

AI-Powered Noise Awareness in Noise-Cancelling Headphones

**Dr.T.J.Jeyaprabha¹, Dr.S.Vijayanand², Ms.Mehzabeen S M³,Mr. Kanagaluru Venkatesh⁴,
Dr.G.Ayappan⁵**

^{1,2,3,4,5} Department of Electronics and Communication Engineering, Sri Venkateswara College of Engineering,
Sriperumbudur 602117, India.

Abstract— Today's market has noise cancelling headphones everywhere that uses ANC (Active Noise Cancellation) that entirely cuts off the outside environmental noise. This creates a situation where the user is not aware of the external environment and is totally cut off. This project aims at creating a vehicular horn detection model using deep learning and embedding it into hardware for use in headphones. Thus, whenever the horn sound is detected by the mic, an alert will be produced to the user to make him/her aware of their surroundings. This feature in a ANC supported headphone helps to ensure the safety of the user when they are totally isolated from their surroundings.

Index terms—Convolutional Neural Networks, Deep Learning, Multilabel Classification, Pattern Recognition, Python, Sound detection, Spectral Analysis, Spectrogram.

I.INTRODUCTION

Almost all the earphones that we use today, are advanced enough for cancelling out the external noise to provide the user with a good experience for hearing the other person on call talk or listen to a music without any disturbance from outside world. Active Noise Cancellation (ANC) is the principle behind all these earphones. ANC is a technology used to enhance the listening experience of the user. It involves the usage of tiny microphones that continuously monitors the external noise and produce an inverted sound wave that cancels out the incoming noise. With this advantage, ANC earphones are being actively used by people in workplaces, public transportation, and even in outdoor activities. However, this feature of the earphones where the total noise is cutoff and the user is isolated from their surroundings, it can also pose significant safety risks. The user becomes unaware of all the potential emergency sound around him, like the vehicular horns, sirens, alarms etc., potentially leading to dangerous situations.

To mitigate these risks, manufacturers have introduced features like "transparency mode" or "ambient mode," which allow environmental sounds to pass through the headphones. However, these modes typically require manual activation and do not differentiate between critical and non-critical sounds. As a result, users are forced to constantly switch between immersive and aware modes or compromise safety for the sake of audio quality. Furthermore, existing

implementations lack contextual intelligence and are unable to adapt dynamically to changes in the environment.

In the context of these limitations, this paper proposes a solution to improve the user safety feature while maintaining the benefits of ANC. We introduce a smart sound detection model developed using deep learning that detects the vehicular horn sound. The convolution neural network (CNN) is a powerful deep learning model that can learn a feature hierarchy for images [2]. By analyzing the audio, the system intelligently produces alerts to the user, thereby providing situational awareness for the user.

Hence, this involves a real time audio classification system that detects the horn sound. It also involves the integration of the detection model into a hardware system by mitigating the memory constraints of the hardware. Thus, it helps in real time vehicular horn detection featuring as a significant safety concern for the users.

II. METHODOLOGY

A. Spectrogram

A spectrogram is a visual representation of the frequency spectrum as it varies over time [2]. It is one of the most widely used tools in audio analysis and digital signal processing, if working with machine learning or deep learning models, for pattern recognition systems that analyze the sound. The spectrogram is generated by dividing the audio signal into small overlapping time windows (typically using a Short-Time Fourier Transform, or STFT), then performing a Fourier transform on each window to extract frequency information. The result is a two-dimensional plot where:

- The x-axis represents time,
- The y-axis represents frequency,
- The colour intensity (or grayscale values) represents the amplitude or energy at each time-frequency coordinate.

This project uses spectrograms to convert raw audio signals to a format suitable for training a deep learning model for sound classification. Spectrogram is being used for classifying the horn sounds from other environmental sounds because of the following reasons:

1) Rich Feature Representation:

Spectrograms capture both temporal and spectral features of a sound. This helps to identify the horn sounds, as different sound differs in pitch and the energy distribution over time and frequency.

2) Suitable format for the Deep Learning models:

By transforming audio into a visual format, spectrograms allow the use of convolutional neural networks (CNNs). Thus, this makes the model highly effective at learning spatial patterns, making it well-suited for detecting specific sound patterns in spectrograms.

3) Compact and interpretable:

A spectrogram provides a compressed yet informative summary of an audio signal, allowing for faster training and inference and also enables visual inspection of the sound features.

