University of Applied Sciences
HOCHSCHULE EMDEN·LEER

Project Report

Magic Wand using Arduino Nano

Author: Adhiraj Walse
Matriculation No.: 7025711

Author: Sudeshna Nanda
Matriculation No.: 7026003

Author: Srikanth Nanda
Matriculation No.: 7026002

Course of Studies : Business Intelligence and Data Analytics

First examiner: Prof. Dr. Elmar Wings
Submission date: January 31, 2025

University of Applied Sciences Emden/Leer · Faculty of Technology ·
Mechanical Engineering Department
Constantiaplatz 4 · 26723 Emden · http://www.hs-emden-leer.de

Contents

List of Figures

List of Listings

Acronyms

API Application Programming Interface

CNN Convolutional Neural Network

FOG Fiber Optic Gyro

IDE Integrated Development Environment

IMU Inertial Measurement Unit

IoT Internet of Things

KDD Knowledge Discovery in Databases

LED Light Emitting Diode

MEMS Microelectromechanical Systems

RGB Rot-Grün-Blau

RTOS Real Time Operating System

tf TensorFlow

TFLite TensorFlow-Lite

TinyML Tiny Machine Learning

Part I.

Introduction

1. Introduction

The development of Machine Learning and the evolution of Internet of Things (IoT) have progressed hand-in-hand within these several years. The IoT Idea has permeated various aspects of our lives, such as healthcare, agriculture, intelligent cities, etc.[**Had:2020**]. Also, lots of IoT applications should be able to process data to respond instantly. Handling such requirements by cloud computing is then not suitable[**shi:2016**].

Tiny Machine Learning (TinyML), therefore, is a rapid enhancement in machine learning area due to its low-latency, low power and bandwidth model inference at edge devices[**sakr:2020**]. TinyML primarily employs small form factor devices, for instance, Real Time Operating System (RTOS) based microcontrollers, to run machine learning models and algorithms[**anh:2009**]. Also, one of the most striking features of TinyML is its small size and, in some cases, its ability to function on battery power for years[**Aba:2023**]. This technology heralds a new era of edge services and applications that are not reliant on cloud processing but rather thrive on distributed edge inference and autonomous reasoning.

Arduino is one of the critical components of this project. It allows for the control of board behavior via instructions relayed to the board's microcontroller. Such instructions are provided using the Arduino Integrated Development Environment (IDE), based on Processing and the Arduino programming language. Arduino's original purpose was to serve as a simple prototyping tool for students without prior knowledge of electronics or programming. However, as the platform developed, Arduino boards began to diversify, expanding their scope from simple 8-bit boards to devices designed for IoT applications, wearable technology, 3D printing, and embedded environments[**kushner:2011**]. This adaptability has enabled Arduino to meet new challenges and needs effectively. The Arduino IDE and its Programming Language allow for a high extent of customization to meet the user's requirements, making it a favored development platform for rapid prototyping and idea validation[**kushner:2011**].

This paper delves into using the 3.3V board Arduino Nano 33 BLE Sense. The hardware features three-axis accelerometers, indicating its ability to detect motion and calculate acceleration in three dimensions[**Ard:2021**]. The objective of this report is to construct a "Magic Wand" and integrate it with the board Arduino Nano 33 BLE Sense. The board is programmed to recognize predefined gestures, indicating movement direction for gesture recognition.

The Light Emitting Diode (LED) can be programmed to flash in response to hand movement commands in Arduino. When the wand is moved in a specified direction, the board interprets the gesture and converts it into a visual output. To create the magic wand, the board Arduino Nano 33 BLE Sense is attached to the end of a stick, and the accelerometer's data is fed into a deep-learning model to ascertain whether a recognized gesture has been made. The Arduino board's integral multi-dimensional detector is used to collect gestures, which are vital for understanding complex data. A model can be trained to understand and embed this complex data into the board Arduino Nano 33 BLE Sense.

Additionally, TensorFlow-Lite (TFLite) is used to implement a Neural Network model to recognize gestures using an accelerometer. Triumphant gesture casting results in visualising the corresponding gesture on the screen and the Arduino board's LED lighting up in response to the human input.

