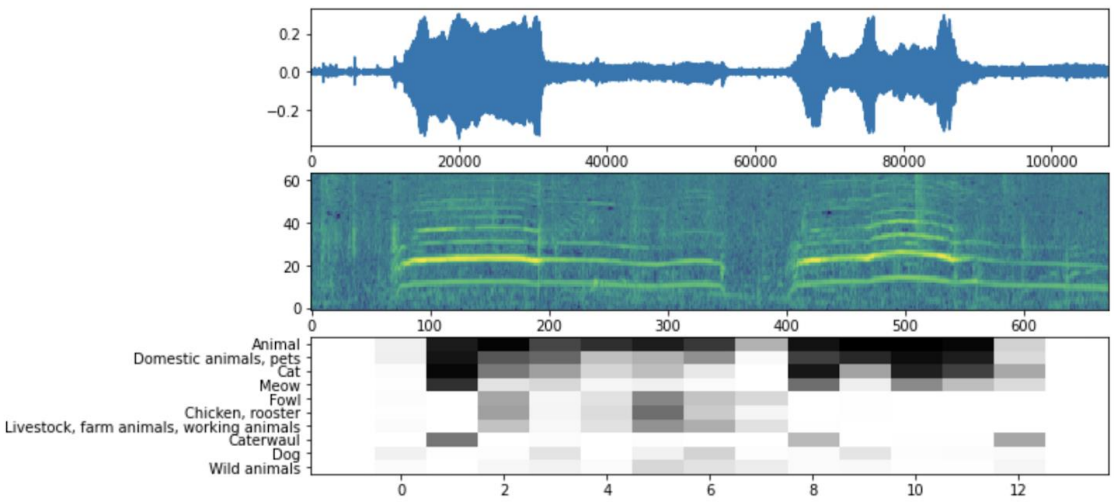

Figure 16: YAMNet outputs. Extracted from [41]

As YAMNET is a model built for deployment, it poses challenges when attempting to modify its internal structure. For the purpose of testing, YAMNet has been employed in a previous step outside the deep learning model to extract 10-dimensional embeddings of

size 1024 from the 5-second-long audios. These embeddings are then used as independent segments, with the classifier being trained to categorize audios that are 0.96 seconds long. After just one epoch, the classifier already exhibits over 60% accuracy on the validation dataset. The model achieves an accuracy of 63% on the test dataset as a 0.96-seconds audio length classifier.

In order to emulate the prediction capabilities of YAMNET, the model is utilized to predict the label scores for the 0.96-second fragments of the test dataset. These are subsequently aggregated for each 5-second audio fragment using the mean average. The label achieving the highest score is selected to calculate accuracy. This prediction processing technique has not been applied to the al2s configuration, as the comparison between predictions on 0.96 second windows and 5 second windows will serve as a reliable reference.

4.3.2. Native Customization of YAMNet

The experiment proposed in section 4.3.1 uses a workaround to understand how the YAMNet model can be used to build a custom audio classification model. Because YAMNet is designed for deployment purposes, TensorFlow Lite offers a native solution to customize YAMNet with TensorFlow Lite Model Maker [42].

It was found that using this library requires a UNIX server, and it might generate some errors when running on online platforms such as Kaggle or Google Colab. For this matter, a machine-learning application has been developed to run TensorFlow Lite Model Maker locally and easily train the model with the preferred dataset (view Annex IV). This machine-learning application has been developed and tested in WSL2.

The advantages of using TensorFlow Lite Model Maker are:

i) It directly customizes a deployment-ready model: Previous experiments require converting the model to TensorFlow Lite, and require architecture adjustments to be deployed on an Android app. The model produced with this ML application can be run directly into the Android app described in 4.3.

ii) It allows the model to train on the entire sample: The workaround proposed in 6.3.1 forces the training to frame audios in 0.96 seconds fragments separately, making the model blind to the true predictions of the audio label for each 5-second long audio sample. TensorFlow Lite Model Maker allows the configuration of the

frame length[1] for training, handling the entire 5-second audios. As suggested in the documentation, it is best to frame each audio by a multiple of the expected waveform length, so the true frame length is 4.875 seconds.

[1] The term frame length can be ambiguous. While the model internally frames the audio into 0.96 seconds, it can create a longer input frame that will handle the predictions internally in a similar way as described in 6.3.1

5. Validation

This section of the work performs the necessary validation steps to check the success of the solution. Section 5.1 outlines the performance results, considering accuracy and computational complexity. Section 5.2 provides an overview of the final product, including a review of the working prototype and its limitations.

5.1. Analysis of the Results and Evaluation

This section outlines the results of the experiments and provides a general evaluation. It is important to note that these results are not statistically proven, meaning that executing the experiments in a different run could yield slightly different results. These results are computed with the average accuracy of 3 runs. The results provide a general reference to understand and compare the models in the context of this project.

The results collect the prediction accuracy on the unseen test dataset of every model after training for 50 epochs with a batch size of 32 for al5s configurations, and 128 for al2s configurations.

It is demonstrated that prediction accuracy is better for longer frames, as training on 5-second-long fragments results in considerably better accuracy in most of the experiments.

As expected, using depthwise separable convolutions is the preferred option. Depthwise separable convolutions generally offer the best results with a decrease of number of parameters of over 50%.

Table 1: Results for standard CNN and DW architectures - 4.1.

	ESC-50				NBAC			
	al5s		al2s		al5s		al2s	
	CNN	**DW**	CNN	DW	CNN	**DW**	CNN	DW
Accuracy	0.635	**0.695**	0.610	0.570	0.755	**0.866**	0.769	0.814
Parameters	139442	**58139**	139826	58523	129676	**48373**	129676	48373

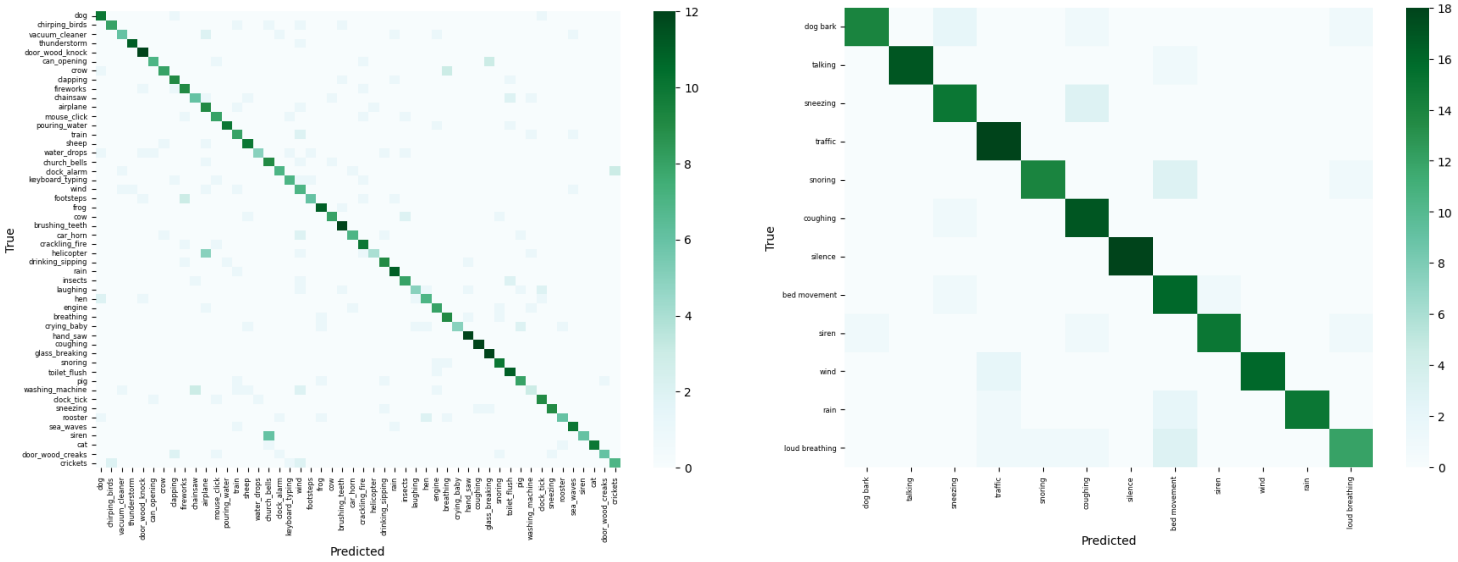

Figure 17: Confusion matrices on test dataset of the DW al5s architecture

Table 2: Results for standard and compact CNN and DW architectures - 4.2.1.

	ESC-50				NBAC			
	al5s		al2s		al5s		al2s	
	CNN	DW	CNN	DW	**CNN**	DW	CNN	DW
Accuracy	**0.670**	0.655	0.569	0.581	**0.866**	0.824	0.763	0.794
Parameters	**139442**	58139	139442	58139	**129676**	48373	129676	48373

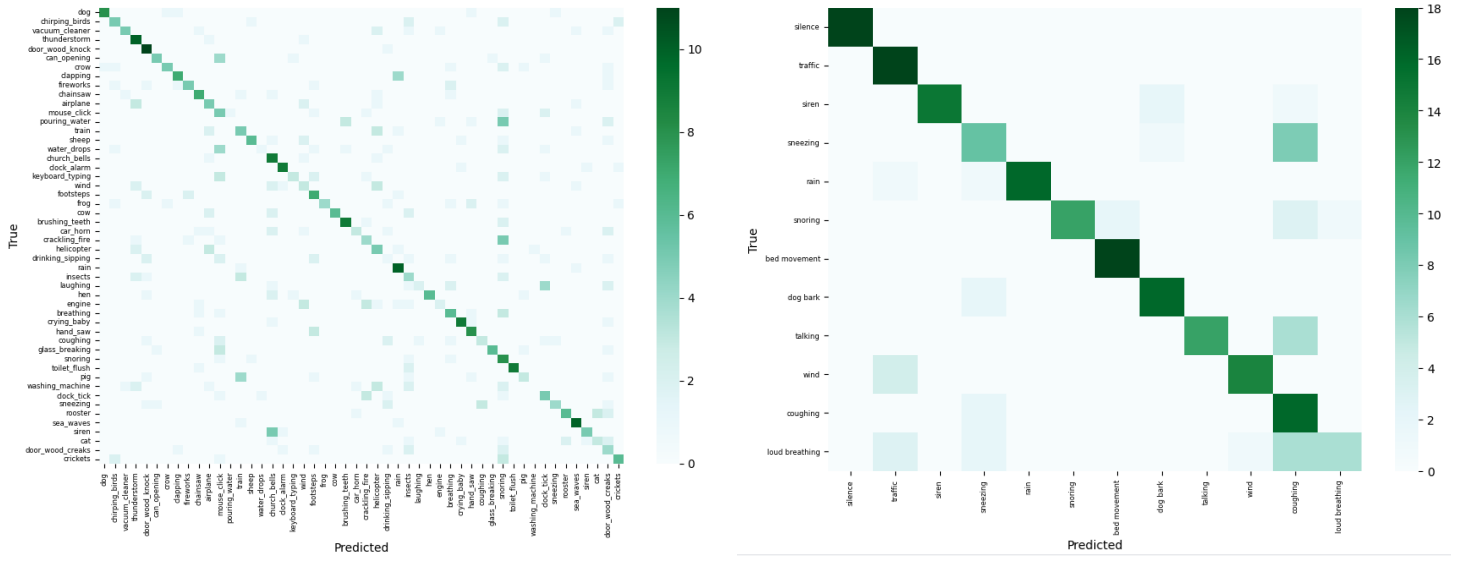

Figure 18: Confusion matrices on test dataset of the compact DW al5s architecture

Using the MobileNetV1 architecture pre-trained on ImageNet resulted in a prediction accuracy of 84.7% on the ESC-50 dataset, near to state-of-the art results.