B. Model Architecture

First, the raw time-domain audio waveform is transformed by a front-end into a time-frequency representation. Then the systems extract features from this representation. In a final step the features are classified.

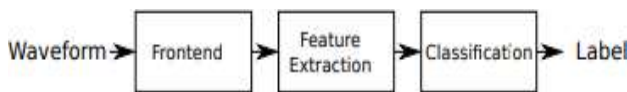


Fig. 1. Model Architecture

The front-end is used to transform the raw audio signal into a time-frequency representation from which features in both, frequency- and time-domain, can be extracted. The feature extraction block uses a CNN to learn the features of the time-frequency representation. Most CNNs are built from very few basic building blocks: convolution, activation and pooling layers. Then classification is done based on the training of the model and output is produced. [12]

C. Linear Mel Spectrogram

The Linear Mel Spectrogram is a time-frequency representation of audio that transforms raw waveform data into a two-dimensional matrix, where one axis corresponds to time and the other to perceptually scaled frequency bands. The Mel scale provides a linear scale for the human auditory system, and is related to Hertz by the following formula, where m represents Mels and f represents Hertz:

$$m = 2595 \log_{10} \left(1 + \frac{f}{1000} \right)$$

The Mel spectrogram is used to provide our models with sound information similar to what a human would perceive [2]. In a Linear Mel Spectrogram, the amplitude values (typically power or energy) of the frequency bins are preserved in their original linear form rather than being converted into logarithmic scale (as in log-Mel spectrograms). This preserves the raw intensity dynamics of

the sound, which can be beneficial for identifying high-energy, urgent environmental sounds such as sirens or horns.

To compute a Linear Mel Spectrogram, the audio signal is first divided into overlapping frames, and each frame undergoes a Short-Time Fourier Transform (STFT) to capture its frequency content [4]. The resulting power spectrogram is then passed through a Mel filter bank, a set of triangular filters spaced according to the Mel scale. The output is a two-dimensional array that represents how the energy in each Mel band evolves over time. This format is particularly advantageous for machine learning applications, as it retains both spectral and temporal information in a compact form, making it highly suitable for pattern recognition tasks.

In the context of this project, the Linear Mel Spectrogram serves as the primary input to the convolutional neural network (CNN). Its image-like structure allows the CNN to extract local and global patterns that are characteristic of specific sounds, while the Mel scaling ensures that the frequency resolution aligns with human auditory perception. By using the linear amplitude values, the model remains sensitive to the true energy levels present in the signal, improving its ability to detect and classify safety-critical audio events amidst background noise.

D. Feature extraction using the CNN Layers

In this study, feature extraction is performed using a Convolutional Neural Network (CNN), which acts as an automated feature learner directly from the input Linear Mel Spectrograms. Broadly, acoustic features can be grouped into two categories: time-domain (or temporal features) and frequency-domain (or spectral features) [1]. Unlike traditional approaches that rely on hand-crafted audio features such as MFCCs, chroma, or zero-crossing rate [1], CNNs learn hierarchical feature representations directly from the raw spectrogram input. A typical convolutional neural network consists of a number of different layers stacked together in a deep architecture: an input layer, a group of convolutional and pooling layers (which can be combined in various ways), a limited number of fully connected hidden layers, and an output (loss) layer [3]. In the initial layers of the CNN, low-level features such as edges, transitions in energy, and simple spectral patterns are detected through localized convolutional operations. These operations involve sliding a set of learnable filters over the input spectrogram to compute activation maps that highlight specific frequency-time regions of interest. As the network deepens, the CNN captures increasingly abstract and complex features, such as frequency modulations, harmonic patterns, or distinctive spectral shapes associated with specific sound types. Max pooling layers are interleaved between convolutional layers to progressively reduce the spatial resolution of the feature maps, thereby condensing the learned information and providing translation invariance. These layers allow the network to focus on the most salient features while reducing computational complexity. Once high-level features are extracted, the feature maps are flattened and passed through fully connected dense layers,

which act as interpreters that combine and weigh these features to form a final representation.

This learned feature vector encodes a compact yet rich description of the input sound, enabling effective classification even in noisy environments. The ability of the CNN to perform data-driven feature extraction from spectrograms [10] significantly enhances the model's robustness and adaptability, making it ideal for detecting safety-critical audio events in real-time applications such as smart noise-cancelling headphones.

