CSEM CHALLENGE: ON-DEVICE KEYWORD SPOTTING WITH TINYML

Luc Reveyron, 15.06.2025

1. Introduction

This report summarizes our approach to designing, training, optimizing, and deploying a keyword spotting model on an ESP-EYE microcontroller. The model detects a set of six keywords and classifies audio samples without those keywords as “unknown”.

2. Literature review

TinyML is a field of machine learning that aims to deploy AI on resource-constrained microcontrollers [1]. One classic application is audio keyword spotting (KWS) that enables real-time, low-power and privacy-preserving speech recognition. The input audio is analyzed locally on a microcontroller without relying on a remote cloud-server for processing. In recent years, many neural network models have been trained [2] on Google speech commands dataset [3]. To reduce memory footprint and computed resources required, the input audio signals are preprocessed to extract meaningful features such as mel spectrograms or mel frequency cepstral coefficients (MFCCs) [4] and model weights are converted from high-precision floating-point numbers (such as 32-bit floats) to lower-precision data types (such as 8-bit int) through network quantization [2].

3. Model architecture and training

3.1. Dataset

To reduce development time, we used the widely adopted Google speech commands dataset [3] instead of creating our own dataset. From this corpus, we selected the following six labels as keywords: “yes”, “no”, “stop”, “go”, “on”, “off”. All other labels were grouped into a single “unknown” class.

For balance, we retained all available samples for the six keywords and randomly selected an equal number of “unknown” samples, resulting in a dataset composed of 50% keywords and 50% unknown samples. This ratio was chosen to ensure a diverse representation of the “unknown” class. Although other projects have used lower proportions of “unknown” samples (e.g., 20% in [5]), we kept this distribution as it yielded satisfactory training performance.

The dataset was then partitioned into training, validation and testing sets using a 70/15/15 split, as recommended by [6].

3.2. Data augmentation

To improve our model robustness to real-world noise conditions, we augmented our training dataset by adding background noise to clean audio samples. For each original audio file, multiple noisy versions were generated using random background clips and varying signal-to-noise ratios (SNRs) within a 15 to 25 dB range.

3.3. Preprocessing

To extract meaningful features for model input, we computed the MFCCs for each sample, following the approach in [2]. This parametric representation of acoustic signals reflects human auditory perception [7], reduces data dimensionality and offers robustness to minor speech or background variations [8].

Standard hyperparameters were used for the transformation, including Short-Time Fourier Transform (STFT) frame length and hop length, number of Mel bands, frequency bounds and the number of cepstral coefficients kept, as described in [9]. The sampling frequency was set to 16 kHz, a common choice in keyword spotting tasks [3].

3.4. Model training

We implemented and evaluated three neural network architectures for keyword spotting from [2] to compare their performance on our dataset. First, we tested a long short-term memory (LSTM) model, the lightest architecture in [2]. Next, we experimented with a hybrid convolutional neural network (CNN) and LSTM model, reported as the most performant in [2]. However, the LSTM component was removed due to incompatibility with the TensorFlow Lite for Microcontrollers (TFLite Micro) optimization module. The final model is CNN-based, efficiently capturing local temporal and spectral correlations in the speech features.

4. Training results

The three implemented architectures - the LSTM-based model, the hybrid model and the CNN-based model - achieved test set accuracies of 90.8%, 94.4% and 91%, respectively.

Figures 1 and 2 illustrate the training process and performance of our CNN-based model, showing no evidence of underfitting or overfitting.

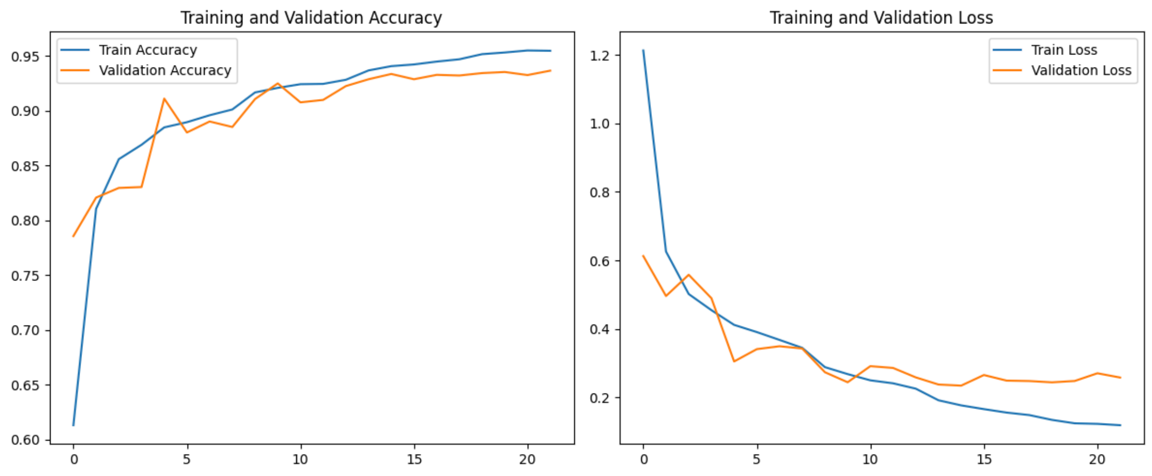

Figure 1 - Training and validation accuracy and loss over epochs for the CNN-based model

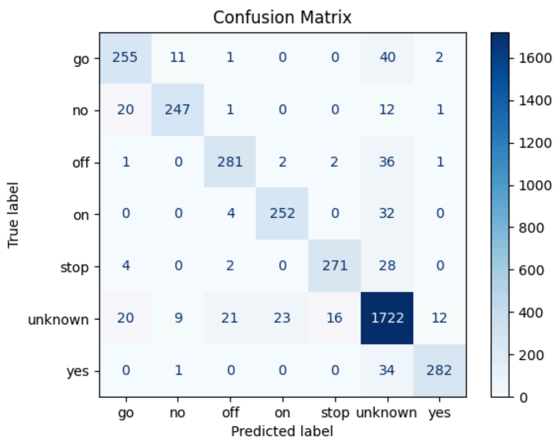

Figure 2 - Confusion matrix on the test set for the CNN-based model

5. Model optimization

To reduce the model size before deployment, we perform the quantization of the CNN-based model using TFLite Micro, which includes weight, activation and bias quantization, operator fusion and constant folding. The model size shrinks from 4.54 MB to 0.39MB, which corresponds to a 91.4% size reduction, while retaining a 92.5% testing accuracy.

