



UNIVERSITÀ DEGLI STUDI DELLA CALABRIA
DIPARTIMENTO DI INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA E SISTEMISTICA

MASTER'S DEGREE IN COMPUTER ENGINEERING FOR
THE INTERNET OF THINGS

Thesis

**Edge Intelligence for speech recognition: a use
case based on Edge Impulse and TinyML**

Supervisors

Dott. Claudio Savaglio
Prof. Giancarlo Fortino
Prof. Pietro Manzoni

Student

Alfredo Carnevale
Matr. 224433

Academic Year 2022-2023

Abstract

In the realm of Information Technology (IT), Speech Recognition (SR) stands as an ever-evolving field, fueled by the growing market demand for customer-centric services and products. Within this domain, existing technologies have undergone adaptations to create systems with the capability to understand human speech. The integration of Machine Learning (ML) into data analysis has introduced a groundbreaking paradigm shift, showcasing how machines can acquire extensive knowledge through algorithms, allowing them to detect emotions in text, voice, or images. Recently, the emergence of TinyML, which tailors conventional ML techniques for devices with limited computational capabilities, such as embedded systems, has unlocked previously unexplored opportunities. Notably, TinyML has enabled the creation of smart sensors that can process data acquired from the surrounding environment in real-time, including the recognition of spoken words.

In this thesis is presented a system designed to assess the degree of appreciation for artworks in a museum, wildlife in a natural park, or any service provided to people. It accomplishes this using smart sensors strategically positioned to capture spoken words from users. We leveraged Edge Impulse, a new innovative platform used to develop Artificial Intelligence (AI) applications on edge devices, giving us the possibility to design and implement a new generation system. Starting from data acquisition and extending to deployment on embedded devices, specifically the Arduino Portenta H7, Edge Impulse offers the flexibility to use pre-built processing blocks for implementation or to design custom models using Python and TensorFlow for learning.

The performances of the developed system have been evaluated in terms of computational complexity and reliability of the results, showing that the lightweight algorithms, supported by microcontrollers, can be used in SR delivering results and performances comparable to traditional computing nodes. Finally, the system has been deployed and tested on Arduino Portenta achieving highly satisfactory results. An average of 80.85% of accuracy has been reached, demonstrating the potential for its practical deployment in public spaces.

Contents

1	Introduction	3
1.1	Objectives	5
1.2	Thesis structure	5
2	Background	6
2.1	Artificial Intelligence	6
2.2	Machine Learning	7
2.2.1	Tiny Machine Learning	9
2.3	Neural Network	10
2.3.1	Convolutional Neural Network	10
2.4	Edge Computing	11
2.4.1	Edge Intelligence	12
2.5	Technologies and hardware	13
2.5.1	TensorFlow	13
2.5.2	TensorFlow Lite	14
2.5.3	Edge Impulse	14
2.5.4	Arduino Portenta H7 and Vision Shield	16
3	Problem description and System design	18
3.1	System Design	19
3.2	Training process and system modelling	21
3.3	Impulse design	23

3.4	Arduino sketch design	24
4	System implementation	26
4.1	Data acquisition	26
4.2	Components Implementation	29
4.2.1	Audio processing component	29
4.3	Data Elaboration	31
4.3.1	Features extraction and data analysis	31
4.4	Neural Network Implementation	34
4.4.1	Training phase	36
4.5	First Model evaluation	37
4.5.1	Training results	37
4.5.2	Testing results	40
4.6	Arduino Portenta H7 Deployment	41
5	Testing phase	45
5.1	First scenario - Silent background	47
5.2	Second Scenario - Voices background	49
5.3	Third Scenario - White Noise Background	50
5.4	Results comparison	51
6	Conclusions	53

Introduction

The **Internet of Things** (IoT) is a system of heterogeneous devices, ranging from conventional computing systems to intelligent everyday objects, with the ability to transfer data over different kinds of networks [1]. In recent years, the proliferation of the IoT has resulted in the interconnection of billions of entities, giving rise to various challenges. Among these challenges, two prominent issues have emerged: heterogeneity and inaccuracy. Heterogeneity arises from the diverse data types and attributes used to describe the 'things' in the IoT ecosystem. Inaccuracy, on the other hand, stems from imprecise or erroneous data generated or collected by IoT devices. The sheer volume of data generated in real-time within dynamic networks, commonly referred to as 'Massive Real-Time Data,' has further compounded these challenges [2]. In light of these developments and in direct response to the challenges posed by IoT, Machine Learning (ML) has emerged as a promising technology with the specific goal of enabling huge volumes of data produced by IoT devices, allowing efficient data processing, analysis and interpretation.

In the past years, the significance of people's appreciations, opinions, and emotions has grown significantly, prompting numerous studies in the realm of Information Technology (IT) to emphasize these aspects. Delving deeper into research becomes crucial to obtain effective and reliable findings regarding the satisfaction levels with products or services. Hence, the results achieved can be used either to increase users' satisfaction or to rectify any deficiencies or inaccuracies within the service itself. In today's society, both public and private organizations are highly

motivated to collect feedback, striving for excellence in both quantity and quality. This feedback has become the primary fuel for assessing the level of appreciation for various subjects, companies, products, or any objects, whether they are presented physically or virtually to the public. Appreciation data can be of many types and sources, such as post-restaurant visit ratings or sharing comments on social networks. The primary aim of scientific research in this specific field is to leverage such data effectively to achieve optimal outcomes in shaping the emotions and perspectives of users. However, discerning an individual's feelings and emotions, whether positive or negative, is a complex job and traditional software and applications are very often unable to optimally perform such an analysis. In specific places such as museums, zoos, and amusement parks it is very often difficult to understand the degree of appreciation of the attractions by users since there are no advanced software systems capable of acquiring the emotions experienced but, usually, there are only simple push-button panels with green or red buttons which indicate, relatively, whether the attraction was to your liking or not, or there are simple questionnaires to fill out. In this direction, therefore, it is feasible to utilize tools such as cameras to capture individuals' facial expressions and microphones to capture spoken remarks, facilitating the categorization of their feelings and opinions, all while adhering to the constraints imposed by privacy laws. ML, and, more broadly, AI serve as vital strategic components for the successful implementation of the aforementioned applications, acting as technology enablers in the field of Sentiment Analysis (SA)[3].

Sentiment Analysis (SA) can be implemented either in the Cloud or on a remote server to handle processing and inference, offering a contemporary and feasible solution utilizing today's technologies. However, this approach may come with its own set of challenges, including significant network resource consumption and the potential for latency that could disrupt system functionality. For example, transmitting results over the network can lead to the loss of some sentiment inferences. Remarkably, Tiny Machine Learning (TinyML) remains relatively unexplored in the context of SA. Despite its recent emergence, TinyML has yielded promising results. It harnesses the capabilities of microcontrollers to reduce energy consumption significantly and circumvent the need for large, resource-intensive deep learning models typically hosted on traditional servers.

1.1 Objectives

This thesis proposal introduces a novel application that leverages TinyML techniques to perform Speech Recognition (SR) for eventually SA applications, presenting a novel approach to addressing the aforementioned limitations typically linked with cloud-based solutions. The work will be structured starting from data acquiring to deployment, on embedded devices, which will take place in strategic point of interest such as masterpieces in a museum or in natural parks. The processed information will then be useful to those who have a wonder in their management to understand the degree of appreciation of people. For this reason, leveraging the available data presents an opportunity to enhance future user experiences. To achieve this, we will develop a specialized software system utilizing TinyML-powered AI tools. This software will be designed to analyze audio data and discern a lexicon of words associated with positive and negative sentiments. To assess the system's performance in terms of both quality and complexity, we will deploy the developed model on a low-power board equipped with a microphone.

1.2 Thesis structure

This thesis is organized as follows: Chapter 2 discusses the paradigms, technologies and software used to implement the system and the hardware over which the system will be deployed. Chapter 3 focuses on the problem description and shows the design of the system. Chapter 4 discusses the implementation details in order to create the ML model. In Chapter 5 all the tests carried out on the device, in real scenarios, are shown and discussed, while conclusions and future developments are reported in Chapter 6.

Chapter 2

Background

The solution proposed in this thesis is based on novel computing paradigms such as **AI, ML, TinyML, Neural Network Edge Computing**, which will be briefly introduced in this Chapter.

Likewise, will be presented some software and frameworks used for implementing ML algorithms like Keras, TensorFlow and TensorFlow Lite (implementing the mathematical operations that are the basis of ML and allow the user to create programs interacting only with high languages level as Python over computers and microcontrollers), and finally the tool **Edge Impulse**, a recent software that offers a web-environment to perform all operations related to the development of embedded ML models, from data collection to deployment of the application on the edge device.

2.1 Artificial Intelligence

AI is the study of building or programming computers to enable them what humans minds can do [4]. AI is a discipline belonging to IT which has evolved with the birth of first computers, and it is used both in the context of business process and also in the daily life. The Turing test marked the birth of AI because for the first time, he focused the problem of a machine's ability to think. During the years this branch of computer science has grown and there are many studies carried out in search of continuous improvements. Obviously, this meets heavy

ethical-philosophical criticism. The AI has evolved and then branched into several minor disciplines and trying to make a classification, we can distinguish, the following interest area of AI:

- Voice
- Natural Language
- Automatic Reasoning
- Knowledge
- Learning
- Robotics
- Vision

2.2 Machine Learning

ML stands out as one of the most prominent subsets within the field of AI, garnering extensive attention from researchers, including computer scientists and mathematicians, over the years. An ML algorithm is a computational process that uses input data to perform a desired task without being literally programmed [5]. Fundamental is the process of adaption or training in which the algorithm, through the training data, configures itself to respond to input already seen or new trying to associate each input to a desired output and the executing a process of generalization.

The ML algorithm to learn use data collections, called **datasets**, which can be created or acquired from the real world. It's possible to distinguish two types of ML [6]:

- **Supervised Learning**
- **Unsupervised Learning**

A classic and simple example of supervised learning is the classification of an input in labels defined a priori. This classification is opposed to that typical on unsupervised learning, the algorithm studying the initial data defines clusters dividing the various data according to their characteristics. In this case there are no labels defined a priori but groups of input grouped by similarities. Therefore, the techniques cited before find applications in practically any field and the type of learning in which take place the thesis proposal is in the field of **Supervised Learning**. In the Figure 2.1, we can see the generic workflow for an ML model.

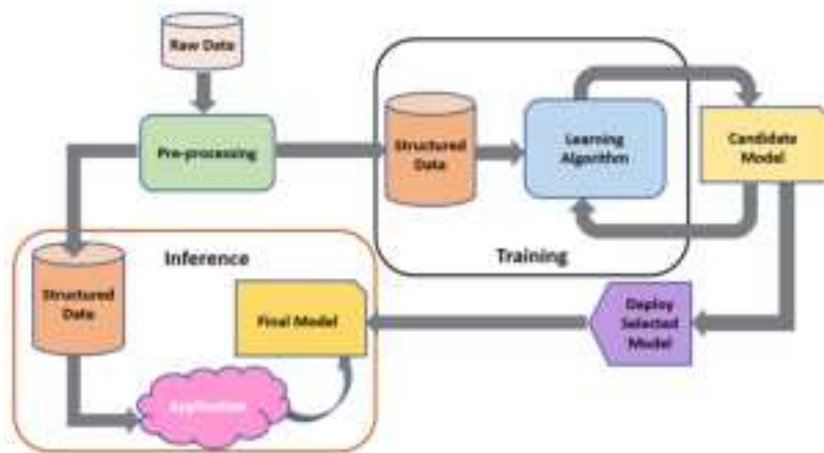


Figure 2.1: Generic ML model workflow

Another important key moment in the designing and implementation of a model is the **training phase**. The training of a model can therefore be seen as the choice of best parameters. These are numerical values that represent the model itself and determine its results. The parameters are chosen according to the results, and through these you choose whether to continue the training phase. In fact, the models can have results that are accurate or not, and the metric usually most used to measure the quality of the model is the accuracy. These metrics in how they are used depending on the model. In this way, the steps needed to create a model of ML can be seen as:

- Study of the problem
- Data collection

- Data preparation (ex. Pre-Processing)
- Model choice
- Model training
- Model evaluation

2.2.1 Tiny Machine Learning

TinyML is part of ML with a reduced or minimal use of the resources of a machine that is usually represented by an embedded system. This approach has spread so rapidly that resources increase day by day. This is due to the usefulness that TinyML has found in every field and the advantages that derive from it to a traditional approach. A lot of research documentation and field of application of TinyML can be found in [7] and [8].