E. Quantization

To enable efficient deployment of the trained CNN model on the resource-constrained ESP32-S3 N16R8 microcontroller, post-training quantization was applied to both the model weights and input features. This process involves converting the original float32 (32-bit floating point) values into int8 (8-bit integer) representations, significantly reducing memory usage and computational load without substantially compromising model accuracy. At its core, quantization is all about finding the right balance between how accurate a model is and how efficiently it can run. By doing this, the model becomes much lighter and faster, making it easier to deploy on resource-constrained devices without a huge drop in performance. [6]

1) *Model Quantization*: In its original state, the CNN model was trained using 32-bit floating-point precision, which ensures high accuracy but is computationally expensive and memory-intensive—especially for embedded systems with limited resources. To address this, TensorFlow Lite's full integer quantization was used to transform all model parameters (weights and biases) and activations from float32 to int8.

Mathematically, each float value x is mapped to an integer x_{int8} using the formula:

$$x_{int8} = \text{round}\left(\frac{x}{s}\right) + z$$

where:

s is the **scale factor** (a positive float value),

z is the **zero-point** (an integer between -128 and 127),

and the operation ensures the float values are linearly projected into the 8-bit integer range. [8]

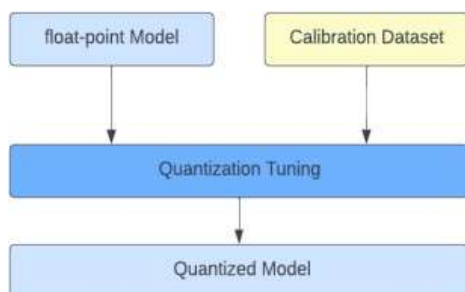


Fig. 2. Post Training Quantization

Post Training Quantization (PTQ) is a method that begins with a pre-trained floating-point model and applies quantization after the training phase is complete. It utilizes a calibration dataset to fine-tune the quantization parameters, ensuring the model performs reasonably well despite the reduced precision. Importantly, the model weights remain unchanged during this process, making PTQ a relatively quick and straightforward approach. While it is highly efficient and easier to implement—especially useful for deploying models on edge devices—it may result in a slight drop in accuracy compared to more complex techniques like Quantization Aware Training (QAT).

This quantization not only reduces the model size but also enables the model to run inference using fixed-point arithmetic, which is significantly faster on MCUs like the ESP32-S3 that lack a dedicated floating-point unit (FPU).

2) *Input Feature Quantization*: In addition to quantizing the model, the input features—linear Mel spectrograms—were also quantized from float32 to int8 to maintain consistency during inference. The spectrogram values, which originally represent power levels in decibels, are normalized and then quantized using the same scale and zero-point parameters derived from the training dataset. This ensures that the feature range during inference remains within the bounds of what the model was trained and calibrated for.

Quantizing both the model and the input features allows for fully integer inference, which is ideal for low-latency, real-time prediction on edge devices. It also helps reduce the overall memory footprint, making it feasible to deploy the model on devices with limited flash and SRAM.

F. Arduino IDE and MicroTFLite Library

The Arduino Integrated Development Environment (IDE) was used as the primary platform for developing, compiling, and uploading the firmware to the ESP32-S3 N16R8 microcontroller. The Arduino IDE provides a user-friendly interface and extensive hardware support, making it ideal for rapid prototyping on embedded systems. To execute the trained and quantized neural network model on the device, the project utilized the TensorFlow Lite for Microcontrollers (MicroTFLite) library—an ultra-lightweight inference engine designed to run machine learning models on devices with extremely limited resources, such as microcontrollers without an operating system. Robert *et al.* introduce the TensorFlow Lite Micro (TFLM), an open-source ML inference framework for running deep-learning models on embedded systems. TFLM tackles the embedded-system resource constraints such as limited memory and limited computation power. TFLM is characterized by low resource requirements and minimal runtime performance overhead. To optimize the required memory size and improve latency, TFLM applies both quantization and weight pruning [9]. The MicroTFLite library was integrated into the Arduino environment, enabling the system to interpret the .tflite model, allocate memory for tensors via a fixed-size buffer (TensorArena), and perform

inference entirely on-device using integer operations. The model, quantized to int8, was converted to a .cc format and embedded directly into the Arduino sketch, along with the preprocessed input features. By invoking MicroTFLite functions within the Arduino code, the system was able to load input data into the model's tensors, execute inference in real-time, and retrieve classification outputs with minimal latency. This setup allowed for a seamless combination of edge AI and embedded programming, showcasing the practical use of the Arduino IDE and MicroTFLite in deploying deep learning models in real-world low-power applications.