Table 2: Results for compact architectures with MobileNetV1 - 4.2.2.

	ESC-50				NBAC			
	al5s		al2s		al5s		al2s	
	Random	**ImageNet**	Random	ImageNet	Random	**ImageNet**	Random	ImageNet
Accuracy	0.511	**0.847**	0.544	0.637	0.653	**0.912**	0.796	0.840
Parameters	3280114	**3280114**	3280114	3280114	3241164	**3241164**	3241164	3241164

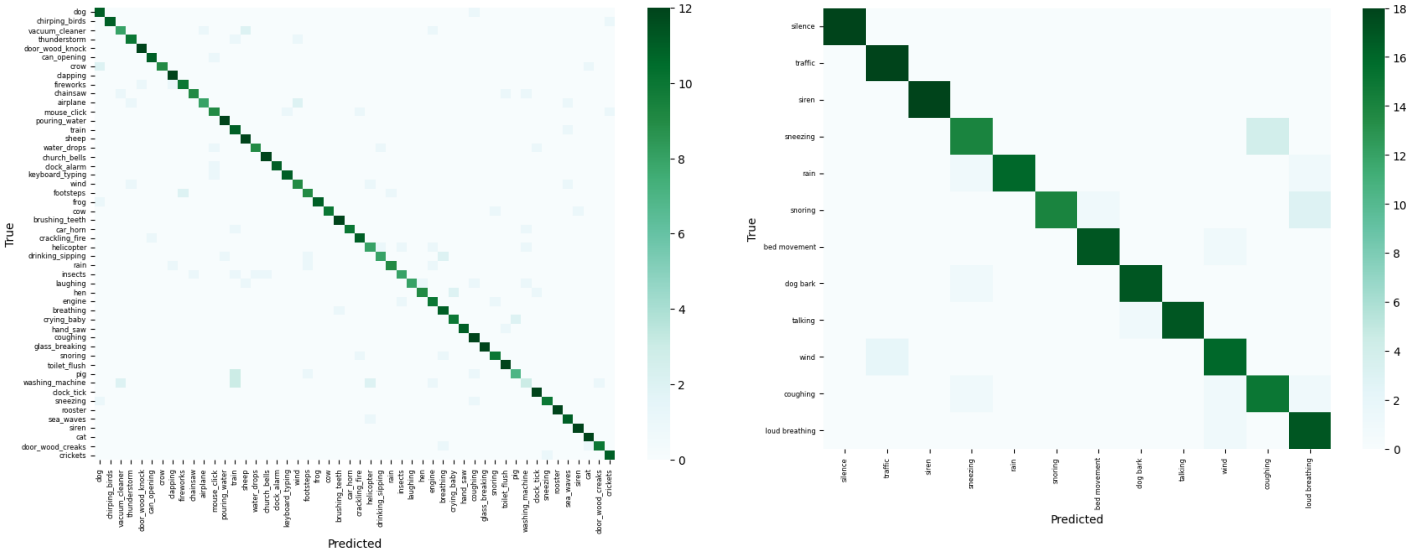

Figure 19: Confusion matrices on test dataset of the pre-trained MobileNetV1 al5s architecture

With native training on the YAMNet model, prediction accuracy increases to 86.5% on the ESC-50 dataset. The accuracy on the NBAC dataset increases up to 95%. Note that these results are only for the al5s configuration.

Table 3: Results for transfer learning with YAMNet - 4.3.

	ESC-50		NBAC	
	Embeddings	**Native**	**Embeddings**	Native
Accuracy	0.825	**0.865**	**0.949**	0.925
Parameters	51250*	**51250***	**10250***	10250*

Note that the parameter evaluation for YAMNet only takes into account the parameters within the classifier. The YAMNet architecture has a number of non-trainable parameters similar to the MobileNetV1 architecture (4.2.2).

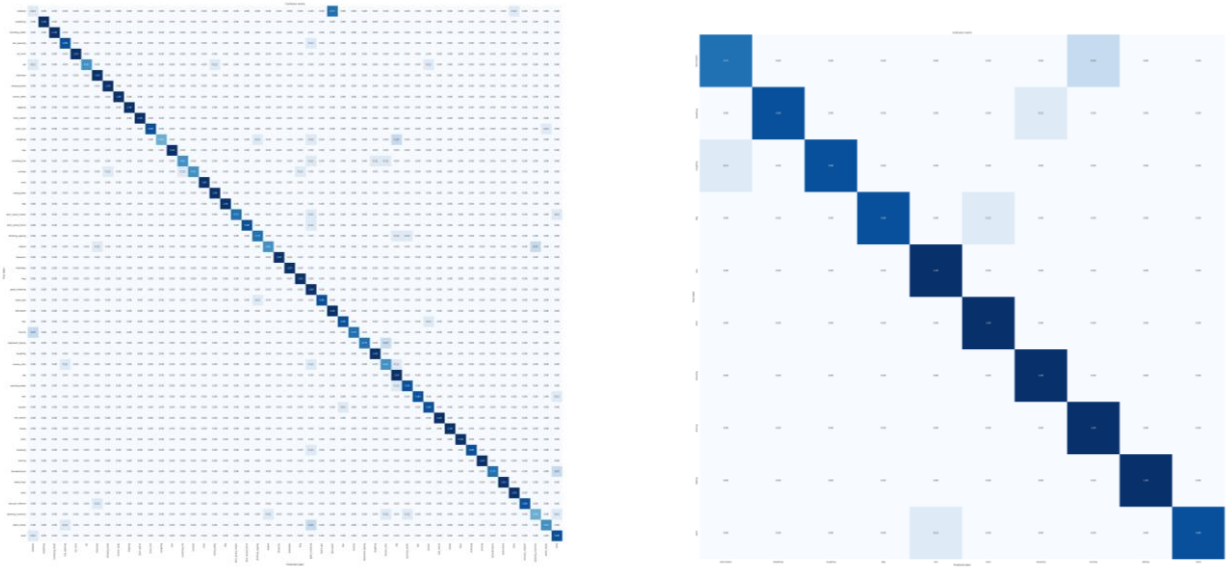

Figure 20: Confusion matrices on test dataset of the YAMNet al5s architecture

Training on the NBAC dataset resulted in high prediction accuracies in most of the models. Pre-trained models achieve an accuracy of over 90%. These results show the predictive capabilities of deep learning models on custom datasets, highlighting the viability of the proposed feature in the *Soundless* project.

5.1.1. Analysis of Minimum Class Sample Size for Convergence

The datasets utilized in these experiments come with a predetermined number of samples per class. To understand if the limitations observed in the derived model arise from the intrinsic nature of the problem or from insufficient data, an analysis of class sample size is undertaken to explore the convergence of performance (View Annex V). This form of analysis is unique to each model architecture. The performance is evaluated on the original test dataset, which comprises 20% of the original datasets: 8 unseen samples per class for ESC-50 and 12 unseen samples per class for NBAC. The results highlight that complex architectures necessitate large volumes of data for successful training, while lighter architectures yield satisfactory outcomes with training only on the available dataset.

This divides effective architectures into two categories: light weight architectures trained from scratch, and encoder-classifier architectures, which use of a pre-trained model with a more complex architecture.

Light weight architectures (refer to section 4.2.1) are more sensitive to class sample size, as the model is learning internal features of the data from scratch.

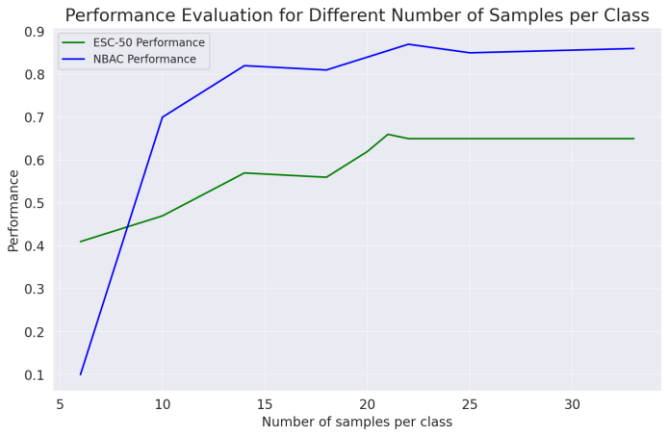

Figure 21: Analysis of performance for the standard DW architecture as a function of number of samples per class

Encoder-classifier architectures, the pre-trained MobileNetV1 architecture in this case, provide effective accuracies with just a few training samples per class, as only the last layers in the classifier need to adapt to the current dataset.

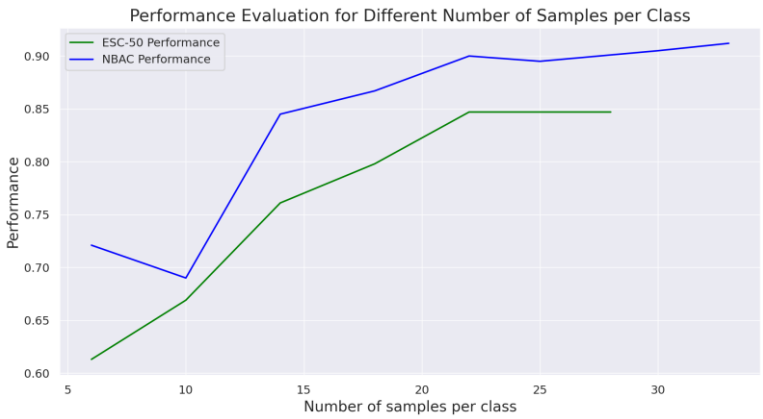

Figure 22: Analysis of performance for pre-trained MobileNetV1 as a function of number of samples per class

The analysis proves the advantages of using pre-trained architectures. The internal representations can be utilized out-of-the box, with accuracies between 60 and 75% on the test dataset with just 6 samples per class. For both datasets, performance converges at 20 samples per class.

5.2. Working Prototype

After the model is constructed, it is ready to be deployed for testing in the Android App. Note that the model needs to be converted to TensorFlow Lite for deployment as detailed in section 3.2. This prototype uses the YAMNet model natively trained on TensorFlow Lite (6.3.2). Find the instructions to run this prototype in Annex VI.

