Intelligent Sports Weights

Olga dos Santos Duarte

(Grau de Licenciatura)

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador:
Doutor Rui António Policarpo Duarte

Júri:
Presidente: Doutor Tiago Miguel Braga da Silva Dias
Vogais: Doutor Mário Pereira Véstias
Doutor Rui António Policarpo Duarte

Outubro 2024

Intelligent Sports Weights

Olga dos Santos Duarte

(Grau de Licenciatura)

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador:
Doutor Rui António Policarpo Duarte

Júri:
Presidente: Doutor Tiago Miguel Braga da Silva Dias
Vogais: Doutor Mário Pereira Véstias
Doutor Rui António Policarpo Duarte

Outubro 2024

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor, Prof. Dr. Rui Policarpo Duarte, and chair of my committee for they knowledge or expertise shared with me. I also could not have undertaken this journey without my defense committee, who generously provided knowledge and expertise. Thanks should also go to Daniela Ribeiro, personal trainer for her advice and suggestions for the exercises.

I am also grateful to my classmates and colleagues, especially my office mates, for their inspiration and moral support. Thanks should also go to the study participants from the university, who in an indirect way helped me achieve the goals of this thesis.

Lastly, I'd like to mention my family, especially my husband and children for all the support. Their belief in me has kept my spirits and motivation high during this process.

Statement of integrity

I declare that this dissertation is the result of my personal and independent research. Its content is original, and all sources listed in the bibliographic references were consulted and are duly mentioned in the text. I further declare that all scientific and technical references relevant to the development of the work are duly cited and included in the bibliographic references.

The author

Olga Duarte

Lisbon, 30th September 2024

Abstract

This report presents the study on the development of a low-power autonomous embedded system for classifying the correctness of gym exercises. The system is designed to be attached to gym weights (dumbbells), enabling real-time monitoring and feedback during workout sessions. It covers the solution design, implementation, and performance results, demonstrating the system ability to process movement data efficiently while maintaining low energy consumption. Using movement data from an IMU accelerometer, the system employs NN techniques adjusted for the limited computational power of a MCU. Real-time inference is achieved with a TFLite, demonstrating classification accuracy ranging from 60% to 94.1% across six exercises. A novel dataset, collected with the guidance of a certified personal trainer, was created to support training and validation. Additionally, the system's potential applications in improving injury prevention are discussed, along with recommendations for expanding datasets and refining accuracy in future developments.

Keywords: Neural Networks, Embedded Systems, BLE, Motion classification, TinyML.

Resumo

Este relatório apresenta o estudo sobre o desenvolvimento de um sistema embebido autónomo de baixo consumo de energia para a classificação da correção de exercícios de ginásio. O sistema foi projetado para ser unido a pesos de ginásio (halteres), permitindo a monitorização e feedback em tempo real durante as sessões de treino. O relatório abrange o desenho da solução, a implementação e os resultados, demonstrando a capacidade do sistema de processar os dados de movimento de forma eficiente, mantendo um baixo consumo de energia. Usando dados de movimento de um acelerometro IMU, o sistema aplica técnicas de Redes Neurais ajustadas para as limitações computacionais de um MCU. A inferência em tempo real é alcançada com um modelo TensorFlow Lite, demonstrando uma precisão de classificação que varia de 60% a 94,1% em seis exercícios. O dataset foi recolhido com a orientação de uma personal trainer certificada, foi criado para dar suporte ao treino e validação da rede. Adicionalmente, são discutidas as potenciais aplicações do sistema na prevenção de lesões, a expansão dos dados recolhidos, juntamente com recomendações para futuras melhorias.

Palavras-Chave: Redes Neuronais, Sistemas embebidos, BLE; Classificação de movimentos; TinyML; Baixo-Consumo.

Contents

List of Figures

List of Tables

Listings

Acronyms

AI Artificial Intelligence

BLE Bluetooth Low-Energy

CNN Convolutional Neural Network

FBT Full-Body Tracking

GPU Graphics Processing Unit

HAR Human Activity Recognition

IMU Inertial-Magnetic Measurement Unit

IoT Internet of Things

LSM6DS3 LSM6DS3 IMU

ML Machine Learning

MCU Microcontroller Unit

MEM Micro Electro-Mechanical System

NN Neural Network

PRA Peak Resultant Acceleration

PVA Peak Vertical Acceleration

TinyML Tiny Machine Learning

TFLite TensorFlow Lite for Microcontrollers

TFLiteLibrary TensorFlow Lite Library

DNN Deep Neural Network

SRAM Static Random-Access Memory

LiPo lithium-polymer

CPU Central Processing Unit

MAE Mean Absolute Error

1

Introduction

This thesis presents the development of the work carried out for the conception of a system for automatically recognizing the correctness of movements in physical exercise using Artificial Intelligence (AI) algorithms. The system is trained using Convolutional Neural Networks (CNNs) and integrates various technologies, including wearable [3] sensors and low-power Microcontroller Units (MCUs). This chapter introduces the motivation and objectives of this work.

1.1 Motivation

Technology miniaturization has improved daily life, especially in health and fitness by providing smaller and more capable devices. Exercise plays a key-role in maintaining a healthy lifestyle [4]. However, the rhythm of everyday life prevents us from making use of expert advice to guide physical exercises. Many people rely on online tutorials and applications to exercise in their spare time. As a consequence, many exercises performed without supervision can lead to injuries.

During and after COVID-19 a problem has been identified: some people, due to time or space constraints are unable to practice exercise under the supervision of professionals. Nowadays it's very easy to purchase and use equipment without this supervision, showing the need for a system that can track the equipment or the user and provide feedback if the exercise is correct or incorrectly performed. Most current systems rely on image acquisition, which is more sensitive to changes in scenery and lighting. The emergence of embedded systems with very compact sensor capabilities and extremely low-power consumption facilitates the introduction of smart movement sensors in gym equipment, like weights, to provide feedback on the

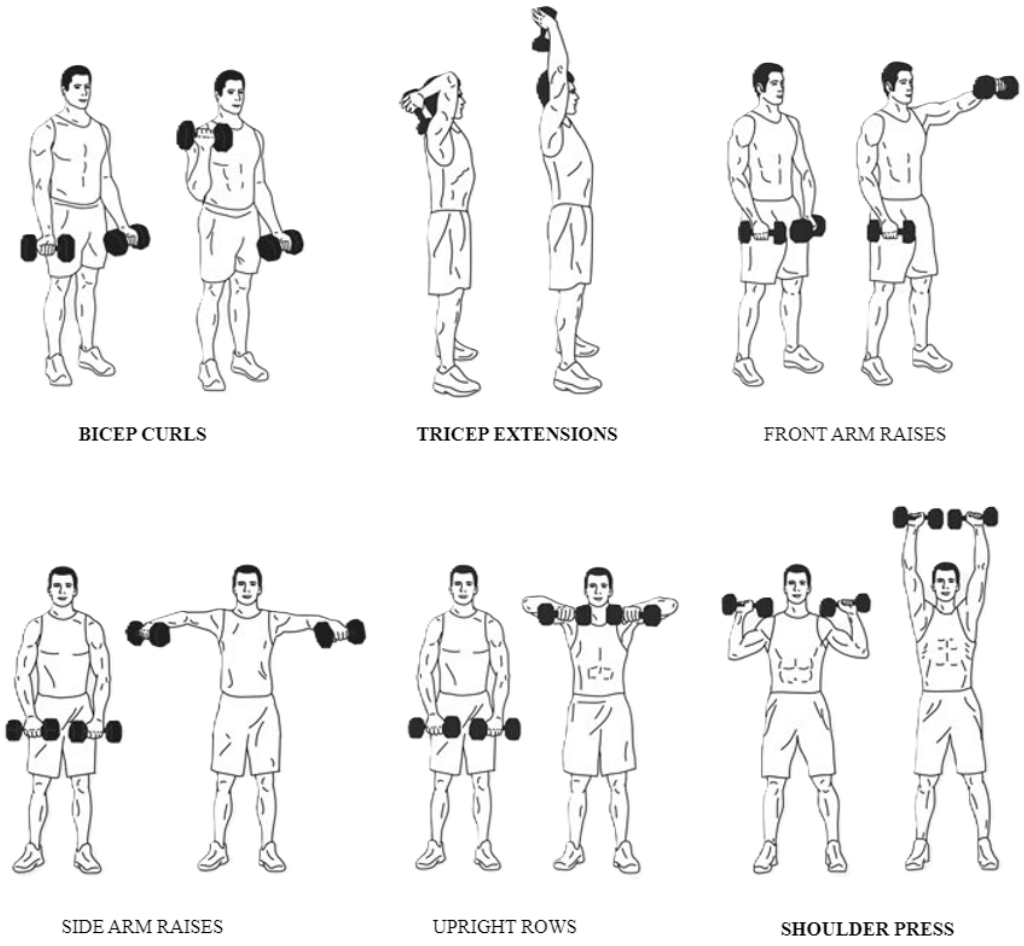


Figure 1.1: Example of exercises done with dumbbells: bicep curls, tricep extensions, front arm raises, side arm raises, upright rows and shoulder press.

correctness of exercise movements in real-time. The system is capable of providing a complete view of the movement being performed by the subject, focusing on a specific set of exercises rather than random ones. The goal is to create a classifier that can identify whether the movement matches a pattern of well-executed exercises.

The motivation to build this system relies on the capacity to have an autonomous system that is capable of identifying the correctness of fitness exercises to avoid injury [5].

1.2 Objectives

The objective of this work is to develop an autonomous embedded system to give the gym user feedback about the correctness of the exercise that is being done in real-time, as illustrated in figure 1.1 from [2].

The proposed solution is capable of providing visual feedback about the correctness of the exercise being performed, either via an LED or OLED display. The system should also be capable of collecting movement data from the IMU, as illus-

Figure 1.2: Dumbbell and the axis.

trated in figure 1.2 for accelerometer readings and show the user in a user-friendly way if the exercise is being well performed, as illustrated in figure 1.3, where green represents the exercise correctly done and red showing the deviation from a correct position for the exercise.

This work planned to bring Machine Learning (ML) models to devices that require low power and low memory to operate, thus increasing the system's efficiency which involves hardware and connection issues. The Tiny Machine Learning (TinyML) concept, explained in more detail in the next chapter, will revolutionize the research scope for Human Activity Recognition (HAR) since this kind of system needs to be responsive, not dependent upon the Internet connection, cost-efficient, size, memory, etc. Being TinyML one of the options [6], HAR research field will now have a wide range of scopes that are more environment-friendly and accessible to everyone (also low cost). This work includes a dataset collected to be trained, a Neural Network (NN) model which was trained for movement recognition purposes and a classifier of the movement based on this NN.

The system goal was to be attached to the gym equipment and follow the key requirements below:

- Autonomy (long-lasting battery) - operates autonomously with a battery for more than 8 hours, ensuring prolonged usage without the need for frequent recharging;
- Compact Design - compact and lightweight facilitating ease of use during exercise and to be attached to gym equipment with a magnet;
- Real-time Data Acquisition - collects movement data from sensors in real-time and communicates this information wirelessly to a host device using Bluetooth Low-Energy (BLE) technology;

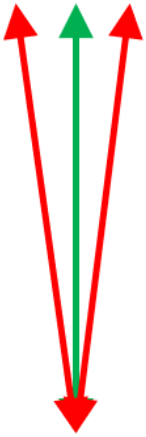


Figure 1.3: Correct and incorrect movements.

- Training the system to support custom exercises - Trained NN with a custom dataset for different exercise movements and with correct and incorrect labels:

  1. Extracting data from the sensor via BLE;
  2. Dataset generation, using different techniques and platforms explained later on in this report;
  3. Selection and training of the most suitable NN.

- Classification - validate the movement via a trained NN. The work to be performed consists of:

  1. Program an application to present different exercises to the user;
  2. Keep track of the user's performance (score/repetitions).

- Feedback - provide real-time feedback to the user. This feedback includes the evaluation of exercise movements as either well or poorly executed, and the number of performed exercises.

1.3 Contributions

The work developed in this thesis is publicly available on GitHub repository, where: the dataset, the experiments, the MCU code and the mobile code are shared. The repository can be found at https://github.com/osduarte/TESE-MEIC-2324.

This work has been published as an article in the Future Generation Computer Systems journal (Elsevier).

1.4 Report Overview

This paper is further structured into few more sections such as follows: Chapter 2 presents relevant related works to understand the state of the art, and highlights the NNs for movement recognition; Chapter 3 introduces the relevant technologies used for achieving the embedded autonomous system, such as MCUs, BLE; Chapter 4 describes the proposed solution for a automatic movement recognition system, showing the high-level architecture; Chapter 5 describes the creation of the dataset, from the raw data and the adjustments done, it also introduces important concepts, such as feature extraction and sliding window; Chapter 6 illustrates the training of the NN in different platforms and technologies; Chapter 7 presents the implementation of the system in the MCU using TensorFlow Lite for Microcontrollers (TFLite) techniques; Chapter 8 shows the implementation of the mobile application that shows the user the feedback from the system; Chapter 9 complements Chapter 6 by comparing and discussing the obtained results. The study concludes in Chapter 10 with its conclusions and future work.

2

Background and Related Works on NN for Movement Recognition

This chapter introduces important concepts needed to understand the work done and the investigated related works and the state-of-the-art of existing solutions. It also covers a brief description of CNN and its layers.

2.1 Human Activity Recognition

HAR is a branch of computational science and engineering that tried to create systems and techniques capable of automatically recognizing and categorizing human actions based on sensor data. It is the capacity to use sensors to interpret human body movement and determine human activity or movement [7]. In 2.1 it is illustrated a summary of the parts combined to achieve a HAR system, while in 2.2 shows the HAR flow.

HAR can analyse sports performance in various ways. It may be utilized to track and analyse athletes' movements during competition and training, anticipate new injury risks, assess the effectiveness of different training programs, follow individual athletes' growth, and examine team sports' tactical and strategic options [8].

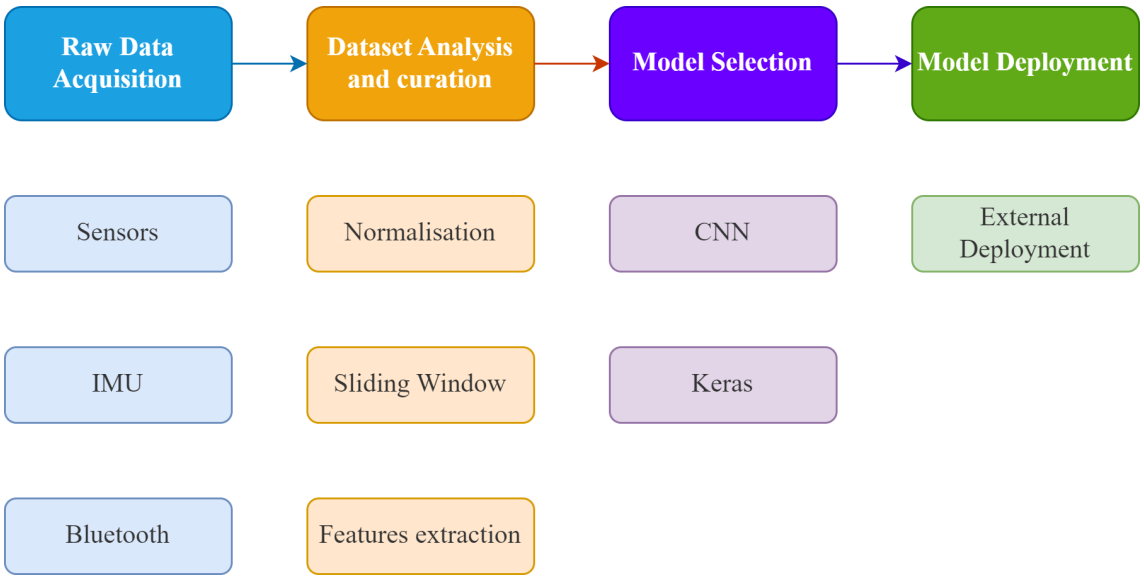


Figure 2.1: Human activity recognition summary for this thesis.

2.2 Tiny Machine Learning and TensorFlow Lite

ML is a subset of artificial intelligence where computers learn from data to make decisions or predictions without being explicitly programmed, including pattern recognition and decision-making, particularly on low-power devices like MCUs for edge computing [9] [1].

TinyML refers to the deployment of ML models on resource-constrained devices such as MCUs and small embedded systems. These devices typically have limited processing power, memory, and energy consumption capabilities, making it a challenge to run traditional machine learning algorithms [10].

TFLite is a version of TensorFlow optimised for running ML models on devices with limited computational resources. It enables the deployment of models on MCUs with minimal memory and processing power, making it ideal for embedded systems.

TFLite is an open-source framework developed by Google that enables the deployment of ML models on mobile and embedded platforms. The key features of this framework are: its optimised ML model for on-device deployment, its compatibility with several platforms including mobiles (iOS and Android) and MCUs, using TFLite for microcontrollers, and its support for multiple languages including Python, Objective-C, etc. [4].

Activity recognition using wearable sensors has become essential for a variety of applications. A typical HAR system usually includes two components: a wearable device, equipped with a set of sensors (i.e., accelerometers, gyroscopes) for capturing human movements during daily life or exercising, and a processing tool that recognizes the movement performed by the subject. MCUs are used to perform the signal acquisition [11] and storage, while an external system such as a PC, tablet, or smartphone - is needed to process signals and recognize the movements. The device

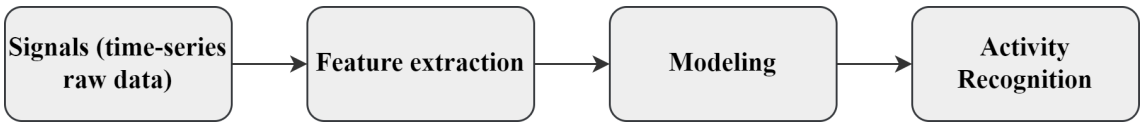


Figure 2.2: Human activity recognition flow.

should be lightweight, small, and easy to use on the subject or objects the user will use (like weights in a gym), provided with a long-lasting battery, and equipped with a MCU having enough internal memory for signals and movement storage and able to support the implementation of a classifier for activity recognition.

2.3 Convolutional Neural Network for Movement Recognition

CNNs is a key technology in modern HAR systems, enabling accurate recognition of complex activities from raw sensor data. This section provides an overview of CNN architecture, discussing how different layers and components contribute to the model's ability to learn and classify movement patterns. It was selected the CNN architecture due to its capability to process raw time-series data.

CNNs combines the feature extraction and classification in an end-to-end approach. The feature extractors are non-linear transformations, and they learn directly from raw data. In contrast, relevant handcrafted features are hard to compute and to scale. By stacking several filters and pooling operations, CNNs extract hierarchically basic and complex human movements, learning their non-linear and temporal relations.

In [12] the architecture consisted of parallel branches. A single branch contains temporal convolutions, max-pooling operations, and a final fully-connected layer. Each of these branches finds temporal relations of time-series per IMU. They create an intermediate representation of the IMUs measurements. The architecture merges each of these representations using a subsequent fully connected layer, which is then used for classification. In addition, they have investigated the effect of using max-pooling, as this operation might not preserve the information. For relatively long sequences, networks using max-pooling operations show better results. Furthermore, evaluations using different learning rates and their reduction during training were presented. These showed some mixed behaviours. The CNN-IMU benefits from a relatively small learning rate as compared to the baseline CNN. For the bigger learning rate, decreasing the learning rate during training is advantageous.

The purpose of this thesis is to describe an automatic system for movement recognition in gym exercises. To this end, it explored the applicability of NNs [13] for sensor-based activity classification and tried to compare it to other state-of-the-art classification methods, while common data inputs are obtained from IMUs, using

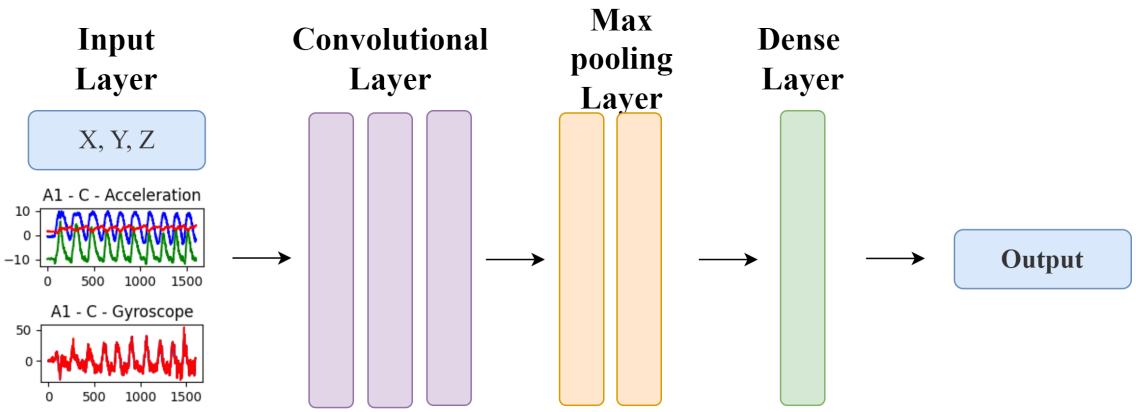


Figure 2.3: Generic convolutional neural network architecture.

afterward a NN platform to train a model with the extracted data.

CNNs works by applying a series of convolution and pooling layers to an input image. Convolution layers extract features from the input by sliding a filter over the image and computing between the filter and the input [14]. Pooling layers then down-sample the output of the convolution layers to reduce the dimensionality of the data and make it more computationally efficient. CNN consists of multiple layers as showed in figure 2.3.

The study [12] presents an evaluation of a CNN architecture for HAR using multichannel time-series data from body-worn sensors, specifically IMUs, for recognizing human activities using data from those sensors. This NN is designed to better capture how people move by considering information from different sensors. The researchers tested this approach on various datasets and found that it outperforms existing methods in accurately identifying human activities. They also explored different factors, like the use of max-pooling and learning rates, to fine-tune the network for better results.

CNN-based approach is practical and achieves higher accuracy than existing state-of-the-art methods [15].

CNNs is composed of multiple layers that process input data hierarchically. This section describes the general architecture of CNNs used in HAR, highlighting the roles of convolutional, pooling, and fully connected layers.

The network structure is mainly composed of an input layer, convolutional layers, pooling layers, and fully connected layers. The input layer is responsible for collecting data and forwarding it to the next layer. The convolutional layer, which contains several convolution filters (kernels) that convolve with the input data, is the main core of a CNN and it is responsible for reducing the dimension of the input data automatically extracting useful features. The pooling layer, also known as the sub-sampling layer, is then used to further reduce the number of parameters and the resulting computational cost by incorporating max-pooling and average-pooling operations. Finally, a fully connected layer takes in input from the previous layer's

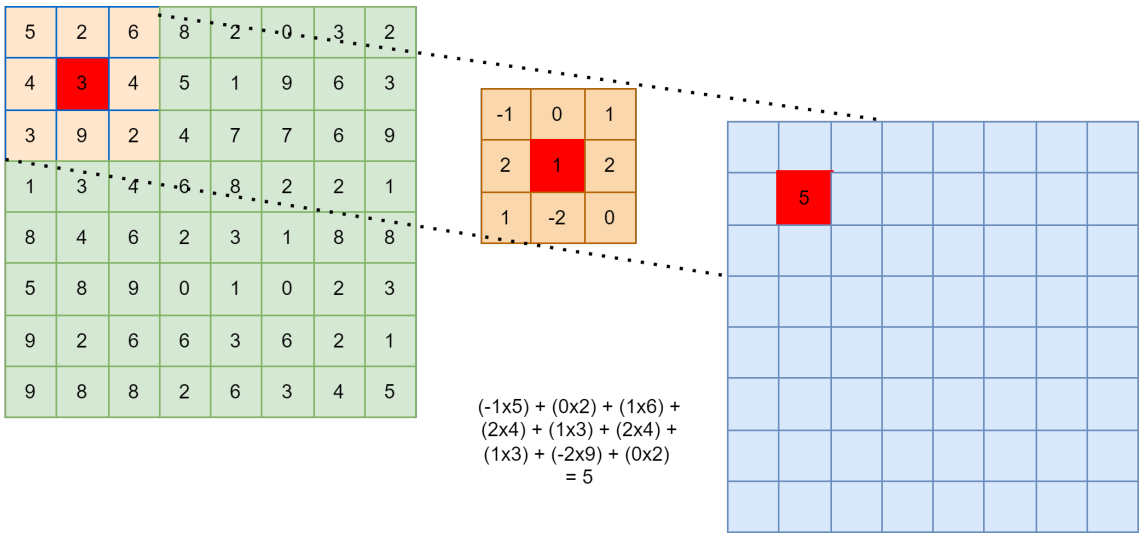


Figure 2.4: Example of a convolutional operation.

features. A CNN has usually been used in the Deep-Learning approach due to its ability to eliminate the need for feature extraction and feature selection, often at the expense of increased computational complexity and memory usage [16].

2.3.0.1 Convolutional Layers

A convolutional layer is a hidden layer that contains several convolution units in a CNN, which is used for feature extraction. Convolution Kernels, also known as filters, are small matrices used for the convolution operation. These kernels slide across the input data, performing element-wise multiplication with the corresponding pixels and producing a feature map that highlights specific patterns or features in the input.

The process of applying a convolution kernel to the input data is referred to as the convolution operation. This operation involves sliding the kernel across the input, computing the dot product at each position, and generating an output feature map, as illustrated in figure 2.4.

In the context of CNNs, features refer to the meaningful patterns or characteristics extracted from the input data by the convolutional layers. These features represent different aspects of the input, such as edges, textures, or more complex structures. Each convolutional kernel is responsible for detecting specific features in the input data.

2.3.0.2 Pooling Layers

The goal of pooling is to transform the joint feature representation into a new, more usable one that preserves important information while discarding irrelevant details. For example, the assumption underlying the computation of a histogram is that the average feature activation matters, but exact spatial localization does

not. Achieving invariance to changes in position or lighting conditions, robustness to clutter, and compactness of representation are all common goals of pooling [17].

2.3.0.3 Fully Connected Layers (Dense Layers)

A fully connected layer, also known as a dense layer, is a layer in which every neuron (or node) is connected to every neuron in the previous layer and every neuron in the following layer. These layers are responsible for combining features learned in previous layers and making a prediction.

2.4 Related Works on Human Activity Recognition

HAR has been a topic of considerable research, with various approaches being explored to achieve accurate classification of physical activities. Traditional methods often relied on handcrafted features extracted from sensor data, but recent advances have seen a shift towards CNNs, which automates feature extraction and can improve recognition accuracy.