G. ESP32 S3 N16R8 Hardware Description

The ESP32-S3 N16R8 is a powerful yet resource-constrained microcontroller developed by Espressif Systems, designed for AIoT (Artificial Intelligence of Things) applications [17]. It features a dual-core Xtensa® 32-bit LX7 CPU, operating at up to 240 MHz, with support for vector instructions that enhance performance for AI and signal processing workloads. The "N16R8" variant specifically includes 16 MB of flash memory and 8 MB of PSRAM (pseudo-static RAM), which provides additional buffer space for dynamic memory allocation—crucial for handling model inference and temporary tensor storage in AI applications. Despite these resources, the ESP32-S3 remains significantly constrained when compared to typical desktop or mobile environments. The execution of deep learning models must therefore be optimized carefully, especially in terms of memory and compute efficiency.

To accommodate these constraints, the deep learning model used in this project was quantized to int8 and deployed using TensorFlow Lite for Microcontrollers (MicroTFLite), which is specifically optimized to operate within kilobytes of RAM and flash. The TensorArena, a statically allocated memory buffer, was carefully sized to fit within the available RAM, ensuring that the model's weights, input/output tensors, and intermediate activations could be processed without memory overflow. The ESP32-S3's onboard peripherals, including GPIOs and serial interfaces, also facilitated easy integration with other components and real-time monitoring via the Arduino Serial Monitor. These characteristics make the ESP32-S3 N16R8 a suitable and cost-effective platform for deploying compact AI models for real-time sound classification, as demonstrated in this horn detection system.

III. EXPERIMENTAL WORKS

A. Dataset

To develop a robust and generalizable audio classification model, a well-balanced and diverse dataset is essential. In this work, a dataset consisting of two primary classes—horn and non-horn—was constructed to support the task of environmental sound recognition in the context of smart noise-cancelling headphones. A total of 1,000 horn sound samples were initially collected, capturing a variety of real-world horn types with varying durations, intensities, and

background conditions. To enhance the diversity and quantity of the horn class and reduce the risk of overfitting, data augmentation was applied using the time shifting technique. This method involves randomly shifting the audio signal along the time axis without altering its spectral content, thereby simulating temporal variations in horn occurrences. This effectively doubled the horn class to 2,000 samples, improving the model's ability to generalize across real-world conditions where horns may appear at different points in time within an audio segment.

In parallel, the non-horn class was composed of 2,000 audio samples encompassing a wide range of environmental sounds such as human speech, bird chirping, water flow, wind, and thunder. These sounds were intentionally selected to introduce high variability and potential acoustic overlap with horn-like features (e.g., sharp onsets, high-pitched tones), thus increasing the classification difficulty and promoting more discriminative feature learning. The non-horn dataset was curated from open-source environmental sound libraries and real-world recordings to ensure diversity in background noise, sound sources, and ambient conditions.

This dataset composition not only ensures class balance—an important factor for model stability—but also enhances the model's robustness to false positives by exposing it to a wide array of non-horn audio events. Furthermore, the inclusion of augmentation techniques introduces temporal randomness, thereby improving the CNN's capacity to learn location-invariant features from spectrograms. Overall, this comprehensive dataset preparation lays a strong foundation for reliable horn sound detection in dynamically changing acoustic environments.

B. Feature Extraction and Spectrogram

The features used for training the classification model are extracted from the raw audio signals through the computation of Linear Mel Spectrograms, which serve as a compact yet information-rich representation of the audio content. These spectrograms are generated by first segmenting the time-domain audio signal using a Short-Time Fourier Transform (STFT), which converts each time frame into a frequency-domain representation. The resulting power spectrogram is then passed through a Mel filter bank, which maps the frequency components to the Mel scale, a perceptual scale that aligns with the frequency sensitivity of the human ear. Unlike traditional Mel spectrograms that apply logarithmic compression to amplitude values, the linear Mel spectrogram preserves the original power values, making it more sensitive to energy variations[11]—an important cue for detecting intense, high-energy sounds like vehicle horns.