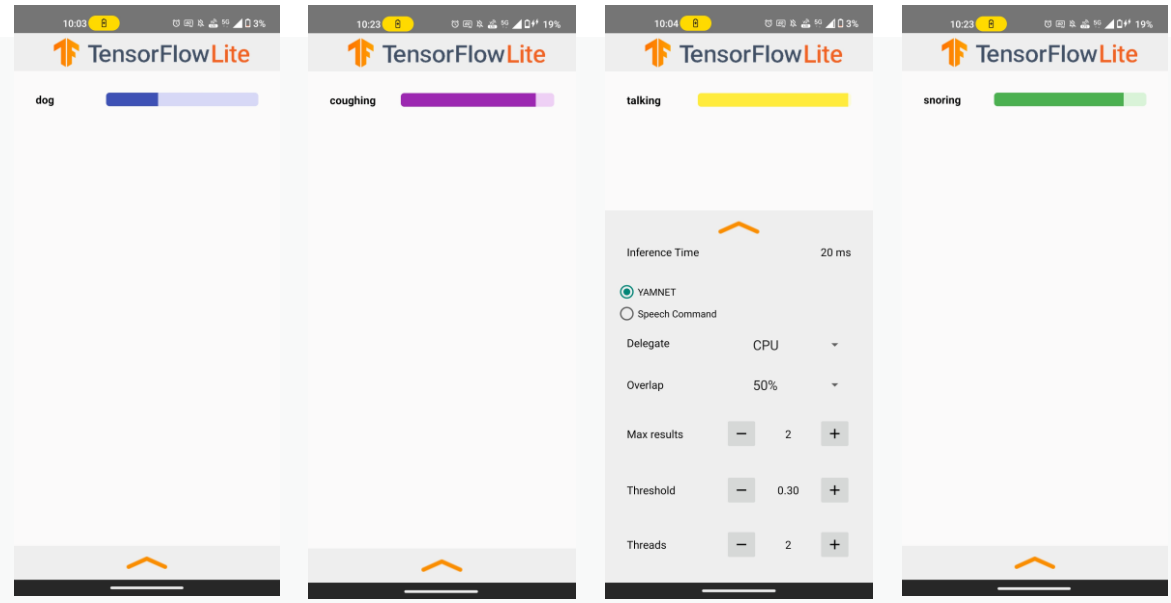

Figure 23: Screenshots of the working prototype

The models developed in section 4 are designed to return a scores distribution for all classes during inference. After on-device inference, the UI handles the scores distribution and outputs the results.

The application is designed to execute inference on audio fragments spanning 5 seconds, with an overlap of 50%. This indicates that every 2.5 seconds, the application produces a prediction based on the last 5 seconds of audio. The screenshot illustrates that the

inference time typically hovers around 20 milliseconds, suggesting that it would be possible to raise the overlapping parameter to achieve denser prediction results.

Moreover, this prototype offers some customization options pertaining to the user interface and the thread count. A crucial aspect of these settings is the threshold parameter. It's common for the model to yield predictions with relatively low scores. However, by setting a threshold value, it is possible to eliminate these lower scoring predictions, allowing the user to only see predictions that score highly. A bar accompanying each prediction visually represents its score. It is challenging to pinpoint a minimum score that guarantees high-quality results, but generally, scores below 0.7 tend not to be highly reliable.

The prototype proves to meet the requirements specified by the *Soundless* team. As outline in section 5.1 the performance accuracies prove successful results, while the prototype works as expected and is ready to be integrated in the *Soundless* app.

5.2.1. Prototype Limitations

This study has so far discussed the use of a centralized multiclass prediction model. Centralized meaning that the model is locally trained on a server and later deployed onto edge devices. There are certain limitations in this design that demand careful examination. These limitations refer to i) how the multiclass prediction model handles unknown sounds and ii) how the centralized design of the model adapts to the expected acoustic context variability of participants.

i) During the inference process, the model assigns a score distribution for each class. For unknown sounds, the model generates a distribution with low scores, signifying that the model is not confident that the sound event belongs to any of the known labels. Hence, in the event of an unfamiliar yet significant sound, the model is unable to provide valuable insights.

ii) The use of a centralized model carries a significant restriction. This model undergoes local training on a server using a specific dataset. After deployment, the acoustic context is likely to shift and diverge from that of the training dataset.

Section 7 delves into these issues and suggests potential solutions. For instance, leveraging the learned feature representations from a centralized model proves beneficial in managing unfamiliar sound events. Moreover, deploying the model and conducting distributed learning across clients through Federated Learning offers a novel approach to avoid the limitations of centralized Deep Learning.

6. Feature Space Analysis

The crafted NBAC dataset aims to cover all the possible relevant sounds that could occur during nighttime within a bedroom setting. It is certain that prior to deployment, the NBAC dataset will have to be refined with additional data, adding more classes and more audio recordings in different acoustic contexts. Nonetheless, there will always exist the possibility of encountering new sounds that the model is not able to predict.

The simplest implementation solution is using the model to generate a class scores distribution. To manage unknown sounds, the score threshold can be adjusted to ignore labels with low scores. This solution will only tag audio segments that the model knows with confidence, leaving unknown or unclear sounds without a label tag.

However, the internal feature representations that the model has learnt, which can be extracted from the Embeddings block, provide useful information even for audio events of unknown class. Instead of using the model to predict the label scores, it can be used to output the embeddings and carry a feature analysis further. This can be useful to understand audio classification within a feature space instead of with label scores.

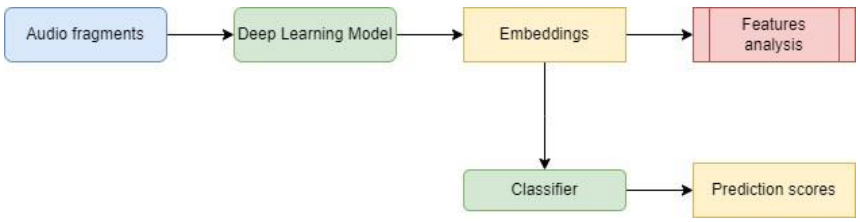

Figure 24: Features analysis diagram

Audio embeddings extracted from pre-trained networks hold significant utility in audio-related tasks due to their ability to encapsulate simple feature representations from complex data [44]. A comparative evaluation against models trained from scratch provides further evidence of the efficacy of pre-trained embeddings, which demonstrate superior performance, particularly when training data is limited.

6.1. Encoder

The features of a sound event are abstract representations of it, encoding the information of the entire clip on a low dimensional variable. The feature extractor is typically referred to as the encoder. For the models developed in section 4, the encoder is the concatenation of the LLF block, the HLF block and the Embeddings block. Depending on the architecture, this feature representation has sizes of 256 or 1024. The only requirement to construct a fully functional multiclass prediction model is to build a classifier on top of the encoder with a single dense layer. All the analysis performed in this section can be found in Annex VII.

Creating a good encoder is the principal task in feature space analysis. The encoder will learn to extract meaningful representations of the data for a specified objective. In the experiments proposed in this project, the objective is to correctly classify each audio fragment according to its label. The embeddings that are fed to the classifier in the last layers are the learned representations. When using pre-trained networks, with ImageNet or AudioSet, the encoder is already built out of the box, as only the classifier is trained on top of it to make the predictions.

Other options exist to create encoders in different contexts. For example, for unsupervised learning, the encoder can be trained to identify audios with the same label by minimizing the difference between learned representations [45]. Similar approaches can be used for few-shot learning as well, where the encoder can be trained to minimize the intra-class differences and maximize the inter-class differences in the learned representations. This means that the encoder tries to make the representations of samples from the same class as similar as possible, while making the representations of samples from different classes as different as possible. By focusing on the differences between classes, the encoder can learn a representation that captures the essential characteristics of each class, even when only a few examples are available [46].

The feature representations produced by the encoder should be useful to find similar audio fragments. By comparing the distance between feature representations, neighboring audio fragments are found.

The encoder is pre-trained on an initial dataset with the aim of maximizing the internal representation identification capacities of the model. Using the encoder with the MobileNetV1 architecture trained on ImageNet the features have been extracted from all audio fragments of the NBAC dataset. In this case, cosine similarity has been used to

compute distance. Figure 25 shows a random sample of 4 audio fragments and their 5 closest neighbors.

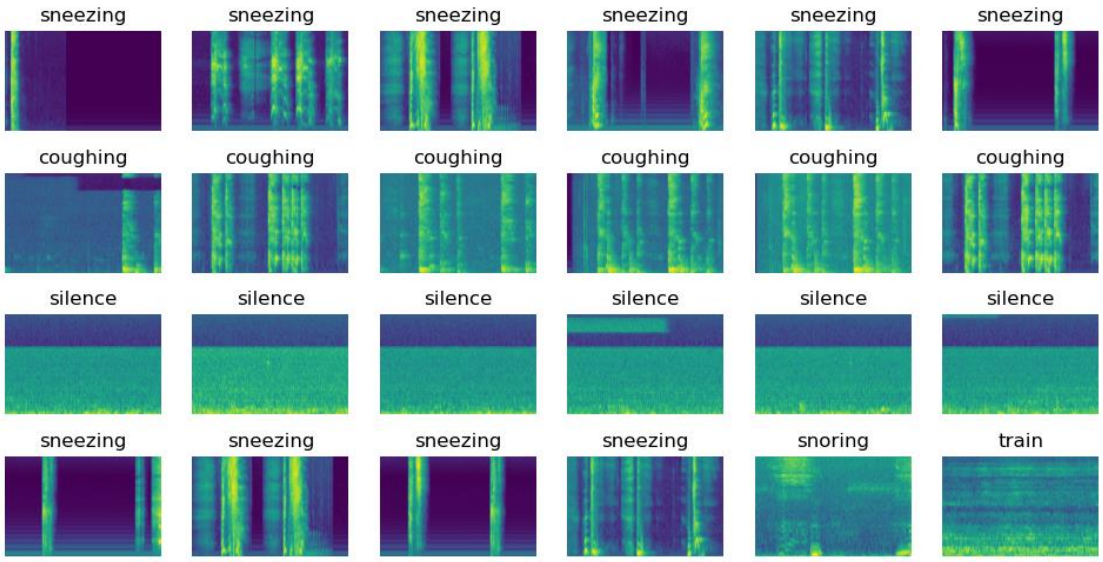

Figure 25: Closest neighbors within feature space

6.2. Semantic Clustering

The objective of this section is to perform unsupervised classification on the NBAC Dataset. While the models trained in section 4 use the labels to learn the internal representations and predict the label (supervised learning), this section classifies the audio samples strictly based on their feature similarity.

The aim is to prove the utility of the feature representations of the audio fragments without a standard classifier. A clustering model has been developed using neural networks in TensorFlow based on the documentation provided by Keras [45]. It utilizes the encoder to produce vector representations of input data, which are then clustered based on their similarities. Two custom loss functions are used: *ClustersConsistencyLoss* ensures that similar inputs yield similar outputs, while *ClustersEntropyLoss* balances the distribution of data across clusters. The script creates two models: one for clustering data, and another for calculating similarity between each data point and its k-nearest neighbors based on the clustering model. If specified, the encoder weights can be frozen during training to preserve its feature extraction capability. Finally, the models are compiled with the custom loss functions and an *AdamW* optimizer, then trained over several epochs. The

goal is to create a model that effectively groups similar data and evenly distributes data across clusters.

In this experiment, 13 clusters are defined with the aim of obtaining one clearly defined cluster for each class. Figure 26 visualizes the clusters assigned to each sample per class. The visualizations highlight high confidence predictions with a confidence threshold > 0.8.