2.4.1 The Accelerometers Effectiveness to Collect Sensor Data

Accelerometers are key sensors for HAR. Accelerometers are a popular choice of sensor due to their size, low cost, and reliability. Studies show that they have been placed in multiple locations, including the waist (belt), wrist, upper arm, chest, pelvis, thigh, and ankle. The wrist has shown the best performance for recognition compared with other positions according to the study [18]. Achieving good results in HAR may include several steps: signal pre-processing, feature extraction, and classification. The approach is to segment the signal using a sliding window with some overlap, which helps reduce noise. The key point of this work was to investigate the significance of the accelerometer signal features. Other studies have applied tri-axial accelerometers to recognize specific sports movements.

For example, in skateboarding, the study [19] achieved high accuracy results when classification tricks based on movement data from IMU attached behind the front axis board. By analysing up to 54 statistical features including mean, variance and kurtosis, among others, the results showed an accuracy of 97.8% for the best performing classifiers. Unlike video-based methods, IMUs does not require any external equipment and offers the advantage of portability which is key for skateboarding environments, and others.

Comparably, in golf, the study [20] developed a mobile golf putt system consisting of an IMU for data collection and wireless transmission. IMUs are small,

non-stationary, and wireless, so they facilitate full integration on wearable systems. The sensor sampled data at 256 Hz and transmitted data wirelessly via Bluetooth to facilitate real-time movement tracking. The results showed that putts are automatically detected with high probability if a basic swing model is followed, 8 out of 11.

In volleyball, the study [21] investigated the possibility of using Peak Vertical Acceleration (PVA) or Peak Resultant Acceleration (PRA) measured by an accelerometer to estimate jump frequency to avoid injuries. Unfortunately, despite the rigorous protocol, the study found that neither PVA nor PRA effectively differentiated between jumping and non-jumping movements, strongly implying that the methodology explored proves ineffective for estimating jump frequency accurately, failing to reach the expected results.

Another study also related to volleyball, [22] used a CNN to achieve high accuracy in classifying volleyball exercises, significantly outperforming traditional methods based on manually defined features, with classification accuracy exceeding 90%, compared to around 75% for manual feature extraction. By combining wearable sensors and pattern recognition, an automatic activity recognition system for player monitoring in beach volleyball was established.

In skiing, the study [23] focused on classifying skiing techniques with high accuracy using IMUs attached to the skier's arm and chest together with a ML algorithm using MATLAB. The use of a gyroscope for cycle detection and an accelerometer for classification was a key innovation in this study. Still in the ski sport the study [24] focuses on the predict jumping errors using a IMU-CNN approach.

For running, the study [25] seeks to determine the capability of IMUs to detect changes in a runner's non-fatigued and fatigued wearable body sensor. The results showed an accuracy exceeding 85%.

Study [26], provides an introduction to the standard procedures and best practices developed by the activity recognition community for designing, implementing, and evaluating HAR systems, not limited to activity recognition using wearable sensors. For educational purposes, the article is complemented with a publicly available dataset and a feature-rich activity recognition framework implemented in MATLAB.

The study [27] introduced an ultra-lightweight, application-agnostic, and on-device learning and inference algorithm for on-sensor movement recognition, reaching 96.7% test accuracy, showcasing the potential for real-time, low-cost, low-power applications across various domains.

Overall, in HAR studies, each study addresses different activities, datasets, feature extraction, and classification, making it very complicated to perform comparisons. Table 2.1 summarized a comparison between the previous studies for better understanding.

Table 2.1: Summary of the analysed studies used in sports activities.

Study	**Sensors**	**NN?**	**Results**	**Data Acquired**	**Classification & Datasets**
Skateboarding [19]	Accelerometer, Gyroscope	No	97.8% accuracy	Data from inertial sensors placed on skateboard	Classification of six tricks performed by seven skateboarders (all male)
Golf [20]	Accelerometer, Gyroscope	No	Detected putts with high probability	IMU data at 256 Hz, transmitted wirelessly via Bluetooth	Applied to analyse putt techniques in golf.
Volleyball [21]	Accelerometer	No	Not effective for estimating jump frequency	Peak vertical/resultant acceleration during high-intensity exercises	Data was collected for jump frequency estimation.
Volleyball [22]	Accelerometer	Yes	Accuracy exceeding 90%	Three-axial acceleration data were recorded from 30 subjects (11 female, 19 male) during beach volleyball training. A total of 39 features were calculated.	CNN used for activity recognition
Cross-Country Skiing [23]	Accelerometer, Gyroscope	Yes	93.9% classification accuracy	Data from two IMUs (arm and chest) across 24 datasets	NN used for classification
Running [25]	IMU	Yes	Accuracy exceeding 85%	IMU data collected during running	Signal processing and classification analyses were completed using MATLAB
General movement Recognition [27]	IMU	Yes	96.7% test accuracy	On-chip processing for real-time movement recognition	TinyML-based

2.4.2 CNN for HAR Using Mobile and Wearable Sensors

Wearable sensors for HAR rely on compact, lightweight devices with sufficient memory and efficient MCUs to capture, store, and process movement data in real-time. A challenging and still open aspect when dealing with HAR is the identification of the correct set of features for the classifier. Many studies focus on the feature extraction for HAR. In [28], the approach was based on online activity recognition on movement sensor data obtained from real-world smart homes, with four methods used to extract features from the sequence of sensor events and explore both fixed static window size and dynamic varying window size. The results demonstrated that dynamic window sizes got the best performance.

Wearable sensors [29] provide a low cost for measuring signals that are related to player movements. Powerful pattern recognition techniques such as Deep Learning could be used for analysing these signals. By combining wearable sensors and pattern recognition, an automatic exercise recognition system for the user was implemented. MCU plays a critical role, it is responsible for running the trained model and making real-time predictions (inference) based on the sensor data. The MCU receives the information from the sensors, runs the model using the TensorFlow Lite library, and gives the result to the user, indicating if the specific exercise is correctly performed or not.

The study [30] aimed to evaluate two different feature sets (time-frequency and time-domain) for HAR using data from 61 healthy subjects who performed seven daily activities - resting, upright standing, level walking, ascending and descending stairs, walking, uphill and downhill - while wearing an IMU-based device. The device captured nine signals, including acceleration, rate of turn, and Earth-magnetic field data, with a sampling frequency of 80 Hz. Each signal was segmented using a 5s sliding window with a 3s overlap. The study demonstrated that the time-frequency feature set consistently delivered higher accuracy.

A traditional HAR uses a sliding-window approach to segment sequences, extracts the relevant hand-crafted features and trains a classifier for assigning certain action labels to those sequences. Sometimes it can be difficult to classify due to the variability of human actions. To overcome this HAR problems in the context of gesture recognition CNNs have been used. CNNs combine conveniently the feature extraction and classification in an end-to-end approach. [12] introduced a CNN for HAR, using multichannel time-series acquired from body-worn sensors, i.e., IMUs and exploring the layers of a CNN architecture. The study concludes that the CNN-IMU architecture, with its use of parallel branches and convolution operations, is effective for HAR using body-worn sensors, particularly when employing max-pooling for longer sequences and optimising learning rates during training.

Health is another area that is investigating wearable devices for HAR recognition. [4] focus their efforts on replacing the cloud servers with an alternative

inference device closer to the sensing platform, like MCUs. The study examines the challenges and trade-offs in using MCUs for health and care applications, they found that power consumption is an important challenge that needs to be taken into consideration during the design process since power will have an impact on the performance.

The [31] aimed to enhance pedestrian safety through the development of wearable systems that leverage advanced sensors and algorithms. Their study focused on creating a system that could accurately detect and predict pedestrian movements, using data from wearable sensors to provide real-time feedback and alerts. By integrating these wearable systems with movement recognition techniques, the researchers tried to reduce pedestrian accidents and improve overall safety.

The [32] study goal was to enable the deployment of Deep Neural Networks (DNNs) on resource-constrained MCUs when their working memory exceeds the RAM size. It was proposed a lightweight run-time working memory compression. Experimental results show that without adding a lot of overhead on memory, the compressed DNNs could maintain the original accuracy or run with moderate accuracy loss.

The increasing growth of tiny wearable devices and, at the same time, the advent of ML techniques that can perform sophisticated inference, represent a valuable opportunity for the development of a system like it will be explained later on in this thesis. Moreover, pushing inference on edge devices can in principle improve application responsiveness, reduce energy consumption and mitigate privacy and security issues. However, devices with small sizes and low-power consumption have some restrictions in computational power, memory, and energy requirements which result in challenging issues to be addressed when implementing this kind of system. It is necessary to explore this trade-off through the characterization of memory usage, energy consumption, and execution time needed by different types of NNs (for example CNNs) trained for HAR on board of a typical low-power wearable device.

Nowadays deep learning is making a major contribution towards the activity recognition research area where CNNs has shown great promise in the field. In [33] authors proposed an architecture that consisted of 3-layers of convolution with varying numbers of filters and the results of the work have proved to be successful, again in [34] CNN based models have been proposed by the author namely CNN reaching accuracy's over 90%.

In [15] was proposed an approach based on CNNs to recognize activities in various application domains. In the training phase, features were extracted from the raw time series data. These features are then used to train a classification model. In the classification phase, they first extract features from raw data and then use the trained prediction model to predict an activity label. The results demonstrated that the CNN approach significantly improved activity recognition

accuracy by effectively capturing temporal patterns and local dependencies in the raw sensor data. According to [15] there are two key advantages when applying CNN to HAR:

- Local Dependency: CNN captures local dependencies of an activity signal. In HAR given an activity, the nearby acceleration readings are likely to be correlated.

- Scale Invariance: CNN preserves feature scale invariant. In HAR, a person may walk with different paces.

A 1D CNN model trained on accelerometer data is suggested in the paper [35] and paper [36] for automatic feature extraction in a HAR system. In the first study, a semi-automatic approach is used that effectively determines the number of convolutional layers in the network, the number of kernels and the size of the kernels, using a Keras based NN modelling library, data from 36 subjects was collected and recorded by an accelerometer with a sampling frequency of 20 Hz, and are labelled for 6 activities – walking, jogging, ascending stairs, descending stairs, sitting and standing. The sensor data was segmented using a 10s sliding window with 50% overlap. The experimental results show that the suggested model outperforms several existing methods with higher accuracy and more efficient feature extraction. In the second the dataset was split into train (70%) and test (30%) sets based on data from 21 subjects for train and nine for test. CNN models were developed for image classification problems, where the model learns an internal representation of a 2D input. This same process can be used on 1D sequences of data, such as in the case of acceleration and gyroscope data for HAR. The model learns to extract features from sequences of observations and how to map the internal features to different activity types. The benefit of using CNNs for sequence classification is that they can learn from the raw time series data directly, and do not features to be selected manually. In [37], they propose an efficient HAR method, namely Iss2Image (Inertial sensor signal to Image - compared in table 2.2), an encoding technique for transforming an inertial sensor signal into an image and a CNN model for image-based activity classification. Iss2Image converts real number values from the X, Y and Z axes into three colour channels to infer correlations among successive sensor signal values in three different dimensions. They evaluated the method using several well-known datasets and their own dataset collected from a smartphone and smartwatch. The proposed method shows higher accuracy than other state-of-the-art approaches on the tested datasets. According to this paper, there are three different methods of HAR, video-based, wearable sensor–based, and environmental sensor–based, and each method has its own pros and cons. In wearable sensor–based methods, devices containing inertial sensor units, such as accelerometers, gyroscopes, and magnetometers, are attached to the body. Activities are then classified into types, with

each activity type showing a different pattern of sensor values. To classify an activity, the features that can best represent it must be extracted from the collected sensory data. Selecting and extracting the most meaningful features are the most difficult problems for achieving accurate HAR systems. The features usually are chosen by humans, called hand-crafted features, and include time-domain features such as mean and standard deviation and frequency-domain features. To find the most efficient and effective features, (1) the programmers must have prior expert knowledge, or (2) they must do a large amount of empirical study to learn which features are useful [30] [38]. Deep learning technology has now been developed and can be used for HAR. The most prominent advantage of using deep learning is that it can automatically extract features and select them without user manipulation. They compared the results of the proposed method with different datasets, different signal transformation methods, different CNNs, and different traditional methods, and calculated the speed of image creation time and inference time. The experimental results show that the proposed method outperforms existing methods using both owned and public datasets.

ML [39] classifiers modelled with generic hand-crafted features, were compared against a NN for classifying nine beach volleyball actions using IMUs. This study [19] it was investigated the efficacy of different transfer learning pipelines towards the classification of skateboarding tricks, by using a NN. The skateboarding tricks signals were acquired through an IMU device. The device is embedded with an MPU6050 sensor, a Bluetooth 2.0 module, a MCU, and a 3.7 V Lithium Polymer rechargeable battery. The use of NNs [40] represents an alternative to this classical classification approach based on hand-crafted features. An important advantage of NN is the automatic extraction of features.

In the master thesis [41], even if not related to sports, the goal remains the same - recognize movements - in this referenced work it was implemented a NN for gesture recognition in low-cost FPGA, it was used an accelerometer placed in a wand to detect gestures from a person with the help of a NN model, they used a MCU Arduino Nano 33 BLE Sense and SparkFun Edge and the model was trained using the TensorFlow Lite framework [42].

More than just CNN, an interesting number of studies combined IMU, NN and their results in the field of sports, the review [13] refers that an assessment of the athletes' performance is important for elite sports to facilitate detailed analysis. The implementation of automated detection and recognition of sport-specific movements overcomes the limitations associated with manual performance analysis methods. The goal of the study was to systematically review the literature on machine and deep learning for sport-specific movement recognition using IMU or computer vision data inputs. The 52 included studies must have investigated a sport-specific movement and analysed via machine or deep learning methods for model develop-

Table 2.2: Methods comparison according to the related works investigated.

	Iss2Image Method	**Wearable Sensor-Based Methods**	**Traditional Methods**
How does it work?	Transforms IMU signals into images for classification using CNNs.	Uses wearable devices with IMU for activity recognition.	Hand-crafted features based on time-domain and frequency-domain.
Accuracy	Outperforms existing methods with higher accuracy.	Accuracy depends on sensor quality and feature extraction methods.	Accuracy depends on feature selection.
Feature Extraction	Automatic extraction using CNNs from transformed images.	Hand-crafted features based on sensor readings; less automated.	Manual extraction of time and frequency domain features.
Data Processing	Involves image encoding and CNN classification.	Involves segmenting sensor data and extracting features manually or using basic algorithms.	Requires explicit manual feature extraction and selection.

ment. Data pre-processing, processing, model development, and evaluation methods were different across the studies. Model development for movement recognition was mostly done using supervised classification approaches. Twelve studies used a deep learning method as a form of CNN algorithm achieving good results.

2.4.3 Human Activity Recognition Using Tiny Machine Learning

TinyML is a good approach to enable applications that want to run HAR systems on resource-limited and low-power edge devices. However, designing efficient TinyML models for these devices remains challenging due to computational resource constraints and the need for customisation to unique use cases [43]. TinyML has seen its research progress in mainly speech enhancement techniques, speech separation, etc. but its advantages have not been fully used in HAR field. This allows for real-time analysis and inference of data at the point of collection, which translates to huge advantages in terms of cost and privacy. MCUs are the ideal hardware platform for TinyML, as they are typically small, low cost and low-power compared to mobile.

MCUs typically integrates a CPU, digital and analog peripherals, on-chip embedded flash memory for program storage, memory for storing data, and commu-

nication peripherals. However, deploying NNs on MCUs is challenging, the biggest limitation being the small memory system in which the model must be stored. Therefore, to achieve the promise of TinyML, the model must be optimised to best overcome the limited resources provided by an MCU hardware and software stack [44]. [43] proposes an approach that uses transfer learning (TL) techniques on edge MCUs to accelerate TinyML development. The strategy involves pre-training generalised ML models on large-scale and varied datasets and fine-tuning them on-device for specific applications using TL. The effectiveness of the approach for HAR was demonstrated by conducting extensive testing using two distinct datasets, using a CNN. Results showed significant model accuracy and reduced training time while maintaining high inference rates and low MCU memory uses. The efficiency, scalability, and ability to adapt to user-specific behaviour make ML an increasingly powerful tool for HAR. Designing this kind of system has challenges in size and battery lifetime, mainly due to limited batteries powering the MCU. The limited battery life impacts the adoption of smart sensors for this type of system, impacting usability due to the difficulty, cost, and time involved in replacing batteries. Achieving a satisfactory battery life requires a low-power system. To further optimise the edge-based HAR implementations, light-weight ML solutions are the key. TinyML specifically addresses space and computational constraints in low-end smart devices and presents a framework for implementing ML solutions for operation on the edge, with a particular focus on HAR. Several innovative techniques have emerged to balance the use of the available resources with the need for data bearing in mind the inherited challenges from the conventional ML task [1].

Study [44] started by analysing on-device, MCU, inference performance. Measurements demonstrate that for models sampled from a given network, the inference latency of the model is linear with the total operation count. Since MCU power is largely independent of workload, operation count is also a strong proxy for energy per inference. Their models optimised for MCUs demonstrate good performance on all three tasks: visual wake words, audio keyword spotting, and anomaly detection. Another study [45] inferred that the models can be compressed up to 10 times after model optimization for CNN, and accuracy seems to be unchanged.

3

Exploring and Evaluating Systems for HAR

This chapter introduces small validation experiments with key technologies used in the development of the HAR Autonomous System. It covers the hardware components, wireless communication protocols, and embedded systems that enable real-time inference on a portable and attachable device.

3.1 Movement Acquisition

Accelerometers [46] detect linear acceleration of devices, that is, the acceleration along an axis. Gyroscopes, on the other hand, detect the angular velocity, i.e., how fast the body is turning. To collect the data that will be trained and used in the model, it is used the accelerometer and gyroscope sensor data. After collecting, this data it will be used in a dataset trained by NNs to recognize which movement the person is performing and if it is correct or not, by means of the data collected by using the accelerometer and gyroscope data coming from an IMU sensor. As a reference, the study [47] discussed the integration of ML algorithms, specifically NN, into edge devices for HAR. It presents experimental findings that evaluated different ML models on an edge device, considering factors such as inference time, memory usage, and classification accuracy, aiming to contribute insights for future research in deploying ML solutions on resource-constrained devices. For the work being done and described in this thesis, it is also important to identify potential limitations and corrections that may be necessary in the future to apply to the collected data. In the bottom-up strategy, some simple data collection tests were

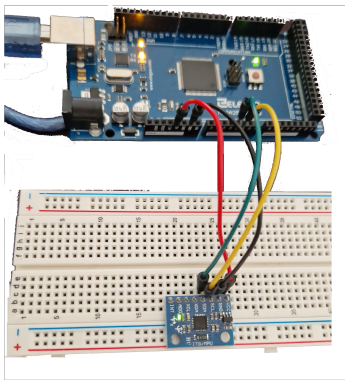


Figure 3.1: Experiments with the MPU6050 on an Arduino.

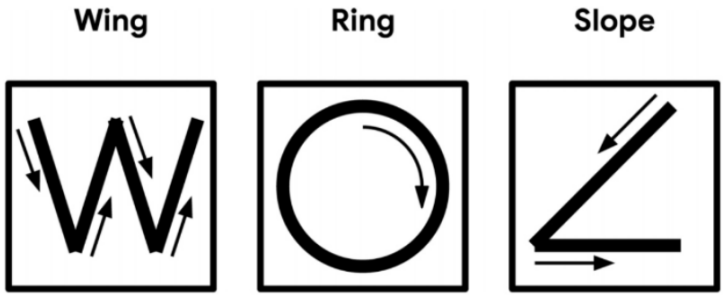


Figure 3.2: Magic Wand movements from [1].

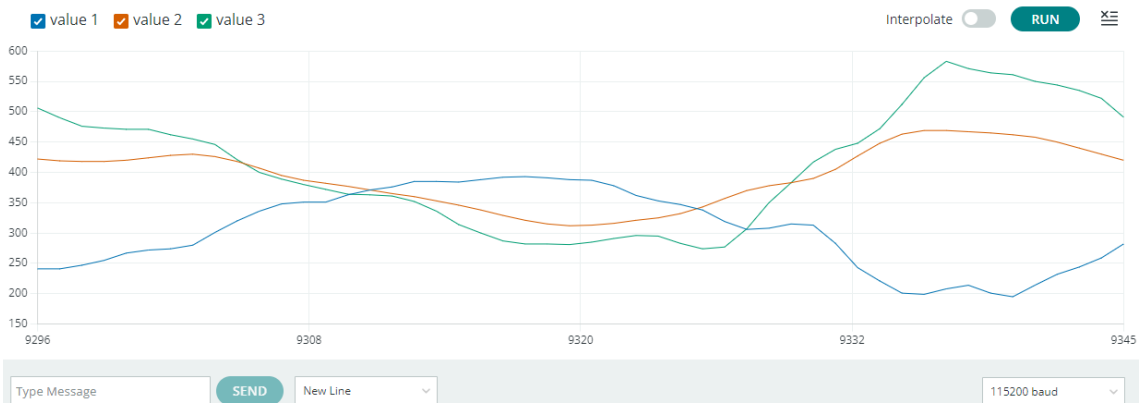
performed using MPU6050 and Arduino nano 328p, as illustrated in figure 3.1 that shows a three-axis accelerometer and three-axis gyroscope based on Micro Electro-Mechanical Systems (MEMs) technology. It facilitates the measurement of velocity, orientation, acceleration, displacement, and other movement-related features. It was not considered or compared, because of the size (too big and heavy to attach to a weight).

3.2 Wireless Communication

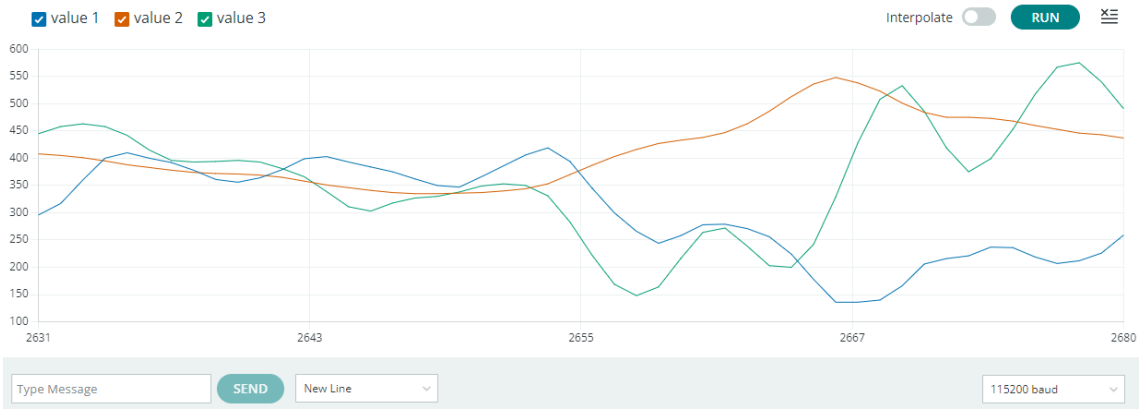
BLE is a wireless, low-power personal area network. Its goal is to connect devices over a relatively short range. BLE is is very commonly used for Internet of Things (IoT) applications. IoT devices tend to be constrained and require extended battery use, so BLE favors low power consumption over continuous data transfer.

It is an attractive solution because of its low cost, long battery life, and ease of deployment. From thermometers and heart rate monitors to smart watches and proximity sensors, BLE facilitates infrequent short-range wireless data communication between devices, powered by nothing more than a battery. Wireless communication is necessary for transmitting sensor data in real-time, especially in wearable devices. This section reviews the wireless protocols, such as BLE, used in the system to ensure efficient and reliable data transfer.

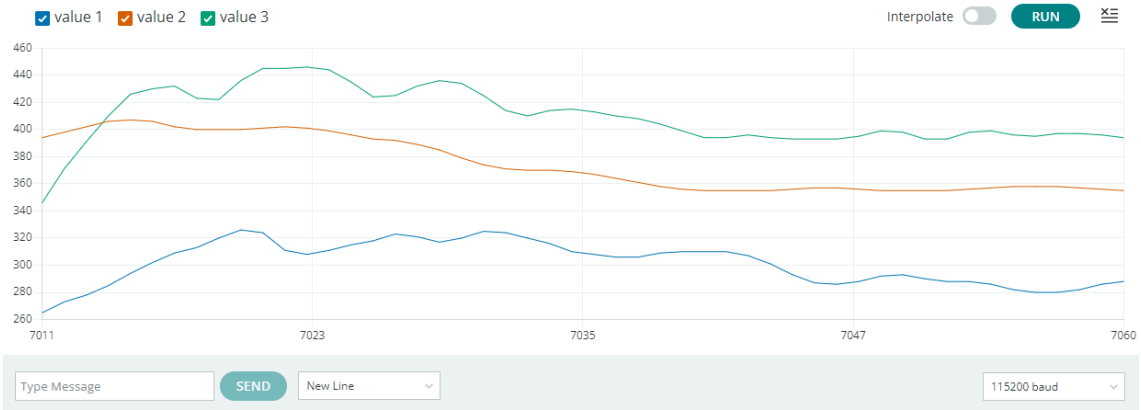
BLE was chosen instead of other wireless communication technologies, like wi-fi. BLE consumes less power and is more affordable, while Wi-Fi provides faster



(a) Wing.



(b) Ring.



(c) Slope.

Figure 3.3: Evolution of data sampled from the execution of the gestures: Wing, Ring and Slope

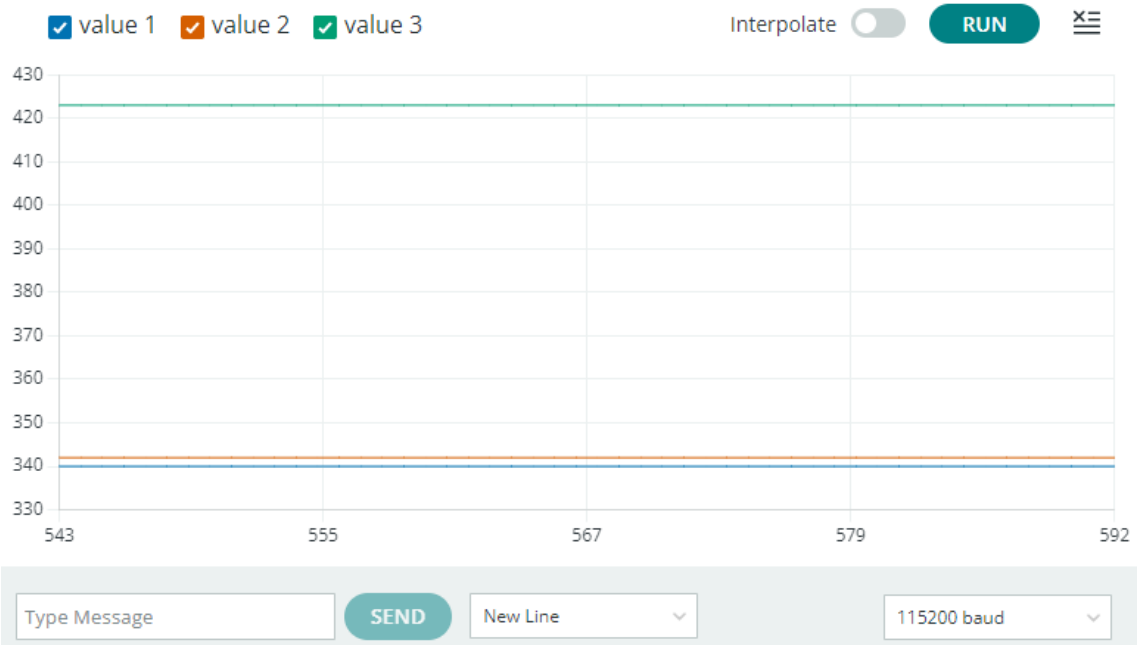


Figure 3.4: Data from the sensor in the rest position.

speeds and broader coverage. BLE fits the purpose of this thesis, showing that it's the perfect choice for communication from the device to the host.

