Sensor based Human Activity Recognition using Edge Computing and TinyML in combination with Convolutional Neural Networks

Master Thesis

Submitted in partial fulfilment of the requirements for the degree of:

Master of Science in Engineering

to the University of Applied Sciences FH Campus Wien

Master Degree Program: Software Design and Engineering

Author:

Franz Julius Gruber, BSc

Student Identification Number:

c1910838014

Supervisor:

Dipl.-Ing. Heimo Hirner

Date:

01.06.2023

Declaration of authorship:

I declare that this Master Thesis has been written by myself. I have not used any other than the listed sources, nor have I received any unauthorized help.\\\\

I hereby certify that I have not submitted this Master Thesis in any form (to a reviewer for assessment) either in Austria or abroad.\\\\

Furthermore, I assure that the (printed and electronic) copies I have submitted are identical.

Date: .... Signature:
....

Kurzfassung

In dieser Masterarbeit wird ein Ansatz zur Erkennung menschlicher Aktivitäten vorgestellt, welcher Edge Computing und Tiny Machine Learning (TinyML) mit Convolutional Neural Networks (CNNs) kombiniert. Das vorgeschlagene System zielt darauf ab, eine hohe Genauigkeit bei der Erkennung menschlicher Aktivitäten in Echtzeit zu erreichen und gleichzeitig den Stromverbrauch und die Datenübertragung zu minimieren. Die schnelle und sichere Auswertung von Sensordaten spielt im Bereich des Edge Computing eine wichtige Rolle, insbesondere wenn viele verschiedene technische Komponenten in einem Prozess verankert sind. Diese Arbeit konzentriert sich dabei auf die Erkennung menschlicher Aktivitäten auf der Basis von Beschleunigungssensor-Daten.

Die Verschachtelung von KI, maschinellem Lernen, Deep Learning und CNNs wird aufgezeigt, um die erste Forschungsfrage zu beantworten. Darüber hinaus wurde eine praktische Anwendung eines CNN-Modells implementiert, um die wesentlichen Schichten und Schritte zum Aufbau und Training eines solchen Modells aufzuzeigen.

Zur Beantwortung der zweiten Forschungsfrage wurde eine Architektur gewählt, um eine praktische Lösung auf Basis von Edge-Technologien zu implementieren. Ziel der Implementierung ist es, die aufgenommenen Daten zu verarbeiten, ohne in weiterer Folge eine ständige Internetverbindung oder die Cloud nutzen zu müssen. Zur anschließenden Darstellung der gewonnenen Resultate wurde eine mobile Anwendung entwickelt.

Im Rahmen der dritten Forschungsfrage werden Hürden und Schwierigkeiten bei der Erfassung von Beschleunigungssensor-Daten und der Erstellung neuronaler Netze diskutiert.

Das Ergebnis ist ein theoretischer Überblick sowie ein praktischer Ansatz für die Verarbeitung von Beschleunigungssensor-Daten mit Deep-Learning-Algorithmen. Durch die schnelllebige Entwicklung ergeben sich immer wieder neue Aufgaben. Aus diesem Grund ist es wichtig, sich mit den Herausforderungen dieser Innovationen auseinanderzusetzen. Um diesen bedeuteten Aspekt hervorzuheben, werden die Themen dieser Arbeit sowohl mit positiven als auch mit negativen Ansichten beleuchtet.

Abstract

This master thesis introduces an approach to human activity recognition (HAR) that combines edge computing and tiny machine learning (TinyML) with convolutional neural networks (CNNs). The proposed system aims to achieve high accuracy in real-time human activity recognition while minimizing power consumption and data transmission. The fast and secure evaluation of sensor data play an important role in the field of edge computing, especially when many different technical components are anchored in one process. This work focuses on human activity detection based on accelerometer data.

The nesting of AI, machine learning, deep learning and CNNs is pointed out to answer the first research question. Moreover, a practical application of a CNN model was implemented to show the essential layers and steps to build and train such a model.

An architecture was chosen to implement a practical solution based on edge technologies to answer the second research question. The goal of the implementation is to process the recorded data without the need of using a constant internet connection or the cloud in further consequence. To subsequently display the results obtained, a mobile application was designed.

As part of the third research question hurdles and difficulties in collecting accelerometer data and creating neural networks are discussed.

The outcome is a theoretical overview as well as practical approach for the processing of accelerometer data with deep learning algorithms. Due to the fast-moving development, there are always new tasks arising. For this reason, it is important to deal with the challenges of these innovations. In order to emphasize this significant aspect, the topics of this work are highlighted with both positive and negative views.

List of Abbreviations

BLE	Bluetooth Low Energy
CNN	Convolutional Neural Network
ConvNet	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DLN	Deep Learning Network
DNN	Deep Neural Network
GPIO	General Purpose Input/Output
GPU	Graphical Processing Unit
HAL	Hardware Abstraction Layer
HAR	Human Activity Recognition
MACC	Multiply-and-Accumulate Complexity
ML	Machine Learning
mW	Milliwatt
NN	Neural Network
OS	Operating System
RAM	Random Access Memory
ReLu	Rectified Linear Unit
RNN	Recurrent Neural Network
RTOS	Real Time Operating System
SPI	Serial Peripheral Interface
TFLite	TensorFlow Lite
TinyML	Tiny Machine Learning
TPU	Tensor Processing Unit
UUID	Universally Unique Identifier

Key Terms

Convolutional Neural Network

Deep Learning

Human Activity Recognition

TinyML

Contents

1. Introduction

The current life situation of people is changing to a faster and tightly planned everyday life. This means that people are often stressed and need a compensation to fulfil their planned tasks. Exercise and a healthy diet are two common ways to achieve this balance. [1]

With first steps in this direction many companies and start-ups try to design and adapt their existing or new products to support people to cope their daily lives. [1]

In recent years, there has been a significant shift towards edge computing, which involves processing data on devices at the edge of a network, rather than relying on cloud-based services. This shift has been enabled by advances in embedded and mobile systems, as well as architectural improvements that make it possible to run machine learning and artificial intelligence algorithms on low-power devices. [2]

One area where edge computing has shown particular promise is in sensor-based human activity recognition (HAR). By leveraging sensors such as accelerometers and gyroscopes, it is possible to accurately recognize human activities in real-time, which allows important applications in the field of healthcare, sports, and other areas. [3]

One of the key technologies powering these advances is convolutional neural networks (CNNs). These deep learning algorithms are particularly well-suited to image and signal processing tasks, making them ideal for processing the data generated by sensors in HAR systems. [4]

To sum up, the combination of machine learning at the edge, artificial intelligence, embedded and mobile systems, architectural improvements, sensor based HAR, and convolutional neural networks has the potential to transform a broad spectrum of industries and applications, from smart cities to healthcare and industrial automation.

1.1. Motivation

The aim of this master thesis is to show that technologies without vast computer system resources can be used to analyze the motion output of fitness exercises.

The intended audience consists of individuals with a technical background which lead an active lifestyle, prioritize a healthy well-being, and maintain physical fitness through exercise routines. Generally, experienced software developers who work with sensor data and deep learning are a crucial audience for this thesis, as they can benefit greatly from the insights and solutions presented. Subsequently they contribute to the ongoing development. The results will be used to give an overview of technologies and existing theoretical as well as practical implementations in these fields.

1.2. Research Questions

1. What are the technical differences between machine learning and deep learning and how do these differences impact their performance in image recognition tasks?
    a. What are the key components of a convolutional neural network, and how do they contribute to its performance in image recognition tasks?
    b. How are convolutional neural networks trained, and what are some common techniques used in this process?
2. How can edge devices and edge computing technologies be combined with deep learning to improve human activity recognition in modern software architectures?
3. What are the specific difficulties of creating a convolutional neural network for human activity recognition, and how do these difficulties impact the accuracy of the network?

1.3. Methodology

To answer the research questions a qualitative analysis of existing theoretical approaches as well as a practical solution is developed that shows an implementation approach with the help of frameworks and tools. The practical result returns reliable outcomes without on-demand availability of high computing power. [2] Due to the necessity of the intensive use of time to determine expressive outcomes of convolutional neural networks already existent dataset were used to train the CNN model. [5]

Furthermore, a conceptional design of an application was outlined, from which a functional result was created by using frameworks and tools. The implementation includes a microcontroller project, an iOS app with the possibility to store collected data without permanent internet connection. The third part of the practical part includes the most important techniques to build and train a CNN model.

The project for the microcontroller unit (MCU), in this case the SensorTile kit, was built with the STM32CubeIDE, the STM32CubeMX and STM32Cube Expansion Packs. [6][7][8][9]

1.4. Goals

The master thesis shows modern technologies and architectures that can support the daily life. This thesis points out techniques and projects that analyze motion data with convolutional neural networks. In conclusion an approach to capture sensor data with the help of a microcontroller and in the next step to return usable results from deep learning algorithms is provided.

Furthermore, architectural improvements of edge technologies and deep learning networks are discussed. Additionally, challenges of convolutional neural networks and hurdles from collecting sensor-based human activity recognition data are considered.

To show an approach of an edge computing architecture the implementation part in chapter 4 shows a quick way to connect technologies like microcontrollers, mobile applications, and most importantly deep learning algorithms.

2. Technical Fundamentals

The chapter on Technical Fundamentals provides a comprehensive overview of the essential concepts and technologies that underpin edge computing (EC), machine learning (ML), and artificial intelligence (AI).

The first topic covered in the chapter 2.1. is edge computing and tiny machine learning (TinyML). EC refers to the practice of processing data at the edge of a network, closer to the source of the data, rather than in a centralized data center. This approach allows for faster processing and lower latency, which is especially important in applications that require real-time decision-making. [10] On the other hand, tiny machine learning describes the application of machine learning algorithms on low-power gadgets like microcontrollers equipped with sensors, where memory and processing capacity are constrained. Together, edge computing and tiny machine learning are enabling the development of intelligent and responsive systems that can operate in real-time on low-power devices. [11]

Artificial intelligence and convolutional neural networks are described in section 2.2. AI is the field of study concerned with creating intelligent machines that can perform tasks that typically require human intelligence, such as visual perception, speech recognition, decision-making, and language translation. [12] Convolutional neural networks (CNNs) are a class of deep learning (DL) algorithms that are widely used for image and video recognition tasks. Knowing the fundamentals of machine learning, such as the building and training of neural networks, is essential to understand how AI and CNNs are nested. [13]

The next subsection 2.3. shows the selection of technologies for processing data with AI and displaying the results. This involves choosing the appropriate data storage, processing, and visualization technologies, as well as frameworks and libraries for implementing AI algorithms. Some of the commonly used technologies in this domain include databases, data processing frameworks, machine learning frameworks, and visualization tools. [14]

Chapter 2.5. relates to architectural improvements for energy and performance enhancements in edge computing and DL networks. As EC and DL applications become more widespread, there is a growing need for architectural improvements to enhance their energy efficiency and performance. Techniques such as model compression can be used to achieve these goals. These improvements enable the development of systems that can operate on low-power devices while still achieving high accuracy and responsiveness. [15]

By covering these four topics, the chapter on Technical Fundamentals provides a comprehensive overview of the essential concepts and technologies necessary for understanding and working with edge computing, machine learning, and AI.

2.1. Machine Learning at the Edge

In specific application areas such as autonomous vehicles, industrial control systems and healthcare, it is becoming increasingly important that the execution of processes takes place more confidently and without interruption. For this reason, the transition to small hardware pieces like microcontrollers and embedded systems advances. Another technology which emerges is Artificial Intelligence in the context of machine learning (ML). In the last decades, a present issue was that the processing power was insufficient to run such ML algorithms. Therefore, the solution was to build big server farms and in conclusion to outsource the execution of algorithms to the cloud. But this changed in the last few years as the processing power required to run ML algorithms is constantly declining. [2]

The decline in processing power required to run ML algorithms has been driven by advancements in both hardware and software. By enabling parallel processing, which is essential for effectively running large-scale ML algorithms, specialized hardware has enhanced processing capacity. Tensor processing units (TPUs) and graphics processing units (GPUs) are two examples. For instance a single GPU can perform thousands of mathematical operations simultaneously, allowing for faster and more efficient computation. [16]

In addition to hardware advancements, software optimization has also played a key role in reducing the computational requirements of ML algorithms. Researchers and engineers have been attempting to build and optimize current algorithms in order to increase the effectiveness and optimization of ML algorithms. Further techniques such as pruning and quantization which are described in subsection 2.5.2. are implemented to reduce the number of calculations needed. [17]

Since the number of smart devices communicating through the internet is constantly increasing the conventional combination of cloud computing and the Internet of Things (IoT) has gradually revealed a few disadvantages such as transmission delays, reliability and latency when using big amounts of data. To minimize these factors the common solution was to bring the computations near the source of the data which is called Edge Computing (EC). But the shift of services to the edge of the network is more an extension than a replacement for cloud computing. [18] To understand what needs to be done to use Edge Computing the next paragraphs show requirements for enabling edge computing:

Management of resources
The predicted service demands of its users are what drive the infrastructure expansion of particular Edge computing service providers. This may result in a reasonable allocation of resources to support a particular user base. Therefore, service providers pool their resources with other interested contributors who have higher service expectations when the demand for their own services is low. Offloading may occur more frequently as service demands rise, which would improve the optimization of variable computational and network resource allocation. To serve a large number of users on heterogeneous edge computing platforms with diverse processor, memory, and network capacities consequently requires effective collaborative resource management. [18]

Support for applications in real-time
By processing data closer to its source, edge computing lowers latency and speeds up response time. Real-time support at the edge enables applications to process data as it is created and enables prompt and informed decision-making. Real-time support also gives

edge devices the ability to react to shifting circumstances and giving the system flexibility and agility. Thus, real-time support is a crucial component of edge computing, enabling it to fulfil the requirements of modern applications. [19]

Expandable architectures

The demand for edge-based services and resources has increased as a result of the IoT device proliferation at an exponential rate in the area of edge computing (EC). Applications running on the edge are anticipated to perform reliably despite the rising load. A scalable EC architecture is thought to be essential for fulfilling this expectation because it can help reduce expenses. This architecture can be created by including a number of characteristics, such as automated orchestration of resources and their virtualization. Resource virtualization facilitates multi-application coexistence by installing each IoT application as a separate segment that shares the same sensing resources. [19]

2.1.1. Machine Learning Architectures

Over the past decade, machine learning architectures become popular due to the success of deep neural networks (DNN). Deep learning, represented by DNNs and their derivatives, convolutional neural networks (CNN) and recurrent neural networks (RNN) has become one of the most prominent AI techniques in recent years. [20]

DNNs are neural networks with multiple hidden layers. They are used for a variety of tasks such as classification, regression, and generation and are trained using backpropagation, where the errors at the output layer are propagated backwards through the network to update the weights. [21]

CNNs are a specialized type of neural network designed to process data with a grid-like structure, such as images or videos. With the help of convolutional layers, CNNs can recognize features like edges and textures. For classification or regression, these convolutional layer outputs are subsequently fed into fully connected layers. [22]

RNNs are designed to process sequential data, such as speech or text and maintain knowledge of prior inputs by using a feedback loop to transmit information from one step to the next. This makes RNNs well-suited for tasks such as language modeling, machine translation, and speech recognition. [23]

In a variety of fields, including speech recognition, natural language processing and computer vision, deep learning has led to remarkable advancements. Also, hardware architectures and platforms are improving always faster to make it possible to meet the demands of computationally intensive deep learning models. Application-specific accelerators are designed to further improve throughput and energy efficiency. In summary, breakthroughs in deep learning and upgrades in hardware architecture are driving continuous success and development in AI. [24]

In recent years, IoT has been attracting attention, and IoT devices are being used in various fields. Many of these implementations running on these devices are based on ML models to make significant decisions. Small hardware devices typically have limited resources, which narrow down their ability to run complex ML models such as DL. In contrast to that, connecting embedded systems to the cloud to transmit raw data and perform processing slows system responses, exposes confidential data, and increases the communication costs. To address these issues an innovative technology originated called Tiny Machine Learning (TinyML). [25]

TinyML allows to deploy DL models on edge devices with tight resource constraints like limited computing power, less memory, and a few milliwatts of power. After applying several techniques to compress the DL model and optimize inference, a pre-trained DL model can now be deployed on a microcontroller. Therefore, TinyML enables data analysis and interpretation locally on the device and returns results in real time. [11]

The practical part described in chapter 4 employs a combination of a microcontroller equipped with an accelerator sensor and an iPhone to transfer and display the result of the processed movement data. Microcontrollers with accelerator sensors are common types of edge devices that can benefit from TinyML and CNNs. By deploying CNN models on microcontrollers, processing can be performed locally, leading to faster and more efficient processing. Accelerator sensors, on the other hand, are small, low-power sensors commonly found in wearable devices and other edge devices. Further the gathered data can be used to perform real-time analysis of movement data, such as recognizing specific types of physical activity or gestures. [26]

2.1.2. Edge-Cloud Collaborative

Figure 1 gives an overview of IoT applications in combination with cloud machine learning (CloudML), edge machine learning (EdgeML) as well as TinyML. The dashed area contains an embedded device combined with TinyML running a CNN that is optimized to be used on a microcontroller unit (MCU) with sensors like an accelerator chip. [25] An additional technology that can be used to train models for traditional supervised learning problems is called EdgeML. It is an open-source machine learning library released by Microsoft and helps to load trained models on edge devices such as IoT devices to return accurate predictions without the need of an internet connection. [2]

Nevertheless, it is important to also mention CloudML as the processing of ML and DL algorithms in combination with low-power devices have a limited range of applications. Therefore, DL algorithms are partially deployed on the edge device and the remaining data is transmitted to the cloud for processing. But the benefits of the cloud are also affected by some disadvantages which must be considered. For example, sensitive data can be intercepted from intruders while transferred to the cloud. [25]

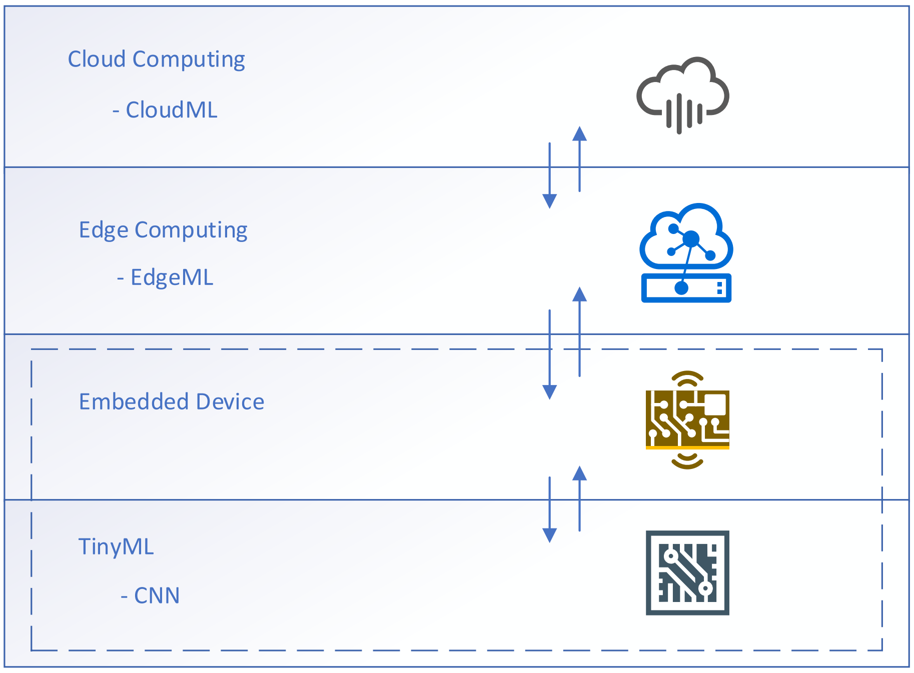


Figure 1: Combination of IoT applications with CloudML, EdgeML and TinyML

Deng et al. [20] state that the convergence of AI and edge computing is natural and inevitable and that they have in fact an interactive relationship. On the one hand, edge computing benefits from the technology of AI and its approaches, and edge computing can employ AI to increase its potential and scalability. Further Edge computing provides scenarios and platforms, and AI can expand its scope in edge computing. [20]

Another important term to consider is the Internet of Everything (IoE). The IoE is a structure of networks that brings people, processes, data, and things together. While the Internet of Things (IoT) is a dynamic global network infrastructure that manages physical devices that are accessed over the Internet, the IoE lays the foundation on top of IoT. In conclusion it brings together network connectivity and intelligent technology to support the creation of new capabilities, better skills and greater economic opportunities for business and society [27]

The IoE is important because in addition to gigantic cloud data centers, more data is being created by widely and geographically distributed mobile and IoT devices. Many other application scenarios, such as autonomous driving, smart homes, smart cities, and real-time data processing in public safety, facilitate the implementation of AI from theory to practice. In addition, AI applications with high communication quality and low computing power requirements can be migrated from the cloud to the edge. In short, edge computing provides AI with a diverse platform that includes a wide range of capabilities. [28]

2.1.3. TinyML

Combining edge computing with the Internet of Things (IoT) offers new ways to enable implicit machine learning techniques on resource-constrained embedded devices at the edge of the network. Traditional machine learning requires tremendous power to predict scenarios. The TinyML paradigm aims to move such richness from traditional high-end systems to low-end clients. Several challenges must be overcome during such a migration. These include preserving the accuracy of learning models, providing capabilities from training to deployment on small, resource-efficient edge devices and optimizing processing power. [29]

Machine learning devices today are hosted in both public clouds and private premises. Organizations use out-of-the-box models from various learning cloud services for many industrial applications. Reliance on such cloud-based machine learning services poses few challenges such as huge power consumption, privacy issues, network processing delays and reliability issues. [30]

TinyML is tackling these issues by processing raw data or data from sensors directly at the microcontroller unit (MCU). This improves the task of proceeding information by raising the efficiency in power and by minimizing the connectivity reliance. [26] According to Ray "such systems can make decisions on embedded edge devices before seeking help from edge AI or cloud AI." [29]

There are several software solutions and libraries that build the basis of TinyML. The mostly known open-source deep learning framework is called TensorFlow Lite (TFLite) [1].