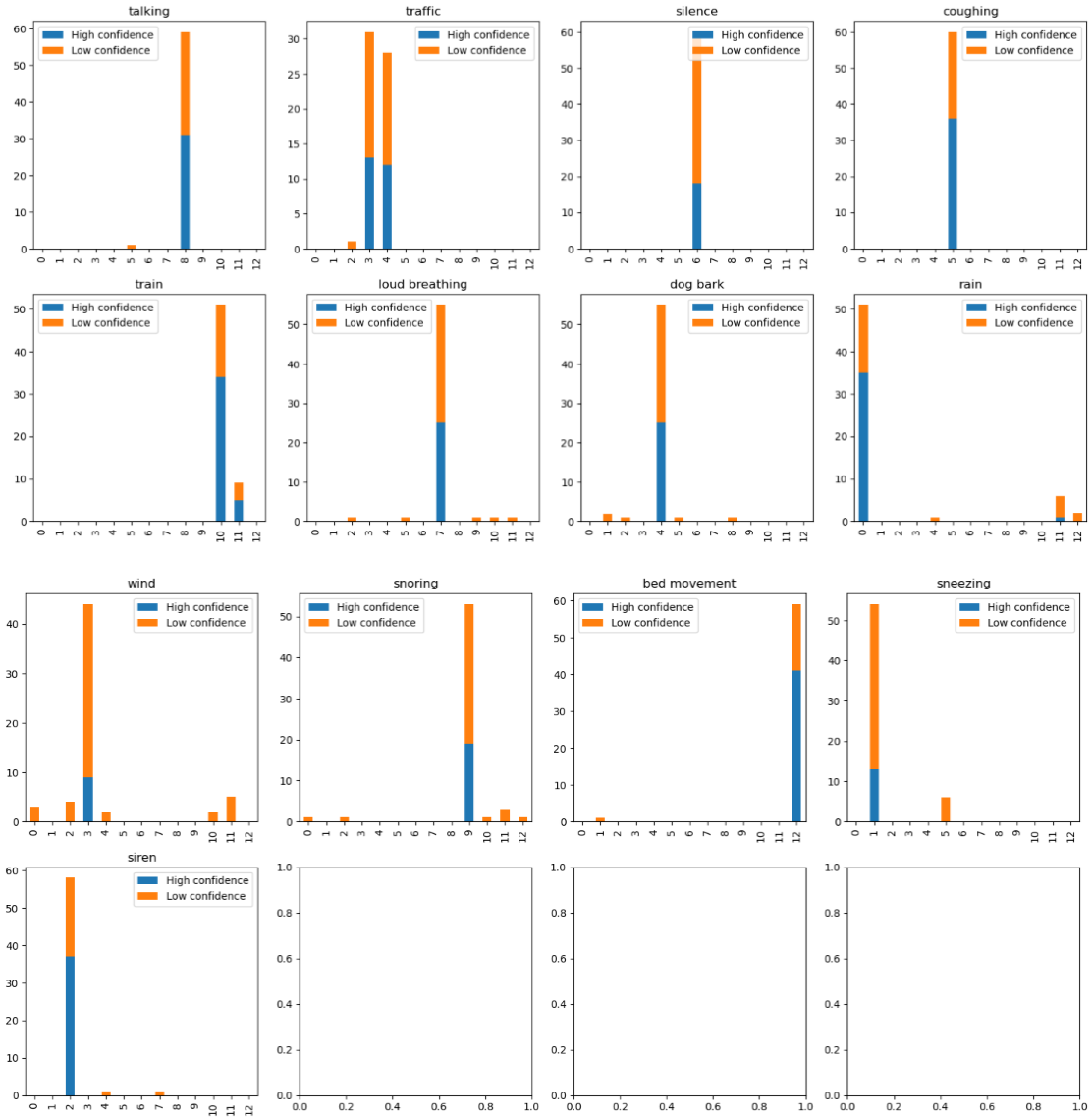

Figure 26: Cluster distribution across audio classes

The results show a classification accuracy of 89%. The model had problems classifying traffic sounds, separated across clusters 3 and 4, mixed with wind and dog bark sounds. Figure 27 visualizes the class distribution within clusters.

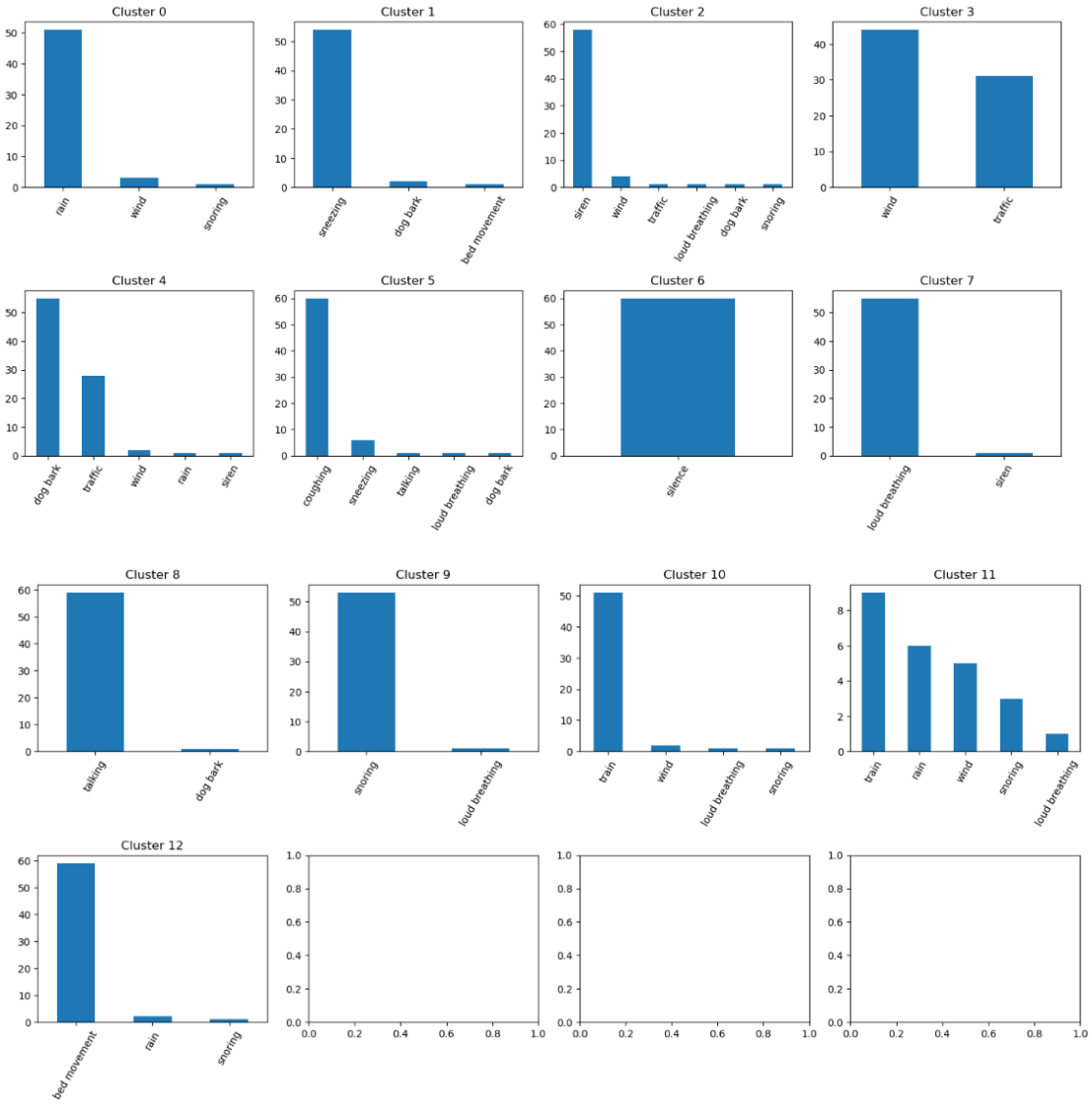

Figure 27: Class distribution within clusters

Clusters with expansive class distributions show clear feature groups. Cluster 3, for instance, contains wind and traffic sounds, which can be understood as a cluster containing grainy, disperse sounds with low frequencies. Cluster 11, which shows the highest dispersion, includes train, rain, wind, snoring and loud breathing sounds. The accuracy of the model can be improved, but the results prove the potential of this type of analysis.

The clusters defined by the feature representations can be visualized with different methods. Figure 28 shows a simple visualization using *T-distributed Stochastic Neighbor Embedding*, a visualization technique used for high-dimensional data. It works by transforming data similarities into joint probabilities and aims to reduce the *Kullback-Leibler* divergence between the low-dimensional embedding (the components) and the high-dimensional data (the feature representations) [47]. Other techniques, such as PCA (Figure 29), can also be useful to create similar representations.

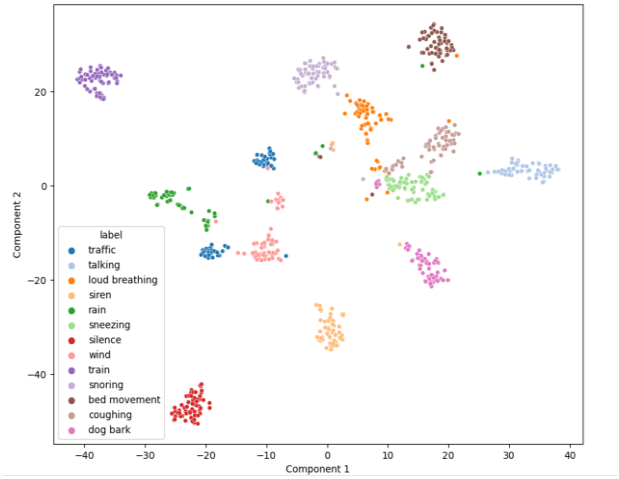

Figure 28: t-SNE cluster visualization

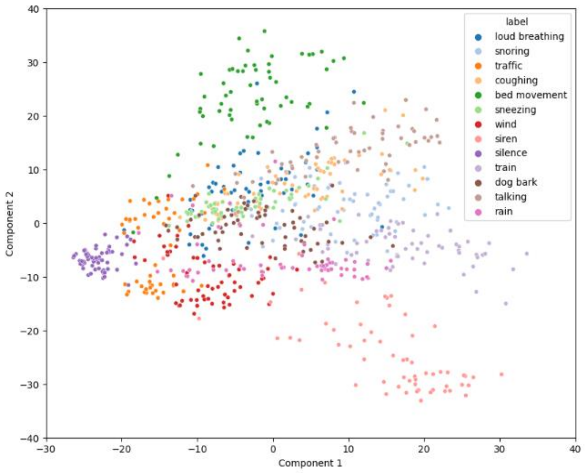

Figure 29: PCA cluster visualization

When dealing with unknown sounds, the feature representation of the new event will fall somewhere in this feature space. This analysis can yield useful information other than a single label. This experiment uses recordings from a garbage truck operating on the streets during nighttime. The data expansion comprises 23 audio fragments of 5 second duration. Once the features of each audio fragments are obtained with the encoder, the new class can be understood in comparison to the other classes.

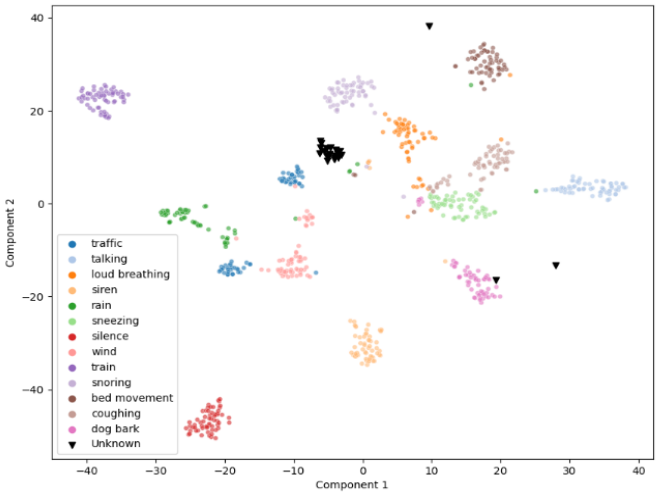

Figure 30: Unknown sound in t-SNE space

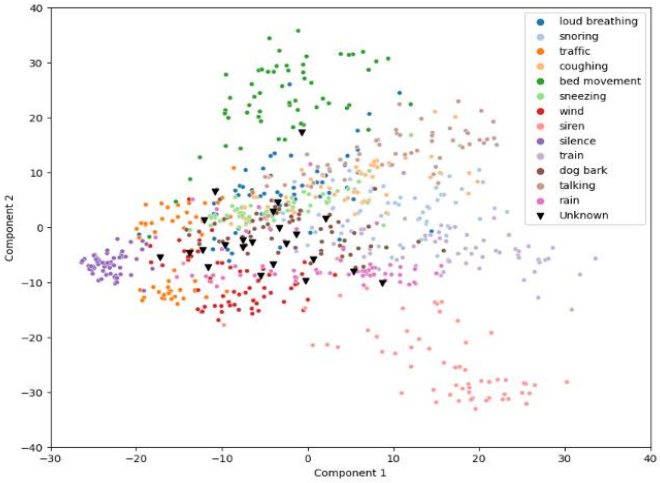

Figure 31: Unknown sound in PCA space

Retraining the cluster learner with the new class, an accuracy of 86% is attained. The results can be visualized in Figure 32.