3.3 Magic Wand TinyML Experiment

As mentioned before TinyML refers to the deployment of ML models on resource-constrained devices such as MCUs and small embedded systems, where ML tasks are executed locally on IoT devices. TinyML provides hardware and software paradigms that enable always-on, real-time, low-cost, and ultra-low-power inference on MCUs. This allows for real-time analysis and interpretation of data at the point of collection, which translates to huge advantages in terms of cost and privacy. MCUs are the ideal hardware platform for TinyML, as they are typically small, low cost and low-power compared to mobile and cloud platforms. MCUs typically integrates a CPU, digital and analog peripherals, on-chip embedded flash memory for program storage and Static Random-Access Memory (SRAM) for intermediate data. To achieve the promise of TinyML, models need to be optimised to best use the limited resources provided by an MCU hardware and software stack.

Machines and their sensors generate huge streams of information that don't map easily onto our human senses. Even when represented visually, it can be difficult for our brains to grasp the trends and patterns within the data. Inspired by the thesis [41], in this example, it was possible to detect some patterns in the results of the accelerometer and gyroscope, just by using the following material. The Magic Wand was inspired by the book about TinyML [1].

Figure 3.5: Magic Wand material.

Necessary material (see figure 3.5):

1. Arduino nano 328p + adxl335
2. Arduino IDE

An Arduino Nano was used, and on top, the accelerometer board was soldered, with the outputs of the analog signals connected to the analog inputs of the Arduino (A0, A2). This example illustrates how to connect sensors, like an accelerometer, to a MCU, like an Arduino Nano, for collecting motion data. This data was processed for gesture recognition, using the implementation source code from [1] as inspiration, but adapted to the setup described. The code in a simple way reads the data from the X, Y, and Z axis of the accelerometer and prints this information on the serial monitor.

In this experience, the goal was to analyse the 3-axis of accelerometers and gyroscope values in different movements, represented in 3.2. In this case, it was used magic wand movements, like in the book [1].

Figure 3.3, represents the raw output from Arduino Serial Plotter. At this stage, the only thing shown is the different patterns. For comparison, in figure 3.4 it can check when the output when the MCU is stopped.

3.4 Portable Ultra-Low Power Embedded Systems

An embedded device is a computing system that is designed to perform dedicated functions or tasks within a larger system. Unlike general-purpose computers like PCs or smartphones, embedded devices are typically "embedded" within the hardware they control and are optimised for specific tasks, often with real-time computing constraints. While a MCU is a small computer on a single integrated circuit. It contains one or more Central Processing Units (CPUs) along with memory and programmable input/output peripherals.

Ultra-low power embedded systems for movement recognition are designed in a very efficient way to read and process movement-related data while consuming minimal energy. Take as an example the SlimeVR [48]. SLimeVR is a set of open hardware sensors and open source software that facilitates Full-Body Tracking (FBT) in virtual reality, with no base station required. These types of systems are crucial in applications where battery life is a critical factor, enabling extended use in portable and wearable devices without the need for frequent recharging, which is important for devices like SlimeVR for fitness tracking.

Ultra-low power embedded systems can be small, low power, low cost, and wireless, for the purpose of the system, the embedded system chosen has to measure and report acceleration, orientation, and other gravitational forces. For that, it is essential to have in the IMU accelerometers and gyroscopes incorporated.

In the context of this work and with the mentioned features above to be considered, a comparison was made between four different embedded systems in terms of energy efficiency, size, weight, and memory. Just to enumerate some of the features that were taken into account to choose the best fit for the built-embedded system.

3.4.1 NRF51 Sensor Tag

The NRF51 Sensor Tag is a compact, low-power device designed for sensor applications. This subsection explores its features, including its integrated sensors and BLE capabilities, and discusses its potential use in HAR.

The first MCU tested was the nRF51822, see figure 3.7, with the following main characteristics:

1. nRF51822 Bluetooth-Compatible 4.0 BLE SOC

2. ARM CORTEX-M0

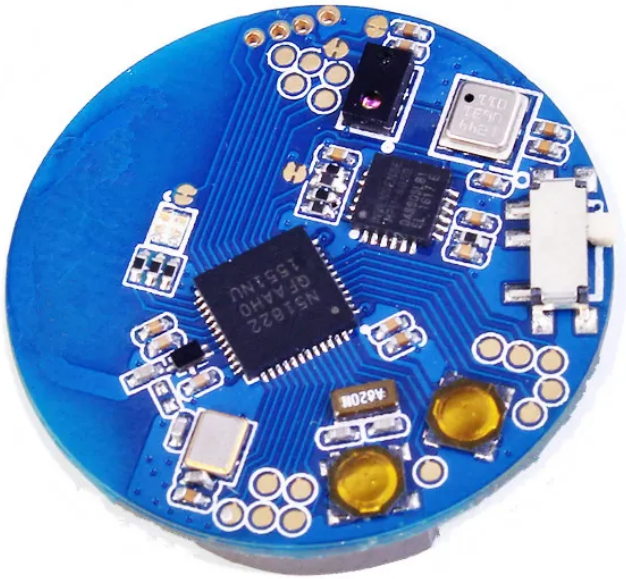


Figure 3.6: NRF51 Sensor Tag.

3. 3-axis acceleration 3-axis gyroscope chip MPU6050

4. BMP180 temperature atmospheric pressure sensor chip

5. Sensing pressure range: 300-1100 hPa; Temperature range: -40- +85 Celsius

6. Ambient light proximity sensor chip AP3216

7. With power switch

8. Support CR2032 button battery to supply power

9. Size:30mm*30mm

Several tries were made, using apps in a mobile phone to extract or even connect to collect the data. Apps used: LightBlue, BLE Scanner, nRF Connect, EFR Connect.

It was excluded due to the lack of examples and public documentation. No data-sheet is available, making it very difficult to use and experiment, as illustrated in figure 3.16 number 2.

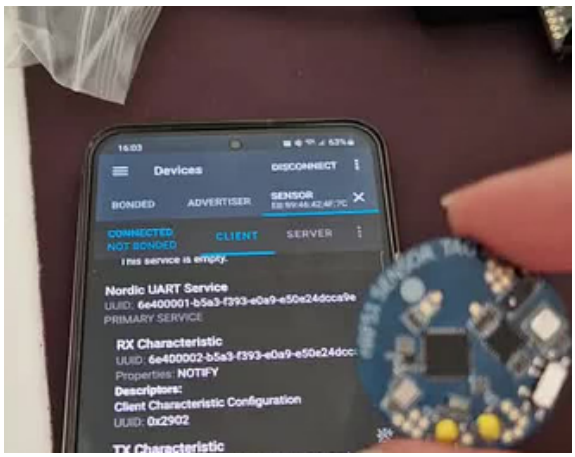


Figure 3.7: Example of a mobile app used with the NRF51 Sensor Tag.

3.4.2 Experiments With the Seeed Studio XIAO nRF52840 Sense

The Seeed Studio XIAO nRF52840 Sense is a compact MCU with good processing capabilities and a range of onboard sensors. This subsection details its specifications and how it was chosen for the HAR system implementation.

XIAO nRF52840 Sense, represented in figure 3.16 number 4, features an onboard microphone and 6-axis IMU, suitable for the TinyML AI+IoT work, making it an ideal option for TinyML AI system [1], see the hardware overview in figure 3.9 from [49]:

3.4.2.1 Pedometer Experiment

A demonstration of how to collect the data from the sensor and use the potential of the libraries. In this case, to show the number of steps walked through time, just by using a simple count. A pedometer, or step-counter, is a device, usually portable, that counts each step a person takes by detecting the motion of the person's hands or hips.

- Onboard digital microphone for real-time audio recognition
- 6-axis IMU for gesture recognition
- Bluetooth communication

Material used:

1. Seeed Studio XIAO nRF52840 Sense
2. Arduino IDE

Figure 3.8 shows the result obtained in the Arduino IDE, by using the built in property *pedometer.readregister()*.

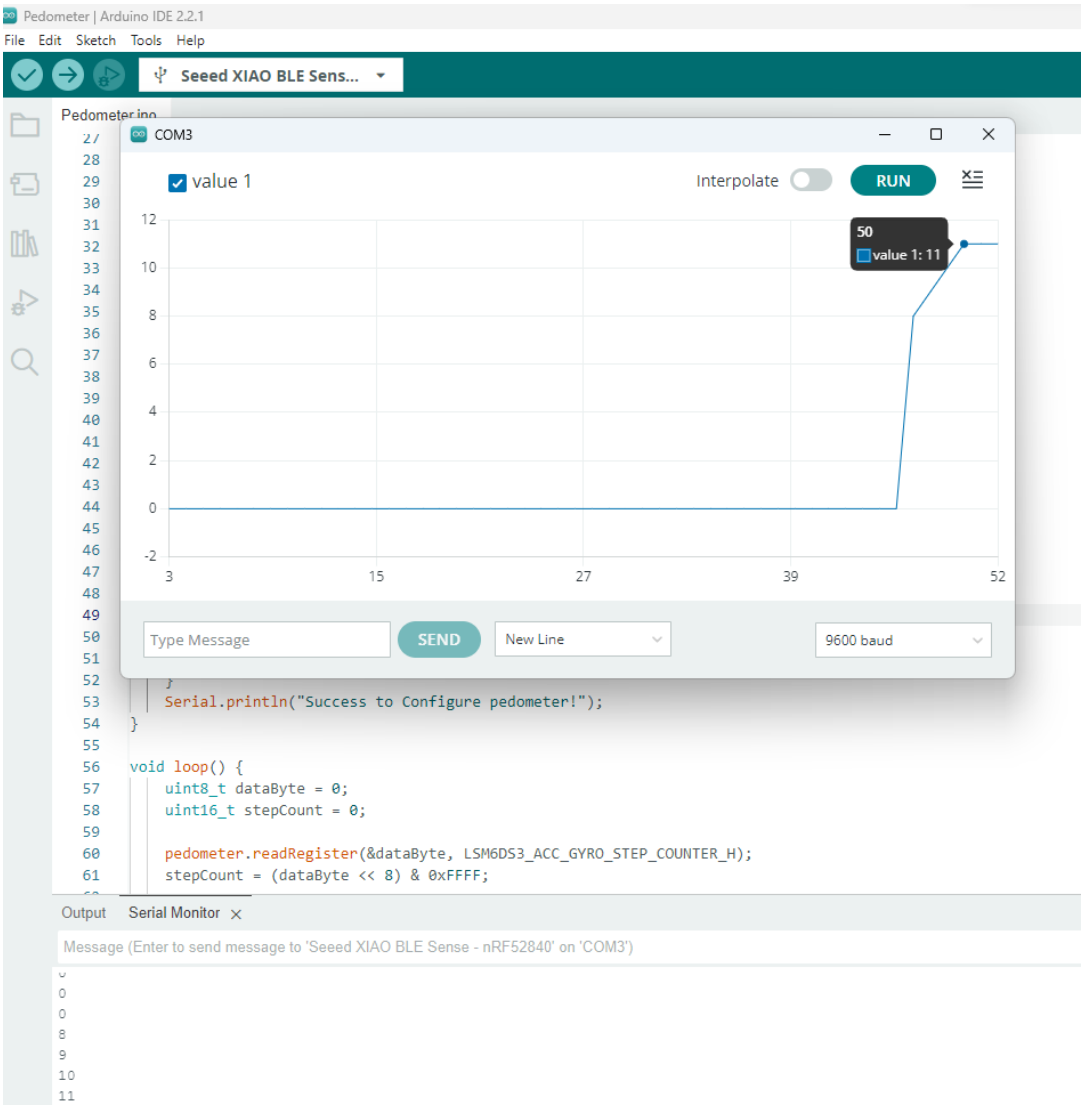


Figure 3.8: Serial plotter with number of steps: 11 in this case.

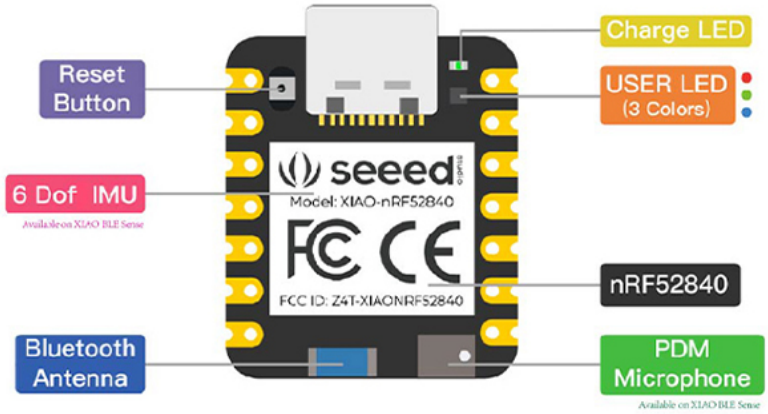


Figure 3.9: XIAO nRF52840 Sense: Hardware overview.

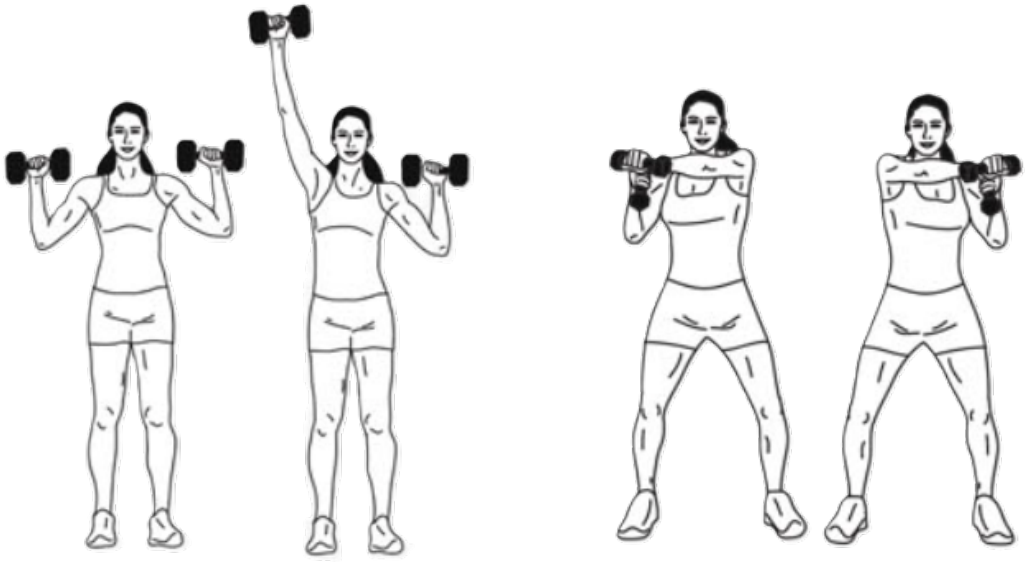


Figure 3.10: Exercises: flex (left) and punch (right) from [2].

This MCU was chosen due to multiple features: small, not heavy, works with a rechargeable battery and not expensive (low cost).

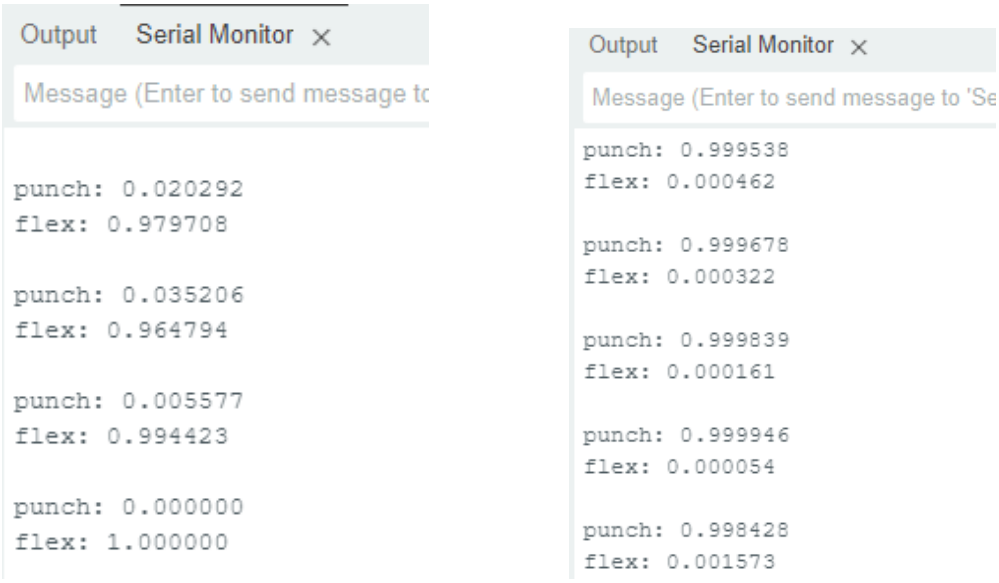
3.4.2.2 Evaluation of TensorFlow Lite Using the Arduino Platform

In this experience, it was used the TensorFlow Lite library on Seeed Studio XIAO nRF52840 Sense. The goal was to detect gestures such as punching and flexing, shown in the figure 3.10, using the on-board accelerometer. The code used can be accessed at [42] and [49]. This code should be opened and uploaded with Arduino IDE.

Components used:

1. Seeed Studio XIAO nRF52840 Sense

2. Arduino IDE



(a) Flex result.

(b) Punch result.

Figure 3.11: Results of the IMU classifier with TensorFlow Lite.

3. Pre-built model (dataset)

4. TensorFlow Lite Library

In the serial monitor, the following results could be observed, see figure 3.11, by monitoring and performing a flexing action, it can be seen that will give a result close to 1 next to flex and by monitoring and performing a punching action, it can be seen that will give a result close to 1 next to punch.

3.4.2.3 Evaluation of TensorFlow Lite Using the Edge Impulse Platform

In this example, it was used the TensorFlow Lite library on Seeed Studio XIAO nRF52840 Sense. The goal was still to detect gestures such as punching and flexing, but now by using a trained model that could also detect if a movement fits in the pattern.

- **Data Collection:** Collect data directly to Edge Impulse.

- **Train the model:** Create and train the model in Edge Impulse.

  Spectral Analysis *- Great for analyzing repetitive motion, such as data from accelerometers. Extracts the frequency and power characteristics of a signal over time.*
  Classification *- Learns patterns from data, and can apply these to new data. Great for categorizing movement or recognizing audio.*

- **Test Results:** Upload the library, TensorFlow Lite with the model to the MCU.

In this case, the following material was used:

1. Seeed Studio XIAO nRF52840 Sense
2. Edge impulse platform (uses tensor flow lite)
3. Arduino IDE

Let's describe the experiment step by step:

Step 1 - Download the code from [42]

Step 2 - Import the code into Arduino IDE, Open Arduino IDE, navigate to Sketch - Include Library - Add .ZIP Library

Step 3 - Navigate to File - Examples - Seeed Arduino LSM6DS3 - IMUCapture to open IMUCapture.ino. This code is going to read the acceleration and gyroscope data from the axis (x,y,z)

Step 4 - Upload the codes, connect Seeed Studio to the computer and open the Serial Monitor

Step 5 - Run the command edge-impulse-data-forwarder, this way data collect will be sent directly to Edge Impulse

Step 6 - On Edge Impulse create a project and select the sensor as "3 axes". Name your label as flex, modify Sample length (ms.) to 20000 and click start sampling. This step needs to be repeated for "punch" also.

Step 7 - With Seeed Studio XIAO nRF52840 Sense in your hand simulate a flex motion for 20 seconds. The raw acquisition data is shown up in 3.12

Step 8 - Split the data by clicking the raw data right top and choose "Split Sample"

Step 9 - Repeat the last 2 steps for punch, label data with different names to click different motion data

Step 10 - Choose Classification (Keras)

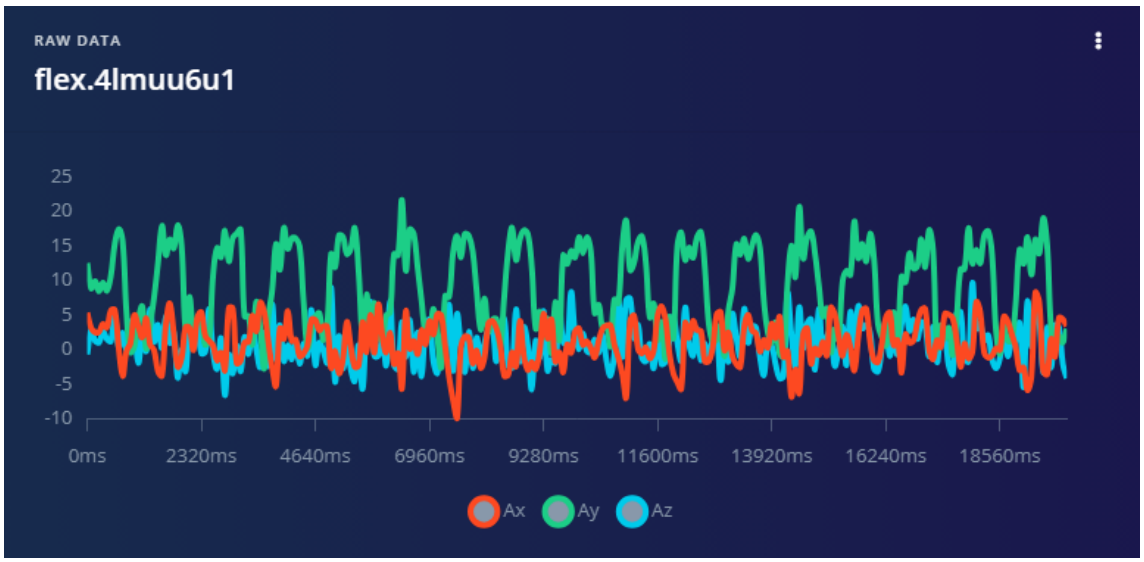


Figure 3.12: IMU classifier with TensorFlow Lite using Edge Impulse: Raw data graph from Edge Impulse Platform.

Step 11 - Choose the Spectral features and generated features

Step 12 - Training your model (Option in Edge Impulse to train model) see figure 3.13

Step 13 - From Edge Impulse download the dataset (zip format)

Step 14 - In Arduino IDE Select the file as "Add the ZIP file" to your Arduino libraries, get the code here [50] and upload it

Step 15 - Punch or flex with Seeed Studio XIAO nRF52840 Sense and check the results

When you move the Seeed Studio XIAO nRF52840 Sense in the left and right direction, the monitor will output something like 3.14 or 3.15.

3.4.3 CJMCU Beetle

The CJMCU Beetle was another MCU option considered for this system. This subsection compares its features with those of other devices, discussing its strengths and limitations. The CJMCU-Beetle USB, based on the ATmega32U4 MCU, does not have built-in Bluetooth capabilities, therefor was not tested, see figure 3.16 number 3.

Figure 3.13: IMU classifier with TensorFlow Lite using Edge Impulse: Model from Edge Impulse Platform.

```
Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 24 ms., Classification: 0 ms., Anomaly: 0 ms.):
    flex: 0.99609
    punch: 0.00391
```

Figure 3.14: MU classifier with TensorFlow Lite using Edge Impulse: Flex result from Serial Monitor in Arduino IDE.

```
Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 24 ms., Classification: 0 ms., Anomaly: 0 ms.):
    flex: 0.01562
    punch: 0.98438
```

Figure 3.15: MU classifier with TensorFlow Lite using Edge Impulse: Punch result from Serial Monitor in Arduino IDE.

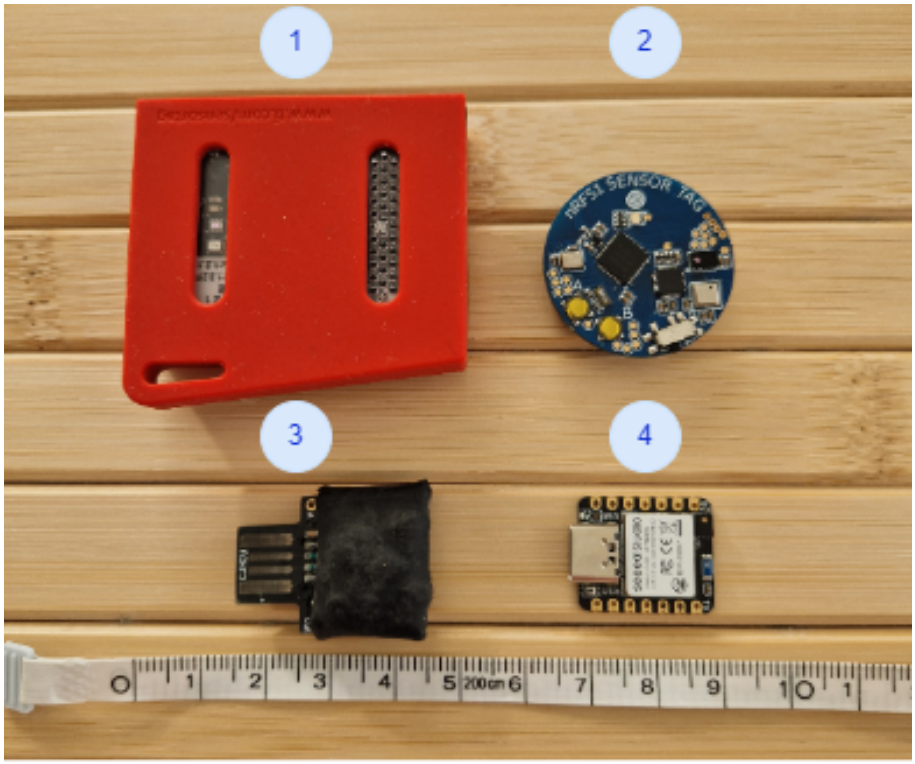


Figure 3.16: The four tested MCUs.