A heuristic function is integral to this application. It encodes gesture knowledge into

code, requiring programming skills, mathematical knowledge, and domain expertise. The data collected from the accelerometer undergoes mathematical conversions to produce the desired output or the intended gestures. Instead of creating a heuristic method from scratch, users can select an appropriate model architecture, gather and label a dataset, and iteratively build a model through training and evaluation. The model uses the accelerometer data as input without pre-processing, performs inference, and analyses the results. Three movements have been trained: wing (w), ring (o), and slope (/). The direction of motion for the gestures is shown in the figures 1.1 1.2 1.3. If the input is valid, it generates a visual output of the gestures on a terminal and responds to each spell by lighting an LED. There is also an output to denote an "unknown" gesture if any movement is not recognized.These "unknown" gestures are the motions which are not belong to the wing, ring and slope movements.

Figure 1.1.: **Wing Gesture [WS20b]**

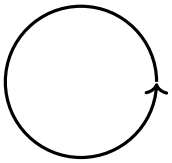


Figure 1.2.: **Ring Gesture [WS20b]**

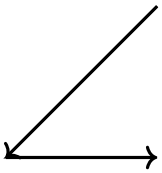


Figure 1.3.: **Slope Gesture [WS20b]**

The planning process encountered several challenges, including:

- The size of the data model should be controlled at a reasonable level due to the limitation of memory in board Arduino Nano 33 BLE Sense [**shi:2016**]
- Creating a Machine Learning data model for the four gestures, especially the "unknown" gesture
- The size of the data model should be controlled at a reasonable level due to the limitation of memory in board Arduino Nano 33 BLE Sense [**Ard:2021**]
- Training the data model to get a more accurate result [**shi:2016**]
- Avoiding extended waved gestures that could lead the accelerometer to record false readings and generate incorrect results.

1.1. Problem's Description

Deploying deep learning (DL) models on edge devices like microcontroller units (MCUs) is a significant challenge due to their limited resources. Most DL algorithms are

designed for high-performance systems such as GPUs or high-performance computing clusters, which makes them incompatible with the memory, computational, and energy constraints of MCUs. This poses a critical issue when adapting DL models for applications requiring real-time processing, such as industrial gauge inspection, where models must be optimized without losing significant accuracy or efficiency.

1.2. Challenges

- **Resource Constraints:**
  Limited memory, computational power, and energy efficiency in MCUs restrict the deployment of traditional DL models. High-performance computer vision models exceed the capacity of edge devices.[**Ard:2021**]
- **Accuracy vs. Efficiency Trade-off:**
  Reducing model size and computational complexity often results in performance degradation, particularly for computer vision tasks.[**shi:2016**]
- **Framework Limitations:**
  Existing tools like TensorFlow Lite and AIfES offer optimization options but are not fully adapted to handle complex computer vision requirements on constrained hardware.[DMM20]
- **Real-world Application Needs:**
  Industrial applications, such as gauge inspection, require fast and accurate real-time image processing, which remains challenging to achieve on MCUs with existing models.[DMM20]

1.3. Solutions

- **Optimization Techniques:**
  - **Quantization:** Convert weights and activations from 32-bit floating-point to lower-precision formats (e.g., 8-bit integers) to reduce memory and computation demands.
  - **Pruning:** Eliminate redundant parameters in neural networks to reduce complexity without significantly affecting accuracy.
  - **Knowledge Distillation:** Use large, pre-trained models to guide the training of smaller, more efficient models.[Mac+23]
- **Specialized Frameworks and Tools:**
  - **TensorFlow Lite (TFLite):** Supports model optimization techniques like quantization and pruning and enables efficient execution on embedded devices.[DMM20]
  - **AIfES:** Tailored for Arduino-based systems, allowing compact model creation and adaptation for 8-64 bit systems.
  - **TinyNeuralNetwork:** Simplifies model compression and supports seamless integration between TensorFlow Lite and PyTorch.
  - **CMSIS-NN:** Provides optimized neural network kernels specifically designed for ARM Cortex-M processors.[DMM20]
- **Real-world Validation:**
  Implement and test optimized DL pipelines for tasks such as industrial gauge inspection, leveraging quantized and pruned models to meet performance and resource constraints. Use converted lightweight models for real-time processing tasks, ensuring that accuracy is maintained within acceptable thresholds.[DMM20]

1.4. Report Structure

Following the introduction, our report progresses to a comprehensive exploration of domain knowledge in second section. This section focuses on the nuances of data collection (**??**), management, and its pivotal role, alongside an in-depth analysis of both the hardware (Arduino Nano 33 BLE Sense) in 4 and the utilized software suite (Arduino IDE, TensorFlow, Python, etc.) in **??** Subsequently, we delve into the utilization of Convolutional Neural Network (CNN) in the realm of data mining in 9, elucidating their components and their pertinence to our project. The report then navigates through section 3, the intricacies of the Knowledge Discovery in Databases (KDD) Process (**??**), covering its development (10), deployment (11), and the materials involved (12). In section 4 conclusion, the report culminates by summarizing the principal discoveries and proposes avenues for future exploration and development.