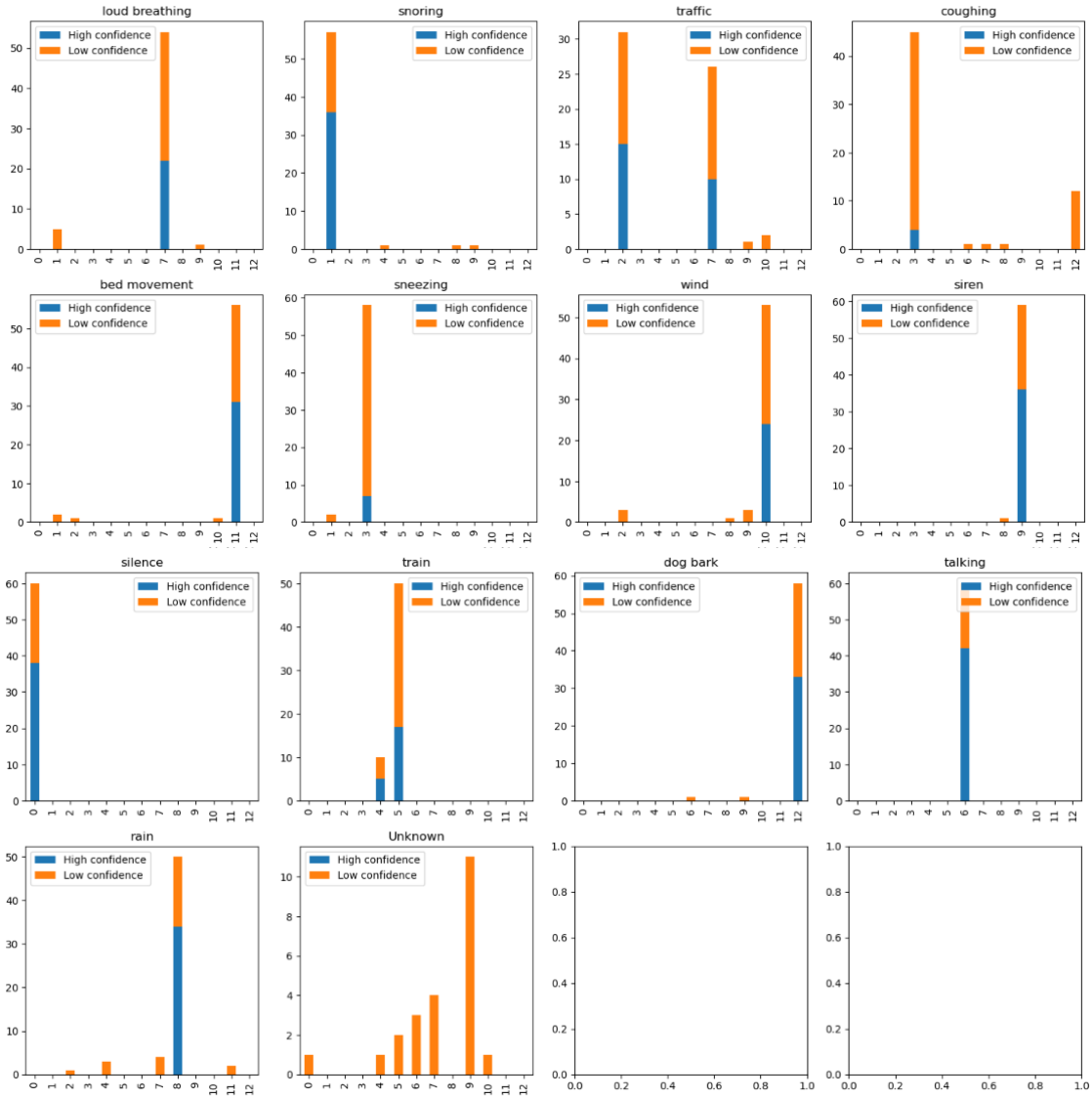

Figure 32: Cluster distribution with unknown sound

Throughout this analysis, no labels have been shown to the model. This unsupervised learning scenario proves the robustness of the feature space created by the encoder.

Dogan et al. [48] take this approach further proposing a zero-shot audio classification model from image embeddings. The feature representation of the audio samples is projected onto the semantic embeddings of a large image dataset.

7. Improving the model

As previously exposed, the different solutions this Master's Thesis explores involve two main architectures: light weight architectures trained from scratch, and encoder-classifier architectures.

The first kind involves using architectures trained from scratch, usually lighter models that do not require vast amounts of data to obtain good performance. The second kind, the encoder-classifier architectures, implies the use of a pre-trained model with a complex architecture that learned feature representations from expansive datasets as an encoder, and train a classifier on top of it to make the predictions.

Depending on the type of architecture, improving the model will require different strategies. Encoder-classifier architectures have undergone solid training and will require simple customizations of the classifier to attain some improvement. Architectures trained from scratch will require a deeper improvement process, as these have been exposed to small amounts of data and can benefit from learning new and more generalized feature representations.

Nonetheless, either approach will require collecting new data. Section 7.1 proposes a simple solution to collect data from participants.

7.1. Using Citizen Science to Collect Data

To improve the models and extend its generalization capabilities across the participant's contexts, data will have to be collected from the participants. Due to privacy and memory constraint reasons, audio files cannot be directly collected from the users and stored in the server. The methods explored in 7.2 and 7.3 will access this data with privacy preserving and memory efficient approaches.

Participants will need to label new data to make effect of any possible improvements. Data will be stored at their devices to be used by the improvement processes. The labeling approach tackles two challenges:

- Storing a large number of audio recordings on the device implies high memory consumption.
- Labeling audio recordings can be time consuming and difficult for participants.

To overcome these challenges, a labeling process is proposed, taking advantage of the Feature Space Analysis discussed in section 6 and applying certain restrictions. The process will store audio recordings with sound levels above a threshold of 30dB and ask the user to label samples within a simple UI that can be integrated in the *Soundless* app. The objective is to design an easy-to-use process that is memory efficient. Find an overview of the labeling process in the pseudocode below:

Labeling process: For each participant, across N=3 nights, assisted labeling with K=5 neighbors, and a sound level threshold of τ = 30 dB, involved in L = 10 labeling rounds

```
Initialize labeling process in each participant
for i = 1,...N do
        Participant starts the nighttime recording X_i
        for each 5-second fragment x ϵ X_i
                if sound level is > τ
                        store in memory X
                else
                        skip
                end for

        Participant stops the nighttime recording
end for
for l = 1,...L do
        Take random audio sample x_l ϵ X
        label = ManualLabelling(x_l)
        LabelNeighbors(x_l, K, label)
end for
procedure ManualLabelling(x)
        Play audio sample in participant's device
        Ask the participant to label the sample
        Store label
end procedure
procedure LabelNeighbors(x, K, label)
        Use the encoder to compute the feature representations of all audio samples
        Find the 5 audio samples with shorter distance
        Tag each neighbor with label
end procedure
```

The proposed labeling process uses the feature space analysis to find the 5 closest neighbors. With this mechanism, each label provided by the user translates into 6 labels. With only 10 manual inputs from the user, 60 samples can be labeled.

Other manners of exploiting unlabeled data can be explored. Gururani and Lerch [49] show the effectiveness of data labeling methods to exploit unlabeled data with student-teacher architectures. The labeling method proposed in this Master's Thesis only takes input from the participants. It would be highly convenient to explore more advanced methods to integrate novel techniques and exploit participant's data to its full potential.

While the proposed labeling process can result in highly heterogeneous and unbalanced datasets, a procedure is presented in 7.3 to balance this data.

7.2. Centralized improvement: Updating the Classifier

Architectures with pre-trained encoders only train the classifier on top of the embeddings. The YAMNet model employed in the working prototype only trains the last layer of the model with a few thousand parameters. As seen in section 6, the features extracted from these types of pre-trained encoders are already good representations of the data.

The approach proposed in this section is to train only the last layer of the model, the classifier, by using feature representations tagged by the participant. The main advantage of this approach is that feature representations can be sent directly to the server with no privacy issues, as the data is encoded in a low dimensional space where it would not be possible to decode the audio sample.

This means that participants can send the feature representation of the recorded audio samples to the central server along with their corresponding labels, without the need of allocating any memory to store audio recordings. The central server will locally store feature representations and labels in a dataset that will be incrementally expanded as more participants are involved.

This approach has been proven effective as models developed in 4.2.2 and 4.3 use this technique. As shown in section 6, this approach is effective even with unlabeled data. One important advantage of this approach is that it can easily integrate new classes into the model from the local server and update the classifier of the participants. The new class with its corresponding feature representations can be added to the dataset stored at the server. Assuring proper class balance, the classifier can be retrained and if successful, it can be sent back to the participants.

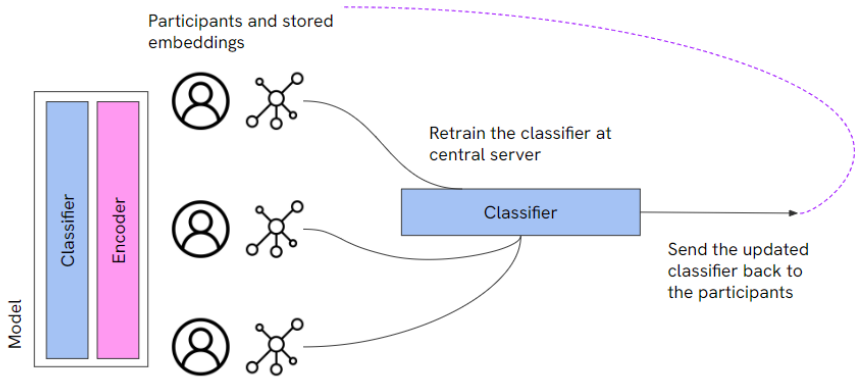

Figure 33: Updating a central classifier with embeddings from multiple clients

7.3. Decentralized improvement: Federated Learning

To train the model on the original audio data, and to avoid storing client data at the server, there is a novel approach using Federated Learning, where all the participants can contribute to the learning process and the model can learn by accessing audio samples in a decentralized manner.

Federated Learning is a machine learning approach that allows for the training of an algorithm across multiple decentralized devices holding local data samples, without exchanging them. This approach is particularly useful in situations where privacy, legal, or commercial constraints prevent data from being centrally stored and processed.

The standard process of federated learning involves five main steps [48]:

Initialize Global Model: The process begins by initializing the model on a central server (f_G). This model could be initialized randomly or from a previously saved checkpoint.