Systems are usually based on IoT devices or sensors that communicate with the Cloud. A classic example of TinyML applied to smart devices concerns the well-known Alexa device, produced by Amazon. In these cases, you can not use the wake-up word and wait for the data sent to the Cloud to be processed and returned. The best choice is to run the ML algorithm on-site. The system which run ML algorithm on-site must be able to be active at all times, so in addition to being very small to ensure a reduced use of energy. In fact, microcontrollers usually consume energy in the order of microwatts or milliwatts, while normal CPUs in the order of the watt. The advantages of TinyML are many and also depend on the use cases.

The execution of local inference is faster than the remote one. So, the software is less complex and produces a low latency output. Embedded devices are designed to consume as less energy as possible. Privacy is also ensured because programs are executed locally, and it's not necessary to send data over the network. In this way is avoided the risk of running into malicious actors that can intercept data.



Figure 2.2: Tiny Machine Learning

2.3 Neural Network

Artificial Neural Networks (ANN) are a type of machine learning model inspired by the structure and functioning of the human brain. They are made up of computational units called “artificial neurons” or “perceptrons” which are organized in layers and connected to each other by synaptic weights. More precisely, in 1943, scientists McCulloch and Pitts theoretically described the first elementary neural network that worked on binary data. The ANNs mimic the neurons and the synapses of the human brain, as well as the ability that neurons have to get information. But, the most important concept imitated is that the flexibility of connections and therefore synapses. In fact, the human brain has dense but flexible synaptic connections that vary depending on the stimulation that receive.

2.3.1 Convolutional Neural Network

Convolutional Neural Networks (CNN) have all the characteristics of traditional networks. The only notable difference between CNNs and traditional ANNs is that the firsts are mainly used in the field of patterns recognition [9]. These networks bring the capabilities of the machines to those of the human brain, especially since they were built imitating the recognition modalities used by the brain.

The operation of CNNs is based on the identification of features, or characteristics, relevant in a some data sequence and their training consists in providing the network with a certain training dataset in which recognizing recurring patterns and associating them to classes the network will learn to classify unknown data in the future. An important advantage of CNNs is that the training of these networks

also teaches them to recognize the dependencies of spatial and temporal data, so you do not get the result only based, for example, on the various bits of an image, which may have not taken individually, but could have a connection if seen as a whole. In addition, thanks to the ability to detect the relevant characteristics, the data to be processed are very reduced by focusing only on what really matters. CNNs have become the most studied and used NNs.

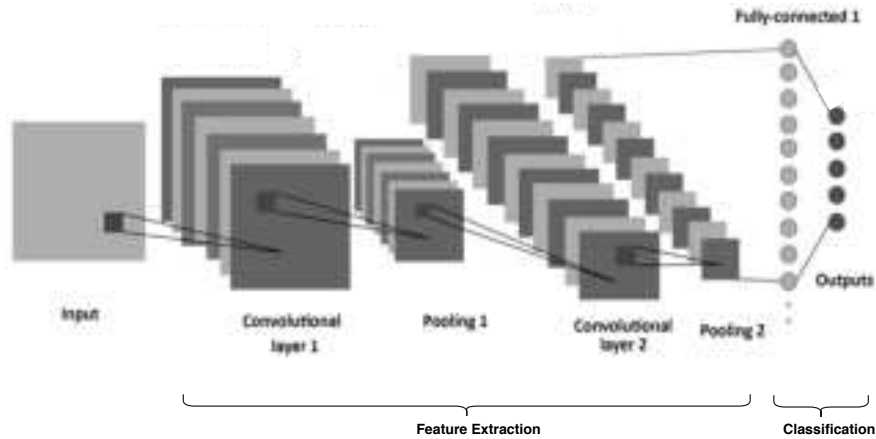


Figure 2.3: Example of Convolutional Neural Network

2.4 Edge Computing

The proliferation of the IoT has meant that much more data is created in a widespread and geographically distributed way, and this data is likely to be greater than that generated by cloud data centers. According to Ericsson study, it is estimated that by 2024, IoT devices would produce 45% of the 40ZB of worldwide internet traffic. [10]

It is difficult to move such massive amounts of data from the edge to the cloud because it's expensive in terms of energy consumption, money, and subject to some downsides like network delay, privacy, and connectivity issues. Therefore, directly addressing user requests at the edge is more practical, giving rise to a brand-new computing paradigm known as **Edge Computing**. In order to deliver services and execute calculations, the premise of Edge Computing is to shift computation and communication resources from the cloud to the edge of networks. This reduces

needless communication delay and enables faster replies for end users. The field of edge computing is currently growing. [11]

The Edge Computing describes the enabling technologies which allow computation to be done at the network’s edge on downstream data for cloud services and upstream data for IoT services. Any computer and networking resources located between data sources and cloud data centers are referred to as the “edge” [12]. For instance, a smartphone is the edge between physical objects and the cloud, a smart home gateway is the edge between things and the cloud. The idea behind Edge Computing is that processing should take place close to data sources.



Figure 2.4: Layered IoT architecture

2.4.1 Edge Intelligence

Nobody can dispute how quickly AI is evolving in the modern world. Big data processing demands more potent techniques, i.e., AI technologies, for obtaining insights that result in wiser choices and tactical business moves.

Edge Intelligence (EI) is a result of the integration of Edge Computing with AI, which is functionally required for the speedy analysis of massive amounts of data and the extraction of insights. [11]

EI appeared to enable the model and ambitious IoT services and scenarios. To extend intelligence outside the cloud, it integrates both cutting-edge and traditional techniques from edge and cloud computing, AI, data science, and networking. This strategy, also known as “Edge AI” promotes local activities above those that take place in the cloud or a centralized data center. It works better if edge devices—such as smartphones, IoT devices, and industrial equipment—are directly involved. [13]



Figure 2.5: Comparison between Central Intelligence and Edge Intelligence. [14]

2.5 Technologies and hardware

2.5.1 TensorFlow

TensorFlow [15] is an ML framework that works on large scale and heterogeneous environment. Developers can test out cutting-edge optimizations and training techniques thanks to TensorFlow. TensorFlow supports a wide range of applications, with training and inference on deep neural networks receiving particularly strong support. TensorFlow has been widely adopted for machine learning research and is utilized in several Google services. It was made available as an open-source project. The main strength of TensorFlow is that it offers a user-friendly environ-

ment, even for less experienced programmers. In fact, it allows you to create a ML model in a few lines of code and working at a high level, without the need to deepen the mathematics and the calculations below. It is implemented in C++, Python and CUDA, and it connects with the Keras API in order to build blocks for ML models.

2.5.2 TensorFlow Lite

TensorFlow Lite (TFLite) [16] is a collection of tools that makes it possible for developers to execute their models on mobile, embedded, and edge devices, enabling on-device ML. TFLite for microcontrollers (TFLite Micro) is a recent adaptation of TFLite (mid-2019), dedicated to the execution of ML on microcontrollers. TFLite itself provides APIs in several languages, such as Java, Swift, Python, and C++. This toolkit mainly presents two basic tools: the converter (TFLite Converter) and the interpreter (TFLite Interpreter) that can be installed independently on the device on which we want to make inference. The converter has the main task of optimizing the model by reducing its size and increasing its speed of execution. On the other hand, the interpreter is responsible for executing previously trained ML models on devices. It loads the model in a compatible format and once it is put in, performs inference, based on the input data provided.

2.5.3 Edge Impulse

Edge Impulse [17] is a cutting-edge platform dedicated to simplifying and accelerate the development of embedded machine learning applications. The platform offers a high-powered method for deploying ML models on edge devices as the world continues to include the IoT and the integration of AI.

The first support, offered by Edge Impulse, is a data collecting system that enables customers to collect and store training and test data with the model and implementation code. Edge Impulse provides several kinds of techniques to collect data in real-world environments, rather than relying on rebuilt dataset or needing customers to build their own data collecting technique.

The second contribution is to help in the stage of combining pre-processing feature extraction with deep learning, enabling users to examine a variety of potential

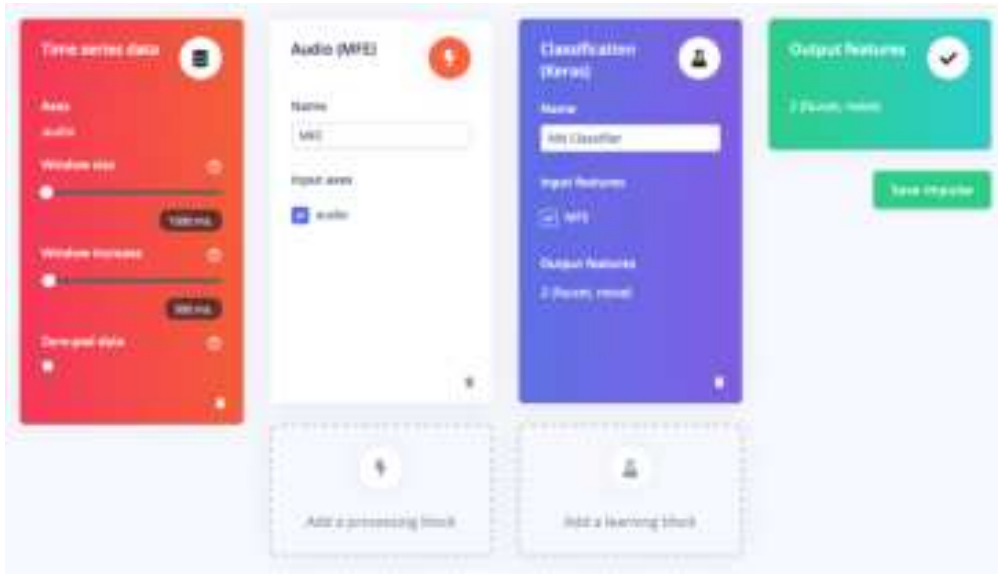


Figure 2.6: Example of creating an Impulse

solutions to each individual job or task.

The third support is a portable and adaptable inferencing library that can be used with a variety of embedded and edge systems. The EON Compiler reduces the amount of RAM and flash space used. [18]

Project developed using Edge Impulse may receive data in different file formats such as CSV, CBOR, JSON, WAV, JPG, or PNG. Additionally, this novel platform provides several ways for users to collect data for the projects, such as command line interface (CLI) that connect to device firmware to receive data in real-time.

One of the most important notions of Edge Impulse, from which the platform takes its name, is that of impulse. The impulse is composed of a set of processing blocks, which can be executed sequentially or in parallel, with other blocks. The impulse can be created after gathering all the needed data for the project's implementation. Therefore, the entire impulse is made up by 3 building blocks: input block, processing block and learning block. In the Figure 2.6, it's possible to see an example of a sound recognition from audio impulse.

The input block points out the kind of input data used for training the ML model. The type of data can be different such as audio, vibration, movements (time series) or images.

The second block, which is the processing block, is a feature extractor. The characteristics that our model learns on are extracted using DSP (Digital Signal Processing) techniques. Depending on the kind of data utilized in your project, these processes change.

The third and last block are the learning block. It is a NN that will be trained to learn on acquired data. The type of data in the training dataset and the function you want your model to perform determine the learning blocks.

2.5.4 Arduino Portenta H7 and Vision Shield

The Arduino family’s Portenta H7 developer board has been built as an “Edge Device” and it has two parallel cores, a graphics accelerator, and can run both real-time and high-level tasks at the same time. Additionally, an onboard wireless module enables it to simultaneously manage WiFi and Bluetooth connectivity [19]. The dual-core STM32H747, which has a Cortex M7 operating at 480 MHz and a Cortex M4 operating at 240 MHz, is the primary microcontroller used by H7. Through a Remote Procedure Call mechanism, the two cores may easily call functions on the other CPU.

