Spoken Language Recognition - Tensorflow Lite for Microcontrollers

A simple convolutional neural network to recognize which language a person is speaking, implemented on an Arduino Nano 33 BLE Sense, with the Tensorflow Lite for Microcontrollers optimized framework

Computer Science and Engineering project for the course: **Hardware Architectures for Embedded and Edge AI**

A project made by:

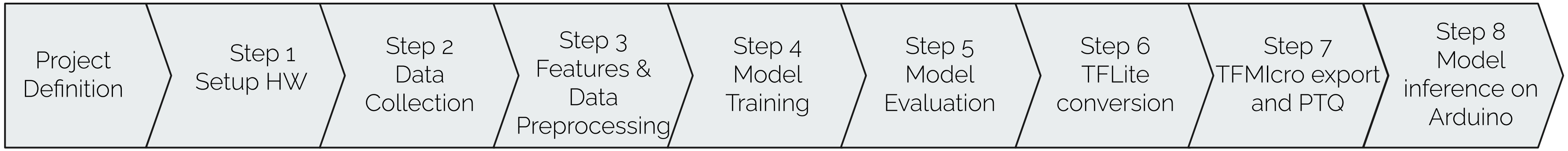
Simone Giampà

Claudio Galimberti

Special thanks to the contributors to the dataset collection:

Simone, Claudio, Elena, Sofia, Rita, Paolo, Lorenzo, Francesco, Gabriele, Alessandro, Guia, Elisa, Josephine, Thiago, Donia, Omar, Chiara, Angela, Lorenzo, Stefano, Martina, Andrea, Davide, Elisa, Paolo, Bob, GioPizzi, PDVG, HumanSafari, Andrea Lorenzon, Yotobi, Synergo, RealEngineering, RealScience, and many other documentarists, teachers, interviewers and youtubers.

Our approach with the project



- Step 1: hardware construction and arduino libraries setup
- Step 2: dataset collection with the device
- Step 3: MFCC features engineering and data preprocessing pipeline
- Step 4: CNN models training
- Step 5: Model evaluation on test sets
- Step 6: Conversion to TFLite and post training quantization
- Step 7: Export to TFMicro and Tensor allocation and model setup
- Step 8: Inference on the device with Arduino and TFMicro Framework

Project definition

Objective of the project: develop a system that is able to recognize in which language is a person speaking a person using a short recording of voice audio.

TFMicro framework used: **the approach followed** is the most “customizable” one. We trained a convolutional neural network in python with Tensorflow, then converted it to a Tensorflow Lite model using its Converter and finally exported it to a C byte array included in the Arduino code.

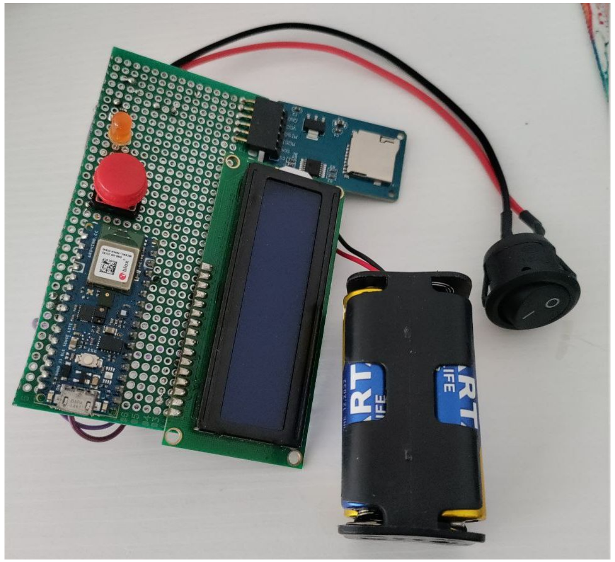
Edge Impulse: this approach **wasn’t feasible** because of the lack of customizability that it offered and that our project required. One of the main problem we faced was the pre processing part, that required a lot of computing power and the free version of the service supported only 20 minutes of CPU power, which wasn’t enough for our experiments.