Send Model to Client Nodes: The parameters of the global model are then sent to the connected client nodes (participants). This ensures that each participating node starts their local training using the same model parameters.

Local Training: Each client node uses its own local dataset (D_m) to train its own local model. This training doesn't aim for full convergence but is rather a brief period of training.

Return Model Updates to Server: After local training, each client node sends its updated model back to the server.

Aggregate Model Updates: The server receives model updates from the selected client nodes and combines all the model updates it received from the client nodes. This process is called aggregation, and there are many different ways to do it. The most basic way is called Federated Averaging, which takes the weighted average of the model updates, weighted by the number of examples each client used for training.

Communication between clients and model updates are repeated rounds until the performance converges.

Standard Federated Learning Algorithm: For M=6 clients, number of rounds N = 20, each client holding a private dataset D_m^i

for $n = 1, \ldots N$ **do**
  for each client $m \epsilon M$ in parallel **do**:
    $\theta_i^m \leftarrow \theta_i^G$
    Client update:
      $\theta_{i+1}^m \leftarrow f_{D_m^i}(\theta_i^m)$
    Send model weights θ_{i+1}^m back to the server
  end for
  Server update:

$$\theta_{i+1}^G = \sum_{m=1}^{M} \frac{N_m}{N} f_m^i(\theta_{i+1}^m)$$

end for

Federated learning is particularly useful for citizen science as it allows for the collaboration of multiple entities, each contributing their local data without having to share it directly. This is especially beneficial for improving audio classification models, as audio data is often sensitive and private. By using federated learning, the privacy of this data can be preserved while still allowing for the improvement of the model. This approach opens new possibilities for the application of machine learning in areas where privacy concerns previously made it challenging.

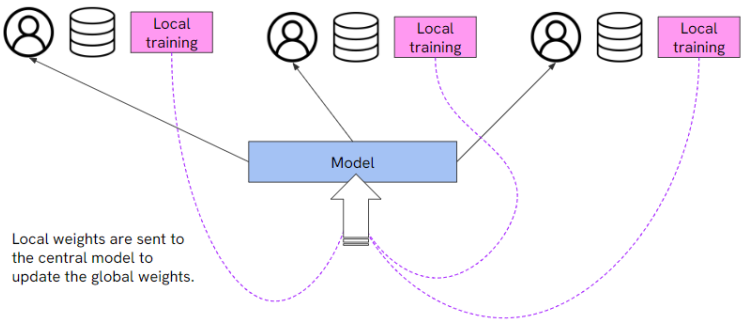

Figure 34: Federated Learning diagram

The Federated Learning setup has been built using Flower, a specialized Python framework that easily enables communication between server and clients to exchange data during the training process and is prepared for full integration with TensorFlow [48].

This project has been built from scratch (view Annex VIII with the GitHub repository) to fully understand how the models developed in section 4 can be configured within a Federated Learning environment.

This set up uses the NBAC dataset, comprised of 780 samples, with 60 samples per class. 6 clients owning 10 samples per class have been prepared to train during 20 rounds.

The setup uses a Depthwise Separable Convolutional Neural Network (DW) with random weight initialization, as built in 4.2.1.

The results in figure 36 show performance evaluation across the 20 rounds, averaging the test accuracy of each client, and showing the theoretical limit with the performances registered in section 5.

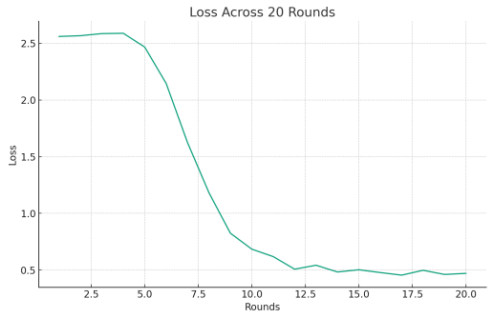

Figure 35: Loss across 20 Federated Learning rounds

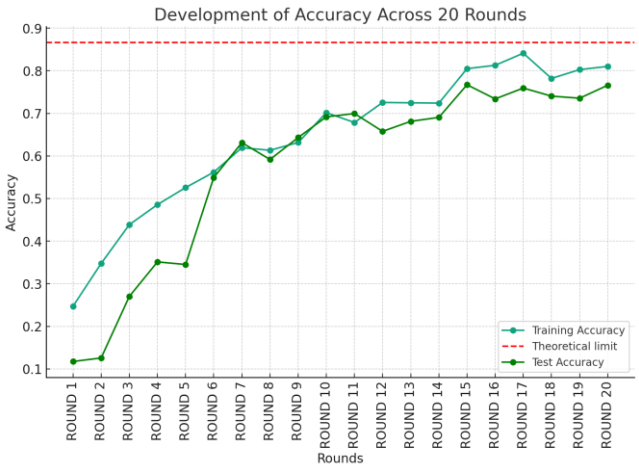

Figure 36: Development of accuracy during Federated Learning

The heterogeneity of the data across different devices in a federated learning setting often results in a non-IID data scenario, potentially leading to unbalanced or skewed learning outcomes. Each device trains on a unique and unpredictable set of data; for instance, in the context of sound recognition, it is not possible to foresee the diversity of sounds a mobile device might record. One participant might be predominantly recording dog barks with an absence of traffic sounds. If this unique dataset is used to update the model, it can

contribute to a skewed learning curve, since it may not accurately represent the broader distribution of sounds in the real world.

This non-IID nature of federated learning poses a significant challenge, particularly in high-dimensionality machine learning problems like the problem approached in this Master's Thesis. In low-dimensionality problems, the challenge may be less severe, but it's still present. To address this, it's essential to employ strategies and algorithms that consider and tackle the non-IID data distribution in Federated Learning environments. Techniques such as weighted aggregation, adaptive learning rates, and advanced optimization algorithms can help achieve a more balanced and representative learning model.

To face this challenge, a modification of the standard Federated Learning algorithm is performed. A public dataset D_0 is created with perfect class balance, with all clients having access to it. For example, this public dataset could be a version of NBAC. Then, before locally updating the model, clients combine their local dataset D_m with D_0 to generate a balanced dataset D_B.

Modified Federated Learning Algorithm: For M=6 clients, number of rounds N = 20, each client holding a private dataset D_m^i, public dataset D_0, and minimum adjustment ratio β = 0.5

for $n = 1, \dots N$ **do**
  for each client $m \epsilon M$ in parallel **do**:
    $\theta_i^m \leftarrow \theta_i^G$
    $D_B = GenerateBalancedDataset(D_m, D_0)$
    Client update:
      $\theta_{i+1}^m \leftarrow f_{D_B}(\theta_i^m)$
    Send model weights θ_{i+1}^m back to the server
  end for
  Server update:

$$\theta_{i+1}^G = \sum_{m=1}^{M} \frac{N_m}{N} f_m^i(\theta_{i+1}^m)$$

end for
procedure $GenerateBalancedDataset(D_{private}, D_{public})$
  for each label in D_{public} **do**:
    max = find maximum class sample size in $D_{private}$
  end for
  for each label in D_{public} **do**:
    x_m^l = take all samples from label in $D_{private}$
    x_o^l = take samples from label in D_{public} to satisfy the equation

$$x_m^l + x_o^l = max + \beta\, max$$

    $x_b = x_m^l + x_o^l$
    Add x_b with its corresponding labels to D_B
  end for
end procedure

This algorithm is tested in Annex VIII by using 10 samples of each class of the NBAC dataset as a public dataset and distributing the rest of the dataset randomly and unevenly across 6 clients.

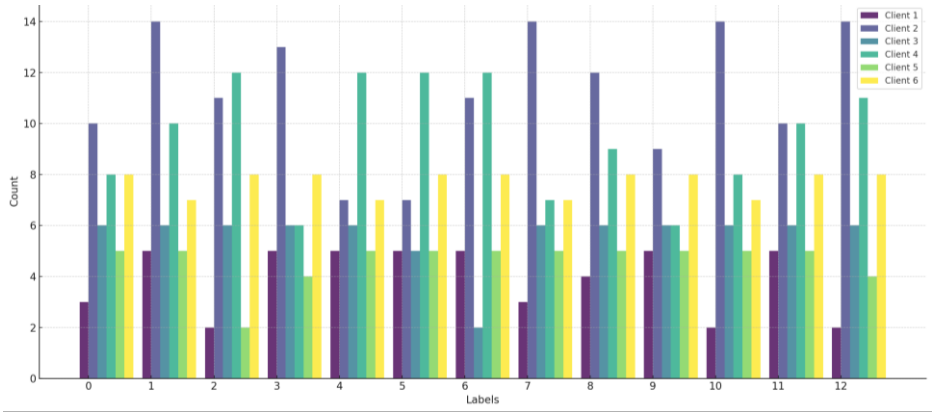

Figure 37: Random distribution of classes across 6 clients

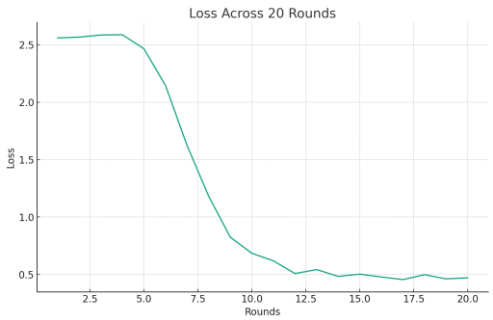

Figure 38: Loss across 20 Federated Learning rounds with non-IID data

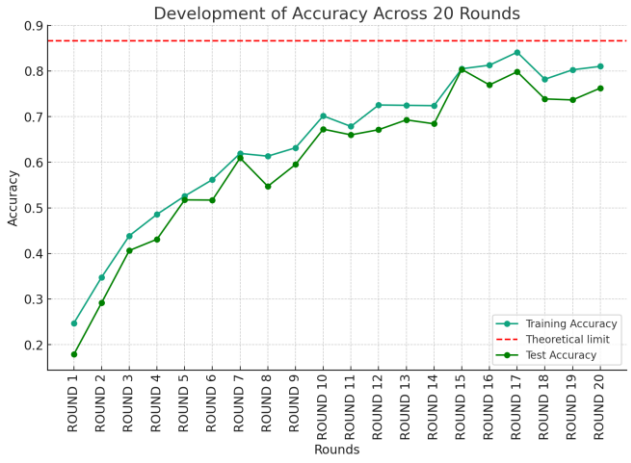

Figure 39: Development of accuracy during Federated Learning with non-IID data

These experiments show that the problem proposed by *Soundless* is suitable for a Federated Learning scenario. The results show that the distributed model can achieve the same accuracies as the centralized model on the NBAC dataset. Even simulating a real case scenario with non-IID data yields satisfactory results. The success of this approach would rely on the effectiveness of the labeling process proposed in 7.1.

One of the main drawbacks of this solution is that it makes it very challenging for users to add new classes. Recent studies manage to tackle this challenge by implementing an advanced algorithm that employs anonymized data impressions to update the public dataset and force the new classes to appear during training of all clients [50]. Leveraging new classes reported by participants is rather simple with a centralized improvement process as proposed in 7.2.

7. Conclusions

In this work, Deep Learning techniques for audio event classification in mobile applications have been successfully applied and refined. The whole process is documented to achieve a highly customizable and flexible product. While the context, with high device and acoustic context variability, holds important challenges, it also offers interesting opportunities to exploit the potential of citizen science to collect data and improve the models.