The Portenta H7 may utilize a camera module with an ultra-low power CMOS image sensor (Himax HM-01B0), a LoRa communication module, and an ultra-compact, low-power, omnidirectional, digital MEMS microphone (MP34DT05), when enhanced with the add-on board Arduino Portenta Vision Shield.



(a) Portenta H7



(b) Portenta Vision Shield

Figure 2.7: Arduino Portenta H7 and Portenta Vision Shield

Problem description and System design

The system has been intended to be used in context of natural parks, museums or similar places with the aim of recognize specific words which express positive or negative feedback by the public. These, either indoor or outdoor, environments are usually crowded and background noises can greatly influence the audio data. The specific information about the public's appreciations is represented by a 12-words dictionary, made of 6 Italian words and 6 Spanish words, chosen to represent two sets of positive and negative feelings.

The presence of numerous visitors around an object of interest naturally encourages people to pause, engage in observation, and share their thoughts and emotions. Therefore, the basic idea is to deploy a microcontroller, which, through the use of an integrated microphone, is able to capture the surrounding audio in a certain spatial range. After audio acquisition, the microcontroller is able to recognize whether the received audio data can be associated with one of the 12 words in the dictionary.

An important element to take into consideration is that in the proximity of the target there might be an overlapping of voices and, therefore, the microcontroller should be able to perform a correct SR regardless the background noise. The results of an initial on-site processing can then be sent to a central server for subsequent processing, for the sake of privacy- and bandwidth-saving. The architecture in Figure 3.1 depicts how the proposed system is designed and deployed.

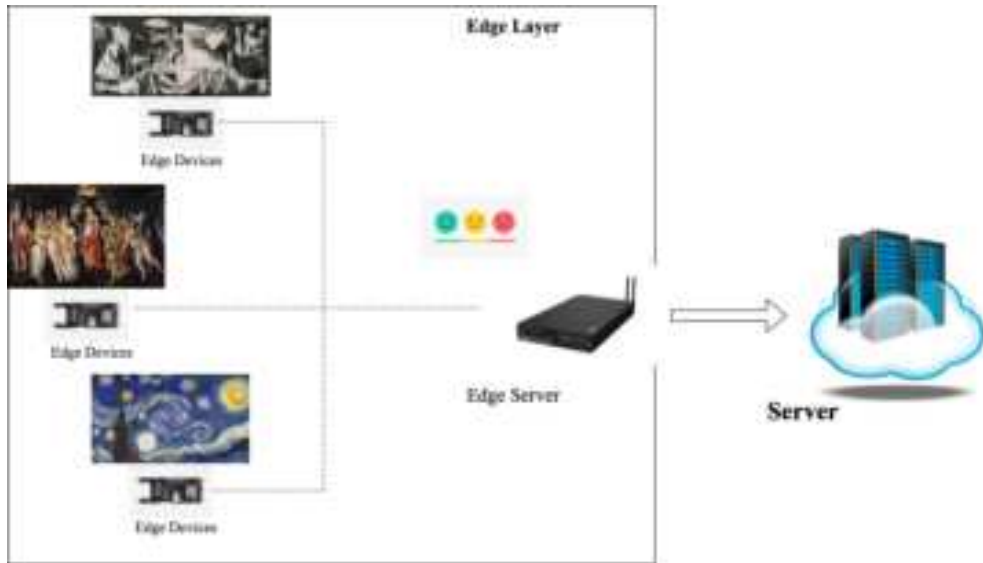


Figure 3.1: System architecture

3.1 System Design

The data handled by the microcontroller are audio recordings, and they must be properly processed. It is important to set processing windows of the same size for each registration in order to be able to properly catch each audio characteristics.

There are two main macro-components that must be present regardless of the technologies used.

- The first component, used to process data captured by the microphone of the Arduino Portenta H7;
- The second component, used to perform the classification on the data taken as input.

The main goal of the thesis is perform SR over a set of a-priori defined dictionary (thus such a type of classification falls within the scope of supervised learning) leveraging on the paradigms and technologies described in Chapter 2. Specifically, we will use the NNs in the context of TinyML. Therefore, there will be additional support components that will make up the final system and which will be necessary for the creation of the model, such as:

- **Code:** the code for the microcontroller is necessary to get ready the Portenta environment and to recall the model in order to perform inferences. The code must be loaded to the Arduino sketch as library.
- **Data:** they represent the system data. There are three different types of datasets available. The first two, validation and training datasets, are used for the training component and the third, testing dataset, will be used for final inferences.
- **Training:** the core of the system. The training component is in charge to coordinate training of the model and to execute it. The part of audio data processing performs operations on data, such as resizing the window. Another specific component performs audio elaboration in order to extract features, such as MFCC (Mel-Frequency Cepstral Coefficients) which work well with the human voices. The definition of the NN is another component of the system. Here, will be also defined the parameters that must be trained to recognize data and to classify them. The last component is the inference model represented by its parameters, and it is the result of the training phase.

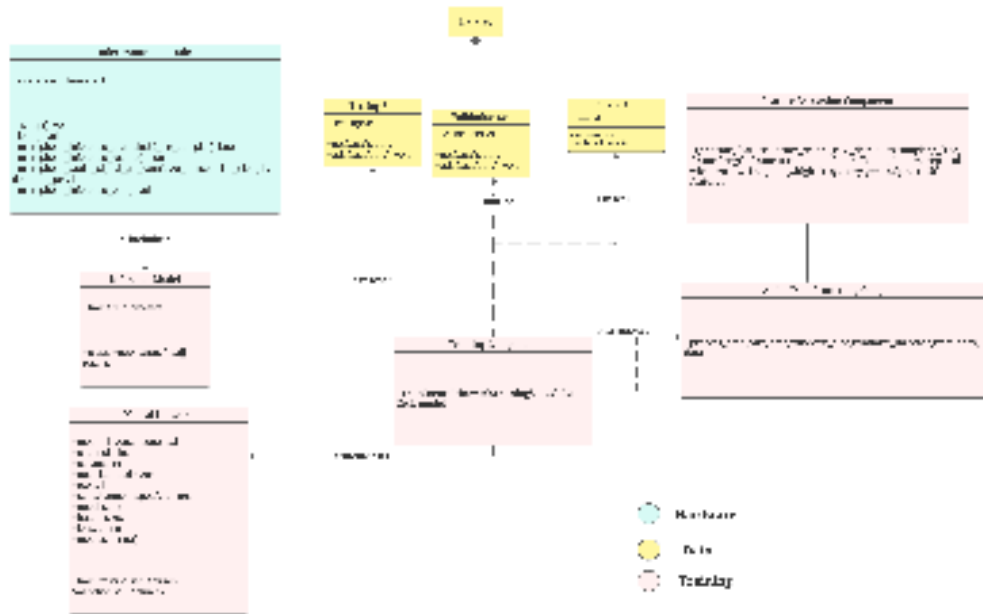


Figure 3.2: UML diagram of architecture design

3.2 Training process and system modelling

In the last section, 3.1, it was shown a UML diagram which highlight the architecture design. Now it's time to see the training process needed for the system to recognize the dictionary's words.

During the development phase, the software is created to enable data recognition. Before this, recordings are carefully labeled to make it easier for the system to recognize words. Following, in a testing phase, will be given new words to the system, in order to see how the software works in labelling them.

Preliminary and final steps of training process were emphasized too in order to contextualise them. We can start by preparing the development environment of the Portenta board and in parallel collect the audio data for the dataset. We can therefore see, going into the specifics of the training process, how it is structured for the case study. In the field of NNs, it is important to develop the structure of the training process and, obviously, to define a good training architecture.

In the Figure 3.3, it's possible to see a UML diagram that defines the steps

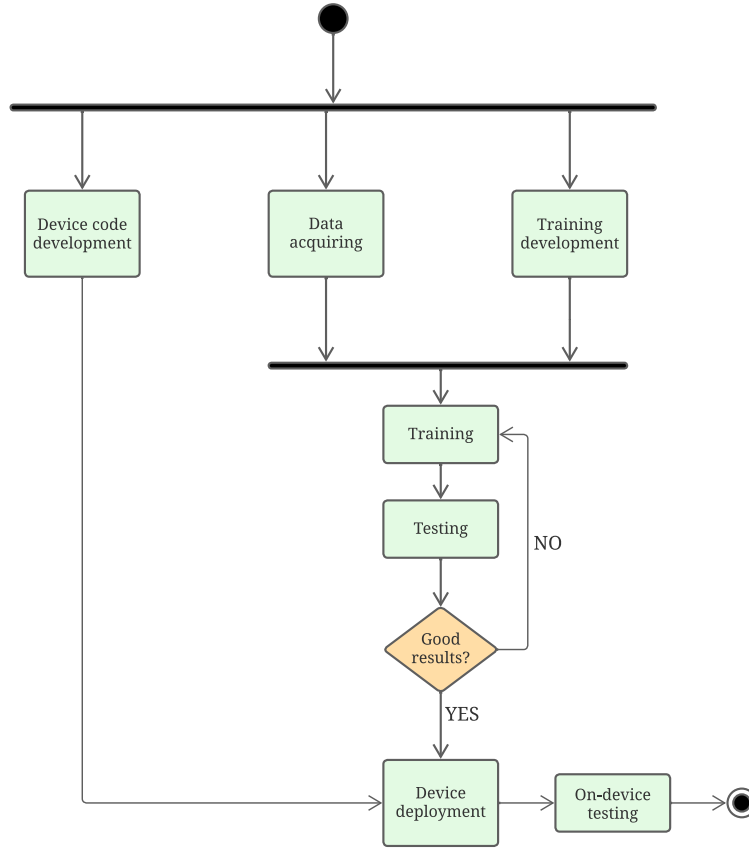


Figure 3.3: UML diagram macro-activities

that must be implemented and, after, executed in order to activate training and obtain the inference model.

Data augmentation is a valuable technique for enhancing both the quality and quantity of the dataset, and will be applied during the training phase. Furthermore, even if it is not indicated in the diagram, the testing phase is important and will be carried out in different steps. In the diagram, showed in the Figure 3.4, it is shown, in a form of note, that each phase is associated with technology chosen and explained in the Chapter 2.

Edge Impulse, 2.5.3, was used in the most crucial phases of development. TensorFlow was used for the training of the model, TFLite for conversion and Python to develop the NN and in data processing flows. In order to develop the Arduino

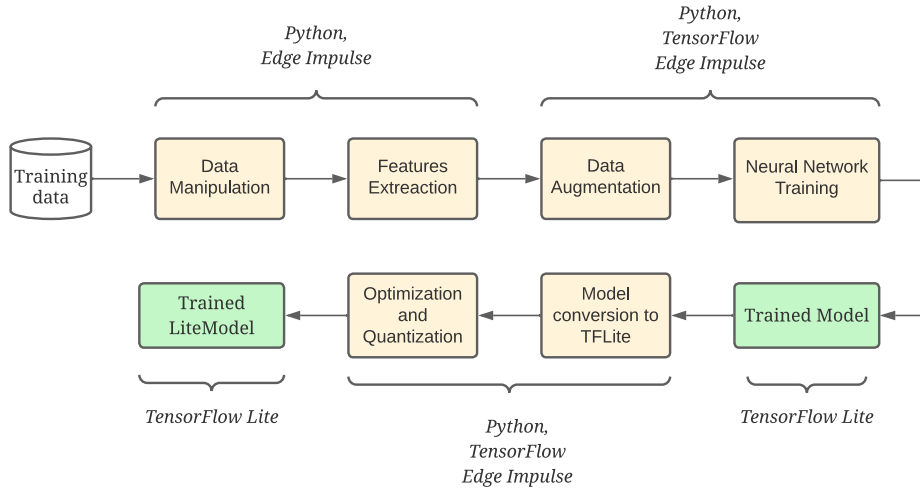


Figure 3.4: System Model Development

sketch, in the Arduino Ide, a simplified version of the C language, with additional functions to easily manage the board’s input/output interfaces, will be used. The trained model that will be developed must have a structure that allows it to perform operations very similar to those done during the training. To perform inferences on a certain audio data, the model will have to perform some data processing and classification operations identical to the training. These operations and parameters will be automatically incorporated by TensorFlow and subsequently by Edge Impulse when they are trained and deployed for the card, respectively. In the end, we will obtain a library that contains the model automatically.