TFLite for Microcontrollers (or TFLite Micro) [2] is a TFLite derivative project. Originally intended to provide deep neural networks on smartphones, it is now available on microcontrollers where deep learning networks can be deployed manually assisted after being created and trained using Keras. [31] Keras is described in more detail in chapter 2.1.4.

NanoEdge AI Studio [3], formerly called cartesiam.ai, has the functionality to select a library, evaluate its performance using an emulator, and then deploy it to the edge. It has many key features such as limiting maximum flash memory usage when creating a project, frequency filtering, flash memory optimization, serial data plotting, real-time search, and library selection with benchmarking. Further it can be used to detect anomalies in datasets and classification tasks and is compatible with STM32 Nucleo-32 boards. [30]

The third software collection that is also used in the practical part of this thesis has the designation STM32Cube.AI [4]. This code generation and optimization software facilitates machine learning and AI related tasks for STM32 ARM Cortex based boards. Neural networks can be easily implemented on STM32 boards by translating neural networks into efficient code for the MCU of choice. It is possible to optimize memory usage at runtime and to use models trained using traditional tools such as TFLite. [29]

There are many more frameworks that form the foundation of TinyML. However, these will not be discussed in detail in this thesis. The following subparts should summarize and show

[1] https://www.tensorflow.org/lite

[2] https://github.com/tensorflow/tflite-micro

[3] https://www.st.com/en/development-tools/nanoedgeaistudio

[4] https://www.st.com/en/embedded-software/x-cube-ai

the key advantages of TinyML in connection with IoT devices such as microcontrollers and microprocessors.

Power consumption
Low energy demand is a huge advantage when deploying TinyML running on MCUs. Because IoT devices running on MCUs rely on batteries and energy harvesting, they consume less power compared to powerful processors and graphics processing units (GPUs) that require a lot of power. Therefore, IoT devices can be placed anywhere without the need to connect to the power grid. Hence, it opens the door to new cognitive applications. In addition, low power consumption allows IoT devices to be combined with larger battery-powered devices to transform them into movable smart devices. One prominent example in the field of personal mobility is the E-Scooter or E-Skateboard. [30]

Cost factor
IoT devices without the ability to process AI directly on edge have played a variety of roles in several IoT applications. Additionally, there was often the need of a large amount of cloud storage space to perform processing and store data. This extensive usage of resources led to high expenses. [25]

In order to eliminate these disadvantages TinyML can process data locally on inexpensive microcontroller devices and apply AI locally on the device itself for high performance data processing. Microcontrollers are less expensive than other alternatives due to their low resource consumption and cost significantly less than other smart IoT devices used to process data locally utilizing DL models. [25]

Response time
In TinyML, computations are performed on the device, so data processing is done locally on the device. Therefore, IoT devices do not suffer from latency. Real-time processing of data on-site within the device enables rapid response and rapid analysis in emergency scenarios. Also, the dependency of the cloud is loosened. [31]

Reliability and security
To transmit raw data from IoT devices to the cloud, a communication channel is required. When data is transferred to the cloud, it is prone to transfer errors and cyberattacks. As a result, transmitted data may be compromised or lost. Consequently, limiting cloud traffic requires processing data locally. TinyML prevents these problems by doing data processing locally on the same device. Another positive aspect is that the forwarding of aggregated or meaningless data to attacks is reduced. [11]

Hardware requirements
To find out if a microcontroller is suitable for handling TinyML applications, there is the need to evaluate it in terms of processor, CPU clock speed, flash memory, RAM size, voltage consumption, connectivity and sensors or connectors. Ray [29] states that most hardware boards with processor frequencies below 100MHz run on average with less than 1MB of Flash and less than 1MB of RAM. Bluetooth Low Energy (BLE) is there the connectivity

technology of choice. Most boards support a variety of onboard sensors such as accelerometers, temperature, humidity, microphones, gyroscopes, and barometric pressure. The power consumption of such boards is in the mW range. Most devices are powered by Li-Po and coin batteries in addition to regular DC power. Furthermore, the ARM Cortex-M4 is the most widely used processor of all available alternatives. [29]

2.1.4. Keras

Keras is a deep learning API written in Python that works on top of the TensorFlow machine learning framework. The goal of the framework is to allow quick experimentation in the field of deep learning. The primary data structure of Keras are layers and models. [32]

Engineers and researchers can take advantage of TensorFlow's scalability and cross-platform capabilities with Keras. It is possible to run Keras on tensor processing units (TPU) or on large clusters of graphics processing units (GPUs). The big advantage is that Keras models can be exported for use in the browser or on embedded devices. [32]

The Keras API is also used for the model that is integrated in the practical implementation. The model is described in more detail in subsection 4.10.2.

2.2. Artificial Intelligence

Machine intelligence is another phrase for artificial intelligence (AI). Furter the designation intelligent agents is often used to describe the study of AI in computer science. In other words, an intelligent agent might be any technology that perceives its environment and takes activities to improve its chances of achieving some predefined goals. [33]

In common language, artificial intelligence is when a computer imitates cognitive functions that humans identify with other human minds, such as learning and problem solving. Self-driving cars and interpreting complex data are among the capabilities currently categorized as AI. Other capabilities include successfully understanding of human speech or competing at an elevated level in strategic game systems. [33]

In Figure 2 the correlation between AI, machine learning (ML) and deep learning (DL) and neural networks (NN) is illustrated. The terms ML, DL and NN are described in the chapters 2.2.1. , 2.2.2. and 2.2.3.

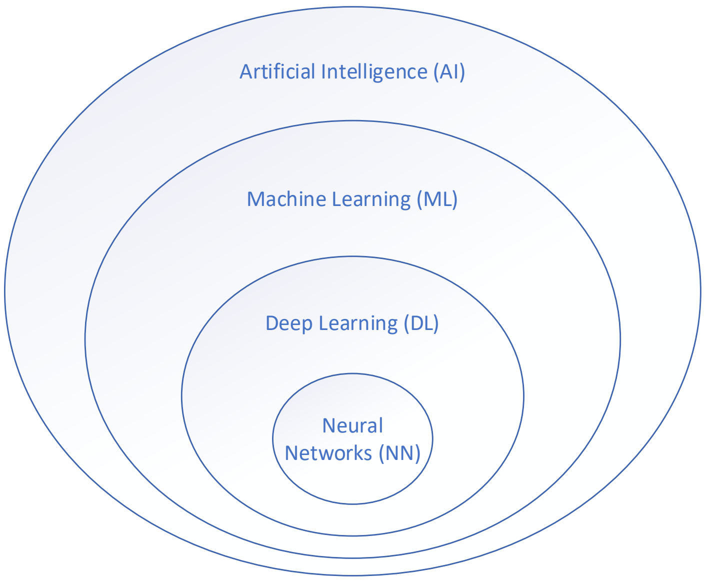


Figure 2: Correlation between AI, ML, DL and NN

2.2.1. Machine Learning

Pariwat states that machine learning is a branch of computer science that, according to Arthur Samuel in 1959 “allows computers to learn without being explicitly programmed." [33] Developed from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning examines the study and construction of algorithms that can learn from data and make predictions. Such algorithms are achieved by being data-driven rather than strictly following static program instructions. [33]

2.2.1.1. Machine learning algorithms

Machine learning algorithms fall into four main categories: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. [34] Below, the types of learning techniques are outlined.

Supervised learning
Supervised learning involves using input/output pairs to learn a function that maps inputs to outputs. This is done by deriving a function using a set of labeled training data and training samples. The basic purpose of supervised learning is to achieve a certain goal using a specified set of inputs. Two commonly monitored tasks in supervised learning are classification, which involves splitting data, and regression which affects fitting data. Examples of supervised learning include predicting the label or main idea behind a text based on forum posts or reviews. [34]

Unsupervised learning
Unsupervised learning occurs when a learning system is designed to recognize patterns without prior labels or specifications. Therefore, the training data consists only of a specific term, and the algorithm aims to find interesting structural information. This could be for example a group of elements with common properties, or a data representation projected from one high-dimensional space onto a low-dimensional space. [4] Janiesch et al. indicate that "a prominent example of unsupervised learning in electronic markets is applying clustering techniques to group customers or markets into segments for the purpose of a more target-group specific communication." [4]

Semi-supervised learning
Semi-supervised learning serves the same purposes as supervised learning. Since obtaining unlabeled data is less expensive, it is common to use both a sizeable amount of unlabeled data and a modest amount of labeled data for the training. This learning style can be combined with methods such as classification, regression, and prediction. In conclusion, semi-supervised learning is helpful when the procedure of labeling is too high for an entirely labeled training approach. Early examples include recognizing a face of a person on a web cam. [33]

Reinforcement learning
Reinforcement learning is a type of machine learning algorithm that allows software agents and machines to automatically evaluate optimal behavior in a specific context or environment to improve efficiency. The purpose of this sort of learning, which is focused on rewards or penalties, is to use insights to take actions that will raise rewards or lower risks. Further, it is a strong tool for building AI models that can help automate or improve the operational effectiveness of complex systems like autonomous driving, robotics, and supply chain management. However, it is not advised for resolving trivial or obvious issues. [34]

2.2.1.2. Real world data

Data availability is usually seen as the key to building machine learning. The differentiation of this data is important since it influences the output of algorithms. Therefore, it is separated into following categories: [34]

- Structured data
- Semi-Structured data
- Unstructured
- Metadata

Structured data
As its name implies, structured data is organized and follows a common ordered data model and is additionally easily accessible. A prominent example of a well-defined schema are relational databases that store data in tabular form. [34]

Unstructured data
Unstructured data has no established structure or format and mostly contains text and multimedia material. This fact is making it much more difficult to collect, process and analyze these kinds of data. Examples of unstructured data include sensor data, audio files, videos, and images. [35]

Semi-structured data
Data that hasn't been formatted in a conventional manner is referred to as semi-structured data. It has no fixed schema like following the format of relational databases. However, there are some organized components in the data, such as tags and organizational metadata, which make analysis easier. This indicates that the data is not completely unstructured or raw. [35]

Metadata
Information that has some meaning in relation to other information and makes it more descriptive is called Metadata. For example, metadata of a document could be expressions like author, file size as well as keywords that define the document. [36]

2.2.2. Deep Learning

Deep learning is a part of artificial neural network (ANN)-based machine learning algorithms with supervised learning. By combining multiple processing layers, deep learning creates a computer architecture that learns from data by utilizing input, hidden, and output layers. Deep learning has a notable benefit over traditional machine learning techniques in some cases, particularly when learning from large data sets. [34]

Deep learning networks (DLN) have gained great popularity in recent years because they can classify image information and recognize objects from image data. Computer vision applications using DLNs, especially convolutional neural networks (CNNs), include object tracking, action recognition, and object counting. [37] A detailed overview about CNNS is shown in chapter 2.2.3.

To better visualize deep learning, it is important to understand that there is always a set of signals which serve as input and a another one which provides the output. When an input layer receives an input, it forwards a modified version of this initial information to the next layer. In deep networks, there are many layers between inputs and outputs, so algorithms can use various processing layers with numerous linear and nonlinear alterations. [33] To better understand the layers in DLN they are listed and explained in chapter 2.2.3.1.

As seen in Figure 3, traditional ML techniques require several sequential steps to complete a classification task, including preprocessing, feature extraction, feature selection, training, and classification. Resulting from this, feature selection significantly affects how well machine learning approaches operate. Additionally, biased feature selection can result in incorrect class discrimination. In contrast to traditional ML techniques, DL can automate feature set learning for multiple tasks and allows training and classification in a single step. [38]

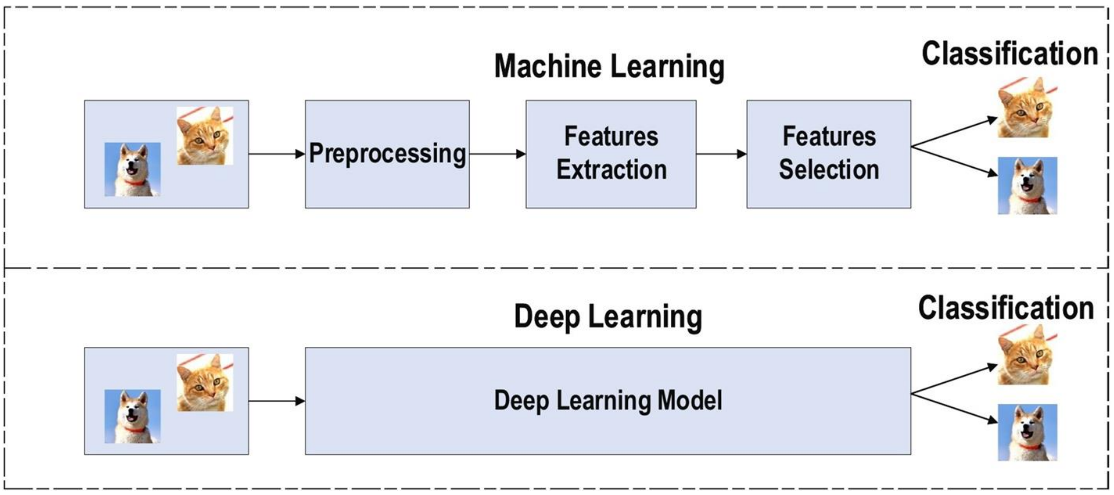


Figure 3: Difference between ML and DL classification steps [38] ©2021 Alzubaidi et al.

2.2.3. Convolutional Neural Networks

A neural network is based on the neuron, the most fundamental unit of the human brain. Further deep learning is a term used to describe the study of how neurons collaborate to form a neural network model. In conclusion, a neural network's output is a deep learning model. Commonly, unstructured data is used to train a neural network model. It draws characteristics on its own by getting trained persistently with data. [39]

Speech recognition and computer vision problems are the main applications for convolutional neural networks. They can solve problems with datasets with spatial relationships where columns and rows are not interchangeable. One prominent example is image data. The network architecture of CNNs contains many stages that allow hierarchical learning of features determined by a specific modeling task. For instance, looking at object detection in images, the first layer of the network is in charge of extracting basic features in the form of edges and corners. These are gradually aggregated in the final layer into more complex features that resemble real-world objects such as people, houses, or animals. Additionally, automatically generated features are used for prediction purposes to identify objects of interest in new images. [4]

As seen in Figure 4, the basic architecture of a CNN is built up from various layers that transform input data to a specific output layout. A CNN is usually structured in an input layer, a feature extraction part, and a classification part which are described in more detail in the chapters 3.3. and 3.4. In the end the output layer returns predefined output classes that show the result. [37]

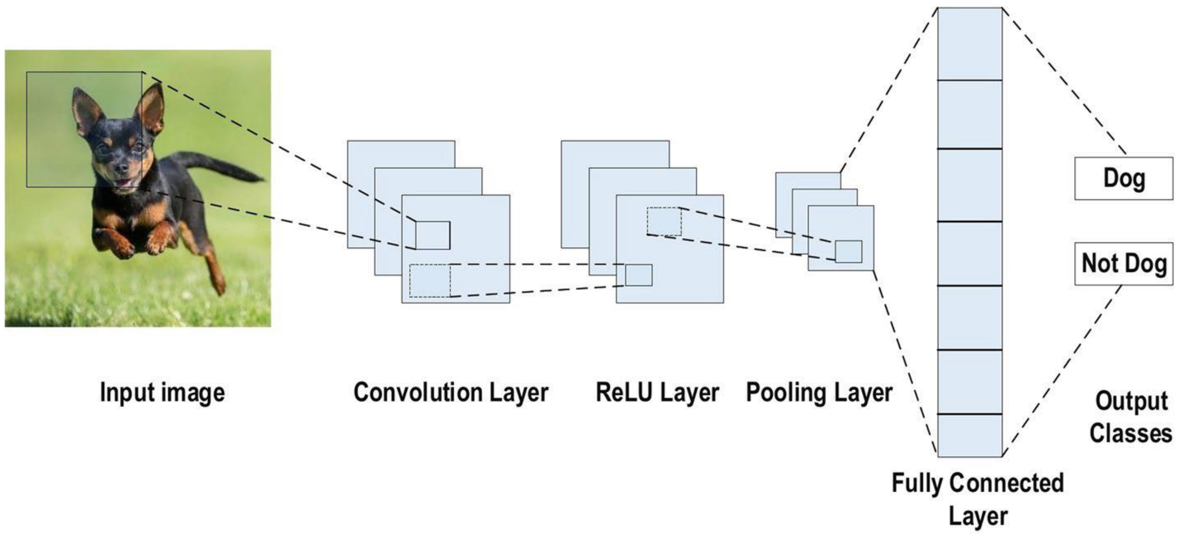


Figure 4: Presentation of different layers of a CNN [38] ©2021 Alzubaidi et al.