2. Data Version Control

Data Version Control (DVC) is an essential practice in managing data science, machine learning projects, and software development, especially when collaboration is involved. It extends version control systems to handle large data files, models, and experiments, ensuring reproducibility, collaboration efficiency, and resource management.

Here are some reasons that we should have the data version control plan during the project:

1. **Track Changes:** Our source code, data changes over time. Tracking these changes allows you to revert to previous versions if needed and understand how your data evolved.

2. **Collaboration:**When working on projects with others, it's crucial to ensure everyone uses the same version of datasets and models. Version control helps synchronize data across teams.

3. **Experimentation:**Machine learning involves lots of experimentation. Tracking data versions alongside code changes helps in reproducing experiments accurately.

2.1. Plan

The first step is to finish the installation of GitHub among all the teammates and have the same version as our version control on the report, manual and even presentations mainly builds on Git's versioning capabilities. After completion of installation, we asked for professor to release the right for us to join the repository and initialize on our side about the project. To update our report and other documents, everyone would send a message in our communicating group first so other would not fetch and push new updates at that moment so crushing on the documents can also be prevented

Secondly, for our different softwares that we used in the project, we have firstly aligned our software version among everyone to make sure all the results replicate under a same environment and prevent issues due to the version difference. All the software aligned versions are stated in the following table.

Thirdly, the data version control of data set is only done on one computer of our teammates. Therefore, every changes in our data set will stay in one computer to prevent the data or code corruption from the others. Moreover, our data set would save every week as a new version in the local computer so that we can track the code again in case there are issues generated in the new version. Our data model and data set can still step back to make sure they can run and get results properly.

Github Version	Version 3.3.8 (x64)
TexStudio Version	TeXstudio 4.6.3
Pycharm Version	2023.2.1
Arduino IDE version	Arduino IDE 2.2.1
Python Tensorflow	2.15.0
Numpy	1.23.5
pandas	1.5.3
matplotlib.pyplot	3.7.1
pathlib	1.0.1
shutil	3.10.12 Built in Python
PIL	9.4.0
math	3.10.12 Built in Python
glob	0.7
json	2.0.9
os	3.10.12 Built in Python
Arduino LSM9DS1 version	1.1.1
Arduino Mbed OS Nano Boards Version	4.0.10
Arduino Tensorflow Lite Version	2.1.0

Table 2.1.: Software and Hardware Version Controls

2.1.1. Installation

Arduino IDE should be installed for every teammate. So a detail installation steps should be provided and extra libraries also includes in this explanation, especially the Tensorflow lite in the Arduino IDE. We would provided requirements of two operation systems that our teammates mainly utilize. The requirements of the Arduino IDE is as follow: Version: Arduino IDE 2 Window System Requirements: Win 10 (64-bit) or newer macOS: 10.14: "Mojave" or newer, 64-bit
The files can be found in the Arduino official website - window version: `https://www.arduino.cc/en/software` MacOS version: `https://www.arduino.cc/en/software` we have also store the files in the cloud to resist from the lose in online. Here is the link: `https://drive.google.com/drive/folders/10TSlasawgUHUaGNCW2ls1JDevt0onTXi?usp=drive_link` Currently, we have uploaded the following files in the drive so that others can still perform the project properly.

- Window .exe file
- MacOS .dmg file
- Arduino Library - Arduino LSM9DS1
- Arduino Library - Tensorflow lite

The detailed installation steps can be found from the software section of Arduino IDE. The two boards can only be installed when you have Arduino IDE thus we do not include these files in the drive. 9.1.1 After that, the Arduino IDE should install the two boards, Tensorflow lite and LSM9DS1.

Part II.