Each resulting spectrogram is a two-dimensional matrix, where one axis corresponds to time and the other to Mel-frequency bands. This structure captures how energy in different frequency regions evolves over time, effectively encoding both spectral (what frequencies are present) and temporal (when they occur) information. These spectrograms are treated as input "images" for the convolutional neural

network (CNN), allowing the network to automatically learn relevant patterns such as frequency shifts, harmonic content, and energy bursts that are characteristic of horn and non-horn classes.

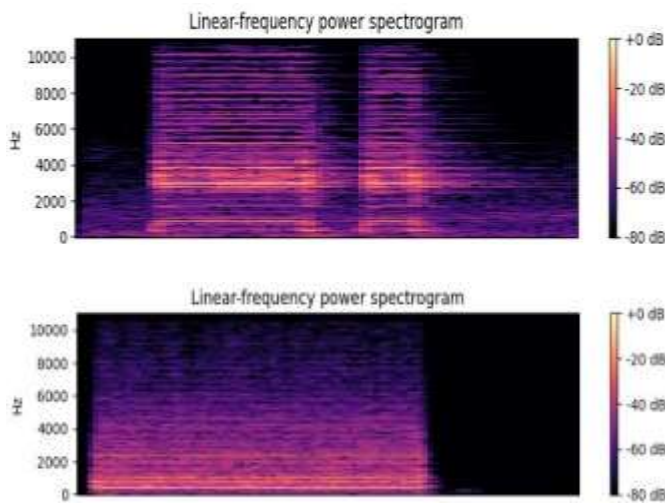


Fig. 3. Spectrogram of horn sounds

The first spectrogram, corresponding to a horn sound, exhibits two prominent bursts of energy. These are clearly visible as vertical bright bands around the 2000–5000 Hz frequency range, with sudden onsets and high intensity. These bursts are temporally localized and sharply defined, indicating the impulsive and high-energy nature of horn signals. The signal contains harmonics and shows abrupt changes in energy distribution, which are strong features for CNN-based models to capture using convolutional filters. The energy is also concentrated in mid to high frequencies, aligning with how most vehicular horns are designed for maximum perceptibility in traffic noise.

The rationale for extracting these features is twofold: first, they offer a dense representation that significantly reduces the dimensionality of raw audio while retaining key acoustic properties; and second, they provide a consistent and structured input for the CNN to perform spatial analysis. By converting variable-length audio waveforms into fixed-size spectrograms, the model is able to process and compare audio samples uniformly. This not only enhances training efficiency but also ensures that the network learns discriminative audio patterns that generalize well across diverse and noisy real-world environments.

C. Model

The core of the proposed system is a Convolutional Neural Network (CNN) designed to classify spectrogram images into two distinct categories: horn and non-horn sounds. The input to the network is the Linear Mel Spectrogram, a 2D time-frequency representation of the audio signal, which captures both the temporal evolution and frequency distribution of sound energy. Each spectrogram image passes sequentially through a series of specialized CNN layers, with each layer

learning and refining feature representations at varying levels of abstraction.

The first component of the model is a convolutional layer consisting of multiple 3×3 filters. This layer performs local operations across the spectrogram image, identifying basic features such as vertical or horizontal edges—representing sudden frequency changes or steady tones respectively.

Following the convolution, the max pooling layer down samples the feature maps by selecting the maximum value within non-overlapping regions (typically 2×2) which reduces computational complexity and introduces spatial invariance, making the model robust to minor shifts in sound events across time or frequency. A dropout layer is applied next, randomly deactivating a percentage of neurons to prevent overfitting and improve generalization.

As the spectrogram features move deeper into the network, they pass through a second convolutional layer with a larger number of filters (e.g., 64), which detects more complex and abstract acoustic patterns, such as harmonics, layered tones, or spectral textures. These higher-level features provide a more semantic understanding of the sound, allowing the model to distinguish nuanced acoustic differences between horn and non-horn samples. Again, this is followed by pooling and dropout to further compress and regularize the learned representations.

After the convolutional blocks, the resulting feature maps are flattened into a one-dimensional vector and passed through a dense (fully connected) layer with ReLU activation. This layer acts as a high-level feature integrator, combining all the localized patterns detected by earlier layers into a comprehensive feature representation. A final dropout layer (with a higher dropout rate) is added before the output to further enhance model robustness.