A major advantage of CNNs over their predecessors is that they automatically identify relevant features without human supervision. The structure of CNNs is inspired by neurons found in the human and animal brains. To simplify the training process and to speed up the interchange between layers shared weights and local connections are used to take full advantage of 2D input data structures such as image signals. [38]

A CNN is a network of neurons connected in an acyclic graph, with some hidden layer neurons connected to a subset of the neurons in the previous layer. This placement is done to train the network implicitly. CNNs are a subset of deep forward coupled artificial neural networks employed in the processing of visual images. Like other artificial neural networks, it has neurons with learnable weights and biases. But not all neurons in a layer are connected to some in the previous layer. Neurons take inputs and the sums of weights to pass them to the activation functions. At the end, the desired output is produced. [40]

The different layers and their underlying functions are described in more detail in the next chapters 2.2.3.1. and 2.2.3.2. Further the different types of real-world data are explained in section 2.2.1.2.

2.2.3.1. Layers

To better understand the need of the different layers of CNNs they are listed in the next paragraphs.

Convolutional layer

It is the base layer of a CNN that contains the entire compute part. A parameter is a learnable filter or kernel. As the meaning of the expression convolve implies, this means that several functions and activation functions are combined. When it is assumed that the input image is 32x32 pixels and it is passed to the next layer with 3 neurons, a 32x32x3 connection needs to be created. [40]

ReLU layer
The ReLU (Rectified Linear Unit) layer is a fundamental component in CNNs. Its basic goal is to provide a digression to the model, which allows the network to learn complex patterns and features. In simple terms, a ReLU layer can be thought of as a filter that lets through only positive values while setting negative values to zero. This layer is widely used in CNNs due to its simplicity and computational efficiency. [38]

Pooling layer
The main task of the pooling layer is under sampling of feature maps. This approach shrinks a large feature map to create a smaller feature map. At the same time, most of the key information is kept at each step of the pooling phase. Like convolution operations, the kernels are initially assigned a size before the pooling operation is performed. Several types of pooling methods are provided that can be used with different pooling layers. These methods consist of average pooling, minimum pooling, maximum pooling and many more. [38]

Fully connected layer
According to its name, the neurons in this layer are completely interconnected with those in the layer before. It is not possible that there is a convolutional layer after the fully connected layer because the neurons are not spatially arranged, and every neuron needs to be connected to the previous neuron. [40]

2.2.3.2. Functions

Loss functions
The previous section introduced different layer types for CNN architecture. Furthermore, the final classification comes from the last layer of the CNN architecture, the output layer. Several loss functions are used in the output layer to compute the prediction error produced across the training patterns of the CNN model. Further the inconsistency between the actual output and the expected output is shown by this error. [41]

Activation functions
Nonlinear activation layers are used after all layers with weights such as fully connected layers and convolutional layers in CNN architectures. In addition, these layers give CNNs the opportunity to gain experience particularly complex things. The following types of activation functions are most used in CNNs: [38]

- **Sigmoid** is a logistic regression function that compresses a number between -1 and +1 and returns the probability that the output is positive.
- **Tanh** flattens the output between -1 and +1.
- **ReLU** functions are primarily used as basic functions for neural networks. Returns 0 if the output is negative and the exact number if greater than zero.
- **Softmax** is frequently applied to normalize the output to a probability distribution over the expected output class.

2.2.4. Challenges of CNNs

In this chapter the main challenges and limitations of convolutional neural networks (CNN) are discussed. An important aspect to keep in mind when using a CNN is that the input and output layers are always built on a specific field of application. As example, if the output classes of the CNN are connected to the correct execution of a fitness exercise there is the need that each exercise has its own CNN model. To summarize the use of CNNs are always limited to a certain extent. [38]

Furthermore, the convolutional neural network is reliable on the quality of the training data which was used for the model creation. Concluding from this having a meaningful and qualitative dataset often represents a major hurdle of designing an own CNN. [42]

As Alzubaidi et al. [38] state there are various difficulties when it comes to deep learning algorithms. One of the factors affecting the result is the training data. To create a useful performance model, deep learning requires a large amount of data. Further the data that is recorded is often imbalanced. This is because negative samples often outnumber positive samples in biological data. Here it is important to note that training a DL model with unbalanced data may yield undesirable results. [38]

Vision networks are frequently used for post-classification of detections, which involves analyzing a tiny portion of an image that contains an object and some context. It is the duty to ascertain whether the patch's center matches to an object and, if it does, to classify the object. The dilemma of how to handle inputs with lower resolutions efficiently arises because objects are frequently tiny and low-resolution. [42]

It is generally accepted that models with greater resolution receptive fields perform much better at recognition. However, it is critical to distinguish between the benefits of greater model capacity and processing and the influence of improved resolution in the first layer receptive field. We end up employing less computationally expensive models to address more difficult problems, but these solutions are probably less effective due to lower computing effort if we merely modify the input resolution without adjusting the model. [43]

The model must examine fuzzy inputs in order to make an appropriate judgment, which is computationally expensive. It is crucial to figure out how much performance can be improved by increasing input resolution while maintaining a constant processing effort. In the case of lower resolution input, reducing the strides of the first two layers or simply removing the network's initial pooling layer are two ways to maintain constant effort. [43]

In the following subsections challenges of CNNs based on different kinds of diverse fields are discussed.

2.2.4.1. IoT challenges

It is difficult to successfully implement DL in specific areas and obtain useful and trustworthy results. It necessitates domain knowledge, statistical analysis, problem solving, and data insight extraction abilities. [44]

One of the main issues in DL models that requires additional effort to solve is complexity. Due to the complexity of the models and the industrial data sets, two issues affecting the performance of DL models are the time-consuming training phase and the high computational effort in the inference phase. [45]

Next, Khalil et al. state that "IoT sensors and devices are designed to record data automatically." [44] But the same sensor doesn't always record the same data on the same instrument or device for a variety of reasons. These occasions range from inconsistent sensor calibrations to device age and further to environmental differences. Additionally, some sensor data may be lost due to poor connections as well as transmission and acquisition noise. [44]

Hybrid cloud/edge computing currently performs fast and efficient computations and provides intelligent computing infrastructure for IoT platforms. However, due to the overall complexity of the problem, as well as the limited storage and processing power of the devices involved, training-in-device processing is not feasible for dealing adequately with the complex-learning issue. To reduce latency and improve the learning process of the network, it is appropriate to use edge-based infrastructure for computing. Nevertheless, integrating cloud and edge-based computing infrastructure into IoT remains an open research challenge. [45]

2.2.4.2. TinyML challenges

TinyML has made notable breakthroughs in many use cases. Additionally, high accuracy using different DL models on resource limited IoT devices could be achieved. Despite TinyML's powerful perceptual capabilities, there are many limitations that hinder its development. [25] The paragraphs below list some of them.

Since embedded devices from different manufacturers usually have their own software stack it is difficult to execute accurate DL models inside of various microcontroller units. The solution to run deep learning algorithms like CNNs with low resources without a decrease in accuracy would be a generalized network. Until this has been found, there will always be difficulties in implementation. [46]

Next, TinyML systems are distinguished by their low power consumption. As a result, a useful benchmark should profile each device's energy efficiency. However, accurately measuring energy consumption presents numerous challenges. TinyML devices can consume vastly different amounts of energy, making consistency across devices difficult. [46]

One of the primary motivations for the development of TinyML was the limited memory of microcontrollers with only a few KB. Traditional DL model inference requires significantly more peak memory, than TinyML devices can provide. This presents a challenge and necessitates novel optimization techniques tailored to each algorithm. As a result of compressing DL models, it is critical that the parameter performance matches that of the original model. But in summary the number of research in this direction is increasing more and more and therefore these limitations are already being worked on. [25]

Dutta et al. [26] are proposing to exchange information over 5G, a key technological development that could transform device-to-device communication. It will be crucial in managing massive data transfers because of its capacity to handle big amounts of data with little delay. A hyperconnected world with all people and items connected is expected to be brought about by the introduction of 5G. Further the interconnection of DL models, TinyML and 5G are shown in Figure 5 in order to demonstrate how these three technologies interact to enable cutting-edge applications in a variety of industries. [26] 5G is described in more detail in subsection 2.4.2.

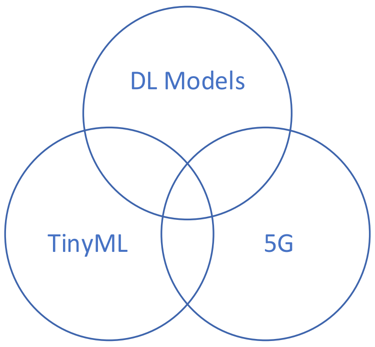


Figure 5: Interconnection of DL Models, TinyML and 5G

2.3. Embedded and Mobile Systems

This chapter shows technologies to process and in the final step to display the gathered information to a user of a mobile system. Related thereto the section 2.4. explores communication standards that permit communication between these systems.

2.3.1. Microcontroller

This section introduces the major hardware components that make up a microcontroller, or more generally a typical embedded system.

The central processing unit (CPU), which is usually housed in an MCU with memory or in other words random-access memory (RAM), is the key component. The CPU and the RAM are intricately linked, as the first component processes data from the second one. To make things easier and quicker, memory is organized hierarchically. That means that faster components are embedded directly into the CPU, and slower components move data to more rapid components. The data stored in memory typically comes from peripherals represented by sensors or transceiver modules for communication. [22]

The steps performed in an embedded application are represented by taking data from sensors, processing it through the MCU, and finally transmit the results via wireless communication such as Bluetooth, cellular networks, or Wi-Fi. This series of tasks is performed periodically, so there is the possibility to save energy by turning off critical electronics when they are not used at the time. [47]

2.3.2. iOS

The embedded device operating system iOS that is developed by Apple Inc. is used in the practical implementation part to connect to the microcontroller and to display the processed results to the user of the mobile application. [48]

It was initially designed for the iPhone and was later expanded to include other Apple devices such as the iPod touch, iPad, and Apple TV. The big advantage of the OS is the user centered design and the consistent performance on all devices of Apple. [49]

The user interface of iOS is called Cocoa Touch. Cocoa Touch includes a framework that allows developers to write an app for the iOS operating system. It defines aspects of the application and provides the fundamental application infrastructure, as well as support for multitasking, touch, and notifications. [48]

Objective-C or Swift is the primary development language. Swift is a compiled programming language developed and released by Apple in 2014 for the iOS operating system. Furthermore. Swift is designed to interact with the Cocoa Touch framework, as well as existing Objective-C code for Apple products. [49]

2.4. Communication Technologies

In the present digital era, communication technologies are playing a significant part of daily activities. To fulfill the growing demand for quick, dependable, and secure communication, new technologies are always being developed. These include 5G and Bluetooth Low Energy (BLE). While BLE is made for low-power devices and is frequently utilized for short-range communication, 5G, the latest generation of cellular network technology, offers high-speed, low-latency connectivity. [50]

2.4.1. Bluetooth Low Energy

Bluetooth low energy (BLE) [5] is a standard that enables energy-efficient wireless communication over short ranges. A physical layer is used that is not backward compatible with legacy Bluetooth devices. Developed for ultra-low power IoT applications, BLE reduces data rates to 1 Mbit/s and therefore decreases the maximum transmit power from 100 mW to 10 mW. [51]

A central aspect of Bluetooth and BLE is the concept of profiles and builds based on two profiles: [52]

- Generic Access Profile (GAP)
- Generic Attribute Profile (GATT)

[5] https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

Device discovery is done through GAP and the basic communication between devices is done through GATT. Because BLE is intended for low-data-rate applications, GATT provides a lightweight data access profile for retrieving data on remote devices. [51]

2.4.1.1. Generic Access Profile

Gadgets, like microcontroller that generate data from sensors, for example accelerator sensors, generally employ the broadcaster and peripheral role of GAP. The broadcaster role is selected if the amount of transmitted data fits within the payload of an advertisement package. If this is not the case and data should also be received, then the peripheral role is preferred. Further the four roles of the fundamental control unit of BLE are listed: [51]

- **Broadcaster** transmits data to its surroundings and is constantly advertising useful data.
- **Observer** examines the information from the advertising packets it obtains while passively listening to broadcasted information in its region.
- **Peripheral** is letting anybody know it has meaningful data and waits until someone decides to connect with it.
- **Central** is the active role and decides to which peripheral it connects to.

2.4.1.2. Generic Attribute Profile

The Generic Attribute Profile defines a framework of methods and formats that organizes and displays data on a BLE device. Once a connection is established between two BLE devices, GATT uses a simple client/server model to exchange data. The sending source becomes the server, and the receiving unit becomes the client. Data is requested by the client and returned by the server. Profiles define attributes to ensure a generic format for organizing data in GATT servers as shown in Table 1 and consist of four fields: [52]

- **Handle** is a server-assigned index relative to the attribute.
- **Type** is a 16-bit or 128-bit universally unique identifier (UUID) that exactly links to the attribute.
- **Value** holds data described by the attribute.
- **Permissions** is used to discover if the client can read or write to the attribute.

Handle	Type	Value	Permissions
2 Bytes	2 or 16 Bytes	Variable length	Variable

Table 1: GATT attribute structure of BLE

2.4.2. 5G

The newest development in telecoms standards, known as 5G [6], or fifth-generation mobile network technology, promises to completely change the connection from various embedded devices to the internet and one another. It is intended to provide internet speeds that are faster, lower in latency, and more dependable than those of its predecessors. This technology has the potential to revolutionize a variety of elements of daily life, including entertainment, transportation, and healthcare. [50]

Fast download and upload rates are made possible by the high data transmission rate of 5G, which can reach 10 gigabits per second (Gbps). High-quality video content may be easily streamed, huge files can be downloaded in a matter of seconds, and new applications like augmented and virtual reality are made possible by this fast connectivity. [53]

Data can be exchanged between devices more quickly because to 5G lower network latency than earlier generations. For applications that demand real-time communication, such as remote surgery, autonomous driving, and gaming, this decreased latency is particularly crucial. [26]

Further the energy-efficiency of the 5G standard is also intended to lessen the environmental impact of telecommunications infrastructure. Finally, 5G networks have built-in encryption and authentication measures that guard against cyber threats, making them more secure than earlier generations. [53]

2.5. Architectural Improvements

In this chapter architecture designs of edge computing and deep learning networks are discussed. Additionally, improvements of CNNs in connection with the consumption of power and memory are considered.

Gamatie et al. state that “no standard heterogeneous microcontroller-based architecture exists for edge computing”. [28] Heterogeneous computing typically refers to systems that include a variety of processing elements in order to meet both performance and power-efficiency goals. The need for heterogeneous architectures suitable for IoT devices has already been identified. Additionally, edge computing applications require such an architecture for power-efficient execution on compute nodes. According to recent research, the top microprocessor technologies and computing paradigms believed to meet the needs of IoT compute nodes. The most important computing paradigms are discussed in the following paragraphs. [28]

The first pattern includes configurable architectures that are designed for various applications to improve the efficiency of energy. The key is that these architectures fit for future applications without being over dimensioned for the current application. [28]

[6] https://www.etsi.org/technologies/mobile/5g

Next, the approximate computing and energy harvesting paradigm allows less precise results while maintaining an acceptable output quality. This concession results in significant performance and energy gains. Harvesting energy from available sources such as solar or radio frequency radiation allows batteries in ultra-low power nodes to be supplemented. Additionally, the archetype in-memory favors local processing of data collected directly on the node and minimizes therefore the off-chip communication. In conclusion, these paradigms make the processing of tasks on edge devices more efficient. [28]

Novac et al. state that “it is usually possible to find a good compromise between the precision and the performance of a given deep leaning network architecture.” [54] But finding the best architecture for a given problem is difficult and remains an open question even when hardware constraints are ignored. In fact, the combinatorial explosion of hyperparameters raises the cost of architectural research significantly. Because these networks are frequently tuned to solve computer vision problems, they are unsuitable for common use cases or simple problems. Therefore, a common approach is that many applications use generic networks. The reason for this is that it should be simple to apply the same deep neural network to different types of data while also simplifying the implementation. [54]

Additionally, there is the possibility of lowering the total amount of parameters in the neural network. This could be done by identifying network parts that are not useful for decision making. After that, these parts can be removed from the architecture. However, the network's operation is complicated by the unstructured nature of the removed parameters. Additionally, in some papers it was suggested that the removal of parameters should be done in a more structured way. One approach here is to remove complete layers according to certain criteria. All of these compression techniques allow for significant reductions in architecture size, but usually at the expense of performance. In practice, it is also common to cut the amount of memory used to store parameters in half or even more while maintaining comparable levels of performance. [54]

2.5.1. Edge AI

The vision of Edge AI essentially involves making it easier for data-driven applications to adapt, enhancing network access, and enabling the creation, optimization, and deployment of distributed AI/ML pipelines that adhere to strict quality and privacy standards. [55]

In service fields like real-time gaming, linked cars, smart factories, and healthcare, edge AI has made steady progress. Edge environments provide a different layer for AI from an infrastructure perspective while opening up possibilities for technologies like embedded AI or federated learning. These technologies concentrate on lowering communication requirements between distributed entities, limiting memory utilization on individual devices, and keeping data locally to improve privacy. [55]

Placement of computer power in close proximity to data sources is a key component of the edge AI idea. In general, any typical desktop or server rack may serve as an edge device, many environments are not ideal for them because of their size and energy requirements. Edge AI devices that are specifically designed to address these issues have been created. These gadgets are perfect for use in industrial settings because they are wirelessly connected and tiny. Low power consumption is essential when numerous devices are running simultaneously. As a result of the necessity for specialized mathematical abilities, accelerator modules for AI have also been developed. [15]

2.5.2. Deep Model Compression

Pruning and quantization, two well-liked methods for deep model compression, will be the main topics of this chapter. A DNN's effectiveness is increased, and its model size is decreased via pruning, which entails deleting any extraneous connections or neurons. Quantization, on the other hand, entails lowering the accuracy of a DNN's weights and activations, which further lowers the model's size and level of computational complexity. [17]

2.5.2.1. Pruning

There are numerous pruning techniques, each having advantages and disadvantages. Due to the fact that these techniques frequently require continual retraining, one of their major limitations is the significant amount of time needed to prune networks. Recently, efforts have been made to skip some of these processes by employing recurrent neural networks to prune neural networks while they are being trained. Nevertheless, pruning can significantly reduce the number of parameters in a network, often between 10% and 30% of the network's weights, independent of the approach employed. Importantly, trimming may accomplish this without materially reducing accuracy. [56]

Li et al. [57] state that the removal of filters that have little bearing on a network's final accuracy is an innovative pruning technique. The appropriate feature maps and kernels in the following layer are automatically removed using this technique. The significance of each filter in a layer is determined by adding their total weights, which produces an estimate of the size of the output feature map. The filters with the least absolute weight sums are trimmed after each iteration. [57]

2.5.2.2. Quantization

Similar to pruning, network quantization is a popular deep learning technique. Its main goal is to reduce the number of bits required to represent each weight, taking advantage of redundancy to cut down on the number of parameters. In order to reduce storage size without significantly affecting performance, quantization entails clustering the parameters of a neural network and setting them to the same value. [45]

3. Sensor based HAR and CNNs

This chapter provides an overview what human activity recognition is, and which data is used to interpret human activity. Further it describes methods which are used to get results from a convolutional neural network with the help of recorded motion data.