3.4.4 Texas Instruments SENSORTAG

This MCU Texas Instruments TIDC-CC2650STK-SENSORTAG is a versatile development kit that includes multiple sensors and wireless communication options. This subsection evaluates its performance and applicability to HAR tasks. Excluded due to the size, price, and the learning curve of the software used to program, no extra tests were made with this MCU, figure 3.16 number 1.

3.4.5 Embedded System Comparison

Figure 3.16 shows the embedded systems that were tested. The table 3.1 presents the characteristics of the different embedded systems considered and evaluated in this thesis.

Table 3.1: Embedded system comparison.

Feature	NRF51 Sensor Tag	Seeed Studio XIAO nRF52840 Sense	CJMCU Beetle	Texas Instruments TIDC-CC2650STK-SENSORTAG
Price	€15.36	€22.20	€8.73	€59
MCU	nRF51822	nRF52840	AT-mega32U4	CC2640R2F
CPU	32-bit industry standard ARM Cortex-M4F processor	Nordic nRF52840 ARM® Cortex®-M4 with FPU runs up to 64 MHz	Atmel AVR 8-bit MCU family	CC2650
On-chip Memory	192 kB flash and 24 kB RAM	1 MB flash and 256 kB RAM	32 KB flash	20 KB of SRAM
Onboard Memory	6 KB to 256 KB or more	2 MB QSPI flash	SRAM: 2.5 KB	128 KB of Flash memory
Interface	UART (RTS/CTS)	1 x UART, 1 x IIC, 1 x SPI, 1 x NFC, 1 x SWD, 11 x GPIO(PWM), 6xADC	UART, I2C, and SPI Interfaces	1 x UART, 1 x SPI, 1 x I2C, GPIO, ADC
Dimensions	Diameter 25mm	21 x 17.5mm	30x20mm	55x50mm
Power	Battery (non rechargeable) - CR2032	Lithium Polymer battery Standby power consumption: ¡5A	Battery (non rechargeable)	Battery (non rechargeable)
Wireless Communication	BLE	BLE	Does not include a wireless communication module such as Bluetooth or Wi-Fi.	BLE
Bluetooth Version	BT 4.1	BLE 5.0	NA	BT 4.2
USB Port	No	Included	No	No
Development Platform	Nordic Semiconductor SDK	Arduino IDE	Arduino IDE	Texas Instruments Code Composer Studio
Portable	Yes	Yes	Yes	Yes
Sensors Included	Yes	Yes	Yes	Yes
Sensors	temperature, humidity, accelerometer, gyroscope, magnetometer	6 IMUs temperature, humidity, accelerometer, gyroscope	accelerometer, gyroscope, magnetometer, pressure	Supports 10 Low-Power Sensors: temperature, humidity, accelerometer, gyroscope, magnetometer, pressure, etc
Led included	Yes	Yes	Yes	Yes
Arduino library	No	Yes	Yes	No
Advantages	Low cost 3 axis Accelerometer, Sensor RF 2.4 GHz BLE 5.0, Low Power Consumption Bluetooth PCBA	Low cost, Low Power, BLE 5, Great documentation, not expensive, easy to start use and program, IMU with extra capabilities - like pedometer, etc	Low-power and extended-range capabilities	Advanced debugging and profiling tools
Disadvantages	Excluded due to the lack of examples and public documentation., Low Power Consumption Bluetooth PCBA	Pay more for the connectivity	Doesn't have built-in Bluetooth capabilities	Excluded due to the size, price and the learning curve of the software used to Develop

4

Proposed Embedded System for Real-Time HAR Architecture

This chapter details the architecture of the Embedded HAR Autonomous System. It covers the system architecture, the integration of sensors, and the deployment of the NN model for real-time inference in the MCU, using TinyML techniques, represented in figure 4.1.

4.1 System Architecture Overview

The implemented embedded system for real-time movement recognition involves multiple components such as: MCU, an OLED display, and a Battery, illustrated in 4.2.

The MCU chosen is the Seeed Studio XIAO nRF52840 Sense, which features a Nordic nRF52840 CPU, 1 MB flash and 256 KB RAM on-chip memory. This compact MCU measures only 21 x 17.5mm and weighs just 4 grams. It also supports Bluetooth 5.0, which is important for the communication with external devices. The embedded sensor LSM6DS3 IMU (LSM6DS3) for its accelerometer and gyroscope functions, allows real-time tracking of movement data, making it ideal for HAR. Additionally, this MCU is Arduino-compatible, which simplifies the development and deployment process. The architecture of the system is designed to ensure low-power consumption, autonomy, and compact design. Each component was selected to maintain a balance between performance and energy efficiency, given the limitations of running on a small LiPo battery and a small MCU. The compact size of the MCU and the battery, coupled with an OLED display, enables this system to

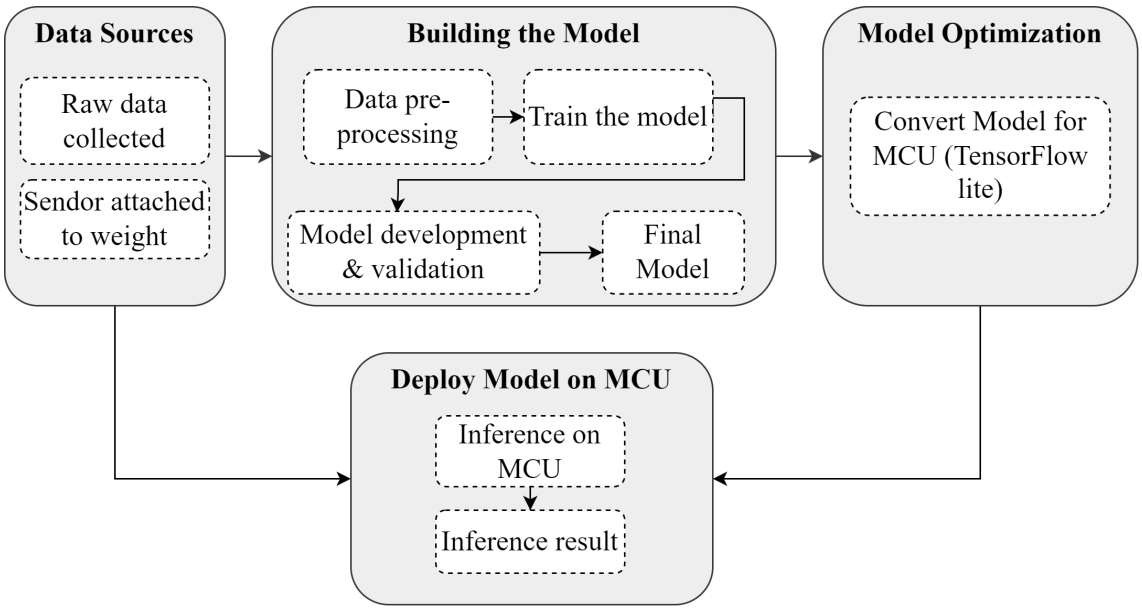


Figure 4.1: System architecture flow.

be portable, compact and lightweight, facilitating ease of use during exercise and to be attached to gym equipment with a magnet. The BLE connectivity allowed the real-time data acquisition and sending of the results to an external system, providing real-time feedback to the user. During the design of the architecture, latency was minimized, ensuring that the user received feedback within milliseconds of performing an action. This responsiveness is achieved through careful memory management on the MCU and reduces the payload of the exchanged messages. Moreover, the system architecture is designed with modularity in mind, see figure 4.2. As new features or sensors are added, the MCU can be reprogrammed to accommodate these changes, making the platform versatile for future upgrades.

To ensure autonomy, a battery is incorporated into the system. A 302030 lithium-polymer (LiPo) battery with a capacity of 120mAh and 3.7V voltage was chosen due to its small size, similar to the MCU. This battery connects directly to the power pins of the XIAO MCU, this way the system can function without depending on another external power source.

Displaying the results of the real-time inference to the user is an important part of the system. A 0.66" OLED display with a resolution of 64x48 was selected. It is small, but perfect in size due to the similarity with the MCU size. The OLED display, like the battery, is powered directly by the Seeed Studio XIAO nRF52840 Sense.

To show the result information to the user the size of the graphical display was also an important feature to be considered, the chosen OLED measures 0.66" with a screen resolution 64x48, check [51]. The battery and OLED display are powered by the Seeed Studio XIAO nRF52840 Sense. Additionally, a dedicated mobile application was developed to show the user the number of exercises performed

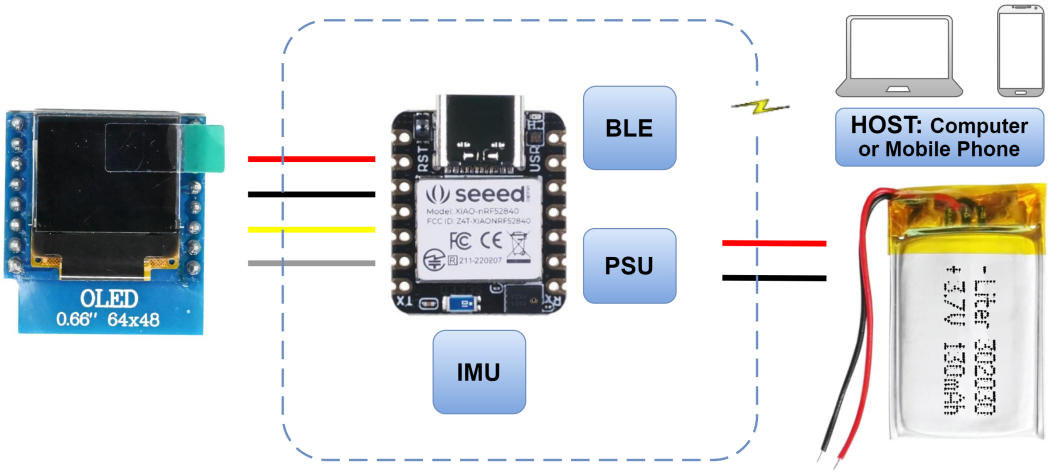


Figure 4.2: System architecture overview: MCU with IMU (6-axis), Power Supply Unit (PSU) and BLE. The OLED Display and Battery outside of the Seeed Studio XIAO nRF52840 Sense.

and if they are correctly or incorrectly done.

The data flow of the system starts at the MCU, where the IMU continually reads data from the accelerometer and the gyroscope. The BLE communication module on the MCU supports efficient, low-latency data transfers, ensuring minimal delay when transmitting sensor readings or inference results. These raw signals, from the accelerometer and gyroscope, are sent from the MCU's by BLE to the host. To facilitate the real-time inference that needs accurate data, the MCU implements a fixed sampling rate for the sensors, ensuring that all movements are recorded with sufficient temporal resolution. For this application, a sampling rate of 50 Hz is chosen, balancing the need for high data resolution and power efficiency.

Once data is captured, the MCU performs transformation to the sensor signals, in this case, the accelerometer readings are converted to m/s2.

Communication between the system components is handled using both physical and wireless protocols. The battery and OLED display are physically connected to the MCU, while BLE is used for wireless data transmission to the host.

4.2 Data Acquisition System

For the goal of this work, the effort that video processing requires was not acceptable, especially for consistent long-term monitoring of the performed exercises. A valid alternative is collecting the movement data with sensors, IMU is the key to achieving this goal. Data was acquired using 6-axis sensors: accelerometer (X,Y,Z) and gyroscope (X,Y,Z).

Data collection for HAR involves multiple stages. Initially, raw sensor data is collected continuously while the subject performs various exercises. These exercises are predefined to represent a wide range of gym exercises. Each exercise is labelled

in real-time, creating a dataset for the NN training purposes.

The system is configured to record from sensor readings, that will be separated in time windows. Each time window is 3 seconds long and contains hundreds of individual sensor measurements. This sliding window approach helps capture the dynamic nature of human movement while minimizing the volume of data required for accurate analysis.

Once the raw sensor data is collected, it is transmitted wirelessly to a computer - the host - via BLE, where more detailed preprocessing occurs. Data preprocessing included removing long samples where the subject was stopped and normalization, and scaling samples to similar ranges.

After collecting the data, the next step was to train the NN, a necessary step in HAR. Later in the next chapters, it's explained in more detail how the NN was training using different platforms and techniques, including Edge Impulse, Google Colab, and Training models in TensorFlow using Python, highlighting the advantages and disadvantages of each. The output of this step was a generated Model to be used in the embedded system, by installing it directly on MCU.

The approach followed was a bottom-up methodology, starting with the platform selection, leveraging the accelerometer for precise data capture, conducting trials for diverse datasets, and implementing the NN within the MCU to achieve accurate real-time movement analysis with low power consumption [44].

4.3 Real-Time Classification System

A central goal of the system lies in its ability to perform real-time inference on the embedded MCU. The system collects continuous movement data through the IMUs accelerometer and gyroscope. The MCU, equipped with a NN implementation [52], using TinyML techniques, analyses the movement data on-the-fly, categorizing movements as well or poorly executed. This information will be displayed using a LED and an OLED display and in a dedicated mobile application, depending on the equipment being used and the performed movement. All these steps are illustrated in figure 4.3.

Real-time inference on the MCU is enabled by deploying a TFLite model optimised specifically for the hardware's limited computational power and memory. After the MCU collects sensor data, the raw readings are transformed into vectors in real-time. These vectors serve as inputs to the pre-trained neural network, which resides in the MCUs flash memory.

Each inference cycle is executed in a matter of milliseconds, ensuring that movement recognition and feedback are real-time as much as possible. The NN model itself is quantized, meaning its weights are converted to 8-bit integers. This process drastically reduces memory usage and computational load without signifi-

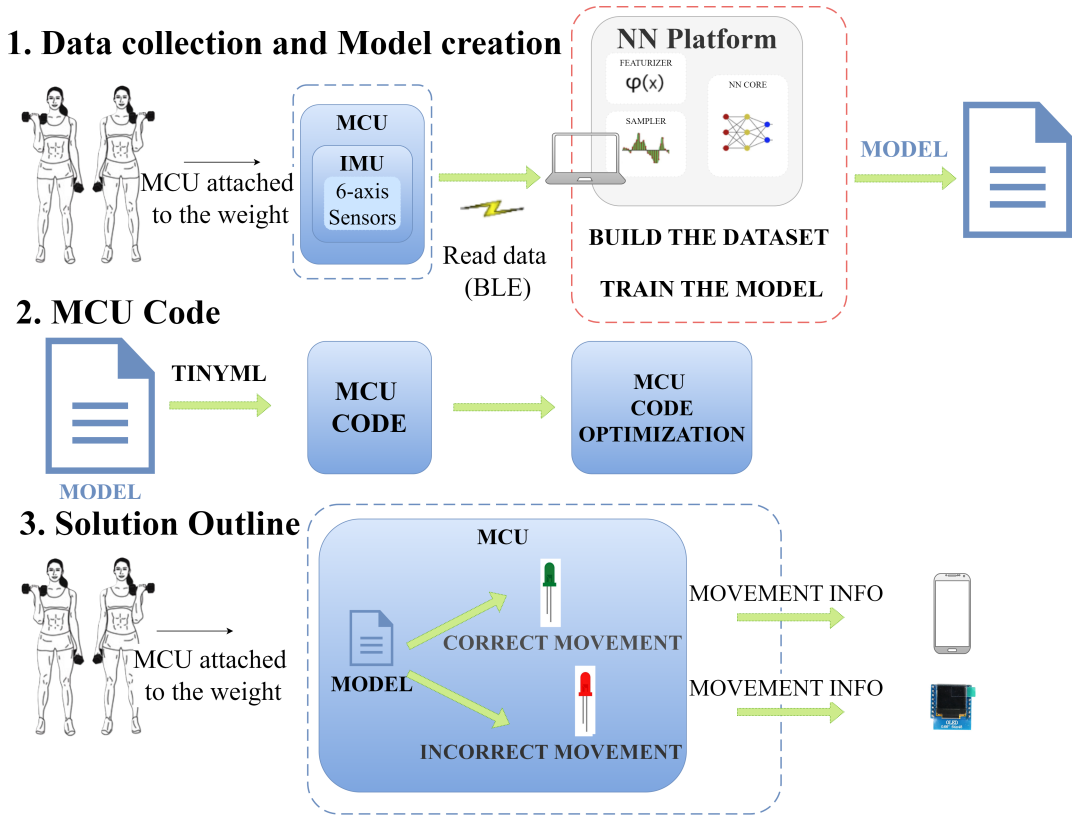


Figure 4.3: Solution outline.

cant loss of accuracy, making it feasible to run complex models on small, resource-constrained devices.

Real-time inference is a key point of HAR systems due to the constraints of embedded devices. In this work the inference process runs on the MCU, and some transformations and optimizations are made to achieve low latency and high accuracy, and also to convert the model to the TensorFlow Lite format. The inference is explained in more detail later on the this thesis.

5

Dataset Construction, Preparation and Curation

This chapter describes the process of construction and preparation of the dataset and processing or organizing the data for use, such as cleaning it. Also highlighting the process of creating and managing the dataset used to train the NN model. It covers the selection of exercises, data acquisition, and the techniques used to prepare and curate the raw data into a dataset for effective training.

5.1 Data Collection

The goal of this thesis is to identify the correctness of a specific set of exercises and during the research process, no existing dataset was found that could be directly used for this purpose. As a result, there was a need to create our own dataset tailored to the unique requirements of this study. This custom dataset will allow us to accurately evaluate and analyse the performance and correctness of exercises.

The dataset was collected in 4h sessions with a total of 8 subjects and includes data from various gym exercises, in total 3 different exercises were performed correctly and incorrectly 10 times each for all the subjects: Shoulder Press, Bicep curl, and Tricep extension, illustrated in figure 5.1 from left to right. The shoulder press is an exercise that involves upward movement of the arms, illustrated in figure 5.1. The bicep curl is an exercise that involves the bending of the arm at the elbow, illustrated in figure 5.1. Tricep extensions involve the extension of the arm at the elbow, illustrated in figure 5.1.

A group of eight participants, age range 19–57 years, were recruited to collect

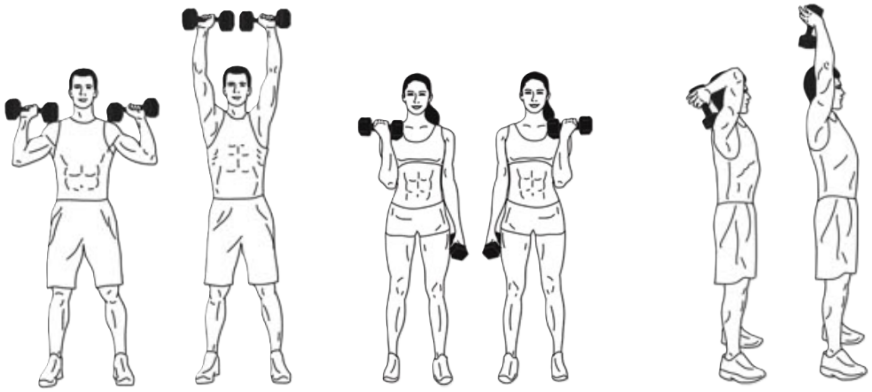


Figure 5.1: Should Press (left), Bicep Curl exercise (centre), Tricep Extensions exercise (right).

Figure 5.2: Session of data collection at ISEL - performing one of the exercises.

Figure 5.3: Session of data collection at ISEL - looking into the raw data patterns.

Table 5.1: Summary of session used to acquire the raw data.

Total duration	4h
Subjects (female/male)	8 subjects: 6 male, 2 female
Range of ages	19 to 57 years
Skill level	Beginner to professional
Sample rate	50Hz
Sensor Placement	On the Dumbbell
Reference	Manually labeled raw data
Local	ISEL F Building (DEETC)
Accelerometer Resolution	The accelerometer in the LSM6DS3 outputs data with a 16-bit resolution for each of the X, Y, and Z axes.
Acceleration (m/s^2)	The constant 9.80665f (standard acceleration due to gravity on Earth) is used to convert the raw accelerometer data from units of g (where 1g +-= 9.81 m/s^2) to m/s^2.

this dataset in a controlled environment at ISEL, building F DEETC. They were asked to perform 3 gym exercises: shoulder press, bicep curl, and tricep extensions. A MCU with sensor XIAO nRF52840, see figure 3.9, placed on the weight using an iman was used to capture the activities at a sampling frequency of 50 Hz using the built-in tri-axial accelerometers and tri-axial gyroscopes, see table 5.1. The sensor attachment is shown in figures: 5.2 and 5.3. In this study, the focus was only on the data acquired from accelerometers and gyroscopes. The accelerometer recorded in g's, where 1g is equivalent to the acceleration due to Earth's gravity (9.81 m/s^2), and gyroscope data recorded the angular speed in dps (degrees per second). To measure acceleration, the tri-axial accelerometer is used and data has x-axis, y-axis, and z-axis. Tri-axial accelerometers provide a low-power and high-fidelity measurement of force along the x, y, and z directions, providing a view into the movement of the person wearing the device, the same technique used in [53]. They capture the user's movements. Here X-axis indicates sideways or horizontal movement of the user, the Y-axis indicates upward or downward movement and the Z-axis indicates forward or backward movement of the user. The data generated from the accelerometer is time series data and it was called by AX, AY, and AZ along X, Y, and Z-axis respectively. The collected data is illustrated in figures 5.4, 5.5, 5.6, 5.7, 5.8 and 5.9.

Most activity classification methods use windowing techniques to divide the sensor signal into smaller time segments (windows), illustrated in figure 5.10, edge impulse example in figure 5.11. With the sliding window method, the signal is divided into windows of fixed length [54].

According to [53], mid-sized time windows (from 5 to 7 s long) perform best from a range of windows from 1 to 15 s for wrist-placed accelerometers. The results are slightly different for other accelerometer placements, but the trend of mid-size windows performing best remains a truth.

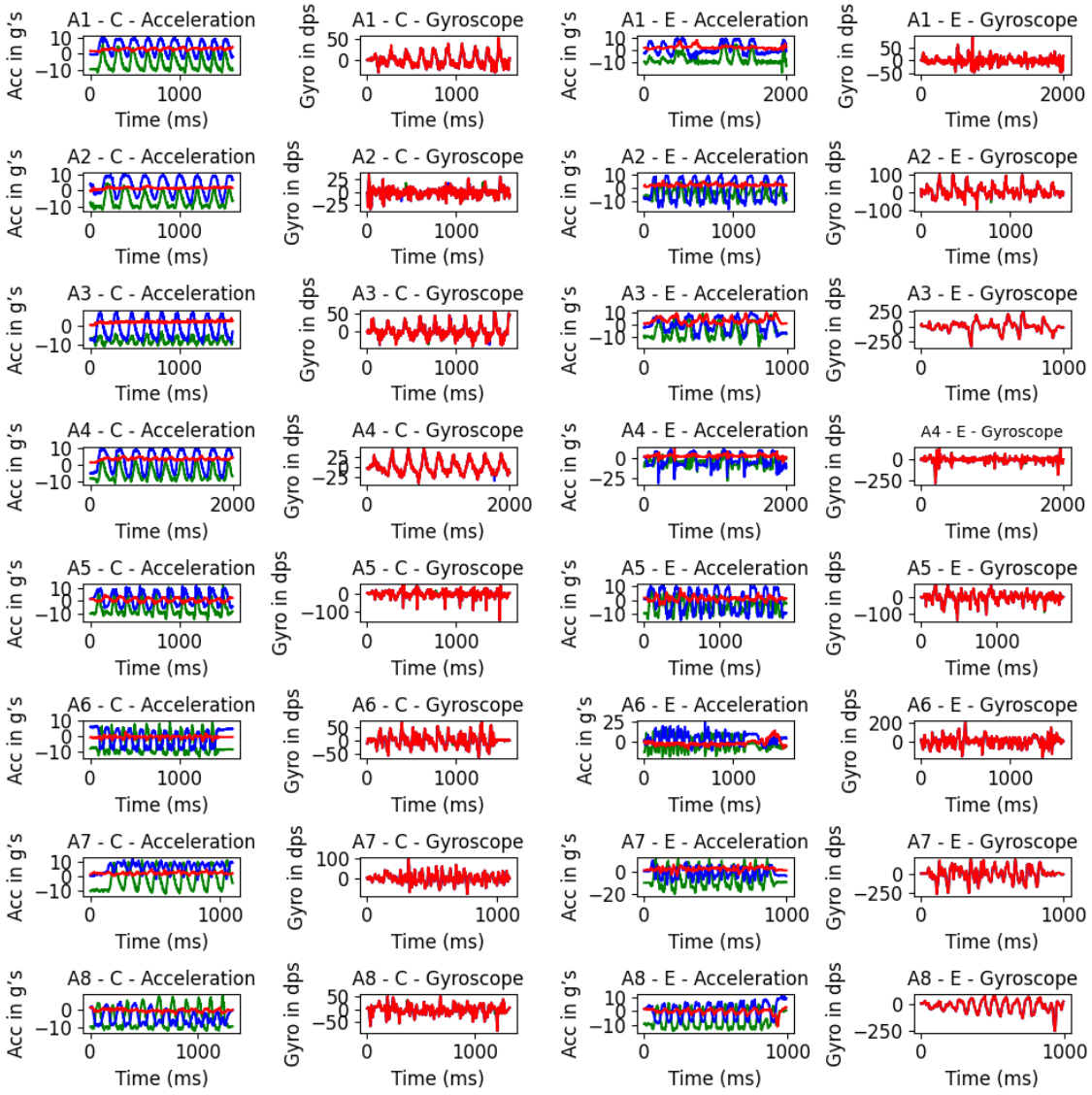


Figure 5.4: Signal acquired on the Bicep Curl exercise by the accelerometer (title Acceleration) and the gyroscope (title Gyroscope) from the 8 subjects during 10 repetitions - correct movement 2 graphs on the left and incorrect movement 2 graphs on the left (X values in blue, Y in green and Z in red. The bottom axis is represented in time(ms)).