3.3 Impulse design

The core of the system architecture, since we choose to use Edge Impulse, must be modeled in the logic of its environment. Remembering the concept of the impulse described in 2.5.3, we then associate, whenever possible, each component to an impulse block. As we can see in the Figure 3.5, the Impulse of the system, the first block is in charge to resize data, the second one to process them to extract the features and the third deals with classification using Keras. The last block

represents the number of output labels.

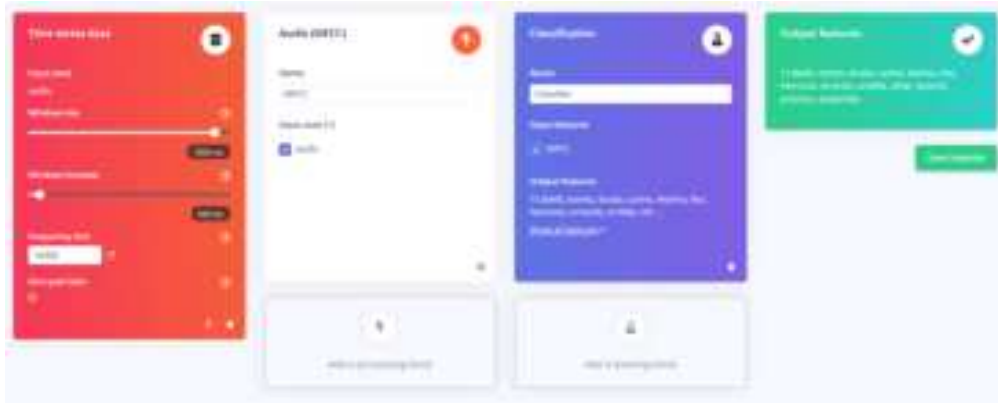


Figure 3.5: Impulse design

3.4 Arduino sketch design

Some characteristics are essential to the functioning of the system and lead to implement the sketch that must be loaded and used by the microcontroller. The development of the model only gets the object that executes the inference, used to recognize words. It's important to take into account how to record words and perform inferences at the same time. If the inference window is set to 1 second and the audio sampling is performed every second, it could happen that the word to identify is between two inferencing windows, and it's difficult to recognize.

By developing the Arduino library for the sketch, some problems are solved. The slicing function is inserted in the Arduino sketch, and it divides the samples into several parts. That, make sure to perform the inference on the sample many times.

The efficient time management is paramount during the recording process to minimize or avoid any time loss. Nonetheless, if voice captures are conducted sequentially, with recording preceding inference, there is a potential risk of losing time-sensitive registrations that could contain critical data. Hence, it is crucial to directly address this issue within the Arduino sketch.

One solution is therefore to follow the activities diagram in Figure 3.6 by performing the two phases of recording and inference, in parallel. Following, in the next Chapter, we will see the implementation details.

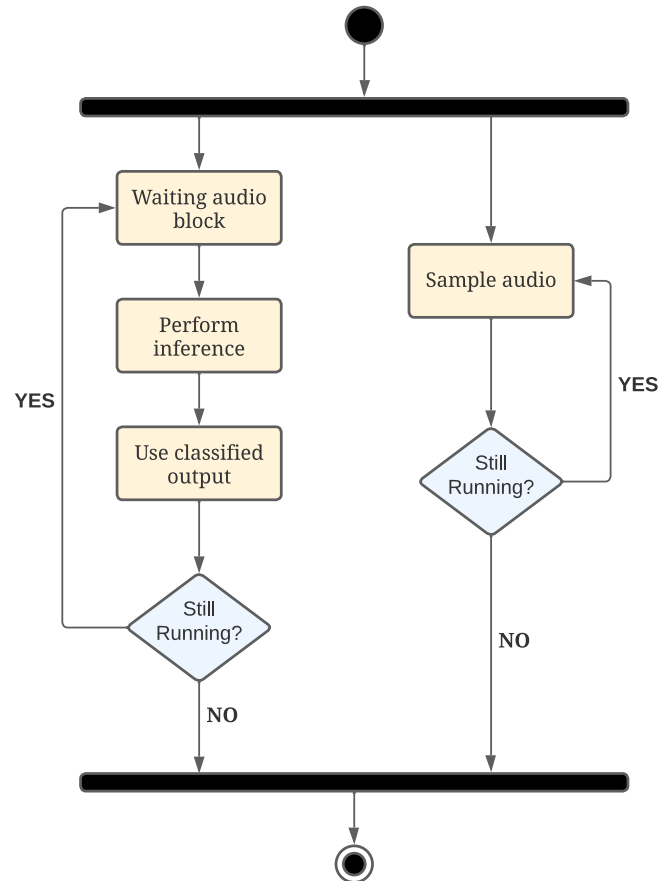


Figure 3.6: Activity diagram of Impulse running

Chapter 4

System implementation

The technologies, the design choices, and the system architecture described before feed the implementation phase, discussed in this Chapter and followed by the deployment phase.

4.1 Data acquisition

The first work, in the implementation phase, was performed focusing on data collection. This is one of the most important step that largely affects the final results. The NN must learn from what is proposed in this stage. For instance, if we propose audio sample that don't generalize well to the specific situation then the network will learn only those details, and when there will be something different that will not be able to recognize it.

The data gathering is performed in Edge Impulse. Here, there is the possibility to connect different devices, but for our case the best solution was to collect data only by using the Arduino Portenta H7 equipped with the Vision Shield which provide a microphone, as we can see in the Figure 4.1. We choose to use only this device because, in the next steps, as for example in the testing phase, we will use this microcontroller.

To connect the board cited before there are several ways, and we chose to use the Command Line of Edge Impulse, installing the Edge Impulse firmware on the Portenta and running the Edge-Impulse-Daemon for device configuration and



Figure 4.1: Portenta on Edge Impulse

association to the project.

The main idea is to have as much data as possible in order to try in the testing phase many combinations as possible.

We moved on to data acquiring after connecting the device to the framework and associating it to the project. Audio data samples of 1-second duration were collected at a sampling frequency of 16,000 Hz. To collect data were considered different criteria. Six people, from different countries, contributed to record all the data. The work was conducted in a multicultural environment, and it was a support to reach the goal. The people, with different accents and tones of voice in pronouncing the words, ensured the creation of a highly diverse dataset, making a fundamental contribution to its creation. It was useful to record in different environments, with noise background, or silent surroundings like at home or in a laboratory. The recordings were made at different distances, very near and so far, and with different angles with respect to the microphone.

All of these considerations were made with the objective of teaching the system with general patterns for each word, thus disregarding factors like proximity, background noises, and other variables

Looking at the wave forms of different recordings, referring to the same word, but pronounced by two different people, it is possible to notice how the signal is different, but it's possible to clearly see the common pattern, as it is shown in Figure 4.2. In the dataset, there were also gathered noises and non-dictionary words that were useful for training the model, and these were designated with the label "other". For which regard the registration of noise, they have been made in the University, in the bar, in the street, with background music, with television and then creating different types of noise like clapping hands, moving chairs and so on.

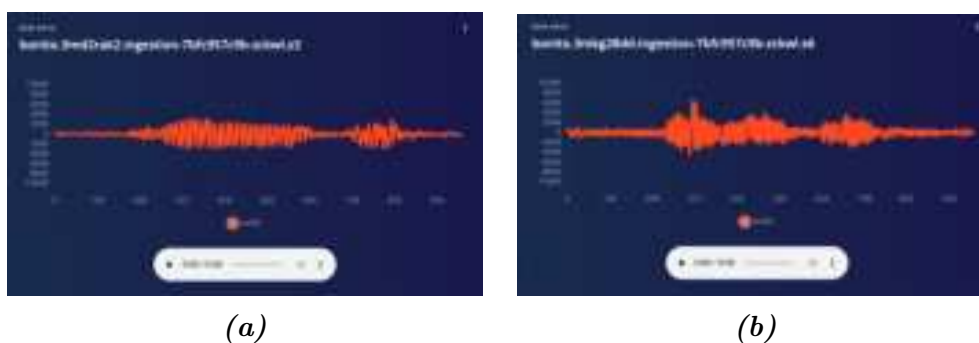


Figure 4.2: Two different wave-forms of the same word

The words for the dataset, as mentioned before, were pronounced and recorded in different positions and angles, with background noises and with silence. Specifically, we choose to record with three different angles: positioned 45° to the left, 45° to the right and in front of the microphones. The background noises, to be as realistic as possible, simulated with [20], were two, the cocktail voices, to simulate the background voices, and the white noises.

An important observation to do, during this phase, is related to the splitting of the data. In ML applications, and also in this case, there are different sets of data and, in this application, each audio sample must be placed in one of the sets. There are only recommended division and doesn't exist the best way to select them. Generally, the recommended range is from 10 to 30 percentage for the testing set and the rest for the training dataset. Moreover, there is a division between training set and validation set that was set to 20% of the training set. The validation set is employed as an independent dataset that was not used while training the model. This dataset serves the purpose of assessing the model's performance on data that has never been encountered before. This aspect is crucial for comprehending the model's ability to generalize patterns from the training data to previously unseen data.

In the end we collected **3 hours and 48 minutes** of data, **19 minutes** for each word of the dataset, for a total of **13,681 samples**, **1140 samples** per word. For the label "other" were collected **6 minutes and 55 seconds**, for a total of **415 samples**.

4.2 Components Implementation

4.2.1 Audio processing component

The implementation of audio processing functions is mandatory for the implementation of the audio data elaborating component. Almost always an identical part will be used to process the raw data, while another more specific part is responsible for extracting the features.

In Edge Impulse, for the implementation of the Impulse, a **Time series data** was chosen in which a 1-second window was initially set for each recording. It was chosen an increase of 1/2 second maximum for audio data that exceeds the preset size and the frequency was set to 16,000Hz for recordings, the same used during the actual recordings. And, finally, the active padding bits were set to 0 if the audio data are smaller than 1 second. Figure 4.3

For features extraction, analyzing the available technologies and the results achieved in other similar projects, it was chosen to use **MFCC**. It is also chosen not to use the default block, but to clone the existing one and to customize it [21].