Human activity recognition (HAR) is a research topic that involves correctly identifying different activities sampled by different methods. In general sensor based HAR uses inertial sensors such as accelerometers and gyroscopes to sense body acceleration and angular velocity. [3]

As Czabke et al. [58] explains the most systems are gathering acceleration data by attaching one or more accelerometers to the subject's body in a given sensor orientation. In this setup the horizontal angle of particular body parts can be determined by using the gravitational acceleration. As an example, an upright chest would imply that a person is sitting and not moving, whereas a horizontal chest position would denote that a person is lying. [58]

Many applications for wearable technology and the Internet of Things (IoT) in industries like healthcare and sports emerged in the last years. The gathering of massive datasets has become routine due to smartphones and wearables' abilities to capture and interpret a wide variety of data. This has invoked possibilities for the creation of data-driven healthcare solutions based on monitoring. The goal of the classification issue known as human activity recognition is to anticipate an activity from a set of predetermined actions that are labeled for further use. In this sector, machine learning is frequently used to create generic predictive models by learning from this labeled data. One of these machine learning techniques, which is discussed in more detail in chapter 4.12. are CNNs. [59]

The focus of CNNs based on human activity recognition will use accelerator data to return output classes which for example describes the type of the exercise. In the next chapter 3.1. the various kinds of accelerometer data are listed. Next section 3.2. describes sensor factors that are influencing the outcomes of CNNs.

To better understand how features are selected and extracted in a CNN the subsections 3.4. and 3.3. explain their basic difference. Nevertheless, there are challenges while processing sensor based HAR data with CNNs and therefore chapter 3.5. shows objections and suggestions for improvement.

3.1. Accelerometer Data

Allahbakhshi et al. [60] state that among the existing wearable sensors, the accelerometer has received the most attention and is becoming increasingly popular among users.

Related to that there are three kinds of accelerometer data: [60]

- uni-axial (1D)
- dual-axial (2D)

- tri-axial (3D)

In this thesis 3D accelerometer data is used to detect the acceleration in different directions which are mapped on three perpendicular axes. The x axis illustrates the horizontal axis and is usually aligned with the motion from side-to-side, or in other word from left to right. Further, the y-axis indicates the vertical axis, where the sensor is shifted up and down. In contrast, the z-axis shows the depth which results in forward and backward movement. [61]

The popularity of accelerometer data is due to its wide range of applications and the valuable insights it provides. Additionally, accelerometers are essential to find out the location and orientation of small devices like microcontrollers. Moreover, features like screen rotation, tilt detection, and gesture-based controls are improving the user experience and give meaningful results to be used in application areas such as human activity recognition during sport activities. [62]

Figure 6 shows an example of a timeseries of tri-axial acceleration data. The blue line (x-axis), the orange line (y-axis) and the green line (z-axis) show the accelerometer output in the different directions measured in the unit g-forces (g).

g quantifies the acceleration of an object in relation to the acceleration of the gravity on earth's surface. Short, it is a force unit equal to gravitational acceleration, or 9.81 meters per second squared (m/s²). Furthermore, this unit is frequently used to represent the stresses and forces experienced by items or persons during rapid changes in velocity or direction. [63]

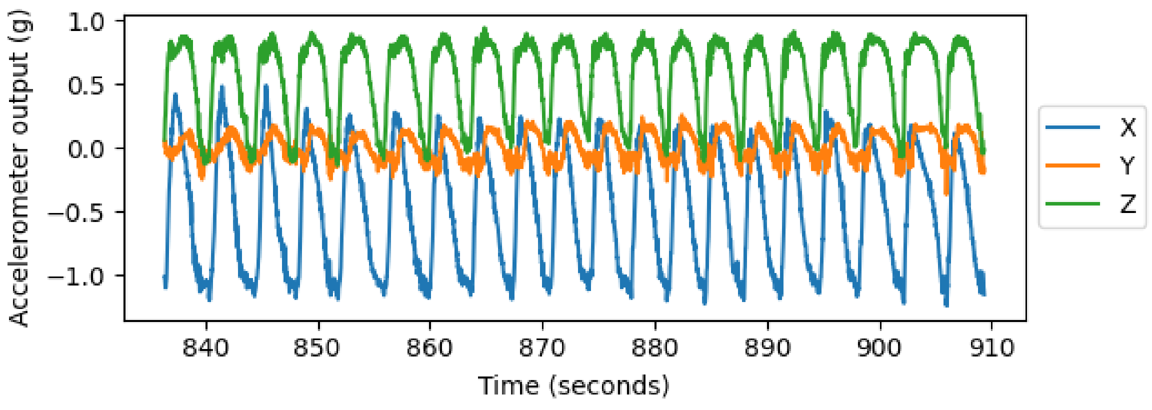


Figure 6: Example of recorded 3D accelerometer data at a sampling rate of 50Hz

3.2. Sensor Factors

The following chapter describes the most prominent factors that influence the outcome of CNNs by changing the recording techniques of movement data. Further these recorded values build the dataset that is used to train the CNN model.

Number of sensors
It is difficult to decide the number of used sensors to collect real-world data. Increasing the number of sensors expands the number of activities that can be classified and improves the classification performance. However, if multi-device configurations are required, data analysis can become complex and time-consuming. In comparison, a single accelerometer may not provide enough information to detect different types of activity. [58]

Sampling frequency
According to Novac et al. [59], a higher accelerometer sampling rate provides more relevant information than a lower rate. Furthermore, using a low sampling rate may make distinguishing transitions between different activities or distinguishing characteristics of a specific activity type difficult. [59]

Sensor placement
The location of the sensor is determined by the type of activity to be detected. The immobility of the device position or the wearing comfort of the participant can also influence sensor placement. The position where sensors can be placed differs into the lower, middle, and upper part of the body or the arms. When only one accelerometer is employed, the waist is the most typically utilized sensor site. Further sensor locations near the center of the body gives the most accurate classification performance for daily activities. [60]

3.3. Feature Extraction

Feature extraction is the process of incorporating important features or patterns out of unprocessed input data including images or audio signals. The procedure aims to minimize the dimensionality of the data by removing unnecessary dimensions from the input data while keeping the most important elements. Subsequently, the output can then be used to make predictions or in other words classifications. [42]

As CNNs are designed to extract features from visual data, such as images, they can automatically learn hierarchical representations of the input through sequential layers of convolutional and pooling operations. The filters in the initial layers find inferior features like corners and edges. In the following, layers that are deeper towards the end find more specific qualities like shapes, textures, and aspects of objects. [64]

3.4. Feature Selection

The goal of feature selection is to identify the most informative and meaningful features in the extracted set of data. This reduces the number of characteristics, the computational complexity of the classification process and further the amount of training data required for parameter estimation. [60]

HAR can be viewed as a classification problem with a time series signals as input and activity class labels as output. The activity detection process is divided into two phases: [65]

- Training phase
- Classification phase

Throughout the training phase, features are extracted from the raw time series data. As next step these features are used to train a classification model. In contrast, during the classification phase features are extracted from new raw data. Further activity labels are predicted using the previously built model. An example would be if the signal-to-feature mapping is not predefined, but instead learned automatically from the training data set. [65]

3.5. Challenges

Bevilacqua et al. state that “the main challenges in sensor based HAR is the information representation”. [66] Traditional classification methods rely on features constructed and extracted from dynamic signals. However, these features are mostly heuristically selected depending on the task at hand. Feature extraction processes often require extensive knowledge of the application domain or human experience but produce only superficial features. [3] Moreover, traditional HAR methods cannot handle complex motion patterns and often perform poorly with dynamic data. [66]

Next, deep HAR is associated with two critical issues. The first one is online deployment and the second one is the usage on low power devices. Although some existing work has used deep HAR on smartphones and other embedded devices, it is still a long way from being ready for online and mobile deployment. The deep learning model is often trained on a remote server and then used afterwards. This method is neither real-time nor conducive to incremental learning. There are two approaches to addressing this issue: [67]

- lowering the cost of communication between mobile and server
- improving the computing power of embedded devices

Furthermore, the deep learning performance is still heavily reliant on labeled samples. Obtaining sufficient activity labels is both costly and time-consuming. As a result, unsupervised activity recognition is critical. Activities that are more complex and advanced should be recorded more frequently compared to simple daily activities. Next, higher-level activities contain more semantic and contextual information. This makes determining their hierarchical structure difficult and for this reason, existing methods frequently ignore signal correlations and further produce poor results. [59]

4. Conceptual Design and Practical Implementation

This chapter describes the theoretical part as well as the practical elements of the implementation. Furthermore, this chapter explains a theoretical outline and is the explanatory part to the practical implementation.

The architecture diagram chapter points out the hardware and software components of the project design. Further protocols and communication standards that connect these components are addressed. In the segment of the requirements analysis the defined requirements are listed. In order to achieve a better delimitation, the requirements are categorized according to the MoSCoW method.

To summarize the building blocks of the practical part this chapter is divided into the different areas of technology that are used throughout the implementation phase. The sections also show the lab setup, the used microcontroller as well as its sensors. Then the tools and integrated development environment (IDE) to create the project are listed. Next, the structure of the microcontroller application and its drivers and middleware components are presented to share the experience for future work on this project.

Further the used dataset and the created CNN model are described in the subsection CNN based HAR. Therefore, the steps to prepare, preprocess and plot the dataset are demonstrated. Afterwards, the architecture and procedure how to compile and fit the built Keras model are displayed. To show the results, the accuracy and operability of the model on a microcontroller are illustrated in the evaluation chapter.

The section user application shows how the iOS app was realized based on application screenshots. Following, an overview of the application flow is provided. Additionally, it is demonstrated how the integration of the BlueST SDK was accomplished. A topic that should not be forgotten is the persistent storage of data. Thus, the captured information is saved in a database platform named MongoDB Realm. In this chapter the advantages of this database solution as well as the underlying technologies are explained.

4.1. Architecture Diagram

Figure 7 includes all embedded devices and sensors that are used in the practical implementation of this thesis. Additionally, the interfaces between them are displayed. Further information of the communication standards and sensors can be found in chapters Bluetooth Low Energy, SensorTile kit, LSM6DSM, BlueNRG-MS, Serial Peripheral Interface and General Purpose Input/Output.

The core components of the architecture are an iPhone and the microcontroller named SensorTile kit that is described in chapter 4.9. The communication between these two components is enabled via Bluetooth Low Energy (BLE). To transfer data over BLE the microprocessor has a built-in chip called BlueNRG-MS which is outlined in section 4.9.1.2.

Further the built-in BLE chip is connected to the SensorTile kit over a communication interface called Serial Peripheral Interface (SPI). In this case a SPI-4-wire interface is used.

To have the possibility to gather accelerometer data the microcontroller has a LSM6DSM sensor onboard that is narrated in chapter 4.9.1.1. In this particular instance the sensor is connected over SPI-3-wire. The description of SPI and the main difference of SPI-3-wire and SPI-4-wire is specified in subsection 4.8.1.

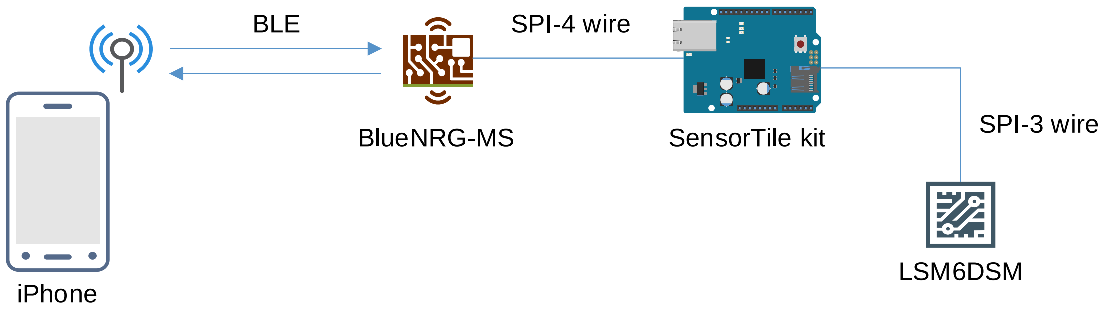


Figure 7: Architecture diagram of the practical implementation

4.2. Requirements Analysis

This chapter describes the MoSCoW requirements of the planned implementation which are fully shown in Appendix B. Therefore, functional requirements (FR) include the features that build the main functionality of the implementation. To show the importance of the individual capabilities the FRs are divided into the four categories must, should, could and will not.

The must requirements define that the microcontroller must be able to gather movement data and further on transmit the values over BLE. Next, the MCU has to have the possibility to read and receive data and to analyze it by a deep learning algorithm.

Another must criterium is the operation without a direct power connection. To fulfill the main functionality the practical implementation also needs to save the analyzed data for later use. Additionally, the solution must offer the possibility to start and stop the recording of data.

Another aspect of the iOS application is that it should show recordings from the past to subsequently compare them to predefined movement patterns. Thus, a customized deep learning model should be used for the specific use case of human activity recognition.

Not that important the app could show the gathered data in a graphical way. To delineate what the app should not be able to do two wont requirements were defined. These include that the app will not be executable on Android devices and the transmission technology will not be Wi-Fi.

4.3. Goals

The main goal of the project that is described in section 4.1. and 4.2. is to support active people to get instant results of recognized human activities.

To reach the aim of the project there are implemented three technical components which build the architecture for the solution. The first part is a microcontroller named SensorTile kit from the company STMicroeletronics. As the name says the MCU is equipped with various sensors which deliver the data for the analysis.

For the graphical presentation, the data is sent over BLE to an iOS app. The programming language which is used for the iOS app is Swift. To consume the data the BlueST SDK is chosen which is provided by STMicroeletronics. This library is applied to transfer the data from the microcontroller to embedded devices like smartphones.

The transmitted data is processed directly on the MCU with a deep learning algorithm which build the third part of the solution. With the help of these algorithms the gathered data is compared to existing movement patterns.

4.4. STM32CubeIDE

The STM32CubeIDE is the preferred integrated development environment to implement applications for microcontroller which are assembled by STMicroelectronics. The IDE is part of the STM32Cube software palette and is a state-of-the art C/C++ development platform including all relevant features to develop, compile and debug STM32 microcontrollers and microprocessors. [7]

The program is based on the Eclipse framework and therefore allows the usage of a set of plugins to support the development process with enhanced functionality. [7]

Following the most prominent features are stated:

- Integrates the configuration and project setup from STM32CubeMX
    - Selection of templates for various boards and microcontrollers
    - Configuration of the hardware components
    - Regenerate code of sensor interfaces without changing the code which is implemented by the user
- Includes analyzers which help to identify memory requirements
- Provides the current information like CPU registers, memories and peripheral registers while debugging code

[7]

4.5. STM32CubeMX

The STM32CubeMX is a graphical user interface that provides an uncomplicated way to configure STM32 microcontrollers. Additionally, it supports the generation of code that initializes the core components of the microcontroller. Therefore, predefined configurations can be used and selected by matching the required set of peripherals of the own microcontroller. [8]

To extend the default software stack there is the possibility to include embedded software for STM32. The standardized packages cover the STM32Cube hardware abstraction layer (HAL) that assure the portability across the STM32 environment and a set of middleware parts including USB, TCP/IP, and Graphics. [8]

4.6. STM32Cube Expansion Packages

The STM32Cube expansion packages extend the standard ecosystem of the STM32 product family. Therefore, additional software can be used to obtain drivers for chips from external manufacturers or specific middleware for many use cases like sensing, power management or connectivity. Optionally they are often accommodating examples of applications for ST boards and supported integrated development environments to minimize the hurdle to implement. [9]

The packages used in this master thesis are X-CUBE-AI and X-CUBE-BLE1. The X-CUBE-AI bundle is used to automate the conversion of pretrained AI (Artificial Intelligence) algorithms like neural network and machine learning models. The major advantage of the AI package is that the required power as well as the performance of the algorithms on the microcontroller can be measured. This ensures that the selected model also runs on the hardware. [68] A practical implementation and the results of this compatibility check are shown in chapter .

The X-CUBE-BLE1 bundle hold the drivers for BlueNRG-MS and BlueNRG-M0 Bluetooth low energy devices. Additionally, the software package comprises sample implementations of the drivers to run on various microcontroller boards. [69]

4.7. STM32CubeIDE Configurations

To successfully connect the microcontroller with the STM32CubeIDE it is necessary to configure the Debug Configurations for the project. The Debug probe that was used to flash the application to the SensorTile kit is called ST-LINK. [7] Therefore, the SWD interface from an additional microcontroller named Nucleo-F411 board is needed, as described in chapter 4.9.2. [70]

4.8. STM32CubeMX Configurations

The first step here is to create a new project. There are two possibilities to generate a template project that serves as basis for further implementations and integrations of new features. For the project which was built for this thesis the initial code basis was generated by selecting the desired micro controller unit. [8] For the SensorTile kit it is the STM32L476RGTx MCU. [6] The other choice is to select a specific board but for the SensorTile kit there is no such option.

The program STM32CubeMX is feasible to configure various configurations of the micro controller. In the tab Pin & Configuration there is a menu on the left that displays all manageable categories like System Core, Connectivity, Middleware and Software Packs. [8] Since the pin configuration is not the focus of the thesis, only the most important points are discussed in this chapter. In the Pinout view the mapping of GPIO (General Purpose Input/Output) pins and the configurations of SPI (Serial Peripheral Interface) interfaces are set. [71][72] Additionally, FreeRTOS is described in section 4.8.3. since it is used in the microcontroller implementation part for multitasking purposes. [73]

4.8.1. Serial Peripheral Interface

SPI is a high-speed, synchronous communication bus and consists of a SPI peripheral and a SPI controller. The types used in this thesis are SPI-3-wire and SPI-4-wire. Usually the bus works in full-duplex mode with a primary device and one or more secondary devices that is called SPI-4-wire. The port occupies the four following lines: [71]

- POCI (peripheral out, controller in)
- PICO (peripheral in, controller out)
- SCK (serial clock)
- CS (chip select)

When an SPI primary needs to send data to one or more secondaries, it selects the controllers by pulling the CS line low and activating a clock signal that can be used simultaneously between primary and secondary. The peripheral sends data to serial data output port POCI and simultaneously receives data from serial data input port PICO. Data is sent bit by bit. Before data is transferred, a clock pulse is provided by SCK so that data is driven out over the POCI line on the rising or falling edge of the clock and read by the secondary on the following falling or rising edge. [71]

SPI-3-wire employs just three wires to establish communication: a SCK line, a POCI wire, and a PICO wire and therefore is called half-duplex transmission. In this case the primary and secondary devices switch between transmitting and receiving data. [74] So, the main difference between the two SPI types is the additional CS wire that permits the connection of several subordinate devices to a single primary and therefore allows data to be sent and received concurrently in both directions. [75]

4.8.2. General Purpose Input/Output

GPIO is a modern microcontroller (MCU) interface that allows easy access to the internal features of the devices. A single MCU has several GPIO pins for various interactions. All GPIOs must be able to define either an input or output mode for individual pins on the chip. Further, the pins must be scalable for a wide range of applications and for instance functional uses in following areas: [72]

- Integrated circuit devices with few pins, such as system-on-chip and programmable logic devices
- Embedded applications that are heavily using GPIOs to read for example the status of various environmental sensors like temperature, triaxial, and acceleration, audio, LCD displays, or LEDs.