The final layer is a sigmoid classification layer, which computes the probability distribution over the two target classes—horn and non-horn—based on the extracted features [19]. This layer outputs the final prediction by identifying the most probable class for each input spectrogram.

Through this multi-layered architecture, the CNN effectively transforms raw spectrogram input into hierarchical feature representations, culminating in accurate and efficient classification of environmental audio events, even under noisy or unpredictable conditions. The model's layer-wise structure mirrors the way in which auditory information is processed in the human auditory system—progressively from low-level signal characteristics to high-level sound recognition.

D. Training and Testing of model

The training and testing of the proposed convolutional neural network (CNN) model are crucial phases in developing a reliable horn detection system. The primary objective of training is to enable the model to learn discriminative features

from spectrogram representations of audio data, while the testing phase evaluates the model's ability to generalize to unseen sound samples.

Initially, the dataset—comprising 2,000 horn samples (including augmented instances) and 2,000 non-horn samples—was split into training and testing subsets. This ensures that the model is trained on a sufficiently large and diverse set of examples while reserving a separate portion for unbiased performance evaluation.

During training, each spectrogram image is passed through the CNN model in mini-batches, and the model's internal weights are optimized using a supervised learning algorithm. This loss is minimized using the Adam optimizer, an adaptive learning rate method that accelerates convergence by incorporating momentum and gradient scaling. The training process involves multiple epochs, during which the model iteratively learns to adjust its convolutional filters and dense layer weights to reduce classification errors. At the end of each epoch, the model is evaluated on a validation set (a subset of the training data) to monitor performance metrics such as accuracy, loss, precision, recall, and F1-score. Techniques like dropout and batch normalization are employed to prevent overfitting and ensure that the model generalizes well to unseen data.

Once training is complete, the model is evaluated on the reserved test set using the same performance metrics. The test set contains real-world horn and non-horn sounds that the model has not encountered during training, providing a realistic assessment of its performance in practical deployment scenarios. The results from the testing phase help verify that the CNN has effectively learned relevant audio features and is capable of distinguishing horn events from various types of environmental background noise with high accuracy.

E. Model Evaluation method and Criteria

An evaluation process is performed with the test dataset from the whole dataset and the evaluation criteria used are as follows: Accuracy, Precision, Recall, f1-score. Below is the confusion matrix for the evaluation on test dataset.

$$\begin{aligned} \text{precision} &= \frac{TP}{TP + FP} \\ \text{recall} &= \frac{TP}{TP + FN} \\ \text{F1} &= \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \\ \text{accuracy} &= \frac{TP + TN}{TP + FN + TN + FP} \end{aligned}$$

Fig. 4. Formula

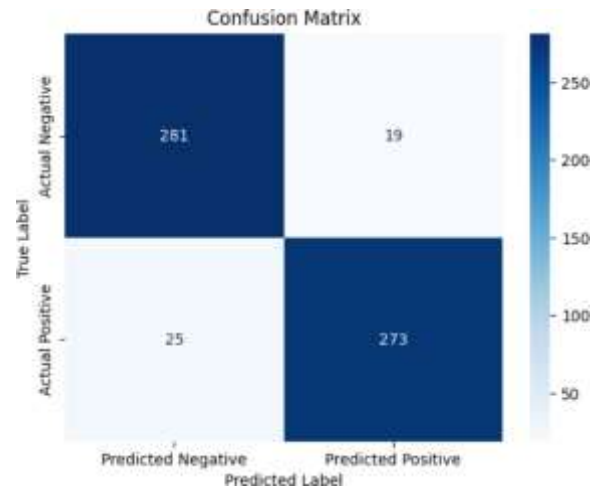


Fig. 5. Confusion matrix of prediction

With the above confusion matrix, it can be concluded that the model has had an accuracy of around 93 percent for the test dataset. Thus the model has been working well for unknown datasets.

	precision	recall	f1-score	support
0	0.92	0.94	0.93	300
1	0.93	0.92	0.93	298
accuracy			0.93	598
macro avg	0.93	0.93	0.93	598
weighted avg	0.93	0.93	0.93	598

Fig. 6. Classification report

The classification report indicates that the model is performing very well, with an overall accuracy of 93%. Both classes (0 and 1) have high precision and recall values, around 92–94%, suggesting the model is making few false positives and false negatives. The F1-scores for both classes are balanced at 0.93, indicating strong and consistent performance across categories. The macro and weighted averages are also 0.93, showing that the model handles both classes equally well and that there is no significant class imbalance affecting performance. Overall, the model demonstrates reliable and robust classification capabilities, making it suitable for practical applications where high accuracy and balanced performance are important.