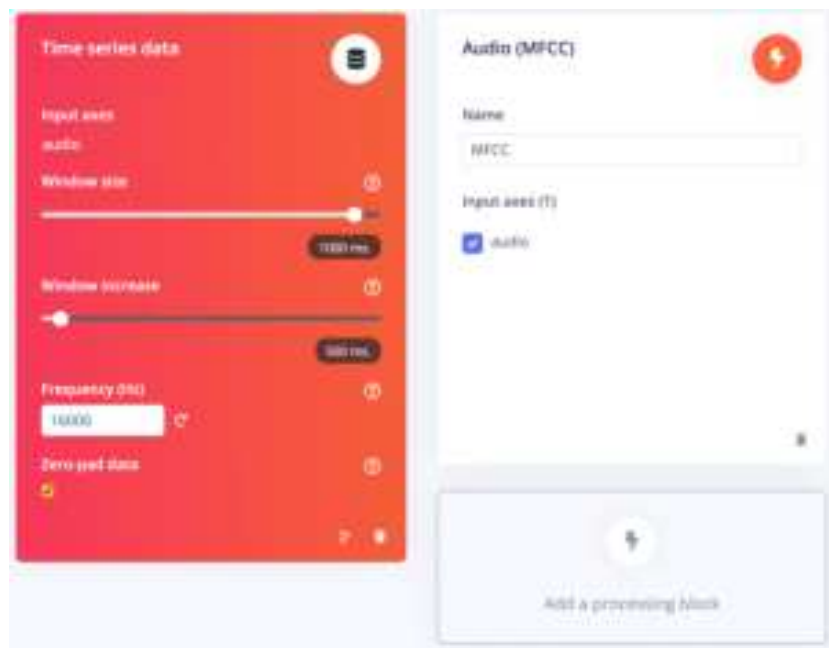
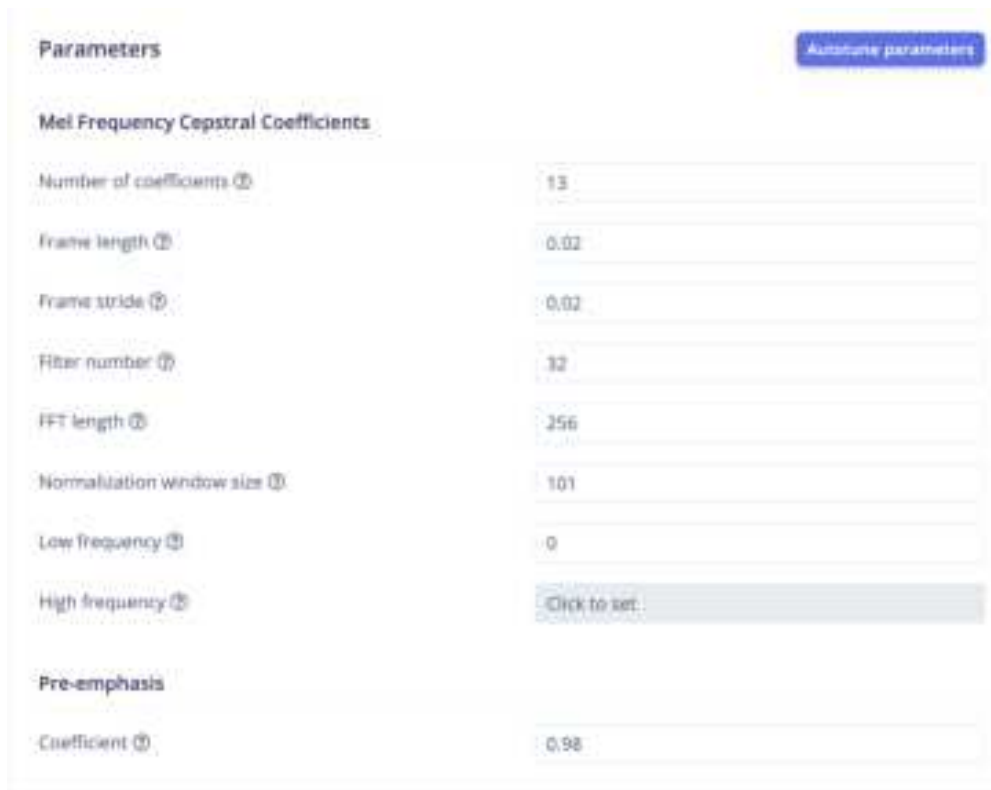


Figure 4.3: Audio processing block

The decision was made to duplicate the existing block to enable the direct modification of parameters and functions within the Python code, thereby affording greater flexibility in configuration. It is feasible to introduce additional parameters or functions or to modify the existing ones. To do all the steps highlighted before, it has been cloned the Git repository of Edge Impulse containing the MFCC block. The parameters used for the MFCC block are shown in the Figure 4.4.



Parameters	
Mel Frequency Cepstral Coefficients	
Number of coefficients	13
Frame length	0.02
Frame stride	0.02
Filter number	32
FFT length	256
Normalization window size	101
Low frequency	0
High frequency	Click to set.
Pre-emphasis	
Coefficient	0.98

Figure 4.4: MFCC Parameters

4.3 Data Elaboration

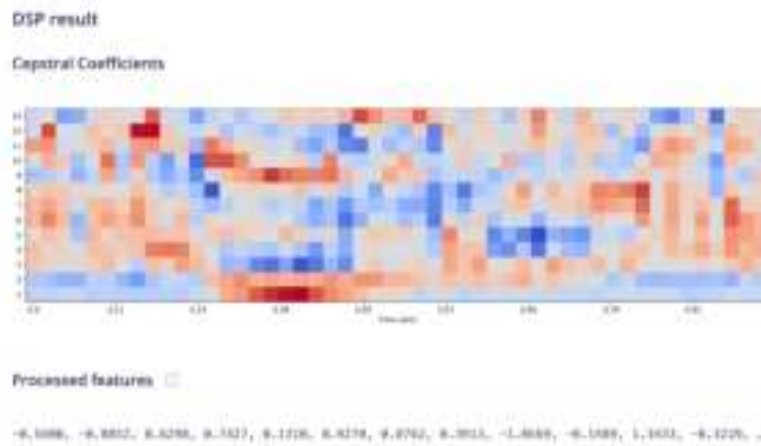
In a typical ML project, the raw data necessitates pre-processing and refinement before being fed into the NN. It is of paramount importance to manipulate this input data effectively to ensure its suitability for NN training.

4.3.1 Features extraction and data analysis

In this phase, we will use the MFCC component described before. The raw data, or raw features, are given as input to the second block and then the processed features are obtained. Edge Impulse allow us to recall the feature extraction process through a graphical interface. The results given by the software is in the form of spectrogram and no longer a sound wave. The raw and processed features in comparison are very different, and this can be seen in the Figure 4.5.



(a)



(b)

Figure 4.5: Comparison between before and after applying the MFCC

The results obtained from the previous operation can be analyzed by the **Features Explorer** tool provided by Edge Impulse. It is also feasible to assess the device's target performances, as illustrated in Figure 4.6. It is crucial to take into account the resulting data to prevent improper memory usage when the model is loaded onto the Arduino Portenta H7.



Figure 4.6: On Device performance

The previous tool also allows performing an in-depth analysis of the data in a two-dimensional environment in which Edge Impulse, through a scaling, reduce the multidimensionality of the features.

In 4.7, the results are presented, where any outliers have been examined through data cleaning, data re-processing, re-execution of the data split, or the analysis of pertinent feature details. Furthermore, it is essential to underscore that the NN has the capacity to classify data optimally and accurately. This is particularly pertinent when the NN is capable of distinguishing between various classes and when distinct clusters are discernible.



Figure 4.7: Data analysis on Features Explorer

4.4 Neural Network Implementation

The component containing the NN is the core of the system, and it must be carefully implemented in order to obtain a good inference model. The system's NN was implemented, starting from a basic one, after an in-depth study of NNs for similar problems [22]. After developing and testing various networks, it was chosen to implement the one that will be described later in this chapter. Subsequently, it was decided to focus, deeply, in the next chapter, on the testing part which is the most important for creating a real and functioning system.

In general, as it is known from the ML state-of-art, there is no single NN that universally outperforms all others; rather, the choice depends on the specific problem or dataset. With this premise in mind, we have opted to analyze two types of CNN, namely the 1D and 2D, respectively.

The choice between a 1D Convolutional Network (1D CNN) and a 2D Convolutional Network (2D CNN) depends on the type of data you are dealing with and the goals of your application. 2D CNNs are suitable for two-dimensional data, such as images or data with a matrix structure, and they can capture spatial relationships in two dimensions. 1D CNNs are usually used for sequential data, such as time series, audio data, text, and one-dimensional signals in general. 1D CNNs are generally more computationally efficient than 2D CNNs. They operate in a single dimension, and they require fewer parameters and fewer convolution operations. Hence, the selection of a 1D CNN for our system was straightforward and logical, aligning with the considerations mentioned earlier.

Now it is time to go into details and explain step by step what has been done to implement it.

The NN was implemented in Python by using the TensorFlow framework, in the Edge Impulse environment. Before, preliminary operations must be carried out. Firstly, the value of the **EPOCHS**, the **LEARNING_RATE** and the **BATCH_SIZE** were set to 150, 0.005 and 32 respectively, then a sequential model with *"model = Sequential()"* function of TensorFlow library was created. Later, a reshape of the input data was inserted to prepare them for the NN, and they were resized with the command *Reshape*. Hence, they have been scaled back to $[[\text{input length} / 13], [13]]$ i.e., input length divided by 13 as the size of the vectors and 13

channels. This is because it was chosen to use 13 coefficients in the MFCC block.

Subsequently, it has been added to the hidden layers of the network. Each level, in addition to the *Convolutional*, *Pooling* and *Activation* levels, also has a *Dropout* layer, used above all to avoid the “overfitting” of the network. In the final levels the data is flattened, through “*Flatten*”, and passed into a final “*Dense*” level, therefore completely connected, which will have 13 neurons, therefore one for each class to be recognised, this is the output level, Figure 4.9.

After defining the model, important parameters and aspects of the network were chosen and configured. Foremost, the **ADAM (ADaptive Moment estimation)** optimizer, Figure 4.8, one of the most used in NNs and which performed well in this case too. The parameters passed are a Learning Rate of 0.005 which seemed ideal from the first test results, and the other two parameters which are almost always standard.

```
39 # this controls the learning rate
40 opt = Adam(learning_rate=LEARNING_RATE, beta_1=0.9, beta_2=0.999)
41 callbacks.append(BatchLoggerCallback(BATCH_SIZE, train_sample_count,
42     epochs=EPOCHS))
43 # train the neural network
44 model.compile(loss='categorical_crossentropy', optimizer=opt, metrics
45     =['accuracy'])
46 model.fit(train_dataset, epochs=EPOCHS, validation_data
47     =validation_dataset, verbose=2, callbacks=callbacks)
48 # Use this flag to disable per-channel quantization for a model.
49 # This can reduce RAM usage for convolutional models, but may have
50 # an impact on accuracy.
51 disable_per_channel_quantization = False
```

Figure 4.8: ADAM Optimizer on the Neural Network

Neural network architecture

```

1 import tensorflow as tf
2 from tensorflow.keras.models import Sequential
3 from tensorflow.keras.layers import Dense, InputLayer, Dropout, Conv1D, Conv2D,
    Flatten, Reshape, MaxPooling1D, MaxPooling2D, AveragePooling2D, BatchNormalization
    , TimeDistributed, Permute, ReLU, Softmax
4 from tensorflow.keras.optimizers import Adam
5 EPOCHS = args.epochs or 150
6 LEARNING_RATE = args.learning_rate or 0.005
7 # this controls the batch size, or you can manipulate the tf.data.Dataset objects
    yourself
8 BATCH_SIZE = 32
9 train_dataset = train_dataset.batch(BATCH_SIZE, drop_remainder=False)
10 validation_dataset = validation_dataset.batch(BATCH_SIZE, drop_remainder=False)
11
12 # model architecture
13 model = Sequential()
14 model.add(Reshape((int(input_length / 13), 13), input_shape=(input_length, )))
15 model.add(Conv1D(8, kernel_size=3, padding='same', activation='relu'))
16 model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
17 model.add(Dropout(0.25))
18 model.add(Conv1D(16, kernel_size=3, padding='same', activation='relu'))
19 model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
20 model.add(Dropout(0.25))
21 model.add(Conv1D(32, kernel_size=3, padding='same', activation='relu'))
22 model.add(MaxPooling1D(pool_size=2, strides=2, padding='same'))
23 model.add(Dropout(0.25))
24 model.add(Flatten())
25 model.add(Dense(classes, name='y_pred', activation='softmax'))

```

Figure 4.9: Neural Network Implementation

4.4.1 Training phase

In the training phase, it's used the NN described before. The output of the MFCC block is inserted as input to the NN and the training is performed. The output of the execution is shown in the software interface, as it's possible to see in the Figure 4.10, and this is used to analyze the progress of the training before to evaluate the results. During this phase, important factors such as overfitting are evaluated.