4.8.3. FreeRTOS

To better understand the parts of FreeRTOS, they are described in more detail in the next paragraphs.

The kernel which supports multitasking is the core component of the operating system and oversees each task that runs a program. Multitasking is the operating system's ability to perform multiple tasks at once. The scheduler is the component of the kernel in charge of determining which tasks should be running at any given time. During a task's lifetime, the kernel can suspend and then resume it multiple times. [73]

FreeRTOS provides messaging capabilities. Messages can be sent to and read from queues by tasks. Therefore, queues have a fixed, limited capacity that is defined when the task is created. A job will block if it attempts to read from an empty queue or write to a full queue. However, there are ways to attribute delays to queuing or non-blocking accesses. [76]

A semaphore is just a special kind of a message queue. Therefore, the two basic types of entities were initially chosen for formalization: tasks and message queues. In combination they form the basic mechanism of task communication and synchronization. [76]

As a task runs, it uses registers on the processor or microcontroller and accesses RAM. Together, these resources include the context of the task's execution. While suspending the execution of a task it is possible that another one is modifying the already used register values of the processor. To ensure that not the altered values are processed after continuing the previous task the saved context is taken for further computation. The feature which describes the process of saving the suspended task's context and restoring the resumed task's context is called context switching. [73]

A core feature which can be used to change the sequence in which processes are executed are included by default. [73] The two approaches are described in the next lines.

The first strategy is called preemptive. In this case the highest available task is executed at first. The initial step in a preemptive scheduler is to stack the context of the current job in case a context transition is necessary. Afterwards the scheduler increases the tick count to check if the activity led to a blocked job becoming unblocked. A context switch is performed to identify whether a task has been unblocked and whether another work has a greater priority than the present job. The scheduler then exits the interrupt, the context is recovered, and registers are unstacked. [73]

The second scheduling approach is named cooperative. The context only switches if the task or process blocks and explicitly executes a macro that prompts context switching. A decision point based on a timer is not implemented by cooperative scheduling. By yielding, processes exchange control with one another. The scheduler is therefore interrupted regularly to increase the tick count. [73]

4.9. SensorTile kit

The SensorTile kit consists of various tools which can be used to design and develop microcontroller-based solutions. One of the key advantages is the integrated development platform that helps to quickly get from an idea to a working result. [6]

The combination of a functional hardware piece with embedded software is used to address the sensors of the microcontroller and to support the development process. Additionally, STMicroelectronics offers a wide range of project examples which can be used under an open-source license. [6]

The SensorTile board embeds different sensors to measure motion and noise. Therefore high-precision inertial sensors with exceptionally low power consumption, a barometric pressure sensor, and a digital microphone are built in. Due to its small construction, it is easy to mount the device on an object to be measured. The battery that is included by default simplifies the collection of data without a direct connection to a power source. To transmit the gathered data to an external source the board uses bluetooth connectivity. [6]

The heart of the microcontroller is a STM32L4 microcontroller unit (MCU) with 80 MHz. There are three sensors connected through SPI (Serial Peripheral Interface) 3-wire with 2.5 MHz. The sensor named LSM6DSM is used for accelerometer and gyroscope data. Secondly, the LSM303AGR gathers accelerometer and magnetometer data and the LPS22HB collects measurements of air pressure. The bluetooth connection to the MCU is handled over a SPI 4-wire with 5 MHz by the sensor BlueNRG-MS. Therefore the BlueNRG-MS has an antenna with an integrated balun that is used to convert the antenna signal to a leveled pair of signals. [6]

4.9.1. Used Sensors

The following chapters show the detailed overview of the most important sensors which are used in the practical part of this thesis.

4.9.1.1. LSM6DSM

The LSM6DSM is a system in a package or short SiP. A SiP is a chip technology that includes various components like dies, passive components and unconnected devices which are small enough to fit in electronic devices. [77]

This chip is made up of a 3D digital accelerometer and a 3D digital gyroscope. The advantage of this combination is that the user has an optimal motion experience while performing at 0.65 mA with low power but therefore in a high-performance mode. [6]

4.9.1.2. BlueNRG-MS

The BlueNRG-MS a Bluetooth low energy (BLE) network processor that performs with low power and underlies the Bluetooth 4.1 standard. It supports various roles and can be used as Bluetooth smart sensor and hub device at the same time. [6]

Following operating modes are supported:

- Reset mode
- Sleep mode
- Standby mode
- Active mode
- Radio mode
  - Receive radio mode
  - Transmit radio mode

[78]

4.9.2. Lab Setup

As seen in Figure 8 the SensorTile kit is connected to another microcontroller called Nucleo-F411. This connection is necessary because the SensorTile kit does not have its own SWD (Serial Wire Debug) port. [6]

4.9.2.1. Serial Wire Debug

Serial Wire Debug (SWD) is used to send and receive commands to and from the Debug Access Port (DAP). A debug access port is a silicon-on-chip device that acts as a bridge between the debug host and the internal components of the microprocessor such as memory and peripherals. [70]

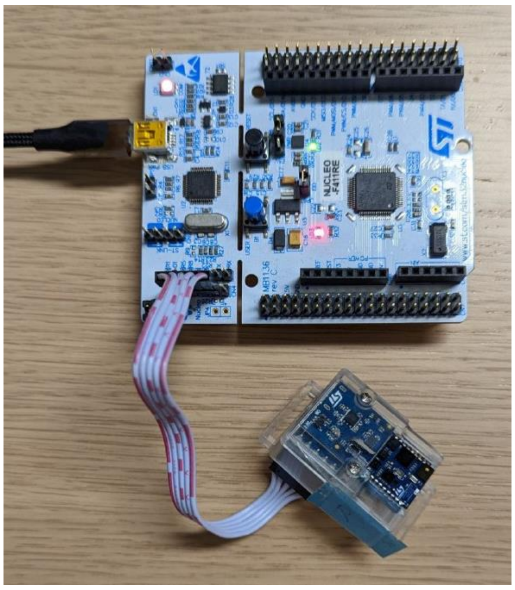


Figure 8: Lab setup of the SensorTile kit and Nucleo-F411

4.9.3. Code Explanation

In this chapter the project structure as well as the practical implementation parts like drivers, middleware components and moreover functions are discussed. The folders of the microcontroller project are described in Table 2.

Folder	Description
Core	Holds the main logic that initializes the HAL and all configured peripherals and components like Bluetooth and AI functions.
Drivers	Houses the drivers to communicate with sensors and specific MCU and HAL
Middlewares	Includes the generic building blocks of the X-CUBE-BLE1 and X-CUBE-AI expansion packages
BlueNRG_MS	Includes the pre-configured pin settings and implemented functions to return values of various sensors to the requesting instance.
X-CUBE-AI	Incorporates the processed model inputs of the pre-selected CNN
Debug	Consists of the built files of the project with debug configuration
Release	Holds the built files of the project with release configuration

Table 2: Folder structure description of the microcontroller project

The HAL (Hardware Abstraction Layer) driver, like the device driver layer, is the intercessor between all tasks, ensuring that all requests that use the peripheral device are recognized correctly. An MCU has a limited number of peripherals, but many processes may use them. A hardware abstraction layer hides hardware-specific details from higher layers. If a HAL is contemplated for a specific MCUs, the upper layer can be used interchangeably. [79]

The following paragraphs describe the crucial files of the expansion packages X-CUBE-BLE1 and X-CUBE-AI. Furthermore, code excerpts are shown in Appendix C. An STM32CubeIDE project include many different files. Therefore, it is important to know that there are .c and .h files.

H files are the standard approach to inform the compiler about functions, variables, and types that exist outside of the file. Further they contain declarations of public functions rather than definitions. The C file has definitions or in other words holds the implementation of the functions. The H file is like the public interface of the module. Further, it is important that the C files include the corresponding H files. This way the C file can accommodate all the declarations in the H file. [80] The upcoming files are listed with the ending .*. This is to state that there are always two corresponding files. One the one hand with the ending .h and on the other one with the ending .c.

BlueNRG_MS (for expansion pack X-CUBE-BLE1)

- app_bluenrg_ms.*
- gatt_db.*
- sensor.*

In the app_bluenrg_ms.* files the name, address, hardware as well as the software version of the bluetooth peripheral is set. Additionally, the function User_Process() is there to implement the desired functionality that is provided to the smartphone that is connected to the microcontroller and updates the acceleration data.

Next the file gatt_db.* includes the logic to transmit the acceleration data to the connected device over BLE. Therefore, it includes the function Acc_Update() that uses GATT which is described in chapter 2.4.1.2. to exchange the updated acceleration values with the smartphone.

The sensor.* files contain the functions and variables to initiate the BlueNRG-MS sensor and to manage different functionalities like setting the microcontroller in a connectable state or to notify if the connected device gets disconnected or in order a device connects in the first place. Further the function DeviceConnectable() allows other devices like smartphones to connect to the sensor. The discovery of the microcontroller is done through GAP that is explained in section 2.4.1.1.

X-CUBE-AI

- network_config.*
- network_data.*
- network.*

The file network_config.* holds various platform and tools versions of the generated convolutional neural network.

Furthermore, the method ai_network_data_weights_get() in the file network_data.* returns an array that consists of weights and biases. These values map the functionality of the CNN. Each neuron in one layer communicates with some or all neurons in the following layer. Weights and biases are applied to inputs as they are sent between neurons. [40]

The network.* files contain the parameter AI_NETWORK_MODEL_NAME that describe the name of the CNN model that was chosen in the X-CUBE-AI expansion package. Further the resolution of the input layer is defined as input structure and the number of output classes can be read out in this file. Additionally, it holds the methods to allocate resources for the NN as well as the initialization of the data structure. Lastly, the function to run the NN model is included.

The software pack X-CUBE-AI is configured to use a Keras model. After selecting the CNN model, it should be checked whether the performance of the MCU is sufficient. Therefore, the accuracy of the deep learning algorithm is validated. The description in Table 3 shows the dimensioning information that indicates the performance of the neural network.

Expression	Description
Complexity	Shows the functional complexity of DL models imported in MACC. It also contains an approximation of the activation function.
Flash occupation	Specifies the size (in bytes) of the read-only memory block generated to store the weight/bias parameters after compression.
RAM	Specifies the size (in bytes) of the expected read-write memory used to store intermediate values for inference computations.

Table 3: Dimensioning information of the X-CUBE-AI model analysis [81]

The term MACC (Multiply-and-Accumulate Complexity) is a unit of complexity for neural network models. It is difficult to estimate the exact CPU cycles per MACC. However, an approximation of about nine cycles per MACC is realistic for a CNN model that runs on an ARM Cortex-M4 processor. With this value it is possible to calculate the needed clock cycles for one prediction. [81] The runnability of the CNN model which was created for this thesis is displayed in subsection 4.10.3.

4.10. CNN based HAR

This chapter includes the main components and procedures to create and train a CNN based on Human Activity Recognition. First of all, the dataset which holds the accelerometer data is described. Before training the CNN there are necessary steps to fulfil. These actions include the preparation, preprocessing and plotting and of data and are mentioned in subsections 4.10.1.1. , 4.10.1.2. and 4.10.1.3.

Further the Keras model, its used layers, the approaches to compile and fit the model and in the end to evaluate the outcome of the model are presented in subsections 4.10.2.1. , 4.10.2.2. and 4.10.3.

4.10.1. Dataset

Due to lack of time and an extensive amount of time which is needed to create an own set of accelerometer data a dataset called Exercise Recognition from Wearable Sensors [7] is used for this thesis. Originally, the set of data is available in two different MATLAB files.

Engineers and scientists use MATLAB to analyze data, design algorithms, and construct models and applications. Further students and people working in different industries are benefiting from this toolset. For example, it enables the processing of signals and images for deep learning and machine learning. [82] Supplementary MATLAB has various functions related to neural networks. The data which is used for these deep learning algorithms is saved in a binary data container with the extension .mat. [83] However, since Keras is used for the creation of the deep learning model in the practical part of the thesis, the data container was converted into a NumPy array. The associated technologies and procedures of the data preparation is explained in section 4.10.1.1.

The first data container of the dataset holds raw acceleration data during exercises and the second one incorporates data of complete exercise sessions. For the reason that the data is used for the training of a CNN where exercises are detected the first document named exercise_data.50.0000_singleonly.mat is used. This consists of recordings from various participants and includes accelerometer data from different fitness exercises recorded with an arm-worn sensor and a frequency of 50Hz. [5]

4.10.1.1. Data Preparation

As mentioned in the previous chapter the outcome of the converted .mat file is an NumPy array. As NumPy is the foremost programming library for arrays in Python it is the ideal tool for research in areas such as engineering, physics, and also deep learning. Further the package is able to operate with most python libraries in the direction of scientific and numerical processing of raw data. To sum up NumPy is a data structure for storing and obtaining multidimensional arrays. [84]

The final structure of the resulting NumPy array consists of tuples of data containing a participant id, the exercise name, timestamp, tri-axial accelerometer data. To understand the steps needed to get to this result the next paragraphs describe these:

[7] https://msropendata.com/datasets/799c1167-2c8f-44c4-929c-227bf04e2b9a

Loading and analyzing the .mat file

To read out the document of the dataset in the .mat format a python package named SciPy is used. The main purpose is to manipulate and visualize data from various environments such as MATLAB. Further, SciPy has a lot of functionalities and domains like clustering algorithms and statistical distributions and functions. But in this case the containing subpackage io (Input and Output) was taken in consideration. [85] The scipy.io package includes various modules to read and write data of different source formats. Therefore, the function loadmat() was taken to load the raw data from the MATLAB file. [86]

In order to get an overview of the read in raw data the number of the overall activity types and participants is issued. As a result, the dataset includes 75 fitness exercises from 94 distinct participants. The script can be found in Appendix E.

Merge data from specific exercises

The used dataset described in section 4.10.1. contains 75 activities. To get an overview of the available training sets, the script offers the possibility to print them. In the practical elaboration three exercises were selected:

- Crunch
- Biceps Curl
- Chest Press (Rack)

Next, the accelerometer data from all participants in the dataset where data for the defined activity is available was read out. During the iteration of the NumPy arrays, attention was also paid to whether there are multiple records of a trainee per exercise. Additionally, the script outputs a line for each appended recording from the participant, as well as the number of added accelerometer data from the chosen exercises. While merging the data each line of the 3D accelerometer was extended by the participant id and labeled with the associated exercise name.

4.10.1.2. Data Preprocessing

The step of normalizing the data is inevitable since recorded accelerometer data or collections of timeseries have a large scale of values. Therefore, data normalization is a common way to establish the possibility to use the data for any deep learning algorithm by narrowing down the area in which the values reside. [87] The approach which was chosen for this work is the min-max normalization and was done with the help of the python package sklearn.preprocessing. These collection of functions and classes provides the possibility to adapt raw data into a format that is more fitting for further processing. The feature which performs the normalization of the timeseries of accelerometer data is called MinMaxScaler [88]

The approach of the min-max normalization technique is that it performs a linear reshaping and are normalized in a certain range. Mostly the scale is chosen that the lowest number is a 0 and the highest is a 1. All values which lie in between are converted to a decimal between zero and one. [89]

To split the normalized dataset into a training and testing set the signals were segmented into overlapping windows of 90 samples with an overlap of 50%. Further the train-test split ratio was set to 80:20.

4.10.1.3. Data Plotting

The technique of plotting or in other word to visualize the data. The python package used for that in this thesis is called matplotlib which has a lot of options to customize and draw a graph out of almost any composition of data. [90] More specific matplotlib.pyplot was used to plot the chart. It is a set of functions that make matplotlib behave similarly to MATLAB. In the end each pyplot collection is changing the figure. For example, that could be steps like creating figures, create a plotting are in a figure or even to add labels to the diagram. Further the bundle offers a variety of ways to customize the chart and display the data by different criteria like keyword strings or categorial variables. [91]

As input of the plots the individual records of timeseries with its related 3D accelerometer data of the normalized dataset was taken. The used data is the output of the data processing step described in section 4.10.1.2. An example of a generated chart can be seen in chapter 3.1.

4.10.2. Keras Model

This chapter shows the architecture of the created CNN model. Further the most important techniques to compile, fit and evaluate a CNN model with the dataset mentioned in section 4.10.1. are described. To show the results, the evaluation of the trained model is demonstrated. In order to perform these steps python scripts were used. The model created for this thesis is a Sequential model which means that the layers are stacked in a linear way and therefore the data flows from one layer to another in a given order until it reaches the output layer [92]

4.10.2.1. Architecture

To have an overview of the used Keras layers, their parameters and used activation functions per layer are shown in Table 4. The layers have been added to the CNN model in the order listed in the table. If no value for the layer parameters or activation function is set in the cell, there is no such setting for the layer in question.

Layers	Layer Parameters	Activation Function
Conv2d	filters=128, size=(2,2)	ReLu

MaxPooling2D	size=(2,2)	
Dropout	rate=0.2	ReLu
Flatten		
Dense	units=128	ReLu
Dense	units=3	Softmax

Table 4: Used Keras layers with their parameters, and activation function

The differences of the used Keras layers are described in the paragraphs below.

Conv2D layer

The Conv2D layer [8] is a convolutional layer that performs two-dimensional convolution operations on input data. The input shape of the layer has the size (90, 3, 1) as the accelerometer data is segmented in overlapping windows of 90 samples with a height of 3 pixels and a width of 1 pixel. The 1 in the input shape suggests that each image is represented by a single channel, corresponding to a grayscale intensity value.

MaxPooling2D layer

The MaxPooling2D layer [9] is commonly used for down sampling the input along its height and width. It further selects the maximum value from each region which is (2,2) in the practical implementation and is therefore reducing the spatial dimension of the image while retaining the most salient features.

Dropout layer

The Dropout layer [10] is a regularization strategy that deactivates a subset of input units at random during training. The frequency when these sections are set to 0 is defined with the rate parameter which is adjusted to 0.2 in the practical part. It prevents overfitting by lowering neuronal interdependence and pushes the network to acquire more robust representations. [93]

Flatten layer

The Flatten layer [11] is a sort of layer that converts multidimensional input into a one-dimensional vector. It reshapes the input while keeping the overall number of components intact, allowing it to be fed into a fully connected layer for additional processing.

[8] https://keras.io/api/layers/convolution_layers/convolution2d/

[9] https://keras.io/api/layers/pooling_layers/max_pooling2d/

[10] https://keras.io/api/layers/regularization_layers/dropout/

[11] https://keras.io/api/layers/reshaping_layers/flatten/

Dense layer
The dense layer [12] allows the neural network to perform complicated nonlinear transformations and learn hierarchical representations of the input data. In conclusion it connects every neuron in the layer to every neuron in the preceding and following layers. To perform these steps where a predefined activation function is used. The first of these layers in the CNN model uses the ReLu function and the second one the Softmax operation.

4.10.2.2. Compiling and Fitting

After building the CNN with the architecture described in chapter 4.10.2.1. the sequential model was configured with the Adam optimizer and afterwards trained with the prepared training set.