6. Deployment on hardware

We deployed the optimized model on an ESP-EYE microcontroller. In a dedicated thread, the microphone signal is sampled at 16 kHz and stored as 1-second samples in a buffer. The main thread retrieves these samples from the buffer, computes the log-Mel energy values using the TFLite Micro `FrontendProcessSamples` function, and extracts MFCCs via Discrete Cosine Transform (DCT). The resulting features are then passed to the CNN model. However, the model raw output remained nearly constant, producing the same predicted label regardless of microphone input.

7. Discussion and possible future work

Due to time constraints, we were unable to fully debug the C++ inference implementation and thus could not evaluate the model performance on the microcontroller. We suspect the issue lies in the C++ preprocessing pipeline, as Figure 3 reveals significant differences between the MFCC distributions of the validation dataset and the live microphone data.

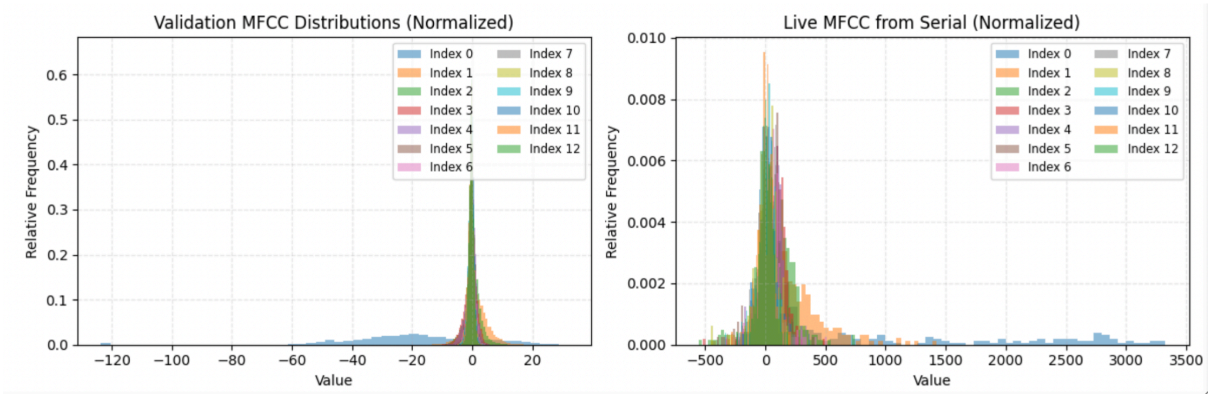

Figure 3 - MFCC distributions for the validation dataset (left) and live samples (right)

8. Conclusion

We successfully designed, trained, and optimized a keyword spotting model for six keywords with data augmentation. However, deployment on the ESP-EYE microcontroller was only partially completed due to an unresolved issue in the C++ preprocessing pipeline, which could not be addressed within the available time.

9. References

1. Lin, Ji, Ligeng Zhu, Wei-Ming Chen, Wei Chen Wang, and Song Han. 2023. “Tiny Machine Learning: Progress and Futures [Feature].” *IEEE Circuits and Systems Magazine* 23 (3): 8–34. https://doi.org/10.1109/mcas.2023.3302182.

2. Zhang, Yundong, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. “Hello Edge: Keyword Spotting on Microcontrollers.” *ArXiv (Cornell University)*, November. https://doi.org/10.48550/arxiv.1711.07128.

3. Warden, Pete. 2018. “Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition.” ArXiv.org. April 9, 2018. https://doi.org/10.48550/arXiv.1804.03209.

4. Toussaint, Wiebke, Akhil Mathur, Aaron Yi Ding, and Fahim Kawsar. 2021. “Characterising the Role of Pre-Processing Parameters in Audio-Based Embedded Machine Learning.” *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, November, 439–45. https://doi.org/10.1145/3485730.3493448.

5. “Speech Command Recognition Using Deep Learning - MATLAB & Simulink.” n.d. Www.mathworks.com. https://www.mathworks.com/help/deeplearning/ug/deep-learning-speech-recognition.html.

6. Bishop, Christopher. 2006. *Pattern Recognition and Machine Learning*. Springer Verlag.

7. Muda, Lindasalwa, Mumtaj Begam, and I. Elamvazuthi. 2010. “Voice Recognition Algorithms Using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques.” *Arxiv.org*, March. https://doi.org/10.48550/arXiv.1003.4083.

8. Faraji, Farnood, Yazid Attabi, ChampagneBenoît, and Wei-Ping Zhu. 2020. “On the Use of Audio Fingerprinting Features for Speech Enhancement with Generative Adversarial Network.” *ArXiv (Cornell University)*, July. https://doi.org/10.48550/arxiv.2007.13258.

9. Jurafsky, Dan , and James H. Martin. 2018. “Speech and Language Processing.” Stanford.edu. 2018. https://web.stanford.edu/~jurafsky/slp3/.