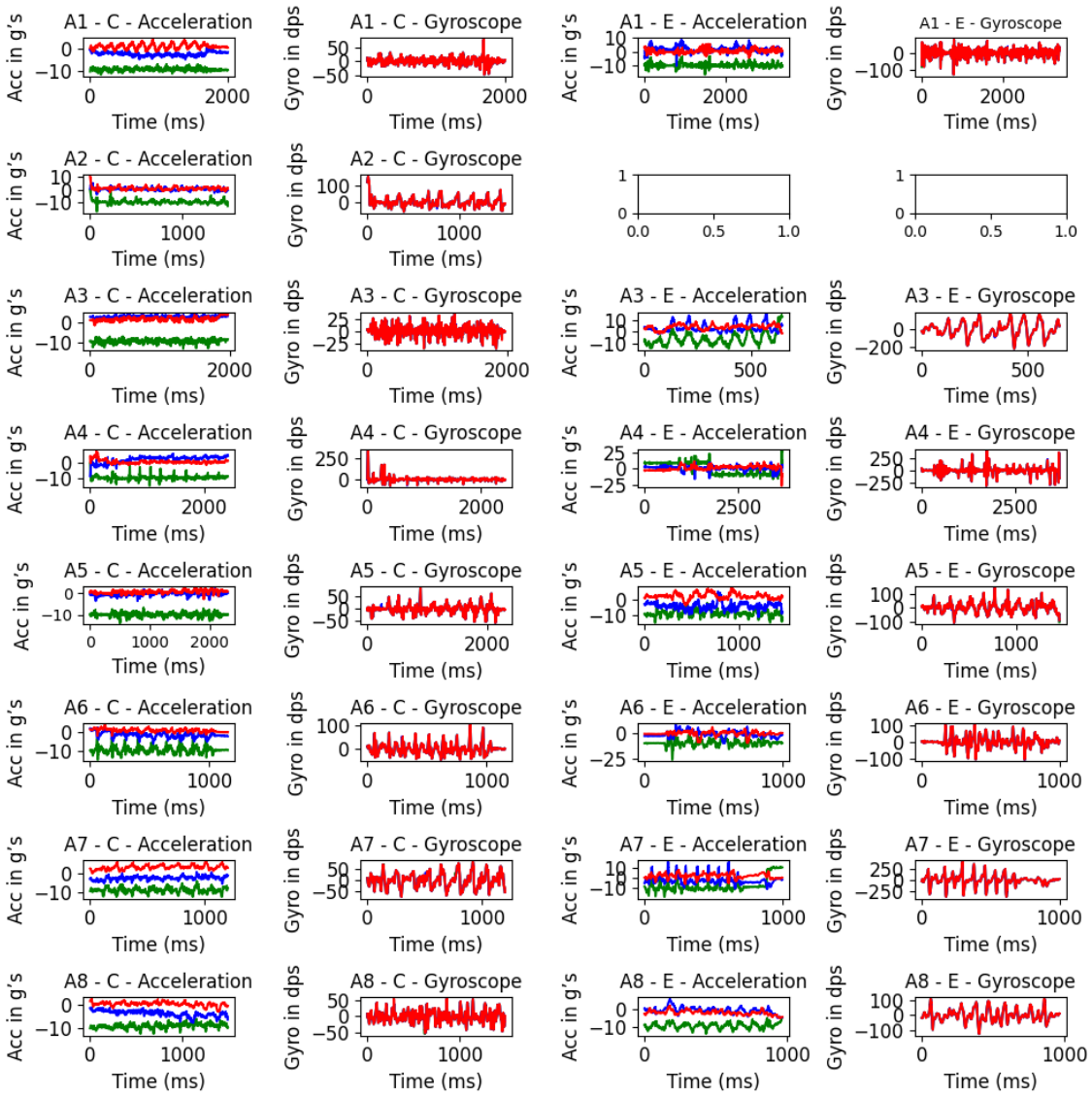


Figure 5.5: Signal acquired on the Shoulder Press exercise by the accelerometer (title Acceleration) and the gyroscope (title Gyroscope) from the 8 subjects during 10 repetitions - correct movement 2 graphs on the left and incorrect movement 2 graphs on the left (Empty graphs are lost samples)(X values in blue, Y in green and Z in red. The bottom axis is represented in time(ms)).

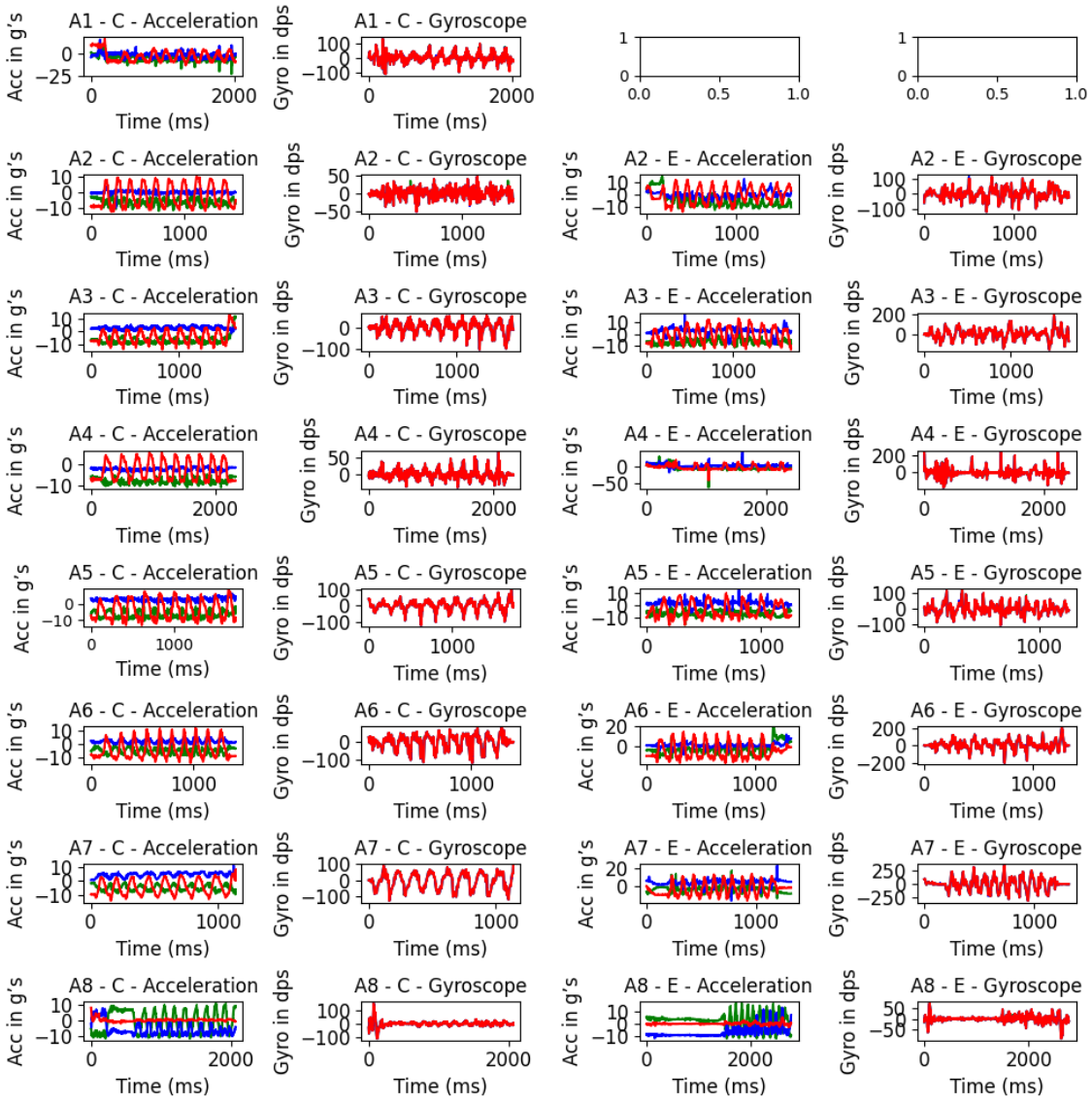


Figure 5.6: Signal acquired on the Tricep Extension exercise by the accelerometer (title Acceleration) and the gyroscope (title Gyroscope) from the 8 subjects during 10 repetitions - correct movement 2 graphs on the left and incorrect movement 2 graphs on the left (Empty graphs are lost samples)(X values in blue, Y in green and Z in red. The bottom axis is represented in time(ms)).

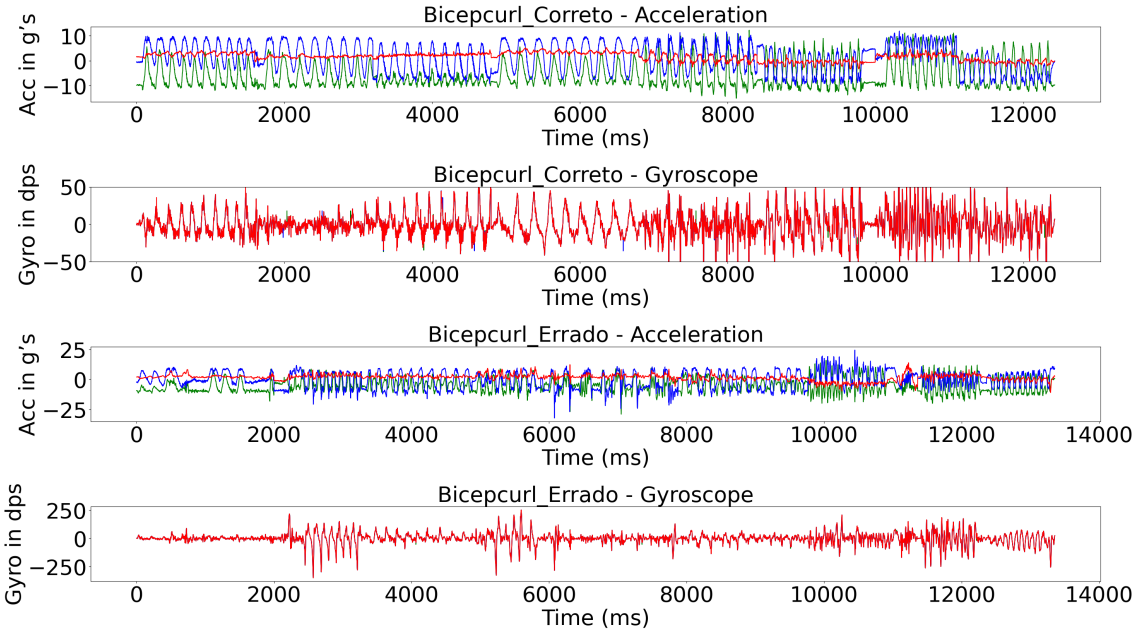


Figure 5.7: Signal acquired of bicep curl for all the subjects by the accelerometer (title Acceleration) and the gyroscope (title Gyroscope) from the 8 subjects during 10 repetitions (X values in blue, Y in green and Z in red. The bottom axis is represented in time(ms)).

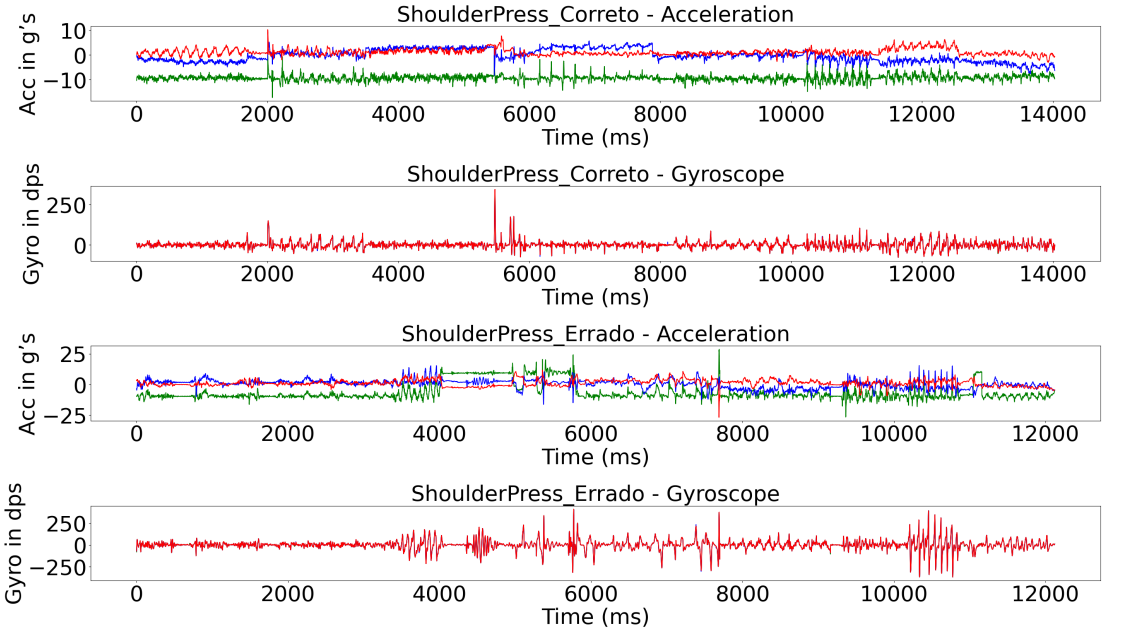


Figure 5.8: Signal acquired of shoulder press for all the subjects by the accelerometer (title Acceleration) and the gyroscope (title Gyroscope) from the 8 subjects during 10 repetitions (X values in blue, Y in green and Z in red. The bottom axis is represented in time(ms)).

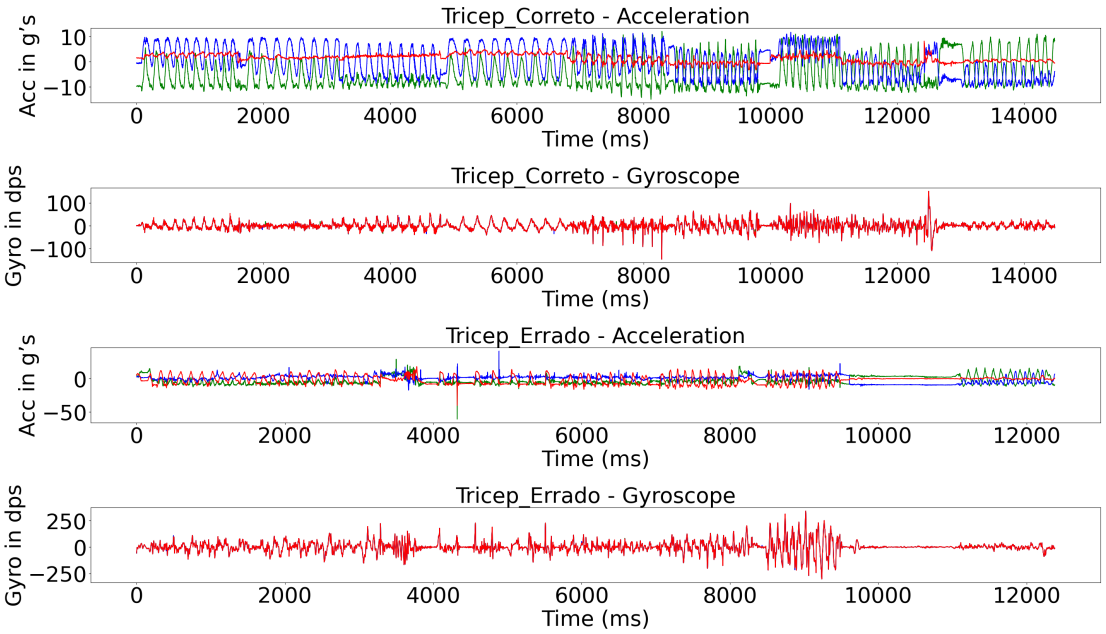


Figure 5.9: Signal acquired of tricep extension for all the subjects by the accelerometer (title Acceleration) and the gyroscope (title Gyroscope) from the 8 subjects during 10 repetitions (X values in blue, Y in green and Z in red. The bottom axis is represented in time(ms)).

The features extraction step involves calculating attributes that are able to capture the patterns over a sliding window, the features were extracted directly from the accelerometer raw data.

According to the study [55], after testing multiple window sizes, they reached a conclusion that in order to maintain a good trade-off between classification performance and resource utilization a window of intermediate size (i.e., 3 s) has been proven to be the best, most of the times 3s were used.

The window size was made in an adaptive and empirical manner, to produce good segmentation for all the activities under consideration. After this investigation, in this thesis it was used a 3s sliding window, with short or no overlap.

5.2 Dataset Analysis and Curation

The first step in data preprocessing is to understand the data that was collected. Just looking at the dataset can give an intuition of what things needed to focus on. Applying NN training on noisy data would not give quality results as they would fail to identify patterns effectively. Data Processing is, therefore, important to improve the overall data quality. Some of the examples are duplicates or missing values that may give an incorrect view of the overall statistics of data, or inconsistent data points that often tend to disturb the model's overall learning, leading to false predictions. With this in mind, the next step was to clean the data. Data Cleaning is done as part of data preprocessing [56] to clean the data by filling in

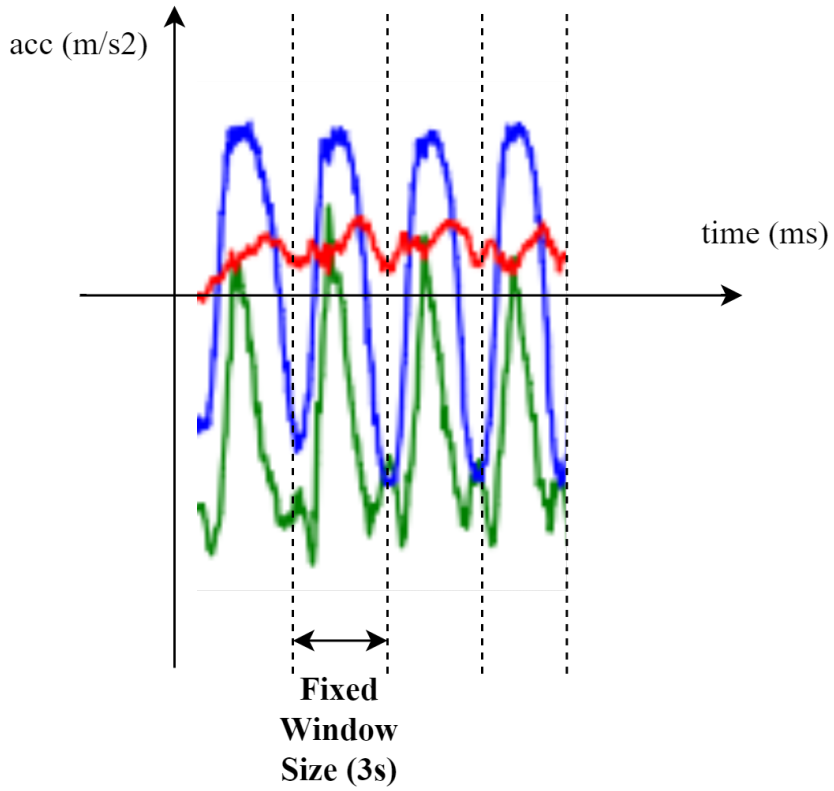


Figure 5.10: Sliding window representation.

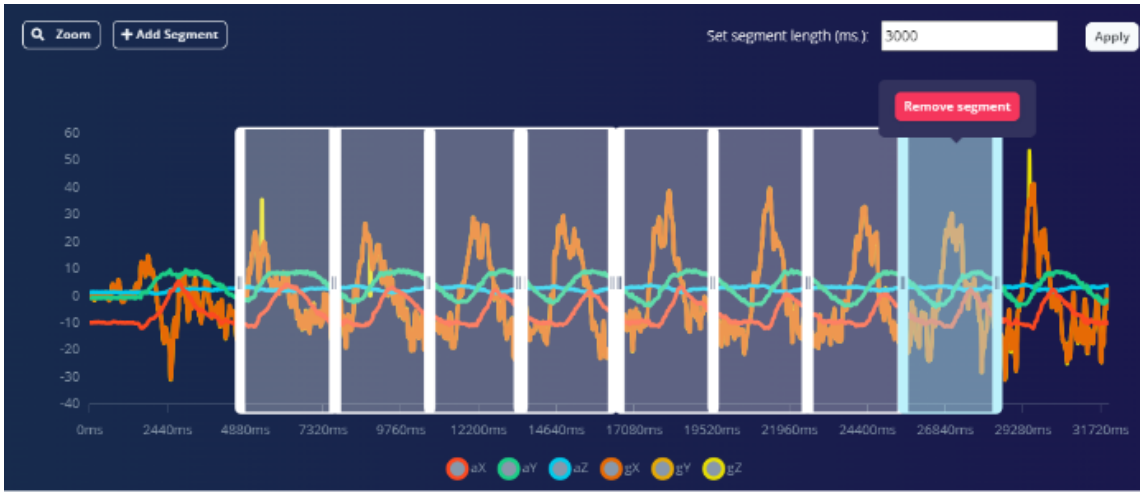


Figure 5.11: Sliding window from Edge Impulse.

missing values, smoothing the noisy data, resolving the inconsistency, and removing outliers. Next, it's described the applied techniques to the raw data acquired.

Involves removing outliers in a measured variable, and also removing the beginning and end of the collected samples, because those were times the participant was at rest. All this information was removed manually.

Normalization was a very important data transformation technique used in this thesis. The numerical attributes are scaled up or down to fit within a specified range. In this approach, the data was constrained to a particular set of values. Normalize the input data, in this case, acceleration values were normalized to: -4 to +4 and the gyroscope is between: -2000 to +2000. This technique allows an immediate perception of the patterns in the data, making it easier to analyse and clean. For the training algorithm to work more effectively the values need to be similar in size.

5.3 Features Extraction

Features are the values that are fed into the NN, the arrays of numbers pass in as inputs. Feature extraction is a step in preparing the data for ML. This section explains the methods used to extract meaningful features from the raw sensor data, such as statistical measures, frequency domain analysis, and others that improve the model's ability to recognize activities. Besides the progress that many studies made [38] [15]. This is caused by the broad range of human activities as well as the big variation in how a given activity can be performed. It is necessary to use features that separate between activities. Feature extraction is one of the key steps in HAR, since it can capture relevant information to differentiate among various activities. The accuracy of HAR approaches greatly depends on features extracted from raw signals such as accelerometer readings. Many existing HAR approaches often rely on statistical features such as mean, variance, entropy, or correlation coefficients. The performance of the activity recognition system is highly dependent on the features extracted from the sensor data which makes the selection of appropriate features a very important part of it.

6

Training of the Neural Network Model

In this chapter and the next it is described how the model was trained, how the performance was evaluated, and how to convert it to run on-device. Training is the process by which a model learns to produce the correct output for a given set of inputs. It involves making adjustments until the prediction accuracy is improved. It also describes the multiple techniques and platforms used to train the NN, having a Model as a result.

6.1 Training Methodology

In wearable sensor–based methods, devices containing inertial sensor units, such as accelerometers and gyroscopes, are attached to the body. Activities are then classified into types, with each activity type showing a different pattern of sensor values. To classify an activity, the features that can best represent it must be extracted from the collected sensory data. Selecting and extracting the most meaningful features are the most significant problems for achieving accurate HAR. Usually, the features are chosen by humans, called hand-crafted features, and include time-domain features such as mean and standard deviation and frequency-domain features. To find the most efficient and effective features, (1) the programmers must have prior expert knowledge, or (2) they must do a large amount of empirical study to learn which features are useful. ML technology has now been developed and can be used for HAR. The most prominent advantage of using deep learning is that it can automatically extract both low- and high-level features and select them without

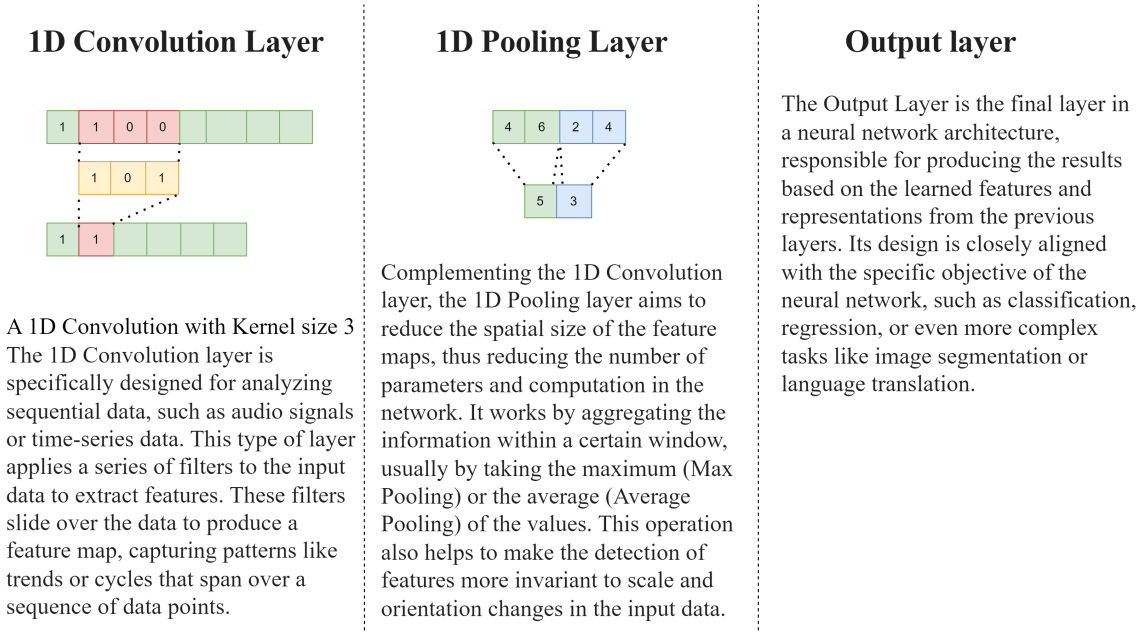


Figure 6.1: CNN 1D Layers.

user manipulation. In terms of sensor signals, low-level features include statistical features and frequency-domain features. Likewise, for images, low-level features are external representations of objects such as lines or colour, and high-level features are semantic. In both cases, high-level features are understandable to humans. In the domain of HAR, low-level features can be signal lines, inflection points, min/max values, or variations, and high-level features are patterns in the signals.

In this system, it was used a three-layer stacked convolution and pooling for extracting features from the raw wearable sensor data. Keras NN [57] is a type of deep learning model, which is integrated into TensorFlow. Keras simplifies the creation and training of NNs, in summary, it can be described as a simple CNN. Keras is the high-level API of the TensorFlow platform. It provides an approachable, highly-productive interface for solving ML problems. Keras covers every step of the ML workflow, from data processing to feature tuning to deployment. Edge Impulse, is based in Keras NN and uses TensorFlow Lite [58], for the local training the same approach was used. These tests are described in more detail in the next chapters.

The data returned by the IMU is then used to train the Model of the NN in a platform, like Edge impulse, Google colab and Python on the PC, using Visual Studio Code IDE and TensorFlow library.

The performance was evaluated using the following metrics: precision, recall, F1 Score and accuracy, and also visually in the confusion matrix, see table 6.1), each defined with the expressions [43] showed below:

$$precision = \frac{TP}{TP + FP}$$

Table 6.1: Confusion Matrix shows the number of correct and incorrect predictions broken down by each class.

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

- TP true positive rate
- TN true negative rate
- FP false positive rate
- FN false negative rate

Accuracy: Percentage of correct predictions.

Precision: Ratio of correctly predicted positive observations to the total predicted positives for a class.

Recall: Ratio of correctly predicted positive observations to all observations in the actual class.

F1 Score: Harmonic mean of precision and recall, providing a single metric that balances both. The harmonic mean is often used to calculate the average of the ratios or rates.

6.2 Edge Impulse

Edge Impulse is a free development platform for ML on edge devices [59]. It provides powerful automation and low-code capabilities to make it easier to build valuable datasets for edge devices. It integrates very well with small portable MCUs.