The Adam optimizer is a popular adaptive optimization technique for training neural networks. It computes per-parameter adaptive learning rate scaling based on the first and second moments of the gradients. Furthermore, bias correction is included in the optimizer to improve convergence speed and stability. [94]

The process of fitting the CNN model is done with the help of the training set divided into two variables. The first one holds the input data or features and the second one the label values corresponding to the input. Further the validation split is defined which is set to 0.2 in the practical implementation. Additionally, the parameters epochs and batch size are configured with the numbers 25 and 10. An epoch is the number of iterations over the provided data and labels. The batch size details how many samples or data points are analyzed and assessed throughout each training cycle. [95]

4.10.3. Evaluation

The goal of the Keras model implementation is to recognize performed fitness exercises with the help of an accelerometer mounted on a microcontroller. The CNN model created for this thesis is able to recognize the activities Crunch, Bicep Curl, and Chest Press (rack).

To verify the results of the Keras model the confusion matrix was chosen. Additionally, the performance and runnability of the created CNN model on a microcontroller was analyzed and validated with the tool STM32CubeMX. A graphical presentation of the model is shown in Appendix A.

Overall, the confusion matrix in Figure 9 shows the confidence with which the predicted values match the defined values. In total the accuracy of the model is 89,76% which is broken down for the individual output classes as follows:

- Crunch: 97,74%
- Bicep Curl: 72,33%

[12] https://keras.io/api/layers/core_layers/dense/

- Chest Press (rack): 95,58%

The y-axis (ground truth) are the labels which were defined as output classes of the CNN model. In contrast the x-axis (predicted values) shows the labels which were assigned with the input data from the testing set.

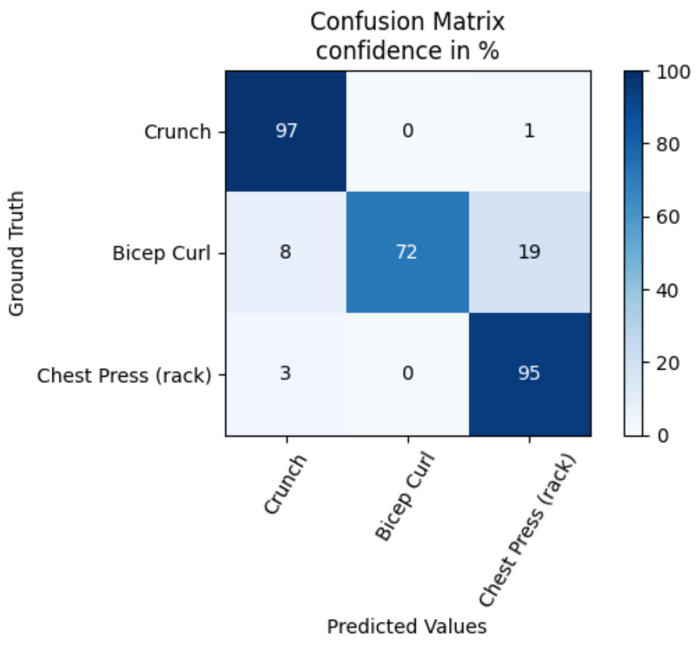


Figure 9: Confusion matrix of the trained Keras model

The analysis of the model with its related indicators within the tool STM32CubeMX is illustrated in Table 5.

Indicator	Value
Complexity	874800 MACC
Used Flash	792.77 KiB / 1024.00 KiB
Used RAM	27.61 KiB / 128.00 KiB
Achieved compression	3.56

Table 5: Analysis of the CNN model with STM32CubeMX

Since the initial size of CNN model was 2.82 MiB and therefore to large to fit on the flash memory of the SensorTile kit it was compressed by a factor of 3.56.

Further the model was validated with the testing set of the used dataset which is described in section 4.10.1. The results from the validation on the microcontroller are listed below:

- Crunch: 97,74%

- Bicep Curl: 72,33%
- Chest Press (rack): 95,58%

Considering the accuracy of the validated CNN model on the microcontroller is the same as in the confusion matrix, it can be said that the compression of the model has not led to any loss of performance of the CNN model.

4.11. User Application

This section shows the graphical user interface where the user is able to create an account and further analyze the motion data of different exercises during the training process. Furthermore, the chapter 0 shows the structure and the context of the iOS application scenes. An overview of BlueST SDK is displayed in subsection 4.11.1.

The iOS application was developed in the course of this thesis to display the results generated by the CNN model on the microcontroller to the user. Additionally, this chapter shows the feasibility how deep learning can improve human activity recognition in modern software architectures and adds a value to the architecture of the practical implementation by visualizing the results.

4.11.1. BlueST SDK

The connection between the user application and the microcontroller sensor BLUENRG-MS is established with an iOS library that is called BlueST SDK and its protocol named BlueST which is described in more detail in section 4.11.1.1.

To understand the main components of the software development kit the most important classes and their need and functionality is described in chapter 4.11.1.2.

4.11.1.1. BlueST Protocol

In this chapter the BlueST Protocol and its main structure is described. The paragraph Advertise lists the vendor-specific fields in Table 6 and its descriptions. Further the subsection Characteristics / Features shows how the SDK detects features and how extended features and known characteristics are elucidated.

Advertise

Length	**1**	**1**	**1**	**1**	**4**	**6**
Name	Field Length	Field Type	Protocol Version	Devic e Id	Feature Mask	Device MAC

Value	0x07	0xFF	0x01	0xXX	0xXXXXXXXX	0xXXXXXXXXXXXX

Table 6: Vendor-specific fields of the BlueST protocol [96]

Field Length is a value that is 7 to 13 bytes long. Next, the Device Id identifies the type of the device. For example, the value 0x02 associates the SensorTile kit that is used in the practical part of this thesis. A feature mask is a bit field that indicates which features are exported from the board. Furthermore, the Device MAC is there to be shown in the GUI of the iOS device and is for that reason optional. [96]

Characteristics / Features

Multiple Bluetooth characteristics can be exported. The SDK looks for known characteristics in all services that are identified by a UUID in the advertise feature mask, such as: XXXXXXXXX-0001-11e1-ac36-0002a5d5c51b and are referred to as basic feature. The first 32 bits are interpreted as the feature mask. If they are set to 1, it indicates that the feature is exporting data. [96]

The other ST characteristics are labeled extended feature and have the following format: XXXXXXXXX-0002-11e1-ac36-0002a5d5c51b. When mapping multiple features to a single characteristic, the data must be in the same sequence as the bit mask as shown in Table 7. [96]

Length	**2**	**>1**	**>1**	
Name	Timestamp	First Feature Data	Second Feature Data	…..

Table 7: Characteristic data format for multiple features [96]

4.11.1.2. Main Library Actors

The following paragraphs show the main library actors of the BlueST SDK. These are classes that are called Manager, Node and Feature.

Manager

This is a singleton class that initiates and terminates the discovery process as well as saves the obtained nodes. You can also define new device IDs and register/add new features to already defined devices before starting the scanning process. The Manager notifies the discovery of nodes via the delegate BlueSTSDKManagerDelegate. [96]

Node

This class represents a remote device. From this class you can restore the functions exported from the node and read and write data to or from the device. The node will export all properties set to the value 1 in the advertised message. Once the device is connected, it scans for available features and activates them. At this point, it is manageable to request and send data coupled to the detected features. [96]

Feature
This class represents data exported from a node. Each feature has an array of BlueSTSDKFeatureFields describing the exported data. Data is received from the BLE feature and is contained in the BlueSTSDKFeatureSample class. The user is notified about data using the listener pattern. The data exported in this example can be extracted using static helper methods in the class. [96]

4.11.2. Screenshots

This chapter illustrates the outcome of the user application for the practical implementation part. To better understand the functionality of the iOS app the application flow and the corresponding scenes are described in section 4.11.3.

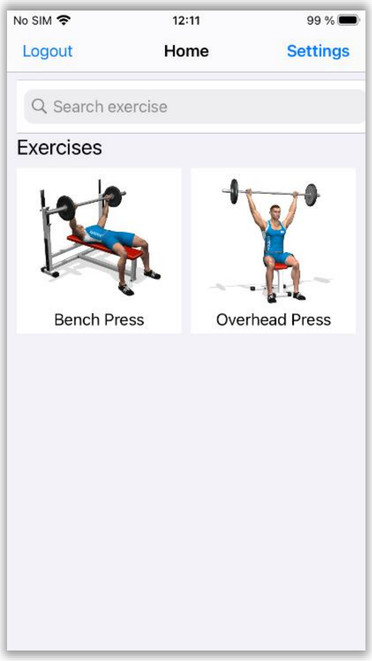


Figure 10: iOS App Home Scene

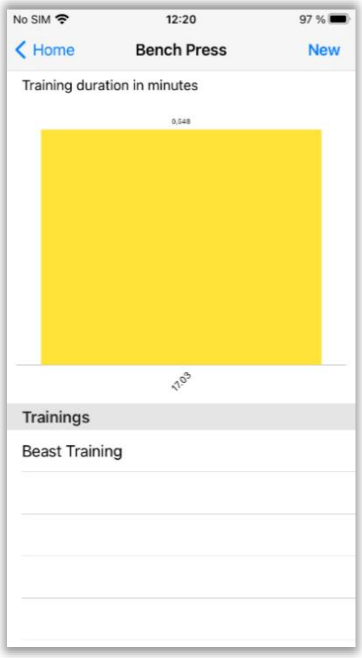


Figure 11: iOS App Bench Press Scene

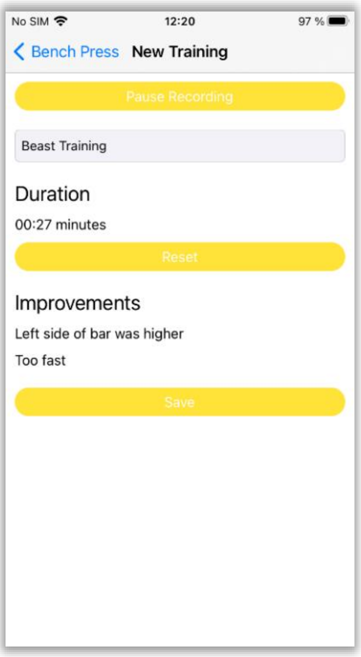


Figure 12: iOS App New Training Scene

4.11.3. Application Flow

The first step is the registration process. There the user must provide an email address as well as a password to register. Afterwards the credentials from the registration are used to login to the application.

The prerequisite that a training can be recorded is the connection with the SensorTile kit. Further the landing page after logging in is the Home Scene as seen in Figure 10. Here all available exercises are displayed. A click on the desired type of training leads to the particular scene which shows the already completed training sessions. In Figure 11 the bench press scene shows an example and additionally, a graph represents the duration values of specific sessions in a graphical way.

To start a new recording of movement data the New Training Scene, as illustrated in Figure 12, allows to start the recording of acceleration data. In addition, there is the possibility to pause the transcription of data. During the exercise the duration is displayed, and improvements of the current session are added right away. After finishing all sets, the training is saved and further displayed in the list of the specific exercise scene.

To have a complete overview of the implemented iOS app, additional screenshots can be found in Appendix D.

4.11.4. Persistent Data Storage

In this chapter the database and its components are shown in chapter 4.11.4.1. Further the project components are explained in section 4.11.4.3. Additionally, MongoDB Realm, the user authentication method and the advantages of the platform are shown in subsection 4.11.4.4. and 4.11.4.5.

4.11.4.1. MongoDB

MongoDB is a database management system intended for the use with mobile and web-based applications. The database is document-based and through its persistence strategy it is built to be easily scalable. Additionally, the database has high read and write throughput and an automatic failover. Furthermore, the MongoDB data format is straightforward to extend since it allows unstructured data by default and no migrations are needed when application prerequisites are modified. [97]

4.11.4.2. Database Components

MongoDB works with collections and documents. That means that the database is a physical storage location for these components and is hosted on a MongoDB server. In the next paragraphs the previously mentioned database fundamentals are described in more detail. [97]

Collection
A collection exists of various MongoDB documents and do not have an own schema. Further different fields can be assigned to documents within a collection and usually have a logical determination. [97]

Document
The document consists of key-value pairs and have a dynamic composition. That means that documents held in the same collection usually do not have the same structure or data types. [97]

4.11.4.3. Project Components

The database for the practical implementation holds the documents for following collections:

- User
- Trainings
- Exercises

As the storage type of the data is document based there is no fixed database structure defined. Therefore, the persistent data storage of practical implementation is prepared to be easily expanded for future work. [97]

4.11.4.4. MongoDB Realm

The database and user authentication are implemented with MongoDB Realm. A Realm database solves many common mobile programming headaches which are described in more detail in the section 4.11.4.5. [98]

For the user authentication an authentication provider named Email/Password is used. In order to work with the iOS app, the user must register with an email and a password.

4.11.4.5. Advantages

The next points describe the advantages of using a persistent data storage in an edge computing architecture.

Local storage
The Realm database runs directly on client devices. Access objects using each platform's native query language, making storing, accessing, and updating data simple and easy. [98]

Network reliability
The preferred usage of the Realm database is offline. By implication this means that reading and writing takes place via the local database. When Realm sync is enabled, local changes are constantly pushed and synchronized with the actual state of the remote server. Additionally, conflicts are solved on each client that is linked to a MongoDB Atlas cluster. [98]

Reactive UI

The latest data stored in the Realm database is shared over so-called live objects. These objects can subscribe to changes and to keep the graphical user interface always up to date. [98]

MongoDB Realm provides a feature that is called DeviceSync. It allows for simple data synchronization between a local client database and a robust, fully controlled backend. Atlas DeviceSync accelerates application development by providing out-of-the-box networking code and conflict resolution, ensuring that the user experience on mobile devices is responsive and performant regardless of connectivity. [99]

4.12. Related Work

During the research phase of the implementation part of this thesis similar projects were found. In this chapter the approaches and used technologies are described to show further possibilities of how to combine edge technologies to present results with the help of CNNs.

Merenda et al. [100] state that the research field of HAR based on motion sensor analysis has been rapidly expanding in recent years. Additionally, there are many applications on the market that are game changers for wearable devices. Many people nowadays use fitness bands while working out. The fitness band feature allows you to track performances, such as exercise duration, or the burned calories. Tracking training progress, recognizing exercises, and counting repetitions are all possible with a variety of solutions. Some rely on computer vision, while others employ smartphones equipped with inertial sensors. [100]

In the work a CNN-powered HAR machine learning (ML) application for fitness environments is proposed. The scope ranges from data acquisition to implementing edge ML systems on a low power MCU. The solution provides a way to automatically detect body movements during exercise execution, as well as repetitive counts [100]

Skawinski et al. [101] build a CNN which detects activities from a single 52 Hz 3D accelerometer which was placed on the chest. The reason why they positioned the sensor on the chest was the outcome of a preliminary study they evaluated. The result was that the chest is the most suitable position for using only one accelerometer sensor. Their approach is detecting following workout types and one category for the timeframe between the exercises: [101]

- Pushups
- Situps
- Squats
- Jumping Jacks
- No Exercise

The recorded set of data was split into a training, validation, and test set with 75%, 17,5% and 7,5% of all tuples. The trained CNN model which was trained with the training data and a batch size of 512 consists of three hidden layers. As a result, the network has an input layer, three fully linked convolutional layers buried in the network, and a Softmax classification output layer. The first convolution uses two filters and a kernel size of 15, the second uses 100 filters and a kernel size of 10, and the third uses 8 filters and a kernel size of 2. After validating and testing the results are that the validation set has an overall accuracy of 90.6%, whereas the test set has an accuracy of 89.9%. [101]

For the thesis of Merenda et al. [100] a group of 15 of her volunteers were selected to receive activity records. Each subject performed a different series of simple exercises while wearing the device on their wrist. Therefore, following three exercises and another category to detect pauses and the improper execution of exercises been selected: [100]

- Squats
- Curls
- Pushups
- NAE (Not an Exercise)

At a frequency of 20 Hz, accelerometer values were collected from a 3-axis accelerometer and a 3-axis gyroscope to create a dataset for both the training and testing phases of the application. Therefore, the research team used the SensorTile kit that is described in chapter 4.9. The collected data was then analyzed and processed in order to correctly label each dataset sample. Further the CNN trained with the dataset has two hidden layers and a kernel size of three. In the first layer, activation functions are ReLu, while in the second, they are sigmoid. Further the redundancy was decreased by changing the content of the information via the MaxPooling layer. In the end the CNN reached an accuracy greater than 95% with a total of 200 epochs. [100]

There are three main reasons why a CNN was selected for this project. To begin, no data preprocessing is required to extract features. This is beneficial for edge machine learning applications because introducing preprocessing operations on the MCU can increase complexity and memory requirements. Nonetheless, because the system is battery-powered, offloading the MCU's computational load reduces power consumption. Weight distribution is the second advantage. As a result, there are fewer inter-layer connections and network memory is saved. Finally, CNNs can benefit from the concept of immutability. In fact, despite simple transformations of the input data, the network's response is largely unchanged. [100]

In comparison the CNN model which was created for this thesis has an overall accuracy of 89,76%. The result is therefore in the range of the discussed related work and shows that CNN models can be used on microcontrollers to improve human activity recognition in modern software architectures.

5. Discussion

The further development of CNNs and its underlying research is becoming an increasingly important topic. Since edge computing is a wide-ranging term the scope of this topic was narrowed down to a few thereof integrable technologies. However, most edge technologies cannot be used without the connection to other components. Even though the computation can take place directly on a small computer like a microcontroller there must be a connection to the outside of the system. These connecting pieces can be for example hardware devices or an internet connection. For this reason, this thesis also covers topics like cloud computing, or even more specific CloudAI. But nevertheless, the main focus lies on running implementations on tiny hardware appliances.

Furthermore, this thesis shows the interrelationships of artificial intelligence, machine learning, deep learning, and most important convolutional neural networks. The scope of application of CNNs was laid on the processing of motion data. Therefore, related fields like accelerometer data and the challenges which possibly occur when processing them with the help of CNNs are discussed.

Due to the fact that during the research as well as the implementation of the practical part difficulties occurred frequently, this work also deals with the challenges of creating neural networks. Additionally, through the lack of experience with microcontrollers and deep learning algorithms the should requirements were not implemented to the full extent. The practical parts of this thesis serve as proof of concept to combine edge technologies. Besides that, a platform was analyzed and chosen to have the possibility to store data locally on the edge device without a constant internet.

Additionally, an overview of the phased structure of AI and CNNs was provided. Basically, this includes computer systems that can perform intelligent tasks without human intervention. In summary, the deeper you dive into the material of artificial intelligence, the more automated it becomes. In relation to this, greater effort is also required in the preparation of models and data.

The research field of neural networks is nevertheless still a work in progress. This applies in particular to the processing of data that may yield different values in the same previous procedure. One of these areas where this can occur is the human activity recognition. In order to better illustrate these characteristics, sensor factors and challenges related to HAR and CNNs are also highlighted in this work.

Comprehensible, ANNS need hardware devices as a basis to run on. In this thesis the viewpoint lies on embedded devices like microcontrollers and smartphones. Therefore, inventions like optimized deep learning models on small hardware devices support these realizations. Equally important is the presentation of the processed data to the user. Further to connect the processing unit and the presentation part a transmission technology has to be chosen. Therefore, the practical implementation represents a possibility how such an edge computing architecture can look like. In order to give further insight into the research already conducted, results of similar projects or papers are presented.

The evaluation of the Keras model shows that it is really important to have a well thought architecture and an expressive dataset which well prepared. Since the set of data used in the practical part was not labeled and had irregularities within the recorded accelerometer data the step of normalizing the values was essential. However, due to the multiple steps taken to preprocess the data, a good result was achieved when crosschecking the compiled CNN model against the test set of data.