F. Quantization

The trained model was quantized using a representative dataset to perform full integer (int8) quantization, which ensured accurate calibration of input activations and preserved the model's predictive performance [6]. This process reduced the model size significantly by converting all weights and activations from 32-bit floating-point (float32) to 8-bit integer (int8), making it highly efficient for deployment on embedded hardware. The quantized model was then

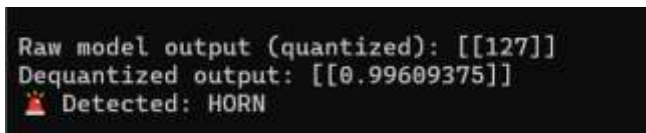
exported in the TensorFlow Lite for Microcontrollers (MicroTFLite) format, which is specifically optimized for execution on devices with limited memory and computational resources. In addition to the model, the extracted input features—linear Mel spectrograms—were also quantized to int8 format to ensure compatibility during runtime inference.

G. Deployment in ESP32

To deploy the model on the ESP32-S3 N16R8 microcontroller, the Arduino development environment was used to create and upload the firmware [16]. Both the quantized model and the preprocessed input features were converted into C array (.cc) files using the xxd utility and embedded into the firmware. The MicroTFLite interpreter was initialized within the Arduino sketch, along with a dedicated memory buffer (TensorArena) to handle tensor operations. During inference, the system loads the input spectrogram into the model's input tensor, invokes the model, and interprets the output to classify the sound as either "horn" or "non-horn." The classification result is then displayed via the Serial Monitor. This efficient implementation enables real-time, low-latency sound classification directly on the device, without the need for external computation, demonstrating the practicality of deep learning-based audio detection in embedded edge applications.

H. Results

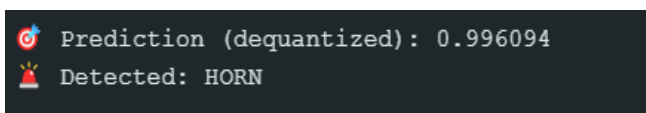
The quantized output features and predictions were compared to ensure the correctness of the predictions. Following figure shows the prediction of the audio file using the python code:



```
Raw model output (quantized): [[127]]
Dequantized output: [[0.99609375]]
🔊 Detected: HORN
```

Fig. 7. Python code output

The following figure shows the output prediction run by the ESP32 hardware using the Arduino IDE:



```
🔊 Prediction (dequantized): 0.996094
🔊 Detected: HORN
```

Fig. 8. Arduino IDE output

By comparing the Fig. 4 and Fig. 5, we could see that the prediction accuracy matches thus giving the conclusion that the model is properly quantized and also the features are correctly extracted and quantized to fit in the ESP32 hardware.

IV. CONCLUSION

In this work, a robust and lightweight deep learning-based system was developed for detecting horn sounds in noisy environments using spectrogram-based feature extraction and convolutional neural networks (CNNs). The system was trained on a carefully curated dataset comprising both horn and non-horn sounds, including various environmental noises such as human speech, birds, etc. Through augmentation techniques like time shifting, the dataset was enhanced to improve model generalization and robustness.

A key aspect of the system involved transforming raw audio signals into linear Mel spectrograms, which preserve the temporal and frequency characteristics of sounds in a visually interpretable format. These spectrograms served as the primary input to the CNN model, enabling the network to learn spatial patterns relevant to distinguishing between horn and non-horn audio events. The CNN architecture was carefully designed with multiple convolutional, pooling, and dense layers to progressively extract and process hierarchical features from the input spectrograms. The model demonstrated strong classification performance, effectively capturing subtle differences in frequency distributions and temporal variations between the two classes.

To enable real-world deployment, the trained model underwent full integer quantization using a representative dataset. This quantization significantly reduced the memory and computational requirements of the model without substantially impacting accuracy. The quantized model was converted into a MicroTFLite format and integrated into the ESP32-S3 N16R8 microcontroller via the Arduino IDE. Both the model and the input features were embedded into the firmware as C arrays, and real-time inference was carried out directly on the device. The successful execution of the model on the ESP32 platform highlighted the feasibility of deploying deep learning-based audio classification systems on low-power, resource-constrained embedded devices.