It's crucial to highlight that all the training process is done in an automatically way by the Edge Impulse.


```
Training output CPU (0) ▲
Epoch 146/150
283/283 - 2s - loss: 0.3436 - accuracy: 0.8934 - val_loss: 0.1819 -
val_accuracy: 0.9477 - 2s/epoch - 9ms/step
Epoch 147/150
283/283 - 3s - loss: 0.3546 - accuracy: 0.8913 - val_loss: 0.1768 -
val_accuracy: 0.9477 - 3s/epoch - 9ms/step
Epoch 148/150
283/283 - 2s - loss: 0.3667 - accuracy: 0.8911 - val_loss: 0.1787 -
val_accuracy: 0.9508 - 2s/epoch - 9ms/step
Epoch 149/150
283/283 - 2s - loss: 0.3550 - accuracy: 0.8888 - val_loss: 0.1865 -
val_accuracy: 0.9415 - 2s/epoch - 9ms/step
Epoch 150/150
283/283 - 3s - loss: 0.3595 - accuracy: 0.8859 - val_loss: 0.1735 -
val_accuracy: 0.9451 - 3s/epoch - 9ms/step
Finished training
```

Figure 4.10: Training output

4.5 First Model evaluation

Once the network training has been carried out and finished, the Edge Impulse software provides us, through its graphical interface, a view on the results obtained from this procedure. All these results are useful for our studies because starting from them, it is possible to carry out all the necessary evaluations. First, we evaluate whether the model has been trained correctly or there is some, or most, data that does not match the specific labels. Subsequently, using the testing dataset it is possible to evaluate the final accuracy value pertaining to the testing phase. This accuracy value can then be compared with the real testing values, through the use of the Portenta board.

4.5.1 Training results

Initially, the evaluation conducted, as depicted in the Figure 4.11, focuses on the confusion matrix following the training phase. This tool is commonly employed in

classification ML problems to evaluate the performance of a model. It allows to clearly and concisely visualize how well or poorly a model is able to classify different input classes. The confusion matrix is organized into a table with rows and columns, each row represents the actual class of the test instances and each column represents the class predicted by the model. The confusion matrix is useful for calculating various metrics for evaluating model performance, including accuracy, precision, recall, F1-score, and others. These metrics provide detailed information on the performance of the model in terms of correct classification and errors made.

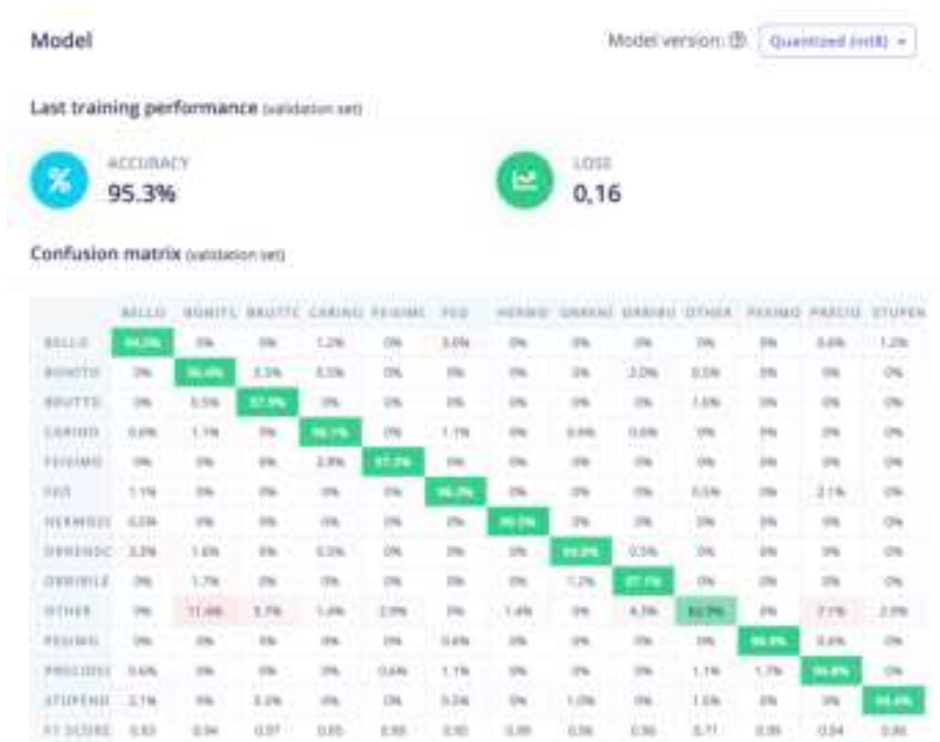


Figure 4.11: Confusion matrix of the model

The final accuracy that model reached is equal to **95.3%**. Each word, trained in the model, reaches a good value of accuracy, they are all above 94%.

A special mention, in this case, should be made for the “other” label. As already mentioned, through this label, we try to train the network for everything that is not part of the 12-word dictionary. As you can see from the table, the

accuracy value compared to all other labels is much lower, equal to 62.9%. This result is achieved because various noises and other words, even similar to those in the dictionary, were used as recordings for the other label. This was done to make the experiments as realistic as possible, and therefore, it occasionally happens that recordings labeled with “other” are recognized as words with different label.

Another very important thing to show, and which is provided by Edge Impulse, is the Feature Explorer, shown in the Figure 4.12. Using this tool, we are able to graphically visualize how the training dataset is clustered, by merging similar data with the same label together. Therefore, as a final consideration, we can assert that, in this case, the closer the data are to each other, the better the model is trained.

The last consideration concerns the data present at the bottom of the figure. These three values represent, respectively, the inferencing time, the peak of the RAM usage, and the amount of flash memory used by the model. It should be highlighted, however, that these three values are only estimates made by Edge Impulse, which knows that the model will be loaded on the Portenta. In fact, in the Chapter 5, in the real testing phase, the inference model, on the real device, will be evaluated and compared with this one.

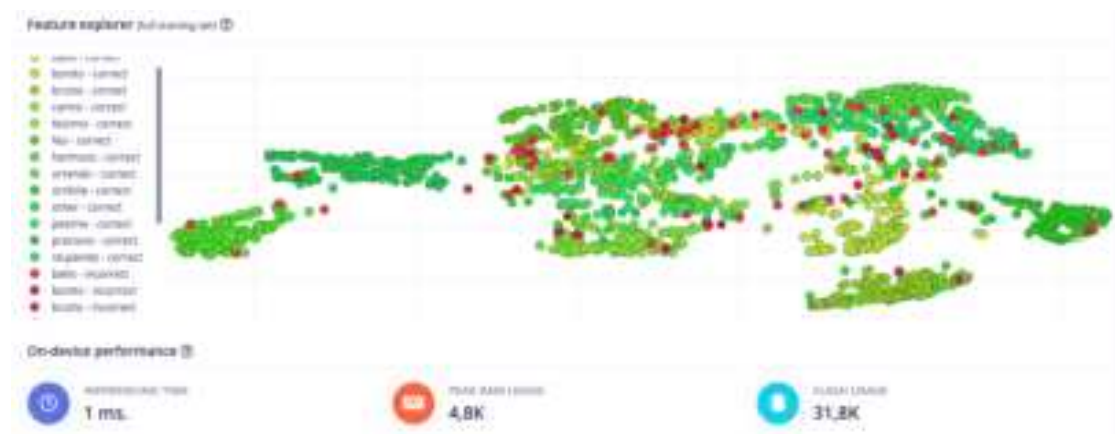


Figure 4.12: Feature Explorer for Training Dataset

4.5.2 Testing results

What was done for the training phase, described previously, was subsequently done for the testing phase, again on Edge Impulse. As it can be seen in the Figure 4.13, the accuracy value achieved is **91.4%**.

The result obtained here is slightly different to the one reached before. This is normal, and it is not due to the overfitting of the network.

This minimal difference between the accuracy during the training phase and that during the testing phase suggests that the model is able to generalize well from information learned during training to new data not seen during training. Thus, the objective was not necessarily to obtain identical accuracies between the two phases, but rather to minimize the difference and seek a balance between accuracy and generalization ability.



Figure 4.13: Testing performances on Edge Impulse

As done previously, also in this case the various clusters were displayed, one for each label. The Data Explorer referring to the testing dataset is shown in the following Figure 4.14.

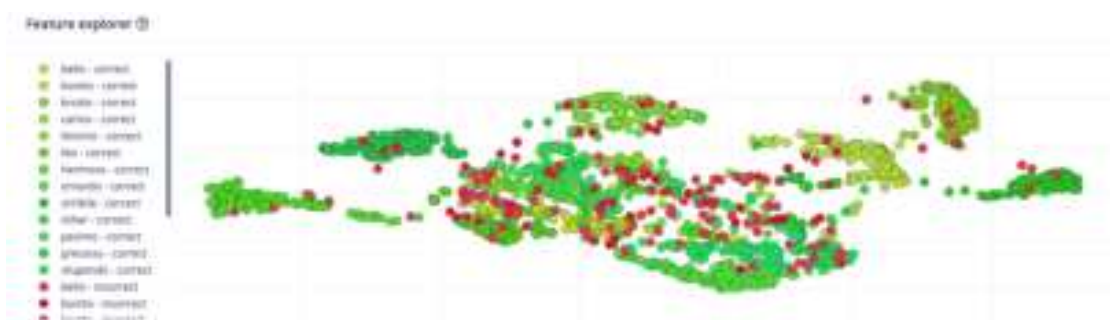


Figure 4.14: Feature Explorer for Testing Dataset

4.6 Arduino Portenta H7 Deployment

Once the model is obtained, the testing phase can start, and the model can be converted into a TFLite model by creating a library model and generating the sketch for uploading onto the board, which will import this library. The conversion of the model into lite model and library generation are managed by EdgeImpulse. Once the model is obtained by all the steps in the software, it was requested as an Arduino library. At this point, to carry out the final step, it is necessary to use an Arduino sketch which includes the library obtained. The sketch implementation is one of the most delicate parts of the project, and it determines the correct functioning of the system.

The goal of the sketch is to run inferences in parallel to recordings, without losing audio data while the inference is going on. One of the most important parameters to take into account is the size of the slices for each recording window. The library, will make inferences by joining the number of consecutive slices indicated and creating a window of second.

There are two important things to take into consideration: the first one, that a few slices per window could cause the inference to lose quality, the second one that many slices increases the number of inferences performed, but the inferences have less chance of missing a spoken word and the computational complexity is increased.

Therefore, a slice size of 250 ms is initially chosen, four slices that scroll in the inference window. In the IDE the implementation of the sketch was performed by including different methods and, obviously, the *setup()* and *loop()*, the classical

method of the Arduino programming. Two buffers were used in order to alternate for recording and inferences, so that inferences and recording can be performed in parallel. The time management is fundamental, when a data is written in one of the two buffers you must be sure that the inference on that data has already been performed.

```

1415 bool microphone_inference_start(uint8_t n_samples)
{
  inference_buffers[R] = (signed short *)malloc(n_samples * sizeof(signed short));

  if (!inference_buffers[R] || !n) {
    return false;
  }

  inference_buffers[L] = (signed short *)malloc(n_samples * sizeof(signed short));

  if (!inference_buffers[L] || !n) {
    at_free(inference_buffers[R]);
    return false;
  }

  inference_buf_select = R;
  inference_buf_count = 0;
  inference_n_samples = n_samples;
  inference_buf_ready = 0;

  // configure the data receive callback
  PDM.onReceive(&pk_data_ready_inference_callback);

  PDM.setBufferSize(2048);

  if (!PDM.begin(1, CI_CLASSIFIER_FREQUENCY)) {
    at_print("PDM: failed to start PDM");
    return false;
  }

  record_ready = true;

  return true;
}

```

Figure 4.15: Buffers and microphone initialization methods

The buffers, cited before, are initialised with a size equal to the size of the slice, as in possible to see in the Figure 4.15. After, the microphone was initialised using the PDM interface of the Arduino library. Later, the *callback* function is set to be recalled when there are available data from the microphone. The function read the data and put them in the active buffer after which is set to ready, warning that data is ready. Figure 4.16

```

//
// @brief FDM buffer full callback
//
// Only write data to the buffers
//
static void am_data_ready_inference_callback(void)
{
    int bytesAvailable = FDM.available();

    // input into the sample buffer
    int bytesRead = FDM.read(char *sampleBuffer, bytesAvailable);

    if (!inference.buf_ready == 0 && !record_ready == true) {
        for (int i = 0; i < bytesRead; i++) {
            inference.buffers[inference.buf_select][inference.buf_count++] = sampleBuffer[i];
        }

        if (inference.buf_count == inference.b.samples) {
            inference.buf_select ^= 1;
            inference.buf_count = 0;
            inference.buf_ready = 1;
            break;
        }
    }
}