5.1. Future work

The result of this work can be used as a basis for the further development of a health-promoting app. The goal is to get expressive feedback by recording accelerometer data of different training exercises and to evaluate them afterwards with a CNN. The current state of the project provides a functional architecture with different edge technologies and the associated protocols.

To fulfill all planned functional requirements of the practical implementation an own dataset of training activities has to be recorded and subsequently labeled. As supporting material, the theoretical research helps to determine the current weak points and what to look for to create a CNN which returns the output classes with a high accuracy.

Further this thesis covers important aspects to be considered in further research on edge technologies in connection with neural networks.

6. Conclusion

In today’s fast moving world technologies are developing at a rapid pace. This makes it increasingly important to get from the idea to the result in the shortest possible time. On the one hand edge technologies support this approach and on the other hand frameworks help to shorten the time to market many times over.

Moreover, edge computing and CNNs have become indispensable in many areas. They are used in a variety of fields and industries to obtain meaningful results quickly and without delay. The hardware assemblies and embedded software packages of STMicroelectronics support the development of advanced architectures based on edge technologies. In addition, the tool chain is very advanced, which can drive the rapid development of novel architectures and is able to save valuable time in the end.

To contribute to this research area, the next step of this work is to gather an own dataset of fitness exercises to develop another neural network based on human activity recognition. The goal of the CNN built in the future is to detect the correct execution of different exercises.

Bibliography

[1] Kevin C. Costley, *The Positive Effects of Technology on Teaching and Student Learning.* [Online]. Available: https://eric.ed.gov/?id=ED554557 (accessed: Nov. 13 2022).

[2] D. K. Dennis *et al., EdgeML: Machine Learning for resource-constrained edge devices.* [Online]. Available: https://github.com/Microsoft/EdgeML (accessed: Nov. 10 2022).

[3] Dogan Gulustan, Ertas Sinem Sena, and Cay İremnaz, "Human Activity Recognition Using Convolutional Neural Networks," in *2021 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, 2021, pp. 1–5.

[4] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electron Markets*, vol. 31, no. 3, pp. 685–695, 2021, doi: 10.1007/s12525-021-00475-2.

[5] Morris, D., Saponas, T. S., Guillory, A., & Kelner, I., *Exercise Recognition from Wearable Sensors.* [Online]. Available: https://msropendata.com/datasets/799c1167-2c8f-44c4-929c-227bf04e2b9a (accessed: May 2 2023).

[6] STMICROELECTRONICS, *Getting started with the STEVAL-STLKT01V1 SensorTile integrated development platform.* [Online]. Available: https://www.st.com/resource/en/data_brief/steval-stlkt01v1.pdf (accessed: Jun. 20 2022).

[7] STMICROELECTRONICS, *Integrated development environment for STM32 products.* [Online]. Available: https://www.st.com/resource/en/data_brief/stm32cubeide.pdf (accessed: Jul. 2 2022).

[8] STMICROELECTRONICS, *STM32 configuration and initialization C code generation.* [Online]. Available: https://www.st.com/resource/en/data_brief/stm32cubemx.pdf (accessed: Jul. 4 2022).

[9] STMICROELECTRONICS, *STM32Cube Expansion Packages.* [Online]. Available: https://www.st.com/en/embedded-software/stm32cube-expansion-packages.html (accessed: Aug. 24 2022).

[10] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile Edge Computing: A Survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, 2018, doi: 10.1109/JIOT.2017.2750180.

[11] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, "TinyML: Current Progress, Research Challenges, and Future Roadmap," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2021, pp. 1303–1306.

[12] I. Ahmed, G. Jeon, and F. Piccialli, "From Artificial Intelligence to Explainable Artificial Intelligence in Industry 4.0: A Survey on What, How, and Where," *IEEE Trans. Ind. Inf.*, vol. 18, no. 8, pp. 5031–5042, 2022, doi: 10.1109/TII.2022.3146552.

[13] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, Antalya, 2017, pp. 1–6.

[14] Y. Chen, "IoT, cloud, big data and AI in interdisciplinary domains," *Simulation Modelling Practice and Theory*, vol. 102, p. 102070, 2020, doi: 10.1016/j.simpat.2020.102070.

[15] T. Sipola, J. Alatalo, T. Kokkonen, and M. Rantonen, “Artificial Intelligence in the IoT Era: A Review of Edge AI Hardware and Software,” in *2022 31st Conference of Open Innovations Association (FRUCT)*, Helsinki, Finland, 2022, pp. 320–331.

[16] A. Shahid and M. Mushtaq, “A Survey Comparing Specialized Hardware And Evolution In TPUs For Neural Networks,” in *2020 IEEE 23rd International Multitopic Conference (INMIC)*, Bahawalpur, Pakistan, 2020, pp. 1–6.

[17] S. Han, H. Mao, and W. J. Dally, “Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding,” 2015. Accessed: Apr. 2 2023. [Online]. Available: http://arxiv.org/pdf/1510.00149v5

[18] H. Xue, F. Dai, G. Liu, P. Cao, and B. Huang, “Edge Computing: A Systematic Mapping Study,” in *2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, AB, Canada, 2021, pp. 507–514.

[19] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, “Edge computing: A survey,” *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019, doi: 10.1016/j.future.2019.02.050.

[20] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, “Edge Intelligence: The Confluence of Edge Computing and Artificial Intelligence,” *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, 2020, doi: 10.1109/JIOT.2020.2984887.

[21] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1,” 2016. [Online]. Available: http://arxiv.org/pdf/1602.02830v3

[22] S. M, S. K, and N. Prasanth, “A novel framework for deployment of CNN models using post-training quantization on microcontroller,” *Microprocessors and Microsystems*, vol. 94, p. 104634, 2022, doi: 10.1016/j.micpro.2022.104634.

[23] A. Sherstinsky, “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network,” *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020, doi: 10.1016/j.physd.2019.132306.

[24] D. Li, X. Chen, M. Becchi, and Z. Zong, “Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs,” in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, Atlanta, GA, USA, 2016, pp. 477–484.

[25] N. N. Alajlan and D. M. Ibrahim, “TinyML: Enabling of Inference Deep Learning Models on Ultra-Low-Power IoT Edge Devices for AI Applications,” *Micromachines*, vol. 13, no. 6, 2022, doi: 10.3390/mi13060851.

[26] L. Dutta and S. Bharali, “TinyML Meets IoT: A Comprehensive Survey,” *Internet of Things*, vol. 16, p. 100461, 2021, doi: 10.1016/j.iot.2021.100461.

[27] V. C. Da Farias Costa, L. Oliveira, and J. de Souza, “Internet of Everything (IoE) Taxonomies: A Survey and a Novel Knowledge-Based Taxonomy,” *Sensors*, vol. 21, no. 2, p. 568, 2021, doi: 10.3390/s21020568.

[28] A. Gamatie, G. Devic, G. Sassatelli, S. Bernabovi, P. Naudin, and M. Chapman, “Towards Energy-Efficient Heterogeneous Multicore Architectures for Edge Computing,” *IEEE Access*, vol. 7, pp. 49474–49491, 2019, doi: 10.1109/ACCESS.2019.2910932.

[29] P. P. Ray, “A review on TinyML: State-of-the-art and prospects,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 4, pp. 1595–1623, 2022, doi: 10.1016/j.jksuci.2021.11.019.

[30] R. Immonen and T. Hämäläinen, “Tiny Machine Learning for Resource-Constrained Microcontrollers,” *Journal of Sensors*, vol. 2022, pp. 1–11, 2022, doi: 10.1155/2022/7437023.

[31] C. Banbury *et al.,* “MLPerf Tiny Benchmark,” 2021. Accessed: Apr. 9 2023. [Online]. Available: http://arxiv.org/pdf/2106.07597v4

[32] F. &. o. Chollet, *Keras: Deep Learning for humans.* [Online]. Available: https://github.com/fchollet/keras (accessed: Jun. 8 2022).

[33] O. Pariwat, *Artificial Intelligence, Machine Learning and Deep Learning*. Piscataway, NJ: IEEE, 2017. Accessed: Jul. 20 2022. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=8250945

[34] I. H. Sarker, “Machine Learning: Algorithms, Real-World Applications and Research Directions,” *SN computer science*, vol. 2, no. 3, p. 160, 2021, doi: 10.1007/s42979-021-00592-x.

[35] A. C. Eberendu, “Unstructured Data: an overview of the data of Big Data,” *IJCTT*, vol. 38, no. 1, pp. 46–50, 2016, doi: 10.14445/22312803/IJCTT-V38P109.

[36] P. Sawadogo and J. Darmont, “On data lake architectures and metadata management,” *J Intell Inf Syst*, vol. 56, no. 1, pp. 97–120, 2021, doi: 10.1007/s10844-020-00608-7.

[37] Kai Heinrich, Björn Möller, Christian Janiesch, and Patrick Zschech, “Is Bigger Always Better? Lessons Learnt from the Evolution of Deep Learning Architectures for Image Classification,” 2019. [Online]. Available: https://www.researchgate.net/publication/337486420

[38] L. Alzubaidi *et al.,* “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions,” *Journal of big data*, vol. 8, no. 1, p. 53, 2021, doi: 10.1186/s40537-021-00444-8.

[39] J. Gupta, S. Pathak, and G. Kumar, “Deep Learning (CNN) and Transfer Learning: A Review,” *J. Phys.: Conf. Ser.*, vol. 2273, no. 1, p. 12029, 2022, doi: 10.1088/1742-6596/2273/1/012029.

[40] D. Arora, M. Garg, and M. Gupta, “Diving deep in Deep Convolutional Neural Network,” in *2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, Greater Noida, India, 2020, pp. 749–751.

[41] K. O'Shea and R. Nash, “An Introduction to Convolutional Neural Networks,” 2015. Accessed: Jun. 11 2022. [Online]. Available: http://arxiv.org/pdf/1511.08458v2

[42] D. Bhatt *et al.,* “CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope,” *Electronics*, vol. 10, no. 20, p. 2470, 2021, doi: 10.3390/electronics10202470.

[43] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna, “Rethinking the Inception Architecture for Computer Vision,” 2016.

[44] R. A. Khalil, N. Saeed, M. Masood, Y. M. Fard, M.-S. Alouini, and T. Y. Al-Naffouri, “Deep Learning in the Industrial Internet of Things: Potentials, Challenges, and Emerging Applications,” *IEEE Internet Things J.*, vol. 8, no. 14, pp. 11016–11040, 2021, doi: 10.1109/JIOT.2021.3051414.

[45] J. Yao *et al.,* “Edge-Cloud Polarization and Collaboration: A Comprehensive Survey for AI,” 2021. Accessed: Nov. 4 2022. [Online]. Available: http://arxiv.org/pdf/2111.06061v3

[46] C. R. Banbury *et al.,* “Benchmarking TinyML Systems: Challenges and Direction,” 2020. Accessed: Nov. 14 2022. [Online]. Available: http://arxiv.org/pdf/2003.04821v4

[47] M. Capra, R. Peloso, G. Masera, M. R. Roch, and M. Martina, “Edge Computing: A Survey On the Hardware Requirements in the Internet of Things World,” *Future Internet*, vol. 11, no. 4, p. 100, 2019, doi: 10.3390/fi11040100.

[48] Abdullahi Umar Umar and Abubakar Wakili, “A Comparative Study of Modern Operating Systems in terms of Memory and Security: A Case Study of Windows, iOS, and Android,” 2023.

[49] O. C. Novac, M. Novac, C. Gordan, T. Berczes and G. Bujdosó, *Comparative Study of Google Android, Apple iOS and Microsoft Windows Phone Mobile Operating Systems*. Piscataway, NJ: IEEE, 2017. Accessed: Nov. 12 2022. [Online]. Available: http://ieeexplore.ieee.org/servlet/opac?punumber=7973337

[50] F. Damien and J. Dungen, “5G ultra-low-power network for collective ephemeral hyper-local context,” in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Milan, Italy, 2015, pp. 597–602.

[51] J. Harris and L. Small, “A Summary of Bluetooth Low Energy,” 2020. [Online]. Available: https://www.dta.mil.nz/assets/Publications/A-Summary-of-Bluetooth-Low-Energy.pdf

[52] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, and D. Formica, “Performance Evaluation of Bluetooth Low Energy: A Systematic Review,” *Sensors (Basel, Switzerland)*, vol. 17, no. 12, 2017, doi: 10.3390/s17122898.

[53] R. Dangi, P. Lalwani, G. Choudhary, I. You, and G. Pau, “Study and Investigation on 5G Technology: A Systematic Review,” *Sensors (Basel, Switzerland)*, vol. 22, no. 1, 2021, doi: 10.3390/s22010026.

[54] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, “Quantization and Deployment of Deep Neural Networks on Microcontrollers,” *Sensors (Basel, Switzerland)*, vol. 21, no. 9, 2021, doi: 10.3390/s21092984.

[55] A. Y. Ding *et al.,* “Roadmap for Edge AI: A Dagstuhl Perspective,” *SIGCOMM Comput. Commun. Rev.*, vol. 52, no. 1, pp. 28–33, 2022.

[56] A. Berthelier, T. Chateau, S. Duffner, C. Garcia, and C. Blanc, “Deep Model Compression and Architecture Optimization for Embedded Systems: A Survey,” *J Sign Process Syst*, vol. 93, no. 8, pp. 863–878, 2021, doi: 10.1007/s11265-020-01596-1.

[57] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning Filters for Efficient ConvNets,” 2016. [Online]. Available: http://arxiv.org/pdf/1608.08710v3

[58] A. Czabke, S. Marsch, and T. Lueth, “Accelerometer Based Real-Time Activity Analysis on a Microcontroller,” in *Proceedings of the 5th International ICST Conference on Pervasive Computing Technologies for Healthcare*, Dublin, Republic of Ireland, 2011.

[59] P.-E. Novac, A. Castagnetti, A. Russo, B. Miramond, A. Pegatoquet, and F. Verdier, “Toward unsupervised Human Activity Recognition on Microcontroller Units,” in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, Kranj, Slovenia, 2020, pp. 542–550.

[60] H. Allahbakhshi, T. Hinrichs, H. Huang, and R. Weibel, “The Key Factors in Physical Activity Type Detection Using Real-Life Data: A Systematic Review,” *Frontiers in physiology*, vol. 10, p. 75, 2019, doi: 10.3389/fphys.2019.00075.

[61] A. Bayat, M. Pomplun, and D. A. Tran, “A Study on Human Activity Recognition Using Accelerometer Data from Smartphones,” *Procedia Computer Science*, vol. 34, pp. 450–457, 2014, doi: 10.1016/j.procs.2014.07.009.

[62] M. Karas *et al.,* “Accelerometry data in health research: challenges and opportunities,” *Statistics in biosciences*, vol. 11, no. 2, pp. 210–237, 2019, doi: 10.1007/s12561-018-9227-2.

[63] C.-J. Holst, *Utilising accelerometer and gyroscope in smartphone to detect incidents on a test track for cars*.

[64] M. Jogin, Mohana, M. S. Madhulika, G. D. Divya, R. K. Meghana, and S. Apoorva, “Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning,” in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, Bangalore, India, 2018, pp. 2319–2323.

[65] M. Zeng *et al.,* “Convolutional Neural Networks for Human Activity Recognition using Mobile Sensors,” in *Proceedings of the 6th International Conference on Mobile Computing, Applications and Services*, Austin, United States, 2014.

[66] A. Bevilacqua, K. MacDonald, A. Rangarej, V. Widjaya, B. Caulfield, and T. Kechadi, *Human Activity Recognition with Convolutional Neural Networks*, 2018. Accessed: Nov. 16 2022. [Online]. Available: https://www.researchgate.net/publication/327667610

[67] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, “Deep Learning for Sensor-based Activity Recognition: A Survey,” *Pattern Recognition Letters*, vol. 119, no. 4, pp. 3–11, 2019, doi: 10.1016/j.patrec.2018.02.010.

[68] STMICROELECTRONICS, *Artificial intelligence (AI) software expansion for STM32Cube.* [Online]. Available: https://www.st.com/resource/en/data_brief/x-cube-ai.pdf (accessed: Jul. 10 2022).

[69] STMICROELECTRONICS, *Bluetooth low energy software expansion for STM32Cube.* [Online]. Available: https://www.st.com/resource/en/datasheet/bluenrg-ms.pdf (accessed: Jul. 2 2022).

[70] T. Cedro, M. Kuzia, and A. Grzanka, “LibSWD serial wire debug open framework for low-level embedded systems access,” in *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*, 2012, pp. 615–620.

[71] L. L. Li, J. Y. He, Y. P. Zhao, and J. H. Yang, “Design of Microcontroller Standard SPI Interface,” *AMM*, vol. 618, pp. 563–568, 2014, doi: 10.4028/www.scientific.net/AMM.618.563.

[72] B. Patel and B. Tarpara, “Design and Implementation of General Purpose Input Output (GPIO) Protocol,” 2015. [Online]. Available: https://www.ijseas.com/volume1/v1i3/ijseas20150368.pdf

[73] Ming-Yuan Zhu, “Understanding FreeRTOS: A Requirement Analysis,” 2011. Accessed: Aug. 5 2022. [Online]. Available: https://www.researchgate.net/publication/308692183

[74] M.-C. Tuan, S.-L. Chen, Y.-K. Lai, C.-C. Chen, and H.-Y. Lee, “A 3-wire SPI Protocol Chip Design with Application-Specific Integrated Circuit (ASIC) and FPGA Verification,” in *Proceedings of the 3rd World Congress on Electrical Engineering and Computer Systems and Science*, 2017.

[75] S.-L. Chen *et al.,* “A Novel Low-Power Synchronous Preamble Data Line Chip Design for Oscillator Control Interface,” *Electronics*, vol. 9, no. 9, p. 1509, 2020, doi: 10.3390/electronics9091509.

[76] D. Déharbe, S. Galvão, and A. M. Moreira, “Formalizing FreeRTOS: First Steps,” in *Lecture Notes in Computer Science, Formal Methods: Foundations and Applications*, D. Hutchison et al., Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 101–117.

[77] L. He, “System-in-Package: Electrical and Layout Perspectives,” *FNT in Electronic Design Automation*, vol. 4, no. 4, pp. 223–306, 2010, doi: 10.1561/1000000014.

[78] STMICROELECTRONICS, *Upgradable Bluetooth® low energy network processor.* [Online]. Available: https://www.st.com/resource/en/datasheet/bluenrg-ms.pdf (accessed: Jul. 2 2022).

[79] J. Dalbins *et al.,* “ESTCube-2: The Experience of Developing a Highly Integrated CubeSat Platform,” in *2022 IEEE Aerospace Conference (AERO)*, Big Sky, MT, USA, 2022, pp. 1–16.

[80] J. Egner, “Guide To C Files And H Files,” 2011. [Online]. Available: http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/c_and_h_files.pdf

[81] STMICROELECTRONICS, *Getting started with X-CUBE-AI Expansion Package for Artificial Intelligence (AI).* [Online]. Available: https://www.st.com/resource/en/data_brief/steval-stlkt01v1.pdf (accessed: Oct. 5 2022).

[82] The MathWorks, Inc., *What Is MATLAB?* [Online]. Available: https://www.mathworks.com/discovery/what-is-matlab.html (accessed: May 14 2023).