The system's design and implementation demonstrate a practical and scalable approach to environmental sound detection, especially in applications such as smart noise-cancelling headphones, or urban acoustic monitoring. By enabling real-time detection of safety-critical auditory events like vehicle horns, the solution contributes to enhanced situational awareness in noisy environments.

Future work may include expanding the dataset to encompass a broader range of horn types and non-horn noises, improving noise resilience through advanced signal processing or attention-based models, and integrating on-device audio preprocessing to enable end-to-end inference directly from raw microphone input. Additionally, power consumption and inference latency can be further optimized for continuous operation in wearable or battery-powered systems. Overall, this work establishes a strong foundation for deploying intelligent, low-cost, and real-time sound classification systems at the edge.

REFERENCES

- [1] S. N. C. K. S.Chu, "Environmental Sound Recognition with Time-Frequency Audio Features," *IEEE Transactions on Audio, Speech, and Language Processing*, pp. 1142-1158, 2009.
- [2] J. L. S. T. Boyang Zhang, "Audio Recognition using Mel Spectrograms and convolutional neural networks," *IEEE*, 2010.
- [3] K. J. Piczak, "Environmental sound classification with convolutional neural networks," *IEEE*, 2015.
- [4] M. A. S. L. K.-R. M. W. S. Soren Becker, "Interpreting and explaining deep neural networks for classification of audio signals," 2018.
- [5] C. D. S. Q. J. L. a. S. D. W. Dai, "Very deep convolutional neural networks for raw waveforms," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2017.
- [6] D. Chenna, "Edge AI: Quantization as the key to on-device smartness," *International Journal of Artificial Intelligence & Applications (IJAIAP)*, vol. Volume 2, no. 2, Issue 01, pp. pp. 58-69, 2023.
- [7] Z. L. R. B. a. D. M. R. Ding, "Quantized deep neural networks for energy efficient hardware-based inference," *IEEE*, p. pp. 1-8, 2018.
- [8] S. K. B. C. M. Z. M. T. A. H. H. A. D. K. B. Jacob, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.
- [9] S. G. Erez Manor, "Custom Hardware Inference Accelerator for TensorFlow Lite for Microcontrollers," in *IEEE Access*, 2022.
- [10] W.-H. T. Y. F. M. G. Van-Thuan Tran, "End-to-End Train Horn Detection for Railway Transit Safety," in *MDPI*, 2022.
- [11] J.-H. K. C. K. Soonshin Seo, "Convolutional Neural Networks Using Log Mel-Spectrogram Separation for Audio Event Classification with Unknown Devices," in *IEEE Access*, 2022.
- [12] L. C. L. T. Matthias Meyer, "EFFICIENT CONVOLUTIONAL NEURAL NETWORK FOR AUDIO EVENT DETECTION," Switzerland, 2017.
- [13] M. C. Josef Palecek, "Emergency Horn Detection Using Embedded Systems," in *IEEE 14th International Symposium on Applied Machine Intelligence and Informatics*, 2016.
- [14] C.-C. L. J.-W. W. W.-C. C. Chiun-Li Chin, "A Wearable Assistant Device for the Hearing Impaired to Recognize Emergency Vehicle Sirens with Edge Computing," in *MDPI*, 2023.
- [15] N. C. S. N. J. M. d. M. Cleyton Aparecido Dim, "HornBase: An audio dataset of car horns in different scenarios and positions," *Elsevier*, 2024.
- [16] P. F. P. S. Marek Babiuch, "Using the ESP32 Microcontroller for Data Processing," in *IEEE*, Krakow-Wieliczka, Poland, 2019.
- [17] A. S. Y. V. Alexander Maier, "Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things," in *IEEE*, Wrexham, UK, 2019.
- [18] M. Tuttle, "Efficient convolutional neural network inference on microcontrollers," Illinois, 2022.
- [19] S. S. A. A. Siddharth Sharma, "Activation Functions in Neural Networks," *International Journal of Engineering Applied Sciences and Technology*, vol. Vol. 4, no. Issue 12, pp. Pages 310-316, 2020.