This Master's Thesis has demonstrated the potential of utilizing standard Deep Learning approaches to enhance the *Soundless* project. The exploratory approach has provided key insights into how audio classification works through Deep Learning, making it crucial for leveraging a simple yet effective solution.

Lightweight Depthwise Separable Convolutions, consisting of only three internal convolutional layers, achieve effective results when employing the proposed data augmentation techniques. These models achieve classification accuracies of over 65% on the ESC-50 benchmark and above 86% on the custom NBAC dataset, utilizing only 58 thousand and 48 thousand parameters respectively. Furthermore, these models are trained from scratch. Strengthening the quality, quantity, and variability of the datasets can further enhance the performance of these lightweight models.

The utilization of pre-trained models has proven to be the most advantageous solution for addressing the audio classification problem, particularly when working with limited data. By employing the MobileNetV1 architecture pre-trained on ImageNet, accuracy rates of 85% and 91% have been achieved for the ESC-50 benchmark and NBAC respectively. Furthermore, utilizing YAMNet, which is internally built with MobileNetV1 and pre-trained on AudioSet, has resulted in a slight improvement in accuracy. Specifically, prediction accuracy of 86.5% has been attained for the ESC-50 benchmark, while an accuracy of 95% has been achieved for NBAC.

With the understanding gained from the exploratory analysis of Deep Learning models, potential implementation solutions are utilized to maximize the model's capabilities. The models can serve as encoders for conducting feature space analysis on audio samples. This analysis is valuable for evaluating and comparing audio samples based on their feature representation. The effectiveness of this approach is demonstrated by employing unsupervised semantic clustering methods, which achieve accuracies exceeding 89% on

NBAC. This possibility proves useful in leveraging participant data to integrate assisted labeling methods or store privacy-compliant data on the server.

The deployment of these models within the citizen science context presents numerous opportunities. This Master's Thesis explores two viable methods for deploying and improving the models. One approach involves locally updating a classifier with feature representations derived from the data collected by participants. The other approach entails training the model in a decentralized manner using Federated Learning. The experiments conducted validate the feasibility of both approaches, as they offer deployment methods with the potential to enhance the model progressively without compromising performance initially.

The aim discussed with Soundless at the beginning of the project has been attained. A model with performance accuracies of over 85% in both datasets has been deployed on the prototype of an Android app with successful results: low latency and accurate predictions.

Despite plans to further advance model deployment, significant efforts were required to develop Deep Learning models compatible with Android applications. This stage demanded a meticulous development approach and testing of various methods to create deployment-ready models. The primary issue was constructing a compact model capable of directly accepting raw wav files as input. While YAMNet offers this functionality out of the box, it was necessary to develop such models from scratch for deployment readiness.

Although this Master's Thesis presents a practical approach, it also proposes novel Deep Learning methods that pave the way for future research. The foremost next step is fully implementing the predictive model in *Soundless*. This includes experimenting with deploying a labeling mechanism for clients, an updatable classifier, or a Federated Learning architecture, all of which are pathways opened by this Master's Thesis.

Furthermore, a more in-depth exploration of deploying TensorFlow Lite models remains an area of interest. Considerable efforts were invested in developing deployment-ready models, making it intriguing to conduct a careful analysis of how different models behave in Android applications.

8. Bibliography

[1] Coll, D., Domingo, S., García, P., "Soundless: Plataforma de ciència ciutadana per a auditar els efectes de la contaminació acústica a la ciutat de Tarragona", *Universitat Rovira I Virgili*, https://soundless.app/, 2022.

[2] https://apps.who.int/iris/handle/10665/326486, Date Visited: March 6, 2023.

[3] https://play.google.com/store/apps/details?id=cat.urv.cloudlab.soundless, Date Visited: March 6, 2023.

[4] Haklay, M., Dörler, D., Heigl, F., Manzoni, M., Hecker, S., & Vohland, K., "What Is Citizen Science? The Challenges of Definition", *The Science of Citizen Science*, 2021.

[5] https://eu-citizen.science/, Date Visited: March 6, 2023.

[6] Irwin, A., "Citizen science comes of age", Nature, https://media.nature.com/original/magazine-assets/d41586-018-07106-5/d41586-018-07106-5.pdf, 2018.

[7] Hong, D., Heacock, H., & Shaw, F., "The practicality of using a smartphone as a sound level meter", *BCIT Environmental Public Health Journal*, 2017.

[8] Wei, Y., Zhang, Q., "Common Waveform Analysis: A New And Practical Generalization of Fourier Analysis", *Springer US*, 2000

[9] Breebaart, J., McKinney, M., "Features for Audio Classification", *Algorithms in Ambient Intelligence*, 2004.

[10] Piczak, K. J., "ESC: Dataset for Environmental Sound Classification", *Proceedings of the 23rd Annual ACM Conference on Multimedia*, 2015.

[11] Tsalera, E., Papadakis, A., & Samarakou, M., "Comparison of Pre-Trained CNNs for Audio Classification Using Transfer Learning", *Journal of Sensor and Actuator Networks*, 2021.

[12] https://www.mathworks.com/help/signal/ref/spectrogram.html, Date Visited: March 6, 2023.

[13] https://www.mathworks.com/help/audio/ref/melspectrogram.html, Date Visited: March 6, 2023.

[14] Kaneko, T., Tanaka, K., Kameoka, H., & Seki, S., "iSTFTNet: Fast and Lightweight Mel-Spectrogram Vocoder Incorporating Inverse Short-Time Fourier Transform", *arXiv.org*, 2022.

[15] Harley, A. W., "An Interactive Node-Link Visualization of Convolutional Neural Networks," *in ISVC, pages 867-877*, 2015

[16] https://adamharley.com/nn_vis/cnn/2d.html, Date Visited: June 13, 2023.

[17] Guo, Y., Li, Y., Feris, R., Wang, L., & Rosing, T., "Depthwise Convolution is All You Need for Learning Multiple Visual Domains", *arXiv.org*, 2019.

[18] Chollet, F., "Xception: Deep Learning with Depthwise Separable Convolutions", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[19] Prince, P., Hill, A., Piña Covarrubias, E., Doncaster, P., Snaddon, J., & Rogers, A., "Deploying Acoustic Detection Algorithms on Low-Cost, Open-Source Acoustic Sensors for Environmental Monitoring", *Sensors*, 2019.

[20] https://www.tensorflow.org/hub/tutorials/yamnet, Date Visited: June 13, 2023.

[21] Choi, Y., Choi, H., Lee, H., Lee, S., & Lee, H., "Lightweight Skip Connections With Efficient Feature Stacking for Respiratory Sound Classification", *IEEE Access*, 2022.

[22] Palanisamy, K., Singhania, D., & Yao, A., "Rethinking CNN Models for Audio Classification", *arXiv.org*, 2020.

[23] Cai, Y., Tang, H., Zhu, C., Li, S., & Shao, X., "DCASE 2022 Submission: Low-Complexity Model Based on Depthwise Separable CNN for Acoustic Scene Classification", *DCASE Challenge, Detection and Classification of Acoustic Scenes and Events 2022*, 2022.

[24] Hou, Y., "Low-Complexity for DCASE 2022 Task 1A Challenge. Technical Report. DCASE Challenge", *Detection and Classification of Acoustic Scenes and Events 2022*, 2022.

[25] https://github.com/tensorflow/models/tree/master/research/audioset/yamnet, Date Visited: May 13, 2023.

[26] Lopez-Meyer, P., del Hoyo Ontiveros, J. A., Lu, H., & Stemmer, G., "Efficient End-to-End Audio Embeddings Generation for Audio Classification on Target Applications", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 601-605*, 2021.

[27] Huang, J. J., & Alvarado Leanos, J. J., "ACLNet: Efficient end-to-end Audio Classification CNN", *arXiv.org*, 2018.

[28] Mohaimenuzzaman, M., Bergmeir, C., West, I., & Meyer, B., "Environmental Sound Classification on the Edge: A Pipeline for Deep Acoustic Networks on Extremely Resource-Constrained Devices", *Pattern Recognition*, 2023.

[29] https://librosa.org/doc/main/generated/librosa.util.normalize.html, Date Visited: May 13, 2023.

[30] Jiang, C., Ahn, J., & Desai, N., "Acoustic Environment Transfer for Distributed Systems", *arXiv.org*, 2021.

[31] Park, D. S., & Chan, W., "SpecAugment: A New Data Augmentation Method for Automatic Speech Recognition", Google AI Blog, 2019.

[32] https://librosa.org/doc/main/generated/librosa.feature.melspectrogram.html, Date Visited: June 2, 2023.

[33] https://www.tensorflow.org/io/api_docs/python/tfio/audio/melscale, Date Visited: June 2, 2023.

[34] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", *arXiv.org*, 2017.

[35] http://research.google.com/audioset/, Date Visited: June 5, 2023.

[36] https://dcase.community/challenge2022/task-low-complexity-acoustic-scene-classification, Date Visited: June 4, 2023.

[37] https://www.tensorflow.org/lite, Date Visited: May 17, 2023.

[38] https://www.tensorflow.org/lite/guide/inference, Date Visited: May 17, 2023.

[39] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L., "Imagenet: A large-scale hierarchical image database", *IEEE conference on computer vision and pattern recognition, pp. 248–255*, 2009

[40] https://www.tensorflow.org/tutorials/audio/transfer_learning_audio, Date Visited: June 12, 2023.

[41] https://blog.tensorflow.org/2021/03/transfer-learning-for-audio-data-with-yamnet.html, Date Visited: June 14, 2023.

[42] https://www.tensorflow.org/lite/models/modify/model_maker, Retrieved June 1, 2023.

[43] https://www.tensorflow.org/lite/api_docs/python/tflite_model_maker/audio_classifier/YamNetSpec, Date Visited: June 1, 2023.

[44] S. Grollmisch, E. Cano, C.Kehling & M. Taenzer, "Analyzing the Potential of Pre-Trained Embeddings for Audio Classification Tasks", eurasip.org, 2020.

[45] https://keras.io/examples/vision/semantic_image_clustering/, Date Visited: June 20, 2023.

[46] P. Wolters, C. Careaga, B. Hutchinson, & L. Phillips, "A study of Few-Shot Audio Classification", arXiv.org, 2020.

[47] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html, Date Visited: June 20, 2023.

[48] D. Dogan, H. Xie, T. Heittola, & T. Virtanen, "Zero-Shot Audio Classification using Image Embeddings", arXiv.org, 2022.

[49] S. Gururani, & A. Lerch, "Semi-Supervised Audio Classification with Partially Labeled Data", arXiv.org, 2021.

[50] G. Krishna, & S. Kumar, “Zero-Shot Federated Learning with New Classes for Audio Classification”, arXiv.org, 2021.