Hardware Device setup, construction and utility

The development of an **hardware device** was done to have a **ready-to-use** and **versatile tool** to record the audio everywhere, and also to have a **more practical experience** and to put into work some of the skills acquired in previous courses.
The hardware device proved to be very useful also for the **inference**.

The HW is composed of a soldered circuit board to which are placed:

1. Arduino connectors
2. LCD display
3. Micro SD card reader
4. Push button
5. Portable batteries



Audio recordings data collection: setup

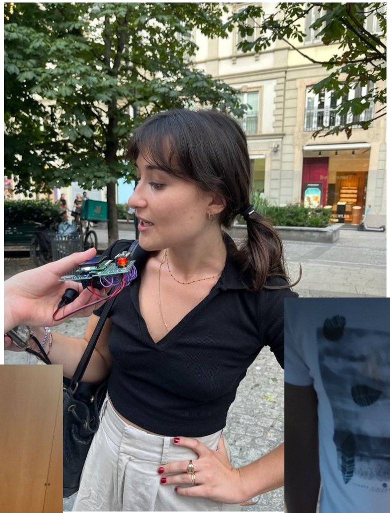
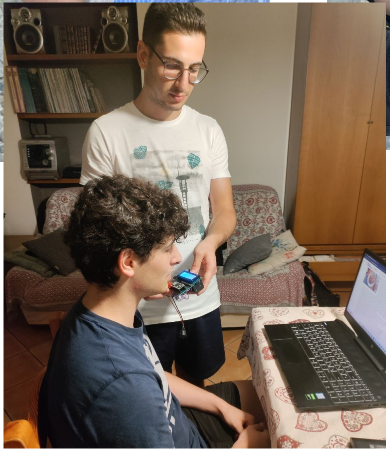
Acquisition

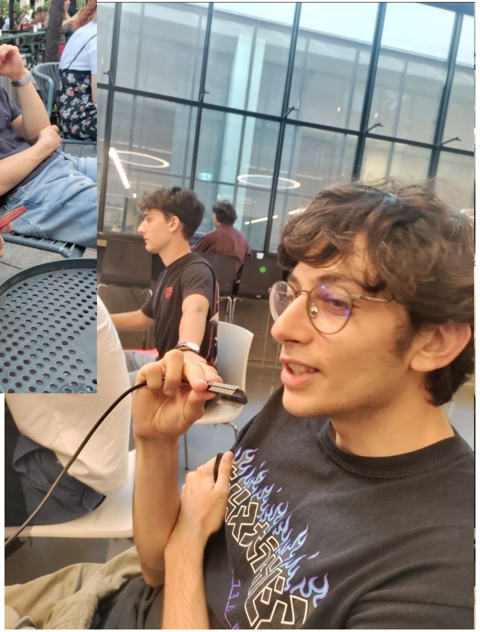
- Entirely done using the arduino microphone sensor.
- One minute audio recordings of a random topic that people talked about.
- Dataset improvement by asking people to provide whatsapp audio and recording youtube conversations.
- The recordings are saved in the micro SD card and then imported in the python environment.

Training-Validation-Test split

- Overlapping sliding windows of data extracted from 1 minute audio samples
- Splitting based on the speaker: 75% training 25% validation split approximately
- The validation set contains only unseen data so that the accuracy values are more representative of the model’s general understanding.