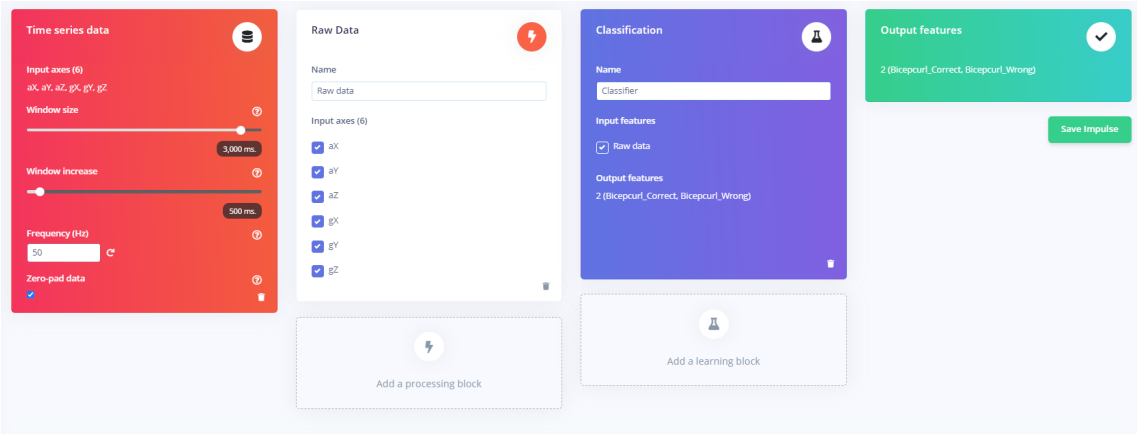


Figure 6.2: Edge Impulse: Create impulse.

The platform builds datasets, trains models, and optimizes libraries to run on any edge device, from extremely low-power MCUs to Graphics Processing Units (GPUs), see figure 6.2. Edge Impulse provides a range of NNs architectures, including CNNs.

Edge Impulse makes use of TensorFlow Lite for training, optimizing, and deploying deep learning models to embedded devices.

The Raw Data block generates windows from data samples without any specific signal processing. It is great for signals that have already been pre-processed and if you just need to feed your data into the NN block. It makes it possible to scale the axis to normalise the data; it was not applied since this was done before in the raw dataset collection.

For Flatten statistical analysis is performed on each window, computing between 1 and 8 features for each axis, depending on the number of selected methods: Calculating: Average: Calculates the average value for the window; Minimum: Calculates the minimum value in the window; Maximum: Calculates the maximum value in the window; Root-mean square: Calculates the RMS value of the window; Standard deviation: Calculates the standard deviation of the window; Skewness: Calculates the skewness of the window; Kurtosis: Calculates the kurtosis of the window; Moving Average Number of Windows: Calculates the moving average by maintaining a rolling average of the last N windows.

The Spectral features block extracts frequency, power, and other characteristics of a signal. Can be chosen statistical features, like RMSS, skewness, Kurtosis or spectral features like maximum value from fast fourier transform frames for each bin that was not filtered out.

Using a platform like Edge impulse helps in the configuration, generation, and training of the model, see figure 6.2. Making it easy to test the multiple possibilities for feature extraction. Table 6.2 represented a summary of the data collected time, number of samples, and split between: train and validation. For example, we can analyse the figures using a Raw data processing 6.3 and Spectral Analysis 6.5. It

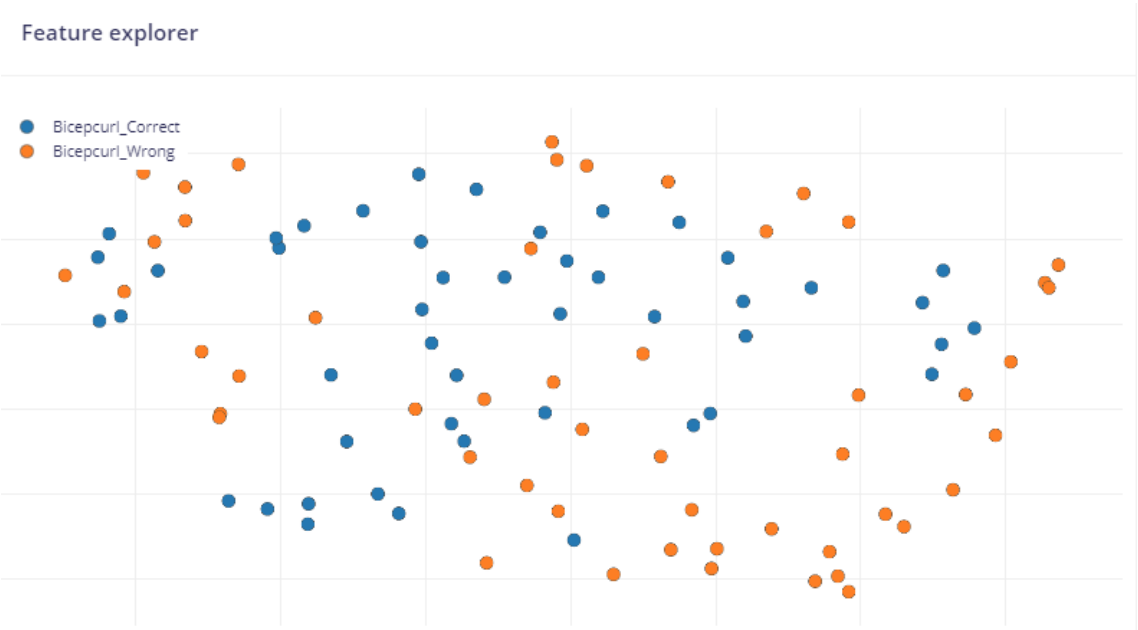


Figure 6.3: Edge Impulse: Feature explorer using processing block raw data.

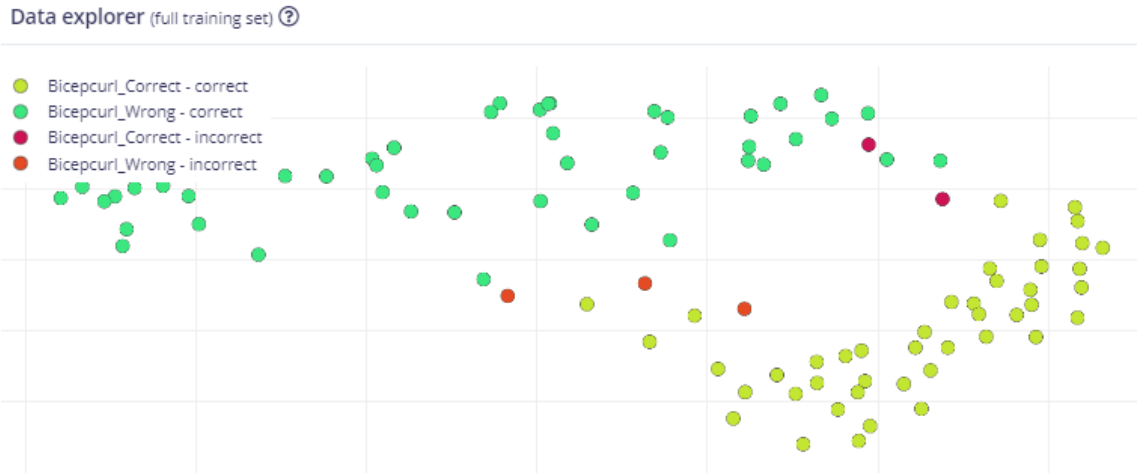


Figure 6.4: Edge Impulse: Data explorer.

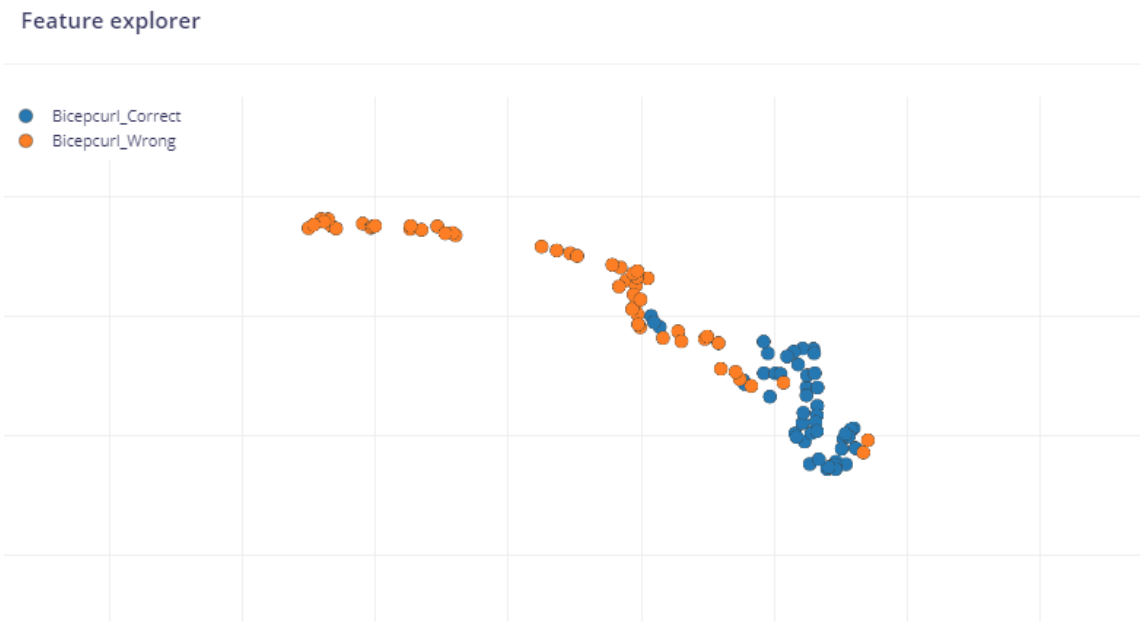


Figure 6.5: Edge Impulse: Feature explorer using processing block spectral analysis.

is visible that data points for different classes are roughly divided, but there is a lot of overlap between the wrong and the correct class, probably because the movement wrongly done was still very similar to the correct one. The feature explorer is a useful tool to help you analyse your dataset, your feature extraction (processing) method, and how well you should expect a machine learning model to classify new samples. First, notice that the samples fall into one of two categories: bicep well performed dataset, "Bicep_Correct", and wrongly performed, "Bicep_Wrong" . These categories come directly from the label names for that dataset. Each sample is represented by a dot on the plot, and the colour of that dot corresponds to its associated label. Using time series data, each window corresponds to one sample. The ideal case would show all the samples in each category in their own grouping separate from other clusters of samples. If that is the case, then you have a very good processing block, and your machine learning model can often be very simple. We can analyse that for Spectral Analysis there is much more separation than in the Raw Data, but still in the middle, we can see a lot of overlap. The overlap among samples that belong to several categories. ML models will often struggle when samples are ambiguous like this. They will have a hard time differentiating among the samples, and you can expect lower accuracy among those groups.

After testing and "playing" we the different types of processing blocks for feature extraction, the next step was to train the model. On Edge Impulse this is done by clicking on a button, no code is required. In figures 6.7, 6.8 and 6.9 are presented the results obtained and summarized in table 6.3, for Bicep Curl was possible to obtain an accuracy between 80% to 95%, Accuracy of 95% instead of 80% with the same data just changing the processing method, for Shoulder press a accuracy of 94.1% and 72.2% for Tricep extensions. It is possible to conclude that

Table 6.2: Edge Impulse data collected times.

Bicep Curl correct Data collected: 4m 8s 59 samples of 3s each 80% Train / 20% test	Bicep Curl correct Data collected: 4m 8s 59 samples of 3s each 80% Train 20% test	Tricep extensions correct Data collected: 4m 38s 59 samples of 3s each 80% Train 20% test
Bicep Curl wrong Data collected: 5m 5s 64 samples of 3s each 80% Train 20% test	Shoulder Press wrong Data collected: 4m 2s 36 samples of 3s each 80% Train / 20% test	Tricep extensions wrong Data collected: 4m 8s 50 samples of 3s each 80% Train / 20% test

Table 6.3: Results obtained in edge impulse.

	Precision	**Recall**	**F1 Score**	**Accuracy (%)**	**Training time (ms)**
Bicep Curl	0.80	0.80	0.80	80.0%	2m 61s
Shoulder Press	0.95	0.94	0.94	94.1%	2m 61s
Tricep Extension	0.73	0.72	0.72	72.2%	2m 60s
All six exercises	0.44	0.49	0.44	49.1%	10m 26s

good accuracy was obtained using this platform.

In the final step of the project a Convolutional 1D network was used, as illustrated in figure 6.6 the CNN layers were: three Conv1D and three Max polling, one Flatten, two Dense Layers. In other platforms, the same structure was used to make it possible and easier to compare.

The model trained on Edge Impulse achieved a good overall performance, with accuracy ranging from 72% to 95%, depending on the exercise performed. This platform increases the velocity of starting to train the model, makes it easier for people with less experience in ML and coding, and generates the model and code to run directly in Arduino, it gives sample code and everything ready to use. TFLite is also included and ready to run. On the other hand limits the customization, the free private projects are limited to two, on the free version there is limited time to train, and the number of supported MCUs is limited and not possible to control.

6.3 Google Colab

The second approach was to do the training using Google Colab, leveraging its cloud-based resources for more intensive training. Google Colaboratory, commonly known as Google Colab, is a cloud-based Jupyter notebook environment that provides a platform for writing and executing Python code through a browser. Especially popular in the data science and ML communities, Google Colab offers free access to a GPU which is particularly useful for training deep learning models due to their parallel processing capabilities.

Like Edge Impulse, Google Colab can make use of TensorFlow Lite and is designed to run ML models on MCUs and other devices with only a few kilobytes of

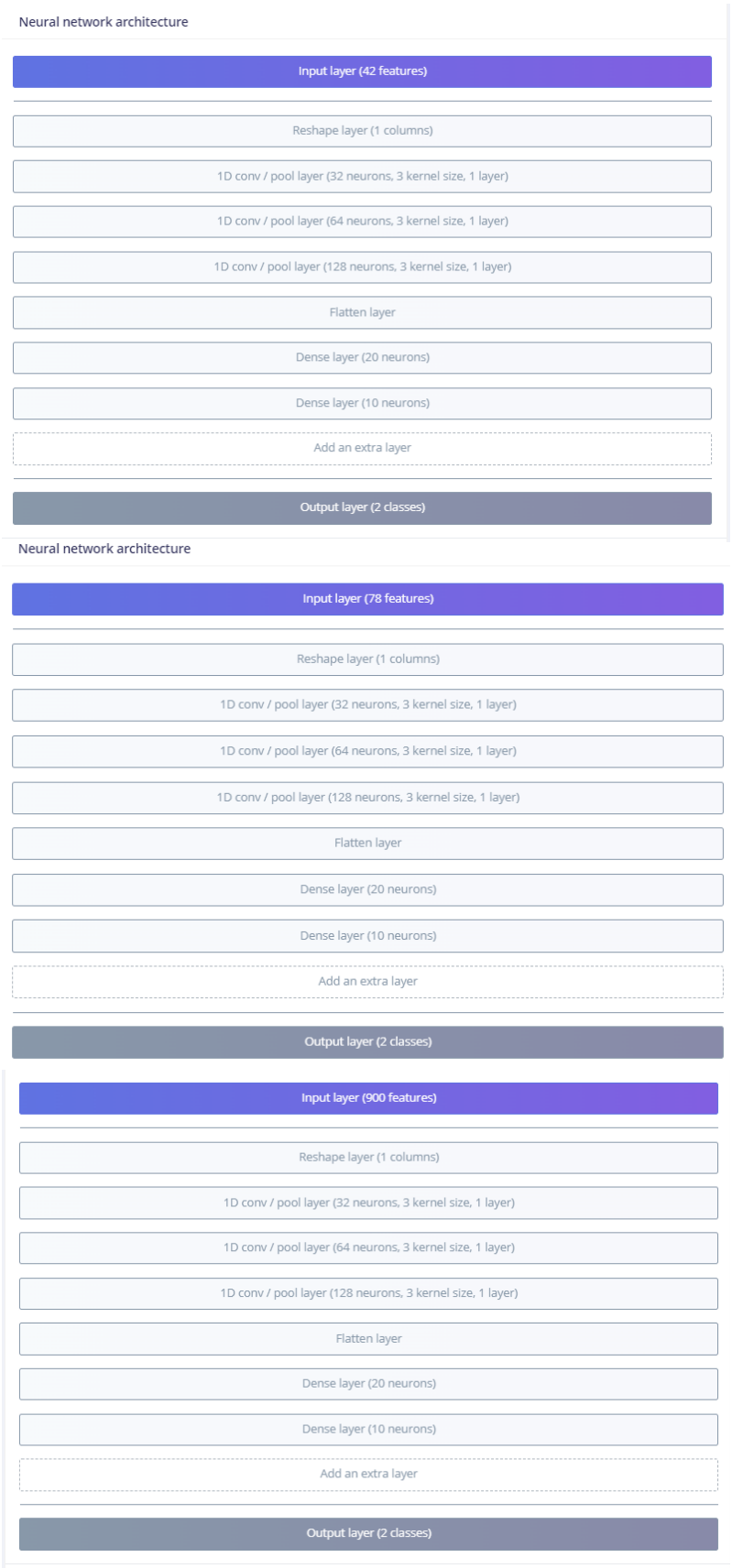


Figure 6.6: Edge Impulse neural network architecture for the exercise bicep curl using processing block: flatten (42 Features), spectral analysis (78 Features) and raw data (900 Features).

	BICEPCURL_CORRECT	BICEPCURL_WRONG
BICEPCURL_CORRECT	80%	20%
BICEPCURL_WRONG	0%	100%
F1 SCORE	0.89	0.91

(a) Bicep curl processing block flatten, resulting confusion matrix.

(b) Bicep curl processing block flatten, resulting accuracy.

	BICEPCURL_CORRECT	BICEPCURL_WRONG
BICEPCURL_CORRECT	90%	10%
BICEPCURL_WRONG	0%	100%
F1 SCORE	0.95	0.95

(c) Bicep curl processing block spectral analysis, resulting confusion matrix.

(d) Bicep curl processing block spectral analysis, resulting accuracy.

	BICEPCURL_CORRECT	BICEPCURL_WRONG
BICEPCURL_CORRECT	80%	20%
BICEPCURL_WRONG	20%	80%
F1 SCORE	0.80	0.80

(e) BicepCurl processing block Raw data, resulting confusion matrix.

(f) BicepCurl processing block Raw data, resulting accuracy.

Figure 6.7: Edge Impulse: Results bicep curl.

	SHOULDERPRESS_CORRECT	SHOULDERPRESS_WRONG
SHOULDERPRESS_CORRECT	100%	0%
SHOULDERPRESS_WRONG	16.7%	83.3%
F1 SCORE	0.96	0.91

(a) Shoulder press processing block Raw data, resulting confusion matrix.

(b) Shoulder press processing block Raw data, resulting accuracy.

Figure 6.8: Edge Impulse: Results shoulder press.

	TRICEP_CORRECT	TRICEP_WRONG
TRICEP_CORRECT	72.7%	27.3%
TRICEP_WRONG	28.6%	71.4%
F1 SCORE	0.76	0.67

(a) Tricep extensions processing block Raw data, resulting confusion matrix.

(b) Tricep extensions processing block Raw data, resulting accuracy.

Figure 6.9: Edge Impulse: Results tricep extensions.

memory. Google Colab provides access to powerful computing resources, including GPU Google Colab advantage would be to use GPU, a disadvantage it's that an internet connection is needed because it's a web interface.

This platform was explored and tested with a basic example, nevertheless like mentioned in the [1] book the option was to run the script in another way and inspired by the setup used in the study [55], where they were able to achieve good results using this experimental setup.

6.4 TensorFlow Training

TensorFlow Library for Python was used to build and train the model, which is a set of instructions that tell an *interpreter* how to transform data in order to produce an output. To use the generated model just load it into memory and execute using TensorFlow.

The NN was implemented using the Keras NN modeling library. The set contains data collected from 8 people under controlled conditions. As mentioned before, the data are recorded by a tri-axial accelerometer and tri-axial gyroscope with a sampling frequency of 50 Hz and are labelled for 6 exercises - bicep curl correct / incorrect, should press correct / incorrect and tricep extension correct / incorrect. The sensor data was segmented using a 3s sliding window with no overlap. Below it is showed the windowing:

For all 6 exercises:

```
Processing index 0 for gesture 'Bicepcurl_Correto'.
 There are 80 recordings of the Bicepcurl_Correto gesture.
Processing index 1 for gesture 'Bicepcurl_Errado'.
 There are 86 recordings of the Bicepcurl_Errado gesture.
Processing index 2 for gesture 'Shoulderpress_Correto'.
 There are 90 recordings of the Shoulderpress_Correto gesture.
Processing index 3 for gesture 'Shoulderpress_Errado'.
 There are 78 recordings of the Shoulderpress_Errado gesture.
```

```
Processing index 4 for gesture 'Tricep_Correto'.
 There are 89 recordings of the Tricep_Correto gesture.
Processing index 5 for gesture 'Tricep_Errado'.
 There are 79 recordings of the Tricep_Errado gesture.
inputs shape = (502, 930)
outputs shape = (502, 6)
inputs len = 502
outputs len = 502
Data set parsing and preparation complete.
```

For a specific one (Bicep curl):

```
Processing index 0 for gesture 'Bicepcurl_Correto'.
 There are 80 recordings of the Bicepcurl_Correto gesture.
Processing index 1 for gesture 'Bicepcurl_Errado'.
 There are 86 recordings of the Bicepcurl_Errado gesture.
inputs shape = (166, 930)
outputs shape = (166, 2)
inputs len = 166
outputs len = 166
Data set parsing and preparation complete.
```

```
The software platform used to build the system is based on the Python (Python is pr
```

6.4.1 Training, Validation, and Testing Datasets

Randomly split input and output pairs into sets of data: 60% for training, 20% for validation, and 20% for testing. To evaluate the accuracy of the model that was trained some data is split to compare its predictions to real data and check how well it matches, an example is shown below.

- the training set is used to train the model
- the validation set is used to measure how well the model is performing during training
- the testing set is used to test the model after training

```
inputs_train len = 99
inputs_test len = 33
inputs_validate len = 34
outputs_train len = 99
outputs_test len = 33
outputs_validate len = 34
Data set randomization and splitting complete.
```

6.4.2 Model Training

The architecture of the CNN, illustrated as a schematic representation of the CNN designed in this work is shown in figure 6.10 consisted of an input layer, three convolutional layers with ReLUs and max-pooling, two fully-connected layers and a soft-max output layer for classification. The kernels applied are small matrix used to filter input data in convolutional layers, while ReLU is an activation function that introduces non-linearity by outputting the input directly if it is positive, otherwise, it outputs zero.

The first hidden layer was a convolutional layer (Conv1D). In this layer, it was applied 8 filters, Kernel size = 3, for filtered each signal. Subsequently, the filtered output of the first convolutional layer was transformed non-linearly using ReLUs. The output of the ReLUs was then compressed using temporal max-pooling. After the temporal max-pooling, it was applied a second convolutional layer. This layer took the transformed and pooled output of the first convolutional layer as input.

In the second convolutional layer, it was applied a 16 filter again to a 1D convolutional layer, kernels again with a length of 3 samples. The output of this layer was transformed and compressed by applying the ReLU non-linearity and overlapping temporal max-pooling into a third Conv1D layer.

Following the two convolutional layers, fully-connected were added which took the non-linearly transformed and pooled output of the third convolutional layer as input.

The fully-connected layers (Dense) consisted of 16 and then 4 units, being units the number of neurons or nodes, with the activation function set to ReLU.

The final output layer was designed to have as many neurons as gestures. For the output layer, the soft-max activation function was employed. This can be found summarized in table 6.4. Below is also shown a result of the training and some training metrics:

```
Epoch 1/600 - 2s 7ms/step - loss: 0.2635 - mae: 0.5057 - val_loss: 0.2401 - val_mae: 0.4868
...
Epoch 600/600 - 1s 5ms/step - loss: 2.6305e-05 - mae: 0.0013 - val_loss: 0.3042 - val_mae: 0.357
```

Training Mean Absolute Error (MAE) represents the average absolute difference between the predicted values and the actual values on the training data. The validation loss is the error measure on the validation data, while validation MAE is the average absolute error on the validation data. Epochs are how many times our entire training set will be run through the network during training.

The decrease in loss (training loss) from 0.2635 to 2.6305e-05 and MAE from 0.5057 to 0.0013 across 600 epochs indicates that the model has minimized the prediction errors. The final validation loss and validation MAE values (0.3042 and

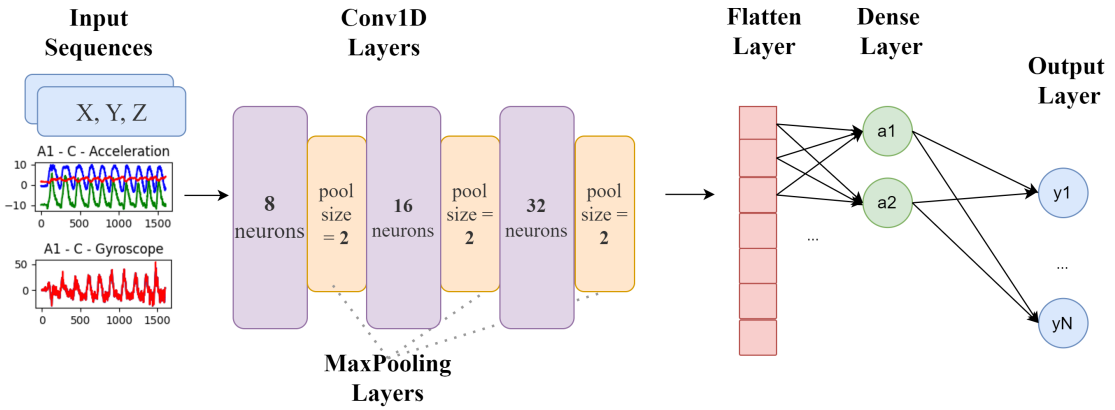


Figure 6.10: CNN model definition showing the included layers.

0.357, respectively) suggest that the model generalizes reasonably well. However, there may be slight over-fitting as the validation metrics are higher than the training metrics.

The training and validation metrics should show similar results, indicating that the model performs consistently on both seen (training) and unseen (validation) data. When the training metrics are better than the validation metrics, it suggests that the model has learned the specifics of the training data too well, including noise or details that do not generalize to new data, this is characteristic of over-fitting.

6.4.3 Result Graphs and Training Metrics

Showing a graphical representation of the lost and Mae results may give another visibility of what happened in the model generation. For the loss, illustrated in figure 6.11, is possible to see that in the first epochs (less than 50) the model improves its predictions since the loss goes almost to zero.