Annexes

The code built to elaborate all experiments and products of this Master's Thesis is stored in different GitHub repositories. The repository `audio-classification-notebooks` collects and organizes all Jupiter notebooks running data preprocessing experiments, models construction and training, and further model analysis. The repository `yamnet-native-training` contains a machine learning application to train and export the YAMNet model in TensorFlow Lite format. The repository `federated-learning-environment` contains a simple Federated Learning application built from scratch to train a custom audio classification model in a Federated Learning environment.

audio-classification-notebooks:

https://github.com/DavidCastello/audio-classification-notebooks.git

yamnet-native-training:

https://github.com/DavidCastello/yamnet-native-training.git

federated-learning-environment:

https://github.com/DavidCastello/federated-learning-environment.git

Annex I: Data Pre-processing Lab

Refer to the file `final notebooks/audio-processing-lab.ipynb` in the **audio-classification-notebooks** repository**.**

This file contains the starting scripts to load the wav files. Data augmentation techniques for wav files and spectrograms are presented. Two different functions to construct the Mel Spectrogram are built. One using the Librosa framework, and the other one using the TensorFlow framework.

```
def librosa_mel_spectrogram(wav, sample_rate=SR, num_mel_bins=64, lower_freq=125, upper_freq=7500, log_offset=0.001):
    # Compute the Short-Time Fourier Transform (STFT) with a window size of 25 ms, a window hop of 10 ms, and a periodic Ha
    n_fft = int(sample_rate * 0.025)  # Window size of 25 ms
    hop_length = int(sample_rate * 0.01)  # Window hop of 10 ms
    window = 'hann'  # Periodic Hann window

    stft = librosa.stft(wav, n_fft=n_fft, hop_length=hop_length, window=window)
    spectrogram = np.abs(stft)

    # Map the spectrogram to 64 mel bins covering the range 125-7500 Hz
    mel_filter = librosa.filters.mel(sr=sample_rate, n_fft=n_fft, n_mels=num_mel_bins, fmin=lower_freq, fmax=upper_freq)
    mel_spectrogram = mel_filter.dot(spectrogram)

    # Compute a stabilized log mel spectrogram
    log_mel_spectrogram = np.log(mel_spectrogram + log_offset)

    # Transpose the output. Librosa outputs spectrogram of shape (n_mel_bins, T)
    log_mel_spectrogram = log_mel_spectrogram.T

    return log_mel_spectrogram
```

Code snippet 1: Mel Spectrogram using Librosa

```
def tf_mel_spectrogram(wav, sample_rate=SR, num_mel_bins=64, lower_freq=125, upper_freq=7500, log_offset=0.001):

    # Convert numpy array to Tensor and normalize based on its actual max and min values
    wav = tf.cast(wav, tf.float32)
    audio_tensor = (wav - tf.math.reduce_min(wav)) / (tf.math.reduce_max(wav) - tf.math.reduce_min(wav)) * 2 - 1

    # Calculate the frame_length and frame_step based on window size and hop size
    frame_length = int(sample_rate * 0.025)  # 25 ms window
    frame_step = int(sample_rate * 0.01)  # 10 ms hop

    # Calculate the spectrogram
    spectrogram = tf.signal.stft(audio_tensor, frame_length=frame_length, frame_step=frame_step)

    # Calculate the magnitude of the spectrogram
    magnitude_spectrogram = tf.abs(spectrogram)

    # Define the parameters for the Mel scale
    num_spectrogram_bins = magnitude_spectrogram.shape[-1]
    linear_to_mel_weight_matrix = tf.signal.linear_to_mel_weight_matrix(
        num_mel_bins, num_spectrogram_bins, sample_rate, lower_freq, upper_freq)

    # Convert the spectrogram to the Mel scale
    mel_spectrogram = tf.tensordot(magnitude_spectrogram, linear_to_mel_weight_matrix, 1)
    mel_spectrogram.set_shape(magnitude_spectrogram.shape[:-1].concatenate(linear_to_mel_weight_matrix.shape[-1:]))

    # Apply a logarithmic scale
    log_mel_spectrogram = tf.math.log(mel_spectrogram + log_offset)

    return log_mel_spectrogram.numpy()
```

Code snippet 2: Mel spectrogram using TensorFlow

The script also contains the function to split the wav files into 2 second long fragments with 0.5 overlap to create the al2s configuration.

Annex II: Architectures with Mel Spectrograms as Input

Refer to the scripts located in the folder `final notebooks/4.1` in the **audio-classification-notebooks** repository.

These scripts build custom models that accept Mel Spectrograms as input with three convolutional layers. This architecture is built using regular CNNs and Depthwise Separable Convolutional Layers.

```
HLF = Sequential([
    tf.keras.layers.Conv2D(32, 3, strides=2, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
])

Embeddings = Sequential([
    tf.keras.layers.Conv2D(256, 1, padding='same', activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.GlobalAveragePooling2D(),
])

# Define the model
model = Sequential([
    HLF,
    Embeddings,
    tf.keras.layers.Dense(NUM_CLASSES),
    tf.keras.layers.Activation('softmax')  # The activation should be softmax for multi-class classification
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(np.array(X_train_specs), np.array(y_train), epochs=50, batch_size=32, validation_data=(np.array(X_val
```

Code snippet 3: Standard CNN model accepting Mel Spectrograms as input

Annex III: Compact Architectures

Refer to the scripts located in the folder `final notebooks/4.2` in the **audio-classification-notebooks** repository.

These scripts integrate the Mel Spectrogram generator within the custom model architecture. This allows the models to accept wav files directly as input.

```python
class LLF(Layer):
    def __init__(self, sample_rate, frame_length, frame_step, num_mel_bins=64, lower_freq=125, upper_freq=7500, log_offset=0
        super(LLF, self).__init__(**kwargs)
        self.sample_rate = sample_rate
        self.frame_length = frame_length
        self.frame_step = frame_step
        self.num_mel_bins = num_mel_bins
        self.lower_freq = lower_freq
        self.upper_freq = upper_freq
        self.log_offset = log_offset

    def call(self, inputs):

        # Convert numpy array to Tensor and normalize based on its actual max and min values
        wav = tf.cast(inputs, tf.float32)
        audio_tensor = (wav - tf.math.reduce_min(wav)) / (tf.math.reduce_max(wav) - tf.math.reduce_min(wav)) * 2 - 1

        # Compute the Short-Time Fourier Transform (STFT)
        stft = tf.signal.stft(wav, self.frame_length, self.frame_step, window_fn=tf.signal.hann_window)

        # Compute the spectrogram
        spectrogram = tf.abs(stft)

        # Compute the mel-spectrogram
        num_spectrogram_bins = stft.shape[-1]
        linear_to_mel_weight_matrix = tf.signal.linear_to_mel_weight_matrix(
            self.num_mel_bins, num_spectrogram_bins, self.sample_rate, self.lower_freq, self.upper_freq)
        mel_spectrogram = tf.tensordot(spectrogram, linear_to_mel_weight_matrix, 1)
        mel_spectrogram.set_shape(spectrogram.shape[:-1].concatenate(linear_to_mel_weight_matrix.shape[-1:]))

        # Compute the log mel-spectrogram
        log_mel_spectrogram = tf.math.log(mel_spectrogram + self.log_offset)

        # Add a channel dimension
        log_mel_spectrogram = tf.expand_dims(log_mel_spectrogram, -1)

        return log_mel_spectrogram
```

Code snippet 4: LLF layer to generate Mel Spectrograms within the neural network

These scripts also use the MobileNetV1 architecture to substitute the HLF block and the Embeddings block. The MobileNetV1 architecture is tested with random weight initialization and weights pre-trained on the ImageNet dataset.

Annex IV: YAMNet Model

The experiments are found in the folder `final notebooks/4.2`, in the **audio-classification-notebooks** repository**.**

These experiments provide a workaround to use the YAMNet model and train a classifier on top of it. The embeddings of the wav files are generated with the YAMNet model and later used as input to train the classifier. Custom logic is then programmed to output the final result.

```python
# Create an empty list to hold the stacked arrays
y_pred_scores = []

# Iterate through array_list in steps of 10
for i in range(0, len(y_pred), 10):

    # Select 10 arrays and stack them together
    stacked = np.stack(y_pred[i:i+10])

    mean_array = np.mean(stacked, axis=0)

    # Append the stacked array to stacked_arrays
    y_pred_scores.append(mean_array)
```

```python
y_pred = np.argmax(y_pred_scores, axis=1)
```

Code snippet 5: Custom prediction logic with YAMNet embeddings

The repository **yamnet-native-training** contains the entire machine learning application to train the custom YAMNet model natively in TensorFlow Lite. Refer to the README.md file for instructions on how to use the application.

```python
def train(args):

    spec = audio_classifier.YamNetSpec(keep_yamnet_and_custom_heads=True,
                                       frame_step=audio_classifier.YamNetSpec.EXPECTED_WAVEFORM_LENGTH // 2,
                                       frame_length=args.length_audio * audio_classifier.YamNetSpec.EXPECTED_WAVEFORM_LENG

    train_data_ratio = args.train_data_ratio
    train_data = audio_classifier.DataLoader.from_folder(
        spec, args.train_dir, cache=True)
    train_data, validation_data = train_data.split(train_data_ratio)
    test_data = audio_classifier.DataLoader.from_folder(
        spec, args.test_dir, cache=True)

    model = audio_classifier.create(
        train_data, spec, validation_data, args.batch_size, args.epochs)

    print("======================")

    print(f'Exporing the model to {args.save_path}')
    model.export(args.save_path, tflite_filename=args.tflite_file_name)
    model.export(args.save_path, export_format=[
                 mm.ExportFormat.SAVED_MODEL, mm.ExportFormat.LABEL])
```

Code snippet 6: Function to train the custom YAMNet model natively

Annex V: Minimum Class Sample Size Analysis

Refer to the scripts located in the folder `final notebooks/5.1.1` in the **audio-classification-notebooks** repository**.**

These scripts evaluate the model performance for different architectures using a different number of samples per class, always guaranteeing class balance.

Annex VI: Android Application Prototype

The README.md file in the repository **yamnet-native-training** contains all necessary instructions to set up the TensorFlow Android application with the custom model.

Annex VII: Feature Space Analysis

Refer to the scripts located in the folder `final notebooks/6` in the **audio-classification-notebooks** repository**.**

These scripts use the embeddings extracted with the pre-trained model with the MobileNetV1 architecture and perform feature analysis and semantic clustering with unsupervised learning.

Annex VIII: Federated Learning

Refer to the repository **federated-learning-environment**. This repository contains the code to train a custom audio classification model in a Federated Learning environment. Find the instructions to run the Federated Learning application in the README.md file.

Execute main.py to simulate a Federated Learning environment with 6 clients and perfectly balanced data. Alternatively, execute main-noniid.py to run a Federated Learning environment with 6 clients and non-IID data. This method proposes a balancing algorithm that uses a public dataset to balance class sample sizes. The standard Federated Averaging method then weighs the impact of each client based on the total number of owned samples.

```python
def balance_dataset(data_subset, label_subset, public_data, public_labels, beta):
    # Convert the labels to Series for easier manipulation
    s_private_labels = pd.Series(label_subset)
    s_public_labels = pd.Series(public_labels)

    # Find the maximum class sample size in the private dataset
    max_size = s_private_labels.value_counts().max()

    # Create empty lists for the balanced dataset
    balanced_data = []
    balanced_labels = []

    # Loop over each label in the public dataset
    for label in s_public_labels.unique():
        # Get indices of the current label from the private and public datasets
        private_indices = s_private_labels[s_private_labels == label].index.tolist()
        public_indices = s_public_labels[s_public_labels == label].index.tolist()

        # Calculate the number of public samples needed to satisfy the equation
        num_public_samples = max_size + int(beta * max_size) - len(private_indices)

        # If there are not enough public samples, take all of them, otherwise take the required number
        selected_public_indices = random.sample(public_indices, min(num_public_samples, len(public_indices)))

        # Collect data samples
        balanced_data.extend([data_subset[i] for i in private_indices] + [public_data[i] for i in selected_public_indices])
        balanced_labels.extend([label] * (len(private_indices) + len(selected_public_indices)))

    return np.array(balanced_data), np.array(balanced_labels)
```

Code snippet 7: Balancing client owned datasets with public dataset