Audio recordings data collection: the fun part

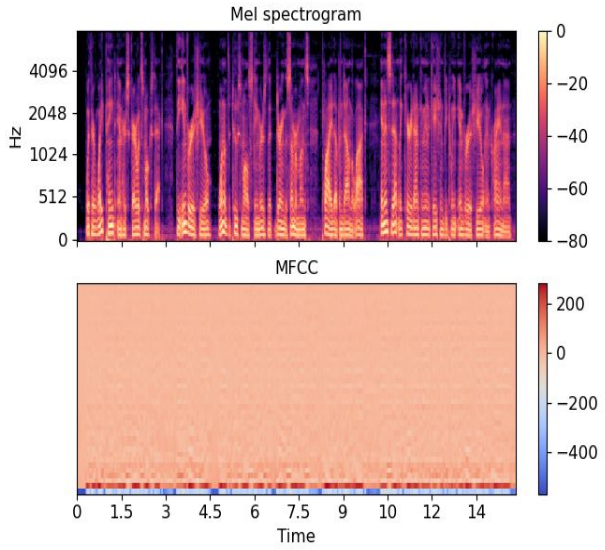


Features for speech recognition: MFCC

MFCC stands for **Mel-Frequency-Cepstral-Coefficient** and they are a feature extraction technique in speech and audio processing. They are extremely useful as features and as a method for efficient dimensionality reduction.

The term “Mel” refers to **Mel scale**, which is a perceptual scale of pitches that is more aligned with the way humans hear sounds. It is used to convert linear frequency scale of audio signals into a logarithmic scale that approximates human auditory perception.

Cepstral Coefficients: a mathematical technique representing the short-term power spectrum of a signal. Cepstral coefficients highlight the **characteristics of the human speech**, such as the timbre and pitch.



Audio data preprocessing and its main challenge

This represented one of the main challenge of the project and it’s the discriminant for the selection of the approach that we actually used.

Preprocessing audio samples with the same exact code in C++ for arduino and Python for the neural network training is a mandatory requirement since the preprocessing has to be unique.

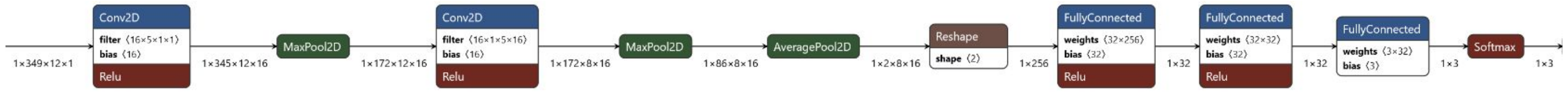
It is required to have the same exact preprocessing pipeline in order to avoid discrepancies in the features computation on two different target platforms.

Extreme memory optimization in the C++ code for the audio preprocessing is employed in order to maximize the audio length for inference

Convolutional Neural Network

Final network: convolutional neural network with 1D and 2D convolutions, global average pooling and fully connected layers.

Many designs for a convolutional neural network were tested: CNN with skip connections (concatenative and additive) demonstrated no improvement on the accuracy.