For the MAE, illustrated in 6.12, the values are high, around 0.3, so predictions are wrong about 30% of the time.

In figures 6.13, 6.14, and 6.15 it is illustrated the confusion matrix for each of the 6 exercises combined 2 by 2 for the correct and incorrect gesture.

6.4.4 Converting the Model for TensorFlow Lite

Convert a TensorFlow model to a TFLite format (TensorFlow Lite format) using Keras API. It is saved as a Flat-Buffer, space efficient format, this will be very useful in the next chapter when the Model needs to be deployed into the MCU. Flat-Buffers are designed to be highly space-efficient and allow for quick access to data without requiring unpacking or parsing, which is particularly beneficial for deployment on resource-constrained devices like MCUs.

Then, the trained model was optimised and converted into a C language representation using TensorFlow Lite framework [58].

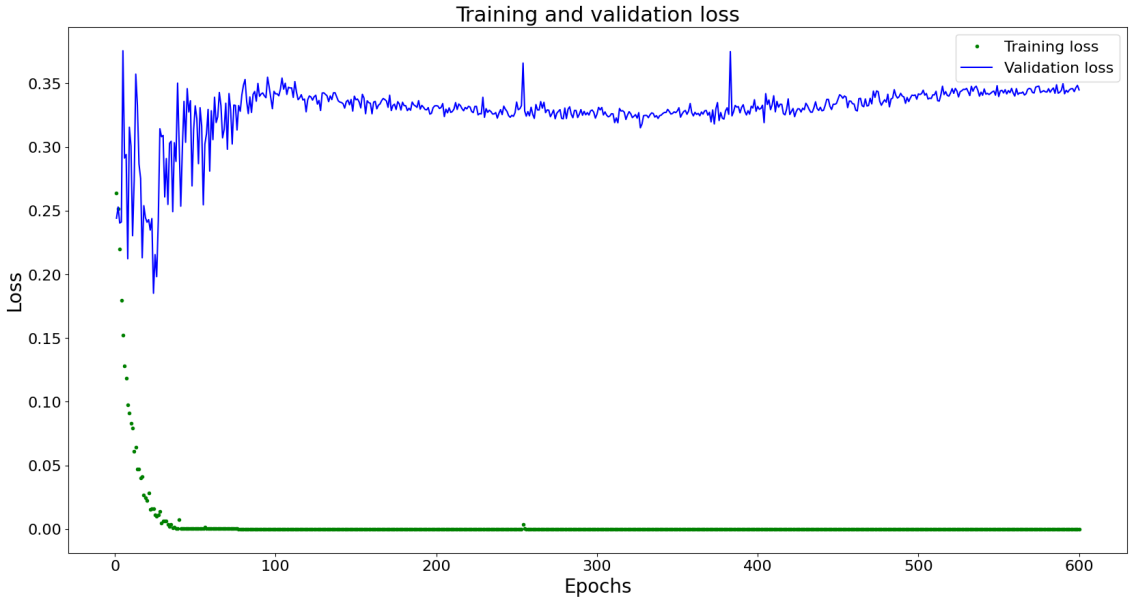


Figure 6.11: Graph the loss to see when the model stops improving, in blue the validation loss and green the training loss.

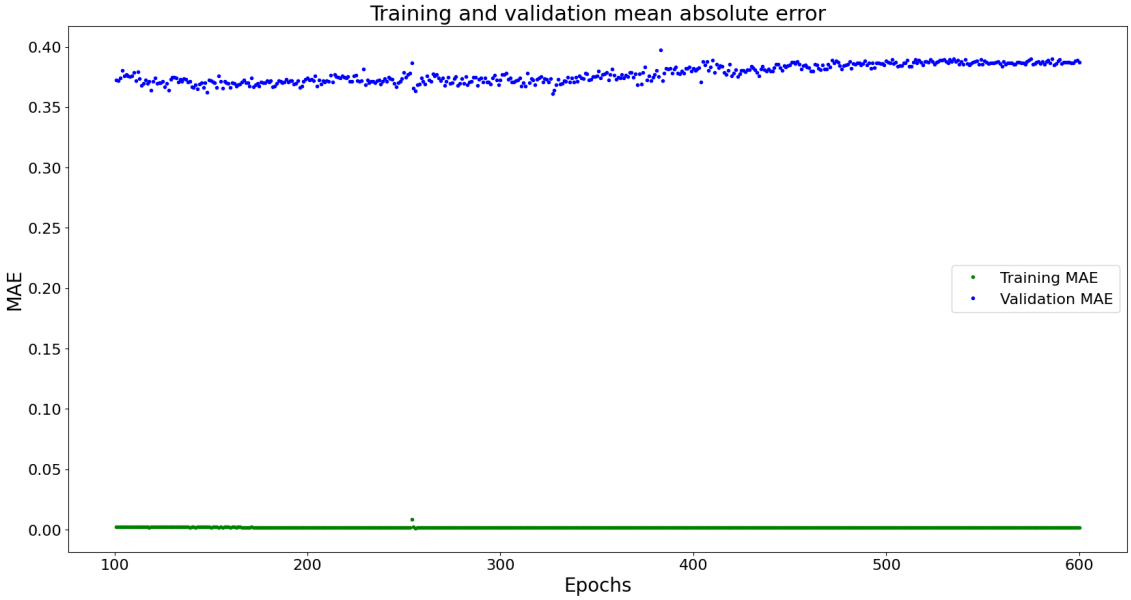


Figure 6.12: Mean absolute error graph, in blue the validation MAE and green the training MAE.

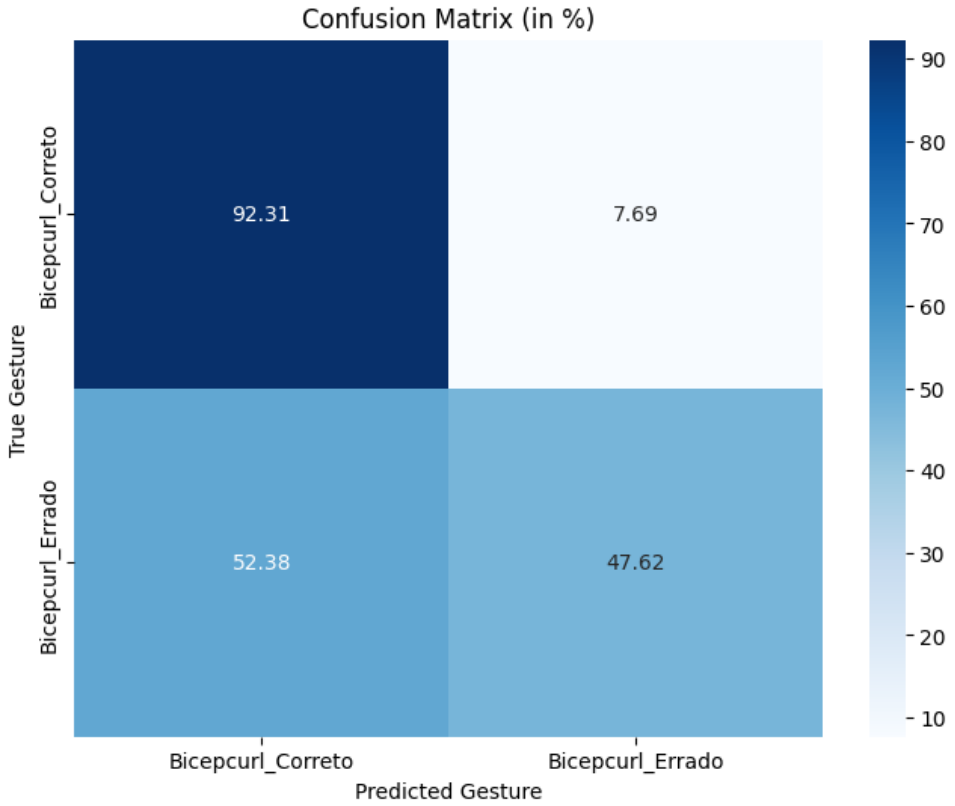


Figure 6.13: Bicep Curl: Confusion Matrix.

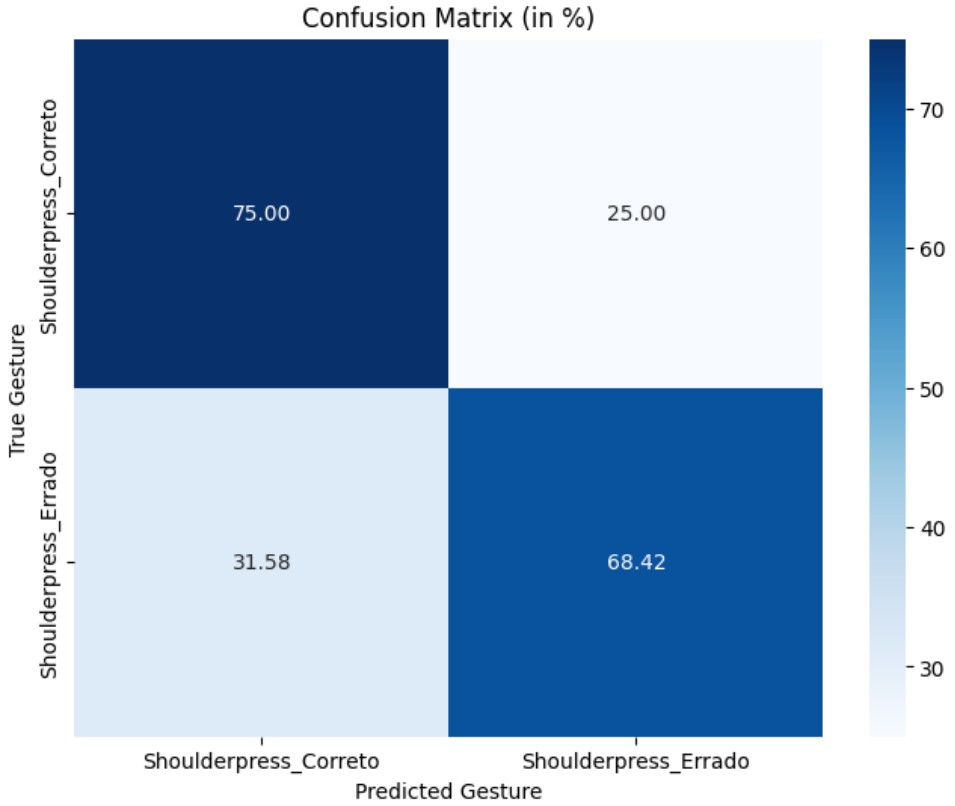


Figure 6.14: Shoulder Press: Confusion Matrix.

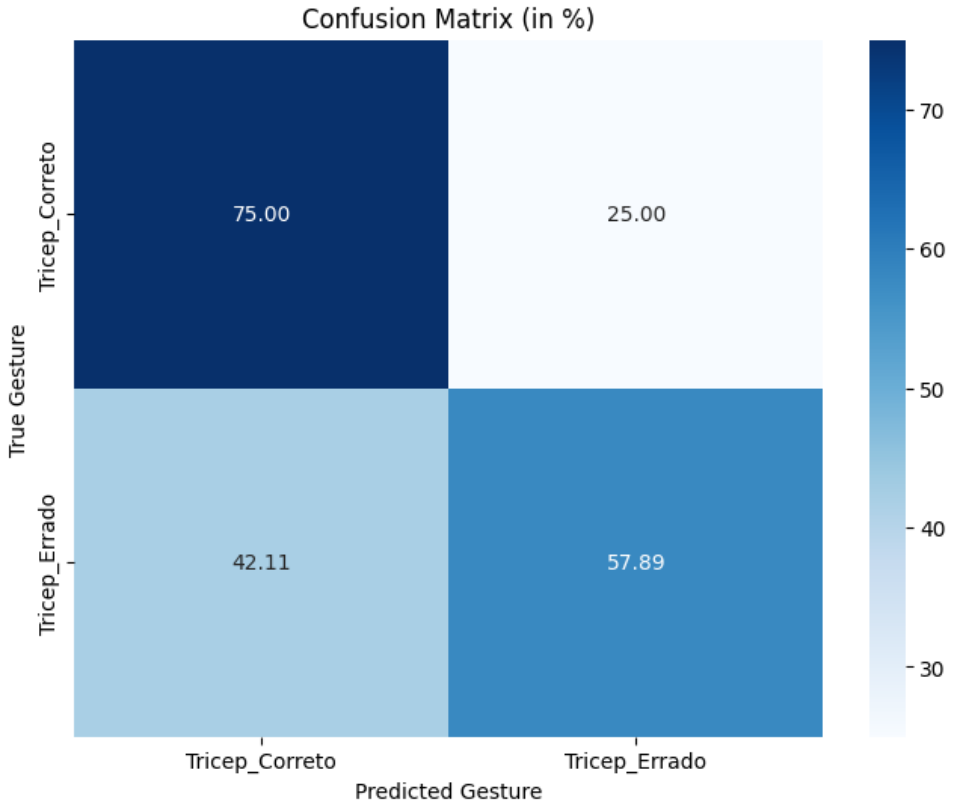


Figure 6.15: Tricep Extension: Confusion Matrix.

Table 6.4: Detailed architecture of CNN Keras used in python in the local script.

Layers	**Filters**	**Kernel Size**	**Size Pooling**	**Relu**
Conv1D	8	3	-	Yes
MaxPooling1D	-	-	2	-
Conv1D	16	3	-	Yes
MaxPooling1D	-	-	2	-
Conv1D	32	3	-	Yes
MaxPooling1D	-	-	2	-

6.4.5 Post-Training Quantization

Post-training quantization is a conversion technique that can reduce model size while also improving CPU and hardware accelerator latency, with little degradation in model accuracy. The technique used was "Full integer quantization", with Benefits of 4x smaller, 3x+ speed-up, most used in: CPU, Edge TPU and MCUs [60]. This technique of quantization reduces the precision of the numbers to fit in 8-bit integers, the Edge Impulse platform does the same when converting the model to be deployed to the MCU.

Afterward, the generated model has then been optimised and converted into TFLite format to be installed into the MCU.

Table 6.5: CNN Keras Metrics - Precision, Recall, F1 Score, and Accuracy — are dimensionless ratios or percentages that represent the model's performance.

Exercise	Precision	Recall	F1 Score	Accuracy (%)
Bicep Curl	0.76	0.65	0.64	65%
Shoulder Press	0.60	0.63	0.60	59%
Tricep Extension	0.60	0.60	0.60	60%
All six exercises	0.33	0.45	0.37	45%

Table 6.6: CNN Keras training time.

	Training Time (m)
Bicep Curl	5m 36s
Shoulder Press	5m 50s
Tricep Extension	5m 17s
All six exercises	16m 10s

6.4.6 Discussion and Evaluation

A summary of the metrics and training times are shown in table 6.5 and 6.6. The results obtained with this technique were less accurate than the ones obtained in the Edge impulse platform, possibly because here the window sliding is done pragmatically and in some of the samples 3s will overlap in the next sample. Nevertheless, this way of training the NN has some advantages like controlling better the Model training: what is used, how to improve, how to evaluate the metrics, and optimize the model. It has also a downside, since it's a programming language and not just a platform that is configured, and as all the options are ready to use, it is more difficult and time-consuming to develop and requires more knowledge to achieve good results, a bigger learning curve.

7

Embedded System for Real-Time HAR Using TFLite

In chapter 6, the model was generated. This chapter presents the implementation of a real-time embedded human activity recognition system using TensorFlow Lite on a MCU platform to run inference on the generated model. The system uses TFLite to run a pre-trained NN model that identifies if an exercise is well performed based on data from IMU sensors. The work demonstrates how embedded systems can be used for ML tasks in a resource-constrained environment, with applications in wearable technology and human-computer interaction.

7.1 System Implementation and Integration

TFLite models are represented in a Flat-Buffers format, which provides reduced size and faster inference compared to TensorFlow's Buffer format, due to its smaller code footprint and directly accessible data. The generation of the TFLite model can be done through three different methods:

1. using existing TFLite models from available examples;
2. designing our model through TFLite Maker; or
3. converting TensorFlow models to TFLite using the TFLite converter and applying optimization method through the process.

In this work, it was used option (3) converting a TensorFlow model to TFLite

using TFLiteConverter from the Keras [57] and Arduino an open-source IDE to write code and upload it to the board.

Once a TFLite model is finalised it is converted to a C source file for running on the MCU. Running the generated C code on MCU requires an MCU specific library version of TFLite, the TensorFlow Lite Library (TFLiteLibrary) used in Arduino to run/interpret the deployed model.

TFLiteLibrary was specifically designed for MCU deployment. The use of TFLite addresses the design constraints of embedded ML namely, power consumption, memory, latency, and privacy.

This system is built around a MCU integrated with an LSM6DS3 IMU sensor. The LSM6DS3 sensor captures movement data, including acceleration and gyroscope measurements across the X, Y, and Z axes, illustrated in figure 4.3, which are essential for recognizing different exercises (like exercise well or poorly done). This sensor is selected for its low power consumption and high performance, making it suitable for continuous monitoring in embedded systems. The work also uses TFLite, a lightweight version of TensorFlow designed specifically for running ML models on devices with limited computational resources. TFLite enables the deployment of pre-trained models in a compact and efficient manner, for applications on MCUs with minimal memory and processing capabilities.

An OLED display (Adafruit SSD1306) is used to show user feedback about the exercise and if it's correctly done, also a LED will turn on green for correctly done and red for incorrectly done, for a better user experience the number of exercises correctly or incorrectly done can be seen in the implemented mobile app in real-time. This display provides visual feedback by showing the recognized exercise and if it's well performed in real-time. The integration of the OLED display improves the system's interactivity and usability, making it easy for users to understand the system feedback immediately.

The system architecture is centered around the acquisition, preprocessing, and inference of IMU data to recognize the exercises in real-time. Initially, the IMU sensor continuously monitors and collects movement data, like exposed before including acceleration and gyroscope data. The system is programmed to detect significant movement by calculating the sum of the absolute values of acceleration across all three axes and comparing it against a predefined threshold. When this threshold is exceeded, indicating a potential exercise is being performed, the system begins recording a fixed number of data samples, which represent the unique signature of the movement involved. In this case, to run the inference needs to detect some movement and identify if that movement can be classified as a correct or incorrect exercise.

Once the data was collected, it was preprocessed to follow the input requirements of the TensorFlow Lite model. This involves normalizing the raw acceleration

and gyroscope data, ensuring that the data is suitable for the ML model to process. This step is critical as it standardizes the input, allowing the model to effectively interpret the variations in the sensor readings. The preprocessed data is then input into the TensorFlow Lite model, which has been previously trained to recognize specific exercises. The model performs inference on the input data, using tensors, generating a set of probabilities that correspond to different predefined exercises. The exercise with the highest probability is identified as the recognized exercise. Finally, the system displays the recognized exercise on the OLED screen, providing immediate feedback to the user, the exercise recognized can be for example: a tricep extension or a wrong tricep extension. Exercises not well performed were also trained and labelled in the model.

The code implementation is divided into two main functions: `setup()` and `loop()`. The `setup()` function is responsible for initializing all the necessary components of the system. It begins by setting up serial communication and initializing the IMU sensor, ensuring that the sensor is ready to collect movement data. Next, the function initializes the TensorFlow Lite interpreter, which involves loading the pre-trained model into memory and allocating space for the model's input and output tensors. The function also initializes the OLED display, preparing it to show output to the user. This setup process is essential for ensuring that all components are properly configured before the system begins operation.

The `loop()` function forms the core of the systems operation. It runs continuously, monitoring the IMU sensor for significant movement recognition. When such an event is detected, the function collects and pre-processes the required number of data samples, which are then input into the TensorFlow Lite model for inference. After the model processes the data, the recognized exercise is identified based on the highest output probability. The exercise is then displayed on the OLED screen, and in the Arduino Serial if we are connected to the PC, illustrated in figure 7.2. The loop is designed to operate in real-time, ensuring that the system remains responsive to new exercises and can provide immediate feedback to the user. This continuous monitoring and processing cycle makes the system effective for real-time movement recognition in multiple applications.

The systems components can be seen in figure 7.1.

7.2 Results and Discussion

The implemented system successfully identifies and classifies exercise correctness based on IMU data. The micro-controller's ability to run the TensorFlow Lite model in real-time demonstrates the feasibility of deploying ML models on low-power, resource-constrained devices. The OLED display provides a simple yet effective means of user interaction, showing the recognized exercise immediately af-

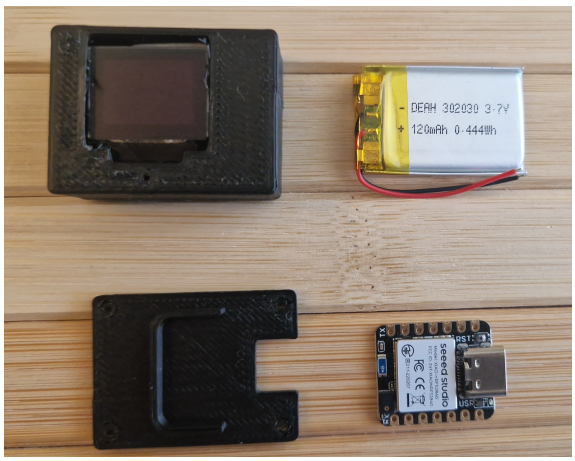


Figure 7.1: System components.

```
20:23:46.235 -> Tricep_Correto: 0.000000
20:23:47.269 -> Tricep_Errado: 1.000000
20:23:48.261 ->
20:23:48.885 -> Tricep_Correto: 0.999392
20:23:49.846 -> Tricep_Errado: 0.000608
20:23:50.877 ->
20:23:51.498 -> Tricep_Correto: 0.000000
20:23:52.476 -> Tricep_Errado: 1.000000
20:23:53.468 ->
20:23:54.082 -> Tricep_Correto: 1.000000
20:23:55.108 -> Tricep_Errado: 0.000000
```

Figure 7.2: Arduino serial monitor showing the name of the movement and the it's results.

ter classification. The immediate feedback to the user, the efficient use of resources, and the low latency in inference: the system is capable of processing input data (from the IMU) and producing an output (recognized exercise) with minimal delay.

The system's performance can be further optimised by tuning the model, adjusting the sampling rate, or optimizing the TensorFlow Lite interpreter for specific hardware. Potential improvements include integrating more complex models, more exercises, or additional sensors for richer data input.

8

Mobile Application for Embedded System Validation

This chapter presents the development of a mobile application that interfaces with the HAR system. The app provides users with a user-friendly way to monitor and track their exercises, displaying real-time feedback based on the data processed by the embedded system.

8.1 Android Application Design and Development

For the app development, it was used a Low Code Platform called OutSystems [61]. Low-code is a visual software development approach that simplifies the creation of applications by minimizing the need for traditional, intricate hand-coding. It introduces a user-friendly, drag-and-drop development environment that makes app development accessible to both novice coders and experienced developers. This approach allows for faster, more inclusive, and streamlined business software creation and innovation. OutSystems is a platform for the rapid development of web and mobile apps. Using a language based on workflows makes it more friendly to program. Significantly faster due to visual development tools and pre-configured components.

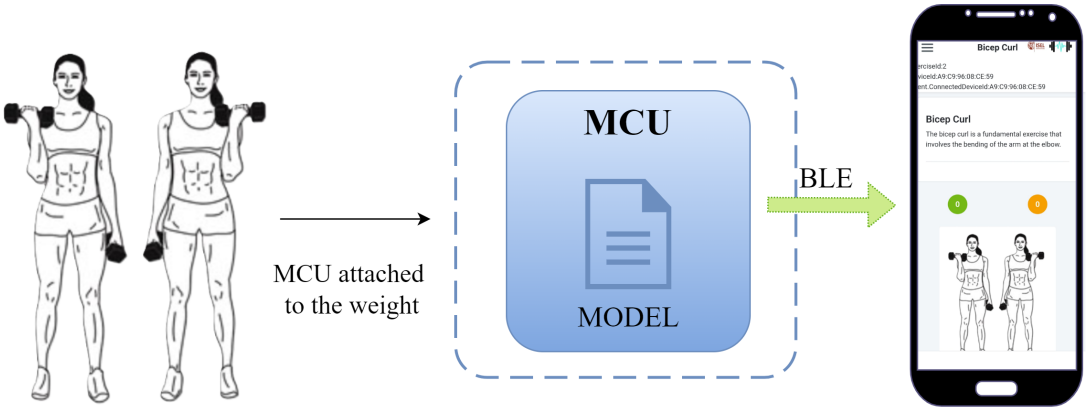


Figure 8.1: Adapted solution outline for the mobile app.

8.2 Integration With the Embedded System

The integration with the Embedded system was made using BLE, like showed in figure 8.1. The Embed system advertises that is ready for connections:

Listing 8.1: BLE advertise code

```
// Create a service and characteristic for the IMU data
BLEService imuService("181A");
// Environmental Sensing service
//BLECharacteristic imuCharacteristic("2A58",
//BLERead | BLEWrite | BLENotify, 40);
// Increased length for string data

// Initialize BLE
  if (!BLE.begin()) {
    if (Serial) Serial.println("Starting BLE failed!");
    while (1);
  } else {
    if (Serial) Serial.println("BLE initialized successfully.");
  }
  BLE.setLocalName("XIAO_IMU_Sense");
  BLE.setAdvertisedService(imuService);
  BLE.setAdvertisedServiceUuid("19B10000-E8F2-537E-4F6C-D104768A1214");
  imuService.addCharacteristic(imuCharacteristic);
  BLE.addService(imuService);
  BLE.advertise();
```

The BLE specification includes a mechanism known as notify that lets you

know when data's changed. When notify on a characteristic is enabled and the sender writes to it, the new value is automatically sent to the receiver, without the receiver explicitly issuing a read command. This is commonly used for streaming data such as accelerometer or other sensor readings. There's a variation on this specification called indicate which works similarly, but in the indicated specification, the reader sends an acknowledgment of the pushed data. The client-server structure of BLE, combined with the notify characteristic, is generally called a publish-and-subscribe model.

On the Mobile app side, the connection is made to the Embedded System and afterward a Listener is defined that is waiting for a notification from the system indicating that a movement was detected. It was defined a protocol of only 1 integer to reduce the payload of the communication:

- 0X00 (HEX) = 0 (INTEGER) - Indicating is connected
- 0X01 (HEX) = 1 (INTEGER) - Indicating a correct movement
- 0x02 (HEX) = 2 (INTEGER) - Indicating an incorrect movement

For the the proof of concept 3 pages were designed and implemented, see 8.2:

- Connection screen - to be possible to connect/disconnect from the embedded system or do a test read.
- Exercises screen - to choose the exercise that will be performed.
- Exercise Detail screen - show the details of the chosen exercise and in green the number of correctly made exercises and in orange the wrongly done, in figure 8.3 it is possible to visualize the Arduino behind and on the front the smartphone with the counters.