```

Figure 4.16: Callback method

In the Figure 4.17 it's shown a small but important detail of the loop method. There, the *microphone_inference_record()* method check the variable *buf_ready* that must be found at 0. Then, the method waits for the recording process to finish populating one of the two buffers, and, subsequently, can set the variable to 1.

```

void loop()
{
    bool r = microphone_inference_record();
    if (!r) {
        // @printf("ERR: Failed to record audio...");
        return;
    }

    signal_t signal;
    signal.total_length = ET_CLASSIFIER_SLICE_SIZE;
    signal.get_data = &microphone_audio_signal_get_data;
    et_inferer_result_t result = IR();

    ET_IMPULSE_TRIGGER r = run_classifier_continuous(&signal, &result, debug_on);
    if (r != ET_IMPULSE_OK) {
        // @printf("ERR: Failed to run classifier (audio)");
        return;
    }
}

```

Figure 4.17: Important detail of the *loop()* method

After that, the inference process immediately puts the variable back to 0 and calls the function *run_classifier_continuous()* in order to perform the inference by passing the signal object through the pointer to the function in the Figure 4.18. This last function accesses the current buffer, the results are printed, and the loop is re-executed.

```
/**  
 * Get raw audio signal data  
 */  
static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)  
{  
    numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1](offset), out_ptr, length);  
    return 0;  
}
```

Figure 4.18: Function referred to the current buffer

Testing phase

This Chapter describes the testing activities performed over an Arduino Portenta H7, equipped with the models previously designed and trained through NN for recognizing words spoken with different accents and in presence of background noises.

Here, some useful variables have been added to the sketch, as detailed in the Chapter 4. These variables aid in assessing the accuracy of the classification process and provide insights into the percentage of correct classifications. Therefore, a new result for a class is scored every time the classification exceeds the score of 0.6, as you can see in the Figure 5.1.

The tests to analyse the system were carried out by seven different people, at different distances and with different background noises. In particular, there were 4 male and 3 female voices. As regard to noises, they were simulated in three different scenarios: the first one without any type of noise, the second one with voices and the third one with white noise, both last two simulated using [20].

With regard to the positions from which the words were pronounced by the people, it was decided to replicate the same positions that were employed in constructing the dataset and the following are recalled. There are three positions that are 45 degrees to the left of the microphone, in front of the microphone and 45 degrees to the right. The pronunciations, in order to test the system, are repeated at two different distances which are 0.5 meters and 1.5 meters.

It was chosen to use this type of scenarios because they are the same ones used

```
20:28:45.016 => Predictions (25P: 22 ms., Classification: 14 ms., Anomaly: 0 ms.):
20:28:45.016 => bello: 0.000000
20:28:45.016 => bonita: 0.000000
20:28:45.016 => brutta: 0.000000
20:28:45.016 => carina: 0.000000
20:28:45.016 => feissima: 0.000000
20:28:45.016 => feo: 0.000000
20:28:45.016 => hermosa: 0.000000
20:28:45.016 => orrenda: 0.000000
20:28:45.016 => orribila: 0.000000
20:28:45.016 => other: 0.000000
20:28:45.016 => pessima: 0.000000
20:28:45.016 => preziosa: 0.996094
20:28:45.016 => stupendo: 0.000000
20:28:45.016 => bello: 2
20:28:45.016 => bonito: 1
20:28:45.016 => brutto: 2
20:28:45.016 => carino: 0
20:28:45.016 => feissima: 1
20:28:45.016 => feo: 0
20:28:45.016 => hermosa: 1
20:28:45.016 => orrenda: 1
20:28:45.016 => orribile: 0
20:28:45.016 => other: 24
20:28:45.016 => pessima: 1
20:28:45.016 => preziosa: 1
20:28:45.016 => stupendo: 1
```

Figure 5.1: Sketch output inference

during the data collection phase to train the model. Therefore, in the following paragraphs there will be an overview of each individual scenario and in the end there will be a recap of all the experiments carried out.

5.1 First scenario - Silent background

The first scenario regards the experiments without any type of background noise. It was chosen to start from this one in order to test the system in a scenario in which there is no disturbance to the words spoken. Therefore, the person finds himself alone in front of the microphone.

To perform the experiments and assess the model's performance in real-world conditions, each word was repeated five times in each position and at each distance. Consequently, in each scenario, every word was spoken a total of 30 times. Subsequently, the results, presented in Table 5.1, will be analyzed later in this section. For more simplicity, in the table, the following notations will be used to differentiate male and female voices: for male we'll use *M. Voice 1* notation and so on, by changing the number for each different person, and for female *F. Voice 1* and the same as before for the number. It's important to highlight that for each person there will be two rows, the first for the number of word recognized by the system and the second for the percentage of accuracy of word recognized.

#	Bello	Bonito	Brutto	Carino	Feisimo	Feo	Hermoso	Orendo	Orribile	Pesimo	Precioso	Stupendo
M. Voice 1	27	28	20	27	18	28	27	27	27	21	23	20
% Accuracy	90,00	93,33	66,67	90,00	60,00	93,33	90,00	90,00	90,00	70,00	76,67	66,67
M. Voice 2	25	27	17	25	18	27	24	28	28	18	27	26
% Accuracy	83,33	90,00	56,67	83,33	60,00	90,00	80,00	93,33	93,33	60,00	90,00	86,67
M. Voice 3	21	23	27	28	26	25	24	22	16	18	22	24
% Accuracy	70,00	76,67	90,00	93,33	86,67	83,33	80,00	73,33	53,33	60,00	73,33	80,00
M. Voice 4	28	26	27	26	24	29	26	27	26	24	22	23
% Accuracy	93,33	86,67	90,00	86,67	80,00	96,67	86,67	90,00	86,67	80,00	73,33	76,67
F. Voice 1	24	26	25	27	22	27	25	22	22	22	24	25
% Accuracy	80,00	86,67	83,33	90,00	73,33	90,00	83,33	73,33	73,33	73,33	80,00	83,33%
F. Voice 2	28	26	28	26	24	25	27	26	27	26	26	28
% Accuracy	93,33	86,67	93,33	86,67	80,00	83,33	90,00	86,67	90,00	86,67	86,67	93,33
F. Voice 3	22	24	23	24	25	27	22	26	26	22	24	25
% Accuracy	73,33	80,00	76,67	80,00	83,33	90,00	73,33	86,67	86,67	73,33	80,00	83,33
Total % Accuracy	82,22	85,71	79,52	87,14	74,76	89,52	83,33	84,76	81,90	71,90	80,00	81,43

Table 5.1: Experiment results - Silent Background

The total average of **Accuracy Percentage** for this first experiment is **81.85%**. As we can see in the Chapter 4, the result obtained is now approximately 10% smaller. This is considered acceptable and good compared to the one obtained previously in Edge Impulse because there are several factors that influence it. Firstly, one reasons concerns the live testing and the fact that despite the slicing set to 250 ms, it is as if we were making the inference on the 1-second window. Consequently, there were instances where words spoken between two slices, causing the model to fail in recognizing them.

Secondly, as is well known from the state of the art of projects implemented with TinyML, the results obtained with models implemented on real devices never reflect the accuracy percentages obtained through implementation on computers, and even in our case there is a small discrepancy. In fact, the model attempts to identify patterns in the training data by learning from it during the training phase. However, generalization, or the capacity to apply discovered patterns to new, previously unseen data, is its primary objective. It is anticipated that there will be differences in performance between the training and testing data sets, as the testing data is designed to represent scenarios that the model has never encountered before.

Another very important thing to underline is the inference time and therefore the time that the model on the device takes to carry out the classification. As seen and described previously, especially in the Figure 4.12, the time estimated by the cloud platform was 1 ms. However, the result obtained through the classification in the on-board testing is **14 ms**, as visible in the top of the Figure 5.1. There is a difference between these two values, and it is normal and expected because the device has reduced computational capabilities. As a result, the execution time in this case is slightly higher than what was previously observed on the Edge Impulse platform. Nevertheless, it remains a relatively good and acceptable result. Finally, it is possible to take into account to use the system in a real place, such as a museum or natural park.

5.2 Second Scenario - Voices background

In the second scenario, all the experiments were carried out with a voices background. It was chosen to use this type of noise in order to simulate a case that was as real as possible. During the analysis phase, it was thought that it would be possible to implement and distribute the system in places where there are many people talking to each other that create a noisy environment. However, it is necessary to test the system also in conditions other than the ideal case of a silent scenario. This evaluation is necessary to assess its capabilities and determine its feasibility for implementation in noisy or crowded places.

With this scenario, therefore, we tried to put the system in difficulty such in a way as to evaluate its efficiency even in conditions in which the microphone is disturbed.

#	Bello	Bonito	Brutto	Carino	Feisimo	Feo	Hermoso	Orrondo	Orribile	Pesimo	Precioso	Stupendo
M. Voice 1	26	20	28	27	18	12	25	18	22	26	23	19
% Accuracy	86,67	66,67	93,33	90,00	60,00	40,00	83,33	60,00	73,33	86,67	76,67	63,33
M. Voice 2	25	26	27	27	20	27	21	22	22	22	21	26
% Accuracy	83,33	86,67	90,00	90,00	66,67	90,00	70,00	73,33	73,33	73,33	70,00	86,67
M. Voice 3	28	24	22	23	24	26	22	24	25	24	28	27
% Accuracy	93,33	80,00	73,33	76,67	80,00	86,67	73,33	80,00	83,33	80,00	93,33	90,00
F. Voice 1	24	23	26	24	25	23	26	24	22	25	27	28
% Accuracy	80,00	76,67	86,67	80,00	83,33	76,67	86,67	80,00	73,33	83,33	90,00	93,33
F. voice 2	27	26	21	21	21	26	24	22	26	23	22	21
% Accuracy	90,00	86,67	70,00	70,00	70,00	86,67	80,00	73,33	86,67	76,67	73,33	70,00
F. Voice 3	25	22	26	25	24	22	24	21	24	25	26	28
% Accuracy	83,33	73,33	86,67	83,33	80,00	73,33	80,00	70,00	80,00	83,33	86,67	93,33
Total % Accuracy	86,11	78,33	83,33	81,67	73,33	75,56	78,89	72,78	78,33	80,56	81,67	82,78

Table 5.2: Experiment results - Voices Background

In this scenario, the total average of **Accuracy Percentage** is **79.44%**. As evident, in this case, the system's efficiency is slightly reduced compared to what was observed in the previous scenario.

In this scenario, we tried to put stress the system to evaluate its response capacity. It has been simulated a new situation, different from the ideal case of

silence background. In fact, it was chosen to test the model and the microphone capabilities even if there are background noises, such as voices and people talking incessantly, even with a high tone of voice.

5.3 Third Scenario - White Noise Background

The third, and last, scenario experimented considered a White Noise background.

The choice to incorporate this type of noise was aimed at assessing the system's performance under a different noisy background condition. Specifically, it was selected to simulate a scenario where background noise might be present but not as perceptible as voices

We will test the quality of the model even with this type of noise and evaluate how much the microphone is disturbed. As done previously, everything will be evaluated with the accuracy percentages of each single word in the dictionary pronounced by each single person. Finally, the total average accuracy will be shown, in such a way as to evaluate the results obtained in this case with the previous one.