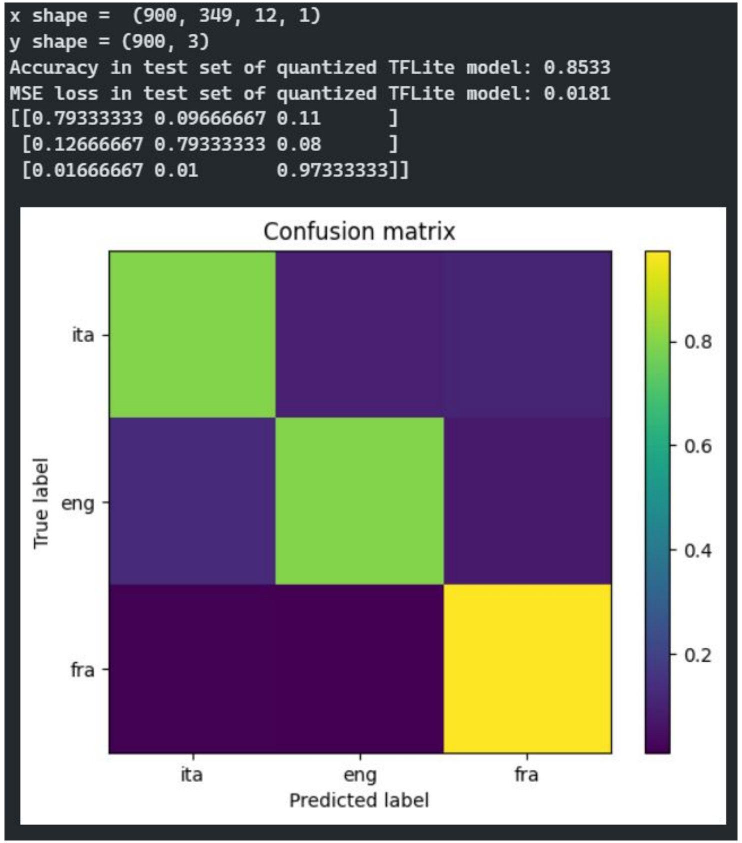
Exporting to TFLite and TFMicro

Post training full-integer (int8) quantization

Tensor Arena memory space constrained the model's size and the first 2 layers (the ones having the most parameters)

Model size: 10771 parameters

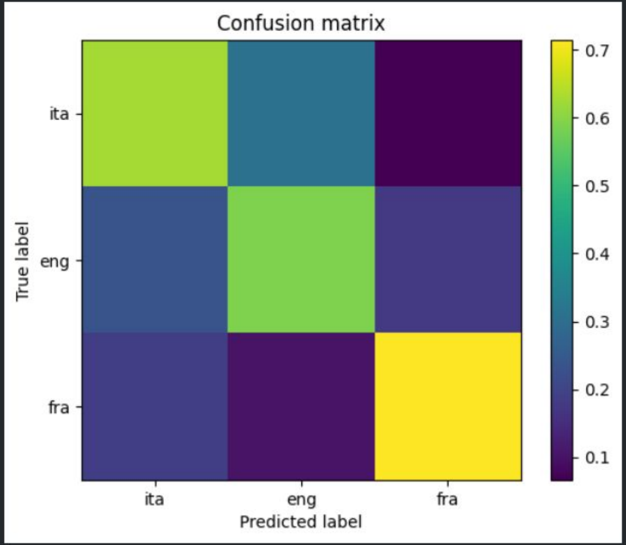
Tensor arena: 115KB memory



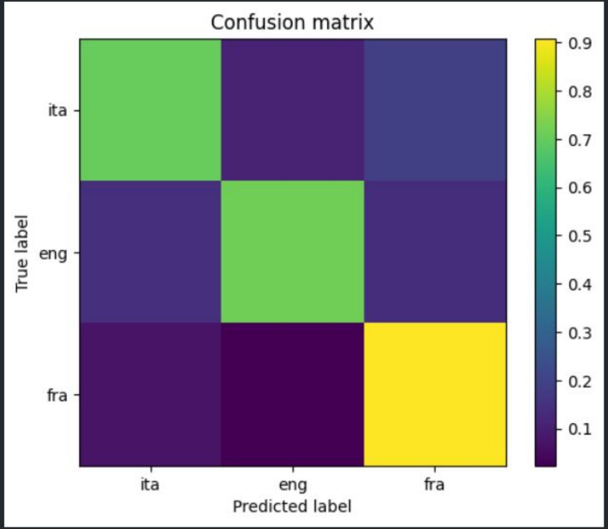
```
x shape =  (900, 349, 12, 1)
y shape = (900, 3)
Accuracy in test set of quantized TFLite model: 0.8533
MSE loss in test set of quantized TFLite model: 0.0181
[[0.79333333 0.09666667 0.11      ]
 [0.12666667 0.79333333 0.08      ]
 [0.01666667 0.01       0.97333333]]
```