[83] M. G. Kay, “Basic Concepts in Matlab,” 2010.

[84] C. R. Harris *et al.,* “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, 2020, doi: 10.1038/s41586-020-2649-2.

[85] The SciPy community, *Introduction.* [Online]. Available: https://docs.scipy.org/doc/scipy/tutorial/general.html (accessed: May 15 2023).

[86] The SciPy community, *Input and output (scipy.io).* [Online]. Available: https://docs.scipy.org/doc/scipy/reference/io.html#module-scipy.io (accessed: May 15 2023).

[87] Samit Bhanja and Abhishek Das, *Impact of Data Normalization on Deep Neural Network for Time Series Forecasting*.

[88] scikit-learn developers, *Preprocessing data.* [Online]. Available: https://scikit-learn.org/stable/modules/preprocessing.html (accessed: May 16 2023).

[89] S. Gopal Krishna Patro and Kishore Kumar Sahu, *Normalization: A Preprocessing Stage*.

[90] I. Stancin and A. Jovic, “An overview and comparison of free Python libraries for data mining and big data analysis,” in *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 2019, pp. 977–982.

[91] The Matplotlib development team, *Pyplot tutorial.* [Online]. Available: https://matplotlib.org/stable/tutorials/introductory/pyplot (accessed: May 19 2023).

[92] M. Spatafora, *Human Activity Recognition using CNN in Keras for SensorTile (STMicroelectronics).* [Online]. Available: https://github.com/ausilianapoli/HAR-CNN-Keras-STM32 (accessed: Apr. 28 2023).

[93] J. Yang and G. Yang, “Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer,” *Algorithms*, vol. 11, no. 3, p. 28, 2018, doi: 10.3390/a11030028.

[94] A. Bhadani and A. Sinha, “A FACEMASK DETECTOR USING MACHINE LEARNING AND IMAGE PROCESSING TECHNIQUES,” *Engineering Science and Technology an International Journal*, 2020.

[95] Keras, *Model training APIs.* [Online]. Available: https://keras.io/api/models/model_training_apis/ (accessed: May 20 2023).

[96] STMICROELECTRONICS, *BlueST SDK.* [Online]. Available: https://github.com/STMicroelectronics/BlueSTSDK_iOS (accessed: Sep. 23 2022).

[97] Hema Krishnan, Research Scholar, CUSAT, “MongoDB – a comparison with NoSQL databases,” 2016. [Online]. Available: https://www.researchgate.net/publication/327120267

[98] MongoDB, *Introduction to Realm.* [Online]. Available: https://www.mongodb.com/docs/realm/introduction (accessed: Nov. 16 2022).

[99] MongoDB, *Device Sync.* [Online]. Available: https://www.mongodb.com/atlas/app-services/device-sync (accessed: Nov. 20 2022).

[100] M. Merenda, M. Astrologo, D. Laurendi, V. Romeo, and F. G. Della Corte, “A Novel Fitness Tracker Using Edge Machine Learning,” in *2020 IEEE 20th Mediterranean Electrotechnical Conference (MELECON)*, Palermo, Italy, 2020, pp. 212–215.

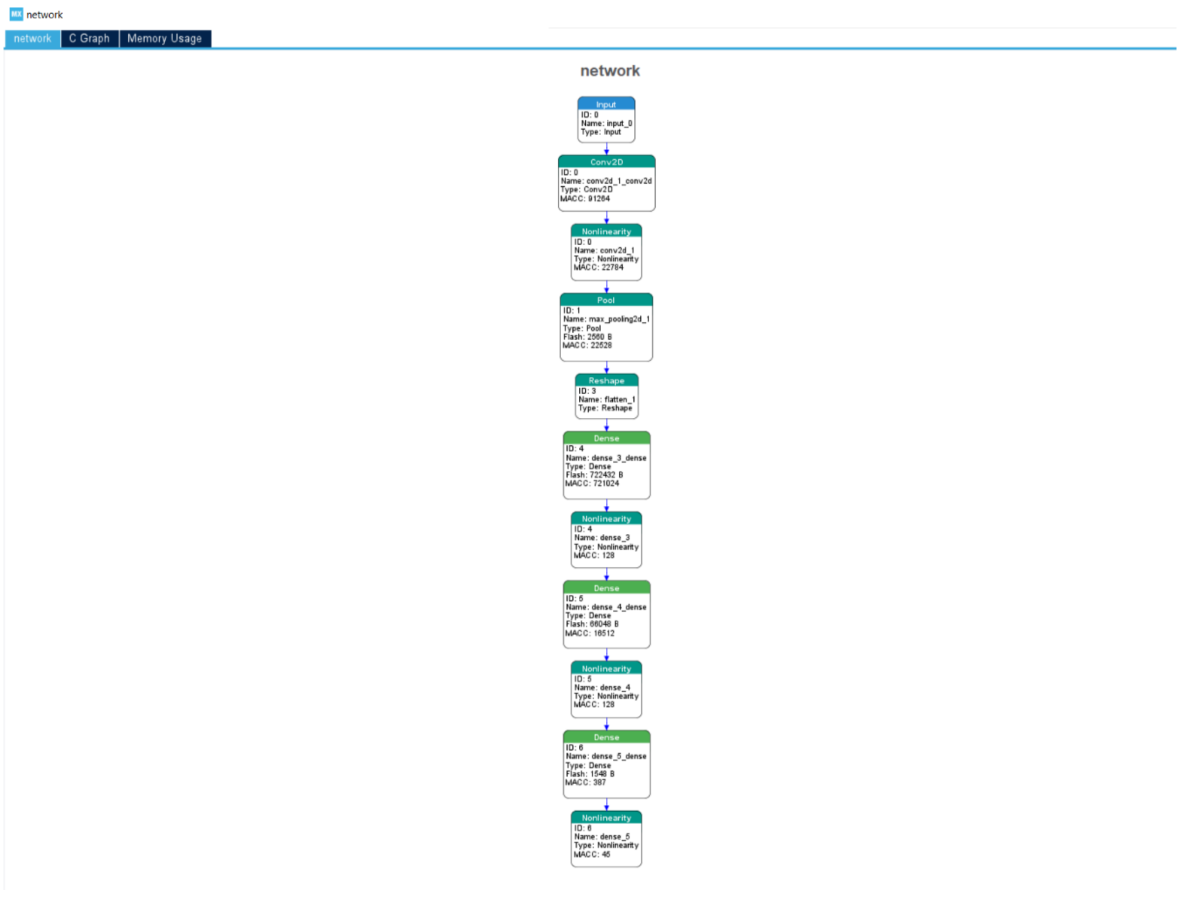
[101] K. Skawinski, F. Montraveta Roca, R. D. Findling, and S. Sigg, “Workout Type Recognition and Repetition Counting with CNNs from 3D Acceleration Sensed on the Chest,” in *Lecture Notes in Computer Science, Advances in Computational Intelligence*, I. Rojas, G. Joya, and A. Catala, Eds., Cham: Springer International Publishing, 2019, pp. 347–359.

List of Figures

List of Tables

Appendix A

Graph of the implemented Keras model from STM32CubeMX



```
Input
ID: 0
Name: input_0
Type: Input

Conv2D
ID: 0
Name: conv2d_1_conv2d
Type: Conv2D
MACC: 91264

Nonlinearity
ID: 0
Name: conv2d_1
Type: Nonlinearity
MACC: 22784

Pool
ID: 1
Name: max_pooling2d_1
Type: Pool
Flash: 2560 B
MACC: 22528

Reshape
ID: 3
Name: flatten_1
Type: Reshape

Dense
ID: 4
Name: dense_3_dense
Type: Dense
Flash: 722432 B
MACC: 721024

Nonlinearity
ID: 4
Name: dense_3
Type: Nonlinearity
MACC: 128

Dense
ID: 5
Name: dense_4_dense
Type: Dense
Flash: 66048 B
MACC: 16512

Nonlinearity
ID: 5
Name: dense_4
Type: Nonlinearity
MACC: 128

Dense
ID: 6
Name: dense_5_dense
Type: Dense
Flash: 1548 B
MACC: 387

Nonlinearity
ID: 6
Name: dense_5
Type: Nonlinearity
MACC: 45
```

Figure 1: Graphical presentation of the CNN from the practical implementation in STM32CubeMX

Appendix B

MoSCoW requirements of the practical implementation

Must

- **FR1:** The microcontroller must be able to gather movement data.
- **FR2:** The microcontroller must be able to transmit the data over Bluetooth Low Energy.
- **FR3:** The iOS app must read/receive the data from the microcontroller.
- **FR4:** The sensor data must be analyzed by a deep learning algorithm.
- **FR5:** The microcontroller must be operated without a direct power connection.
- **FR6:** The results of the analysis of the data must be saved for later use.
- **FR7:** The iOS App must offer the possibility to start and stop the recording of data.

Should

- **FR8:** The iOS App should show recordings from the past to compare the results.
- **FR9:** The gathered data should be compared to predefined movement patterns.
- **FR10:** The deep learning model should be customized for the specific use case.

Could

- **FR11:** The iOS app could show the gathered data in a graphical way.

Wont

- **FR12:** The first version of the app will run-on Android devices.
- **FR13**: The connection between the microcontroller and the iOS app will be Wi-Fi.

Appendix C

Code lines of X-CUBE-BLE1 expansion package

```
static void User_Process(void)
{
  float data_t;
  float data_p;
  static uint32_t counter = 0;

  if (set_connectable)
  {
    Set_DeviceConnectable();
    set_connectable = FALSE;
  }
    BSP_LED_Toggle(LED1);

    if (connected)
    {
      /* Set a random seed */
      srand(HAL_GetTick());

      /* Update Acceleration, Gyroscope and Sensor Fusion data */
      Acc_Update(&x_axes, &g_axes, &m_axes);
      Quat_Update(&q_axes);

      }
      HAL_Delay(100); /* wait 1 sec before sending new data */
    }
}

tBleStatus Acc_Update(AxesRaw_t *x_axes, AxesRaw_t *g_axes, AxesRaw_t
*m_axes)
{
  uint8_t buff[2+2*3*3];
  tBleStatus ret;

  HOST_TO_LE_16(buff,(HAL_GetTick()>>3));

  HOST_TO_LE_16(buff+2,-x_axes->AXIS_Y);
  HOST_TO_LE_16(buff+4, x_axes->AXIS_X);
  HOST_TO_LE_16(buff+6,-x_axes->AXIS_Z);

  HOST_TO_LE_16(buff+8,g_axes->AXIS_Y);
  HOST_TO_LE_16(buff+10,g_axes->AXIS_X);
  HOST_TO_LE_16(buff+12,g_axes->AXIS_Z);

  HOST_TO_LE_16(buff+14,m_axes->AXIS_Y);
  HOST_TO_LE_16(buff+16,m_axes->AXIS_X);
  HOST_TO_LE_16(buff+18,m_axes->AXIS_Z);

  ret = aci_gatt_update_char_value(HWServW2STHandle, AccGyroMagCharHandle,
                                   0, 2+2*3*3, buff);
  if (ret != BLE_STATUS_SUCCESS){
    PRINTF("Error while updating Acceleration characteristic: 0x%02X\n",ret)
;
    return BLE_STATUS_ERROR ;
  }
  return BLE_STATUS_SUCCESS;
}
```

```c
void Set_DeviceConnectable(void)
{
  uint8_t ret;
  const char local_name[] = {AD_TYPE_COMPLETE_LOCAL_NAME,SENSOR_DEMO_NAME};

  uint8_t manuf_data[26] = {
    2,0x0A,0x00, /* 0 dBm */  // Trasmission Power
    8,0x09,SENSOR_DEMO_NAME,  // Complete Name
    13,0xFF,0x01, /* SKD version */
    0x02,
    0x00,
    0xF4, /* ACC+Gyro+Mag 0xE0 | 0x04 Temp | 0x10 Pressure */
    0x00, /*  */
    0x00, /*  */
    bdaddr[5], /* BLE MAC start -MSB first- */
    bdaddr[4],
    bdaddr[3],
    bdaddr[2],
    bdaddr[1],
    bdaddr[0]  /* BLE MAC stop */
  };

  manuf_data[18] |= 0x01; /* Sensor Fusion */

  hci_le_set_scan_resp_data(0, NULL);

  PRINTF("Set General Discoverable Mode.\n");

  ret = aci_gap_set_discoverable(ADV_IND,
                                (ADV_INTERVAL_MIN_MS*1000)/625,
                                (ADV_INTERVAL_MAX_MS*1000)/625,
                                 STATIC_RANDOM_ADDR, NO_WHITE_LIST_USE,
                                 sizeof(local_name), local_name, 0, NULL, 0, 0);

  aci_gap_update_adv_data(26, manuf_data);

  if(ret != BLE_STATUS_SUCCESS)
  {
    PRINTF("aci_gap_set_discoverable() failed: 0x%02x\r\n", ret);
  }
  else
    PRINTF("aci_gap_set_discoverable() --> SUCCESS\r\n");
}
```

Code lines of X-CUBE-AI expansion package

```c
/*!
 * @brief Get network weights array pointer as a handle ptr.
 * @ingroup network_data
 * @return a ai_handle pointer to the weights array
 */
AI_API_ENTRY
ai_handle ai_network_data_weights_get(void);
```

```c
#define AI_NETWORK_MODEL_NAME          "network"
```

Appendix D

Additional screenshots of the implemented iOS app

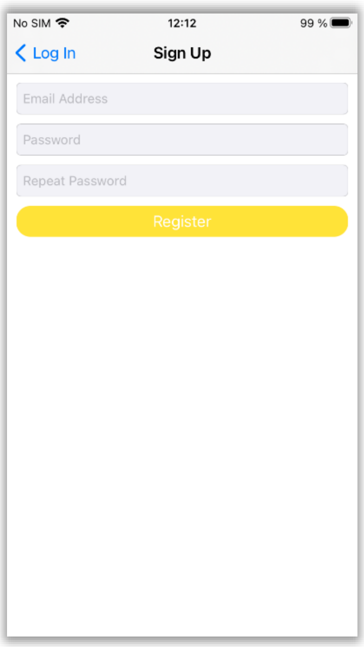


Figure 2: iOS App Sign Up Scene

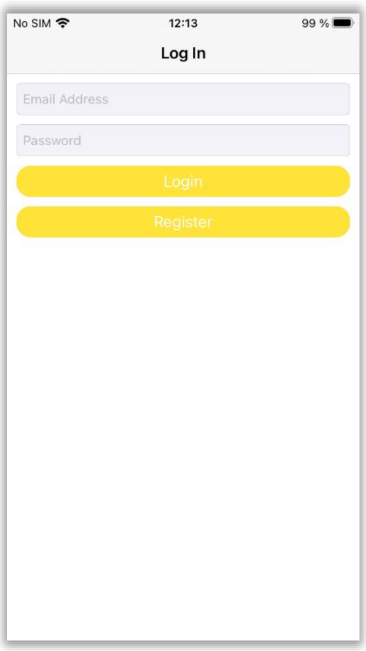


Figure 3: iOS App Log In Scene

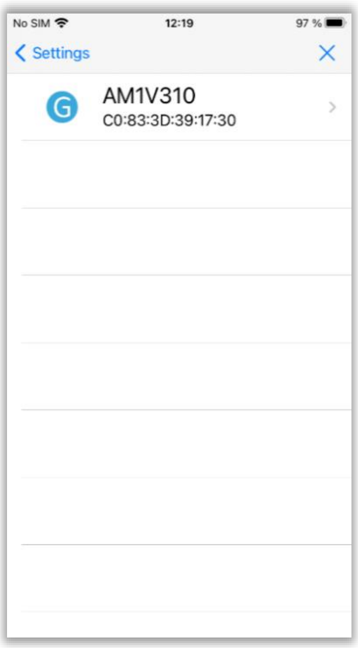


Figure 4: iOS App MCU Selection Scene

Appendix E

Python script to load & convert a MATLAB dataset

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import os
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt

dataBaseDir = r'C:\Datesets\exerciserecognitionfromwearablesensors'
dataFileSingleActivity = os.path.join(dataBaseDir, 'exercise_data.50.0000_singleonly.mat')

print("Loading single-activity data...")
exerciseDataSingleActivity = sio.loadmat(dataFileSingleActivity)
print("Loaded single-activity data...")

# In[2]:

exercises = ['Crunch', 'Bicep Curl', 'Chest Press (rack)']
activities = exerciseDataSingleActivity['exerciseConstants'][0][0]['activities'][0]
nActivityTypes = len(activities)
#print('Available activities:\n', activities)
print(f"Number of overall activity types in the dataset: {nActivityTypes}")

nParticipants = exerciseDataSingleActivity['subject_data'].shape[0]
print(f"Number of overall participants in the dataset: {nParticipants}\n")

allAccelData = []
allGyroData = []

def add_accelData_from_exercise_to_allAccelData(exerciseName):
    exerciseIndex = np.where(activities == exerciseName)[0][0]
    print(f"\nIndex of exercise '{exerciseName}': {exerciseIndex}\n")

    for participantId in range(nParticipants):
        recordings = exerciseDataSingleActivity['subject_data'][participantId][exerciseIndex]

        if recordings.shape != (1, 0):
            recording = recordings[0]
            for i in range(recording.shape[0]):

                accelData = recording['data'][i][0]['accelDataMatrix'][0]
                accelData = accelData.astype(object)

                accelData = np.insert(accelData, 0, participantId, axis=1)
                accelData = np.insert(accelData, 1, exerciseName, axis=1)

                allAccelData.append(accelData)

                print(f"Appended recording from participant '{participantId}' with shape {record-
ing.shape}")
        else:
            continue

    print(f"\nNumber of added accel data from exercise '{exerciseName}': {len(allAccelData)}")

for exerciseName in exercises:
    add_accelData_from_exercise_to_allAccelData(exerciseName)
```

```python
# In[3]:

from sklearn.preprocessing import MinMaxScaler

normalizedDataset = []

def show_plot_of_accelData_from_exercise(accelData):
    legendArray = ['X', 'Y', 'Z']
    accelT = accelData[:, 2]
    accelXYZ = accelData[:, 3:6]

    plt.subplot(2, 1, 1)
    plt.plot(accelT, accelXYZ)
    plt.xlabel('Time (seconds)')
    plt.ylabel('Accelerometer output (g)')
    plt.legend(labels = legendArray, loc='center left', bbox_to_anchor=(1, 0.5))
    plt.show()

# Apply transformation using MinMaxScaler for each array in the data structure
scaler = MinMaxScaler()
original_data_structure = []
for accelData in allAccelData:
    numeric_data = accelData[:, 2:].astype(float)
    original_data = scaler.fit_transform(numeric_data)
    original_array = np.concatenate((accelData[:, :2], original_data), axis=1)
    original_data_structure.append(original_array)

# Plot each row of the normalized accelerometer data
for row in original_data_structure:
    show_plot_of_accelData_from_exercise(row)
    for value in row:
        normalizedDataset.append(value)

# In[4]:

import csv

# Specify the txt file path
txt_file = 'Dataset.txt'

# Open the file in write mode with newline=''
with open(txt_file, 'w', newline='') as file:
    writer = csv.writer(file, delimiter=',')

    # Write the data rows
    writer.writerows(normalizedDataset)

print(f"Txt file '{txt_file}' has been exported successfully.")
```