#	Bello	Bonito	Brutto	Carino	Feisimo	Feo	Hermoso	Orrendo	Orribile	Pesimo	Precioso	Stupendo
M. Voice 1	22	27	21	28	15	24	28	24	22	11	17	21
% Accuracy	73,33	90,00	70,00	93,33	50,00	80,00	93,33	80,00	73,33	36,67	56,67	70,00
M. voice 2	27	28	23	30	24	26	23	23	28	24	25	26
% Accuracy	90,00	93,33	76,67	100,00	80,00	86,67	76,67	76,67	93,33	80,00	83,33	86,67
M. Voice 3	22	29	25	28	25	28	28	27	25	26	22	24
% Accuracy	73,33	96,67	83,33	93,33	83,33	93,33	93,33	90,00	83,33	86,67	73,33	80,00
F. voice 1	24	22	24	24	22	24	25	24	24	25	26	22
% Accuracy	80,00	73,33	80,00	80,00	73,33	80,00	83,33	80,00	80,00	83,33	86,67	73,33
F. Voice 2	28	27	28	27	22	26	24	25	26	23	23	23
% Accuracy	93,33	90,00	93,33	90,00	73,33	86,67	80,00	83,33	86,67	76,67	76,67	76,67
F.Voice 3	24	26	22	26	24	26	23	25	22	26	25	22
% Accuracy	80,00	86,67	73,33	86,67	80,00	86,67	76,67	83,33	73,33	86,67	83,33	73,33
Total % Accuracy	81,67	88,33	79,44	90,56	73,33	85,56	83,89	82,22	81,67	75,00	76,67	76,67

Table 5.3: *Experiment results - White Noise Background*

In this third and last scenario, the total average of **Accuracy Percentage** is

81.25%. Indeed, as evident from the results, in this scenario with different noise compared to the previous one, there is a slightly improved accuracy. This suggests that the system, in this case, exhibits a better ability to recognize spoken words compared to when background voices were present.(Table 5.2).

Another important consideration to make is that relating to the people who carried out the experiments. It was decided to test the system with people who speak Italian as their native language and with a person who speaks Spanish as his native language (in particular, M.Voice 2 in the tables). This is important because, as already described in the previous chapters, the dictionary that the system is able to recognize is made up of Italian and Spanish words.

Therefore, looking the results of tables 5.1, 5.2, 5.3, it is possible to state that the system does not notice differences between people of different native languages, and this is important because it is able to recognize what is said to it regardless of the person's nationality and spoken language.

5.4 Results comparison

In conclusion, all the results obtained from the aforementioned experiments are compared and presented in Table 5.4. This table provides the average accuracy values for each individual word within each respective experiment, facilitating a comprehensive comparison.

Therefore, all the values regarding the accuracy, both for each word and scenarios, are pretty good and in line with what was previously described.

In conclusion, it can be state that the system developed in this thesis, starting from data collection to testing on the real device, is reliable. After the several in-depth tests, it is evident that every part of the project aligns with expectations. Consequently, should the need arise, the system can be deployed in places such as museum or natural parks, in order to test furthermore its capabilities.

	First Scenario	Second Scenario	Third Scenario
Bello	<i>66,67%</i>	<i>86,11%</i>	<i>81,67%</i>
Bonito	<i>85,71%</i>	<i>78,33%</i>	<i>88,33%</i>
Brutto	<i>79,52%</i>	<i>83,33%</i>	<i>79,44%</i>
Carino	<i>87,14%</i>	<i>81,67%</i>	<i>90,56%</i>
Feisimo	<i>74,76%</i>	<i>73,33%</i>	<i>73,33%</i>
Feo	<i>89,52%</i>	<i>75,56%</i>	<i>85,56%</i>
Hermoso	<i>83,33%</i>	<i>78,89%</i>	<i>83,89%</i>
Orrendo	<i>84,76%</i>	<i>72,78%</i>	<i>82,22%</i>
Orribile	<i>81,90%</i>	<i>78,33%</i>	<i>81,67%</i>
Pesimo	<i>71,90%</i>	<i>80,56%</i>	<i>75,00%</i>
Precioso	<i>80,00%</i>	<i>81,60%</i>	<i>76,67%</i>
Stupendo	<i>81,43%</i>	<i>82,78%</i>	<i>76,67%</i>
Total	<i>81,85%</i>	<i>79,44%</i>	<i>81,25%</i>

Table 5.4: Results comparison

Chapter 6

Conclusions

This thesis introduces a novel application that leverages TinyML techniques to perform Speech Recognition for evaluating public perceptions of artworks in a museum. By harnessing the potential of microcontrollers to substantially minimize energy consumption and eliminate the requirement for large, resource-intensive deep learning models typically hosted on conventional servers, on-site analysis becomes feasible, allowing for immediate processing at the network’s edge. This brings several benefits over Cloud-based solutions, such as reducing data traffic and communication delays as well as preserving privacy.

The proposed approach offers a means to outperform the constraints imposed by conventional evaluation tools (*explicit feedback*). Present-day user feedback is indeed provided hastily, resulting in less-than-complete sincerity. By discerning the level of appreciation from people’s speech (*implicit feedback*), the obtained results are expected to surpass the efficacy of current state-of-the-art approaches.

The classification model was meticulously developed and trained on the Edge Impulse platform, achieving an accuracy score of 91.40% during the platform’s test phase. To assess the model’s real-world performance in a dynamic, real-time inference environment, it was seamlessly integrated into the Arduino Portenta board. Then, three distinct scenarios were thoughtfully crafted for the model’s evaluation. In the first scenario, characterized by a noise-free background, the model exhibited an impressive accuracy rate of 81.85%. In the second scenario, with background voices present, the model maintained a robust accuracy level of 79.44%. Lastly,

in the third scenario featuring a backdrop of white noise, the model demonstrated a commendable accuracy of 81.25%. The proposed research underscores the viability of utilizing TinyML for the development of SA systems, offering a practical and promising approach to meet the evolving demands of a customer-centric and increasingly demanding market. Notably, Edge Impulse emerges as a cutting-edge software solution, showcasing its prowess in optimizing the application of TinyML techniques. Moreover, it proves its adaptability even on systems with limited CPU resources.

Future works

The research proposed in this thesis represents a starting point for future investigations in this field. It offers several avenues for potential improvement and expansion. Firstly, there is room for enhancing the dataset. This could involve increasing the volume of data, improving data quality, and diversifying the content. Expanding the vocabulary to include words commonly used by users to describe attractions at places of interest is one possibility. Additionally, adding support for languages such as English, which is not currently present in the dataset, would broaden its applicability.

In the future, it may be worthwhile to transmit analysis results obtained on the microcontroller to an edge server, possibly using protocols like LoRa. This would enable further data processing and analysis. To ensure data privacy, only the analysis results could be transmitted, which could then be used to create models and track trends through a dashboard. This would provide real-time updates on the management of places of interest based on visitor sentiment.

Expanding the ML model is another avenue for improvement. Beyond word-based analysis, incorporating additional features such as tone of voice, facial expressions, and other multimodal data could significantly enhance the depth and accuracy of the results. Complex computations performed on the server-side would further refine these insights.

Lastly, optimizing components beyond ML, such as the recording sketch running on the Arduino Portenta, is essential. This component has a direct impact on the system's output and should be a final focus for achieving the desired results.

Bibliography

- [1] Chamandeep Kaur. “The Cloud Computing and Internet of Things (IoT)”. In: *International Journal of Scientific Research in Science, Engineering and Technology* (2020). DOI: <https://doi.org/10.32628/IJSRSET196657>.
- [2] Meng Ma, Ping Wang, and Chao-Hsien Chu. “Data Management for Internet of Things: Challenges, Approaches and Opportunities”. In: *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*. 2013, pp. 1144–1151. DOI: [10.1109/GreenCom-iThings-CPSCoM.2013.199](https://doi.org/10.1109/GreenCom-iThings-CPSCoM.2013.199).
- [3] Ansari Fatima Anees et al. “Survey paper on sentiment analysis: Techniques and challenges”. In: *EasyChair2516-2314* (2020).
- [4] Margaret A. Boden. “Artificial Intelligence”. In: *Artificial Intelligence: A Very Short Introduction*. Oxford University Press, Aug. 2018. ISBN: 9780199602919. DOI: [10.1093/actrade/9780199602919.001.0001](https://doi.org/10.1093/actrade/9780199602919.001.0001).
- [5] Issam El Naqa and Martin Murphy. “What Is Machine Learning?” In: Jan. 2015, pp. 3–11. ISBN: 978-3-319-18304-6. DOI: [10.1007/978-3-319-18305-3_1](https://doi.org/10.1007/978-3-319-18305-3_1).
- [6] Mohamed Alloghani et al. “A Systematic Review on Supervised and Un-supervised Machine Learning Algorithms for Data Science”. In: *Supervised and Unsupervised Learning for Data Science*. Ed. by Michael W. Berry, Azlinah Mohamed, and Bee Wah Yap. Cham: Springer International Publishing,

- 2020, pp. 3–21. ISBN: 978-3-030-22475-2. DOI: [10.1007/978-3-030-22475-2_1](https://doi.org/10.1007/978-3-030-22475-2_1). URL: https://doi.org/10.1007/978-3-030-22475-2_1.
- [7] Yap Yan Siang, Mohd. Ridzuan Ahamd, and Mastura Shafinaz Zainal Abidin. “Anomaly Detection Based on Tiny Machine Learning: A Review”. In: *Open International Journal of Informatics* 9.Special Issue 2 (Nov. 2021), pp. 67–78. DOI: [10.11113/oiji2021.9nSpecialIssue2.148](https://doi.org/10.11113/oiji2021.9nSpecialIssue2.148). URL: <https://oiji.utm.my/index.php/oiji/article/view/148>.
- [8] Sridhar Gopinath et al. “Compiling KB-Sized Machine Learning Models to Tiny IoT Devices”. In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 79–95. ISBN: 9781450367127. DOI: [10.1145/3314221.3314597](https://doi.org/10.1145/3314221.3314597). URL: <https://doi.org/10.1145/3314221.3314597>.
- [9] Keiron O’Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: [1511.08458](https://arxiv.org/abs/1511.08458) [cs.NE].
- [10] Ericsson. *Ericsson Mobility Report*. <https://www.ericsson.com/en/reports-and-papers/mobility-report/reports>.
- [11] Shuiguang Deng et al. “Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence”. In: *IEEE Internet of Things Journal* 7.8 (2020), pp. 7457–7469. DOI: [10.1109/JIOT.2020.2984887](https://doi.org/10.1109/JIOT.2020.2984887).
- [12] Weisong Shi et al. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (2016), pp. 637–646. DOI: [10.1109/JIOT.2016.2579198](https://doi.org/10.1109/JIOT.2016.2579198).
- [13] Vincenzo Barbuto et al. “Disclosing Edge Intelligence: A Systematic Meta-Survey”. In: *Big Data and Cognitive Computing* 7.1 (2023). ISSN: 2504-2289. DOI: [10.3390/bdcc7010044](https://doi.org/10.3390/bdcc7010044). URL: <https://www.mdpi.com/2504-2289/7/1/44>.
- [14] Dianlei Xu et al. “Edge Intelligence: Empowering Intelligence to the Edge of Network”. In: *Proceedings of the IEEE* 109 (Nov. 2021), pp. 1778–1837. DOI: [10.1109/JPROC.2021.3119950](https://doi.org/10.1109/JPROC.2021.3119950).

- [15] Martín Abadi et al. *TensorFlow: A system for large-scale machine learning*. 2016. arXiv: [1605.08695](https://arxiv.org/abs/1605.08695) [cs.DC].
- [16] *TensorFlow Lite*. <https://www.tensorflow.org/lite>.
- [17] *Edge Impulse*. <https://edgeimpulse.com/>.
- [18] Shawn Hymel et al. *Edge Impulse: An MLOps Platform for Tiny Machine Learning*. 2023. arXiv: [2212.03332](https://arxiv.org/abs/2212.03332) [cs.DC].
- [19] *Portenta H7*. <https://docs.arduino.cc/hardware/portenta-h7>.
- [20] *Background noises generator*. <https://noises.online/>.
- [21] *Building custom processing blocks*. <https://docs.edgeimpulse.com/docs/edge-impulse-studio/processing-blocks/custom-blocks>.
- [22] Howard B. Demuth et al. *Neural Network Design*. 2nd. Stillwater, OK, USA: Martin Hagan, 2014. ISBN: 0971732116.