Model’s prediction accuracy in test sets

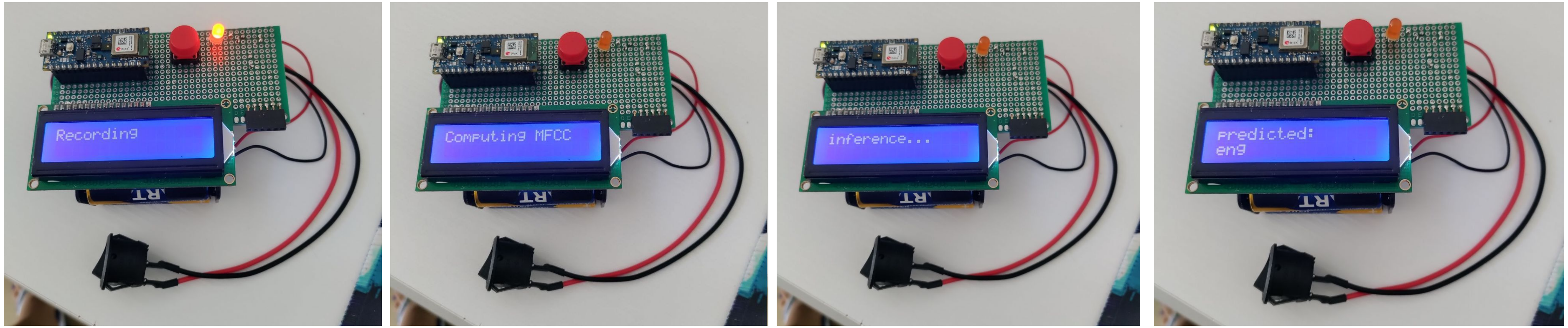
```
evaluating accuracy of quantized model on test set: unheard speakers
x shape =  (1260, 349, 12, 1)
y shape = (1260, 3)
Accuracy in test set of quantized TFLite model: 0.6429
MSE loss in test set of quantized TFLite model: 0.0237
[[0.62619048 0.30714286 0.06666667]
 [0.23571429 0.58809524 0.17619048]
 [0.18571429 0.1        0.71428571]]
```



```
evaluating accuracy of quantized model on test set: known speakers with noisy audio recordings
x shape =  (1170, 349, 12, 1)
y shape = (1170, 3)
Accuracy in test set of quantized TFLite model: 0.7752
MSE loss in test set of quantized TFLite model: 0.0198
[[0.7025641  0.10769231 0.18974359]
 [0.14615385 0.71538462 0.13846154]
 [0.06923077 0.02307692 0.90769231]]
```

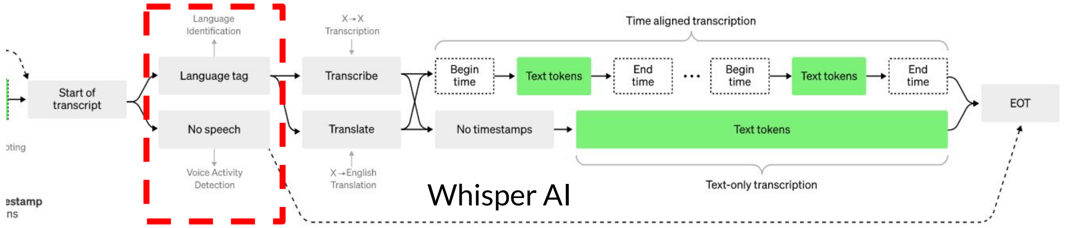


Inference with the trained model on Arduino



Recording audio sample >> Features computation >> TFMicro Model inference >> Output prediction

Market opportunities



Google Translate recognizes languages and translates with online connection using their servers

OpenAI's Whisper model uses a language recognition module before translating audio data

Customer support and call centers where the language has to be identified from the user's words in order to call an appropriate human interpreter and redirect the conversation

Language recognition robust to dialects and accents differences to preserve cultural heritage

Language independent interaction with IoT devices in places where multiple languages are spoken on a daily basis, such as voice assistants in airports or train stations

Data privacy: tackling with the potential issues

Absence of personally identifiable data: speakers were warned to talk about something without mentioning personally identifiable or sensitive information, such as addresses, private names, banking information, sensitive life info, etc.

This is needed to avoid the potential harm of reverse engineering dataset information from the model's inference predictions.

To save the recordings and use them for model training, **consent must be given.**

Requirements for the audio samples quality: clear speech with few interruptions and no onomatopoeic sounds or highly noisy environment.