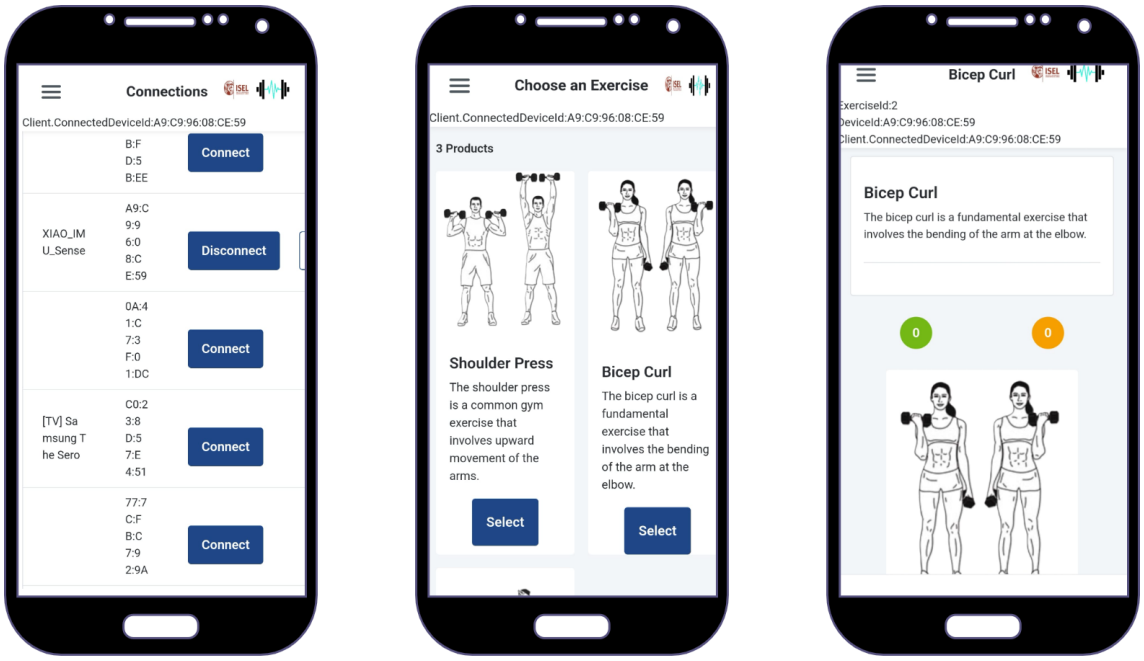


Figure 8.2: Mobile app: screens flow.

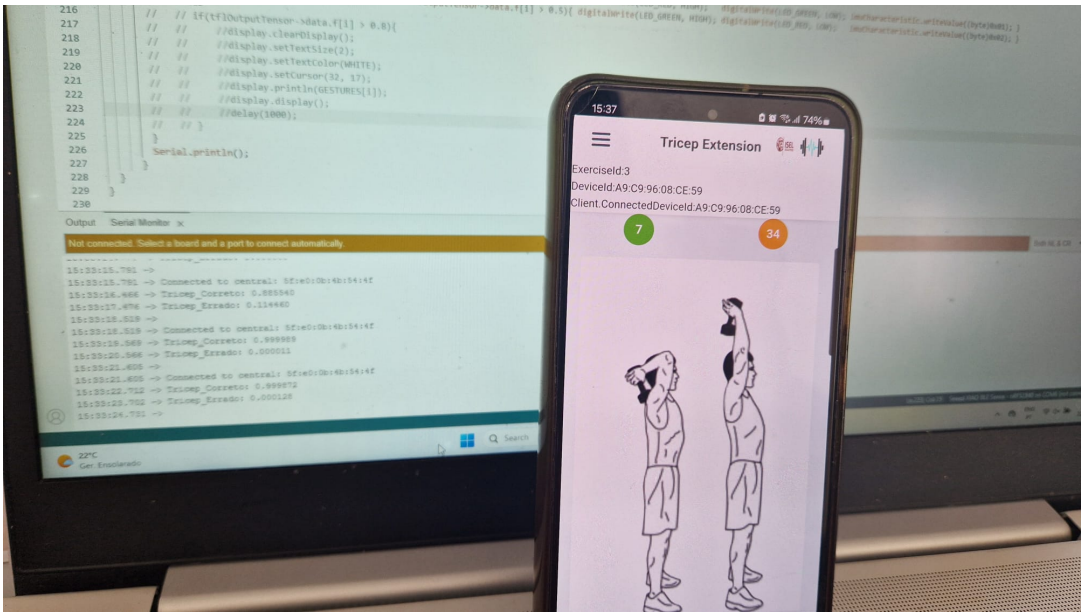


Figure 8.3: Result screen of the android application running on a mobile phone.

9

Results and Discussion

This chapter discusses the performance of the entire embedded system and the accuracy comparison between the different NN training methods. It also summarized limitations and potential improvements for the future.

9.1 Overall System Performance

Overall the system performed well reaching accuracy above 70% in all the platforms and techniques used. Most of the collected metrics also indicate that it's a moderated system. Moderate means it is fairly accurate in distinguishing between a well or wrongly performed exercise. It is possible to conclude looking into the table 9.1 that the metrics obtained from the model tested in Edge Impulse are higher than the ones in the local code. In Edge Impulse the accuracy was between 70% to 95% while training the model in TensorFlow using Python, the accuracy was around 60%. The accuracy of all exercises is shown in figure 9.1 and tables 9.1 and 9.2. As explained before there is some suspicious that it can be related to the window size and split that makes the pattern more accurate, also Edge Impulse it's a No Code online platform that is optimized to train NN, while training the model in TensorFlow using Python, all the code is built, and all the optimizations need to be developed, nevertheless the used library helped a lot to achieve good results. Another factor that seems visible in the data, but requires future work and more investigation is the difference in the data collected from subjects male or female.
Regarding the model size it was possible to conclude that when converting to TensorFlow Lite format and optimizing what memory was used to store each information the selected MCU Seeed Studio XIAO nRF52840 Sense performed very well to the

Figure 9.1: Edge Impulse training performance including all the three exercises using as a processing block raw data.

objectives of this work.

*Binary to Hexadecimal Expansion: Each byte in a binary file is represented by two hexadecimal characters in the C array. Since each hexadecimal digit is represented by an ASCII character, and each byte (8 bits) in the binary file is represented by 2 hexadecimal digits (2 bytes), the size can approximately double.

Regarding Current consumption, it was possible to check that the current consumption on average was 11.9mA, so using a Battery 120mA we have an autonomy of approximately 10h (302030 LiPo battery (120mAh, 3.7V)).

Figure 9.2: Edge Impulse training performance including all the six exercises using as a processing block spectral analysis.

Table 9.1: Results obtained in the tested platforms.

	Precision		**Recall**		**F1 Score**		**Accuracy (%)**		**Training time (ms)**	
	Edge Impulse	**Python**	**Edge Impulse**	**Python**	**Edge Impulse**	**Python**	**Edge Impulse**	**Python**	**Edge Impulse**	**Python**
Bicep Curl	0.80	0.76	0.80	0.65	0.80	0.64	80.0%	65%	2m 61s	5m 36s
Shoulder Press	0.95	0.60	0.94	0.63	0.94	0.60	94.1%	59%	2m 61s	5m 50s
Tricep Extension	0.73	0.60	0.72	0.60	0.72	0.60	72.2%	60%	2m 60s	5m 17s
All six exercises	0.44	0.33	0.49	0.45	0.44	0.37	49.1%	45%	10m 26s	16m 10s

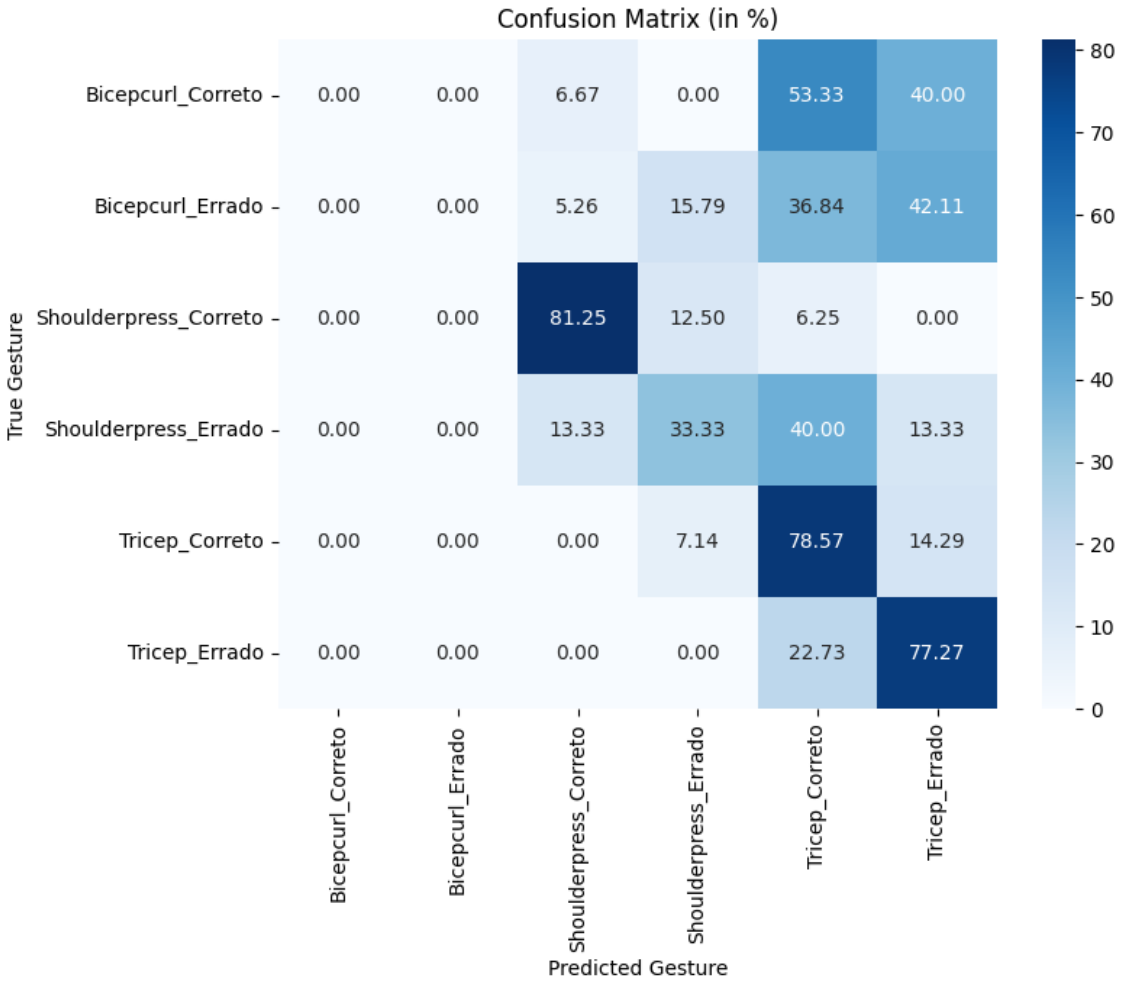


Figure 9.3: Python with TensorFlow training performance including all the three exercises using as a processing raw data.

Table 9.2: Model size comparison.

	Edge Impulse				**Python with TensorFlow**			
	Bicep Curl	**Shoulder Press**	**Tricep Extension**	**All Exercises**	**Bicep Curl**	**Shoulder Press**	**Tricep Extension**	**All Exercises**
Model Size (KB)	NA	NA	NA	NA	256.8	256.7	256.4	256.8
Quantized Model (KB)	NA	NA	NA	NA	73.5	73.4	73.2	73.6
Converted to TFLite Model (KB)	348.0	348.0	348.0	348.0	453.7	453.0	451.8	453.9

9.2 Limitations and Potential Improvements

One of the limitations is the number of exercises, only three are being trained. Actually 6, because the exercises wrongly done are also considered an exercise. There is room to improve and apply the entire flow to more exercises. Collect more raw data to improve the dataset samples and train the model with more information, leading to more accurate results. Because the results obtained from the 6 exercises training were moderate, it indicates that the system is unable to distinguish between different exercises simultaneously and can only assess whether a single type of exercise is performed correctly or incorrectly. For the purpose of this thesis, the overall results were good, if we analyse that in some cases was possible to reach accuracies up to 90%. Still in this topic is the size of the model, with the optimizations, it would be possible to have the Model with the 6 trained exercises included in the MCU memory, the alternative would be 3 Models, but with the size of each Model it would not fit the MCU memory with all the rest of the parts involved, LED, OLED, BLE, etc. Speaking of memory, even with only one Model and with the size shown in previous chapters, it was necessary to move the model to the RAM memory, by removing the *"const"* of the Model array. The Model was saved in SRAM and the rest in the flash memory. When checking the size of what the program occupies:

```
Sketch uses 618888 bytes (76\%) of program storage space.
Maximum is 811008 bytes.
Global variables use 127952 bytes (53\%) of dynamic memory,
leaving 109616 bytes for local variables.
Maximum is 237568 bytes.
```

Another important improvement would be to find the perfect window size for the sliding window because it may have an impact on the training of the NN.

In summary, this thesis is an exploratory work and not 100% reliable, so it has its limitations.

9.3 Real-World Application

After obtaining all the results of the complete systems working, it is possible to say that a prototype could be assembled as a next step to be added to gym weights and indicating to the user if the exercise is correctly done, this way like already mentioned before this system could help prevent possible injuries from exercising without supervision.

A box was built to make it a compact fully integrated system, illustrated in figure 7.1, lightweight, weighs 5 grams, and all the components from: MCU, OLED, and battery are inside the box, making it completely portable.

10

Conclusions and Future Work

The goal of this work was to develop an autonomous embedded system to give the gym user feedback about the correctness of the exercise that is being done in real-time. An accessible, low-power consumption, autonomous, small, compact, and lightweight system consisting only of a MCU Seeed Studio XIAO nRF52840 Sense, a battery, and an OLED display using NN techniques to train the model was created capable of categorizing movements based on their correctness and give that feedback in real-time.

The data acquisition process, driven by the IMUs accelerometer, ensured the capture of movement data, while the integration of BLE facilitated the transmission of raw sensor data to a host. The use of a NN implementation on the MCU made the system perform inference with minimal latency. The mobile application implemented improved the ability to give feedback in real-time.

Sample datasets were necessary for testing specific components of the system and the MCU, but collecting live data enabled the training of the NN to specific exercises chosen for this work, including both correctly and incorrectly performed. During the research activities, no existing dataset was found that could be directly used for the purpose of classifying the movement correctness. As a result, there was a need to create our dataset tailored to the unique requirements of this study, collected with the supervision of a certified personal trainer.

After collecting the data, the NN was trained using different platforms and techniques, including Edge Impulse and TensorFlow implemented in Python. Edge impulse provides better accuracy results and is easier to understand for less experienced people, but on the other hand, it limits the customization of the training configurations. TensorFlow model training implemented in Python gives more free-

dom of customization, but at the same time, it takes a longer learning curve to achieve good results. It ended with the model generation to be used in the embedded system.

Each component was selected to maintain a balance between performance and energy efficiency, given the limitations of running on a small LiPo battery and a small MCU.

Real-time inference on the MCU is done with a TFLite model optimised specifically for the hardware's limited computational power and memory. In this work, the inference process runs on the MCU. Each inference cycle is executed almost instantly, ensuring that movement recognition and feedback are real-time as much as possible.

The results indicated an accuracy from 60% up to 94.1% in identifying correctly or incorrectly performed exercises, these results support the idea that real-time feedback can help with workout safety. Nevertheless, the study was limited to a small dataset of six exercises, so future research should explore a broader range of movements and incorporate larger datasets. Overall, these findings can contribute to the ongoing development of smart fitness technology and can impact injury prevention in exercise practices. To improve the accuracy of the results it is necessary to expand the dataset.

Bibliography

[1] W. Peter and S. Daniel, *TinyML - Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. O'Reilly Media, 2023. [Online]. Available: https://books.google.pt/books?hl=en&lr=&id=tn3EDwAAQBAJ&oi=fnd&pg=PP1&ots=jqllbr-7A4&sig=GAqcAA99f2EPyvOjOaddMu8gzss&redir_esc=y#v=onepage&q&f=false

[2] Darebee, "Darebee - an independent global fitness resource," https://darebee.com/, 2023.

[3] T. Ploetz, N. Y. Hammerla, S. Halloran, and T. Plötz, "Deep, convolutional, and recurrent models for human activity recognition using wearables," 2016. [Online]. Available: https://www.researchgate.net/publication/301818728

[4] M. S. Diab and E. Rodriguez-Villegas, "Embedded machine learning using microcontrollers in wearable and ambulatory systems for health and care applications: A review," *Sensors*, 2022.

[5] X. Guo, J. Liu, and Y. Chen, "When your wearables become your fitness mate," *Stevens Institute of Technology*, 2020.

[6] N. Lanraoui and C. Touati, "Tiny ML for gesture recognition," *Journal of Sensors*, 2023.

[7] D. Shah, "Human activity recognition (har): Fundamentals, models, datasets," 2023. [Online]. Available: https://www.v7labs.com/blog/human-activity-recognition#h3

[8] J. Brownlee, "Evaluate machine learning algorithms for human activity recognition," *Machine Learning Mastery*, 2020.

[9] P. Baheti, "A simple guide to data preprocessing in machine learning," *Website V7 Labs*, 2023. [Online]. Available: https://www.v7labs.com/blog/data-preprocessing-guide

[10] N. Rashid, B. U. Demirel, and M. A. A. Faruque, "Ahar: Adaptive cnn for energy-efficient human activity recognition in low-power edge devices," *IEEE Internet of Things Journal*, vol. 9, pp. 13 041–13 051, 8 2022.

[11] S. D. Bersch, D. Azzi, R. Khusainov, I. E. Achumba, and J. Ries, "Sensor data acquisition and processing parameters for human activity classification," *Sensors (Switzerland)*, vol. 14, pp. 4239–4270, 3 2014.

[12] F. M. Rueda, R. Grzeszick, G. A. Fink, S. Feldhorst, and ten Michael Hompel, "Convolutional neural networks for human activity recognition using body-worn sensors," *Informatics, volume 5, issue 2 of the journal in 2018*, 2018.

[13] C. Emily, S. A. J, B. Kevin, and R. Sam, "Machine and deep learning for sport-specific movement recognition: a systematic review of model development and performance," *Journal of Sports Sciences*, 2018. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/02640414.2018.1521769

[14] GeeksforGeeks, "Introduction to convolutional neural networks," https://www.geeksforgeeks.org/introduction-convolution-neural-network, 2024.

[15] L. T. Nguyen, B. Yu, O. J. Mengshoel, P. W. Jiang Zhu, and J. Z. M. Zeng, "Convolutional neural networks for human activity recognition using mobile sensors," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2014.

[16] T. Zebin, P. J. Scully, N. Peek, A. J. Casson, and K. B. Ozanyan, "Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition," *IEEE Access*, vol. 7, pp. 133 509–133 520, 2019.

[17] Y.-L. Boureau, J. Ponce, J. P. Fr, and Y. Lecun, "A theoretical analysis of feature pooling in visual recognition," 2010. [Online]. Available: https://www.researchgate.net/publication/221345753

[18] A. K. B. Blaine A. Price, Daniel Gooch and B. N. M. Bennasar, "Significant features for human activity recognition using tri-axial accelerometers," *MedCentral*, 2022. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9572087/

[19] B. H. Groh, T. Kautz, and D. Schuldhaus, "Imu based trick classification in skateboarding," *Workshop on Large-Scale Sports Analytics as part of the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2015.

[20] U. Jensen, M. Schmidt, M. Hennig, F. A. Dassler, T. Jaitner, and B. M. Eskofier, "An imu-based mobile system for golf putt analysis," *Sports Engineering*, 2015.

[21] J. M. Jarning, K.-M. Mok, B. H. Hansen, and R. Bahr, "Application of a triaxial accelerometer to estimate jump frequency in volleyball," *Sports Biomechanics*, 2015.

[22] T. Kautz, B. H. Groh, J. Hannink, U. Jensen, H. Strubberg, and B. M. Eskofier, "Activity recognition in beach volleyball using a deep convolutional neural network," *Data Mining and Knowledge Discovery*, 2017.

[23] O. M. H. Rindal, T. M. Seeberg, J. Tjønnås, P. Haugnes, and Øyvind Sandbakk, "Automatic classification of subtechniques in classical cross-country skiing using a machine learning algorithm on micro-sensor data," *Sensors*, 2017.

[24] H. Brock, Y. Ohgi, and J. Lee, "Learning to judge like a human: Convolutional networks for classification of ski jumping errors," *Proceedings of the 2017 ACM International Symposium on Wearable Computers (ISWC)*, 2017.

[25] C. Buckley, M. O. Reilly, A. V. Farrel, L. Clark, and V. Longo, "Binary classification of running fatigue using a single inertial measurement unit," *Proceedings of the IEEE 14th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 2017.

[26] A. Bulling, U. Blanke, and B. Schiele, "A tutorial on human activity recognition using body-worn inertial sensors," *ACM Computing Surveys*, 2014.

[27] M. Chowdhary and S. S. Saha, "On-sensor online learning and classification under 8 kb memory," *Journal of Machine Learning Research*, 2023.

[28] B. F. N. Yala, "Feature extraction for human activity recognition on streaming data," *Journal of Ambient Intelligence and Humanized Computing*, 2015.

[29] R. Yao, G. Lin, Q. Shi, and D. C. Ranasinghe, "Efficient dense labelling of human activity sequences from wearables using fully convolutional networks," 2018.

[30] G. Balestra and M. K. S. Rosati, "Comparison of different sets of features for human activity recognition by wearable sensors," *PubMed Central*, 2018. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6308535/

[31] S. Xia, D. de Godoy, B. Islam, M. T. Islam, S. Nirjon, P. R. Kinget, and X. Jiang, "Improving pedestrian safety in cities using intelligent wearable systems," *IEEE Transactions on Mobile Computing*, 2019.

[32] Z. Wang, Y. Wu, Z. Jia, Y. Shi, and J. Hu, "Lightweight run-time working memory compression for deployment of deep neural networks on resource-constrained," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[33] P. Zhou, W. Shi, J. Tian, Z. Qi, B. Li, H. Hao, and B. Xu, "Attention-based bidirectional long short-term memory networks for relation classification," *54th Annual Meeting of the Association for Computational Linguistics (ACL 2016) - Short Papers*, pp. 207–212, 2016.

[34] R. Mutegeki and D. S. Han, "A cnn-lstm approach to human activity recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020.

[35] M. Lazarova and A. A.-P. S. Tsokov, "Accelerometer-based human activity recognition using 1d convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, 2021.

[36] J. Brownlee, "1d convolutional neural network models for human activity recognition," *Machine Learning Mastery*, 2020. [Online]. Available: https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/

[37] J. Bang, T. Huynh-The, J. Lee, J.-I. Kim, S. Lee, and T. Hur, "Iss2image: A novel signal-encoding technique for cnn-based human activity recognition," *Sensors*, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/11/3910

[38] N. A. Capela, E. D. Lemaire, and N. Baddour, "Feature selection for wearable smartphone-based human activity recognition with able bodied, elderly, and stroke patients," *PLoS ONE*, vol. 10, 4 2015.

[39] S. Shalev-Shwartz and S. Ben-David, "Understanding machine learning: From theory to algorithms," *Published 2014 by Cambridge University Press.*, 2014.

[40] R. Sanchez-Iborra and A. Skarmeta, "Who is wearing me? tinydl-based user recognition in constrained personal devices," *Convergence*, 2021.

[41] T. A. Gonçalves, "Convolutional Neural Network for Hand Gesture Identification on FPGAs," Master's thesis, Instituto Superior Técnico de Lisboa, Nov. 2022.

[42] Arduino, "Github - lakshanthad - tflite-micro-arduino-examples," https://github.com/lakshanthad/tflite-micro-arduino-examples, 2012.

[43] A. M. H. M. H. S. A. R. Z. D. McLernon, "Tinyml empowered transfer learning on the edge," *IEEE Transactions on Emerging Topics in Computing*, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10459233?denied=

[44] C. Banbury, C. Zhou, I. Fedorov, R. M. Navarro, U. Thakker, D. Gope, V. J. Reddi, M. Mattina, and P. N. Whatmough, "Bambury micronets neural network architectures for deploying tinyml applications on commodity microcontrollers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2021.

[45] S. Gupta, "A tinyml approach to human activity recognition," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 2022.

[46] E. M. Tapia, S. S. Intille, W. Haskell, K. Larson, J. Wright, A. King, and R. Friedman, "Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor," *Journal of Medical Internet Research*, 2007.

[47] R. Yauri and R. Espino, "Edge device for movement pattern classification using neural network algorithms," *IEEE Access*, 2022.

[48] (2023) Slimevr full-body tracker. [Online]. Available: https://www.crowdsupply.com/slimevr/slimevr-full-body-tracker

[49] S. Studio, "Seeed studio xiao nrf52840 sense edge impulse getting started," https://wiki.seeedstudio.com/XIAOEI/, 2023.

[50] ——, "Xiao ble motion recognition," https://files.seeedstudio.com/wiki/XIAO-BLE-Motion-Recognition/XIAOEI.ino, 2012.

[51] Arduino, "Oled display code source," https://create.arduino.cc/editor/PLAY-ZONE/07da3ca0-4ead-4fbf-a88e-e1723135279a/preview, 2024.

[52] A. Ignatov, "Real-time human activity recognition from accelerometer data using convolutional neural networks," 2017.

[53] N. Twomey, T. Diethe, X. Fafoutis, A. Elsts, R. McConville, P. Flach, and I. Craddock, "A comprehensive study of activity recognition using accelerometers," *Informatics*, vol. 5, 5 2018.

[54] S. J. Preece, J. Y. Goulermas, L. P. Kenney, D. Howard, K. Meijer, and R. Crompton, "Activity identification using body-mounted sensors - a review of classification techniques," 2009.

[55] E. Lattanzi, M. Donati, and V. Freschi, "Exploring artificial neural networks efficiency in tiny wearable devices for human activity recognition," *Sensors*, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/7/2637

[56] M. K.R.Baskaran, "Machine learning algorithms for human activity recognition," *International Journal of Engineering & Technology*, 2019.

[57] Keras, "Keras API," https://keras.io/, 2024.

[58] G. AI, "Tensorflow lite - deploy machine learning models on mobile and edge devices," https://www.tensorflow.org/lite, 2012.

[59] E. Impulse, "Edge impulse website," https://docs.edgeimpulse.com/, 2023.

[60] GoogleAPI, "Google api for post-training quantization," https://ai.google.dev/edge/litert/models/post_training_quantization, 2023.

[61] OutSystems, "Outsystems website," https://www.outsystems.com/, 2023.