Domain Knowledge

3. Application

3.1. Application of Magic Wand with an Arduino Nano 33 BLE Sense

The Magic Wand project aims to develop a handheld device that can interact with a computer or other systems through Bluetooth Low Energy (BLE) technology. The device is designed to recognize specific gestures made by the user and translate them into commands or actions. The Magic Wand uses the Arduino Nano 33 BLE Sense board, which is equipped with various sensors, including an accelerometer, gyroscope, magnetometer, and temperature sensor. These sensors allow the device to capture the user's movements and orientation in real-time, enabling gesture recognition and interaction with the system. The Magic Wand project involves programming the Arduino Nano 33 BLE Sense board to read sensor data, process the data to recognize gestures, and communicate wirelessly with a computer or other devices. The project also includes developing a user interface or application on the computer to receive and interpret the gesture commands sent by the Magic Wand. The Magic Wand project demonstrates the capabilities of the Arduino Nano 33 BLE Sense board and explores the possibilities of gesture-based human-computer interaction using BLE technology. The project has various applications in fields such as gaming, virtual reality, human-computer interaction, and assistive technology. By developing a responsive and efficient gesture recognition system, the Magic Wand project aims to provide an intuitive and interactive way for users to control and interact with digital systems and devices. The project also highlights the potential of using low-cost and accessible hardware platforms like the Arduino Nano 33 BLE Sense for developing innovative and interactive applications in various domains. The Magic Wand project showcases the power of combining hardware, software, and sensor technologies to create novel and engaging user experiences and demonstrates the possibilities of using gesture-based interaction as a means of human-computer communication. The project also emphasizes the importance of user-centered design and usability testing to ensure that the Magic Wand is intuitive, easy to use, and accessible to a wide range of users. By exploring the capabilities of the Arduino Nano 33 BLE Sense board and developing a gesture recognition system, the Magic Wand project aims to inspire creativity, innovation, and exploration in the field of interactive technology and human-computer interaction. The Magic Wand project represents an exciting opportunity to experiment with sensor technologies, wireless communication, and gesture recognition algorithms and to create a unique and engaging user experience that combines physical movement with digital interaction. The project has the potential to open up new possibilities for interactive technology and to inspire future projects and applications that leverage the capabilities of the Arduino Nano 33 BLE Sense board and other sensor-based platforms.

3.2. Problem

The project represents an exciting challenge in interactive technology using board Arduino Nano 33 BLE Sense, whereby the project derived is referred to as the Magic Wand. This project aims to develop and implement a handheld device similar to the magic wand by using the board Arduino Nano 33 BLE Sense due to its characteristics such as wireless communication implementation. The wand is expected to interact

with a suitable system, such as a computer, through the implementation of Bluetooth Low Energy (BLE) technologies. The challenge would be to develop a responsive and efficient gesture recognition system that can remap the wand movement of the user into actual commands. Moreover, the project might incorporate sensors like accelerometers or gyroscopes for accurately capturing and interpreting gestures. Thus, Successful implementation requires an understanding of programming Arduino, sensor integration, and BLE communication protocols. In this Magic Wand project, it will be essential to resolve the intricacies by documenting the board Arduino Nano 33 BLE Sense, sensor datasheets, and other relevant online resources covering gesture recognition algorithms. For this reason, the project not only comes up with a fantastic and exciting way of testing board Arduino Nano 33 BLE Sense capabilities but also opens new horizons in gesture-based human-computer interaction.

3.3. Data Acquisition

3.3.1. Database

In the lack of pre-existing databases adapted to our specific model, we developed a new database for our Magic Wand project. The information is organised as a.txt file and contains raw data from accelerometer readings taken by the Arduino Nano 33 BLE Sense. Recognising the inherent heterogeneity in how people capture gestures, we asked three members of our group to contribute, resulting in a richer and more diverse dataset. Each participant scrupulously documented at least 50 trials of each gesture W, O, and L to ensure a complete representation of gesture variability. This strategy takes advantage of the distinct hand movements of three different users, improving the database's accuracy, efficiency, and efficacy. [WS20b]

In the next part of our project, named Data Preparation and Quality," we intend to process the raw data collected from the database. This includes identifying and resolving outliers as needed, as well as further refining the dataset to ensure optimal performance when training our machine learning model. The comprehensive quality of our dataset, enriched by diverse contributions and purposefully limited in size, provides a solid foundation for the project's succeeding phases, promising an effective and adaptable model for Magic Wand gesture identification. Detailed Information about database in teh section **??**.

3.3.2. Data Preparation

This section discusses how to shape our data and how we can use it for training. Our data model will recognise three gestures: wing, slope and ring. Since we do not intend to use existing databases to prepare the data, we will give multiple inputs for a single gesture and save it as model data for the gesture. When a gesture, for example - "wing" is waved using the wand, the accelerometer present in the microcontroller detects the movement by using the coordinates in the x,y and z axes. The exact process is repeated multiple times for the dataset to be as large as possible. Creating a database with more inputs allows the data model to be more accurate and provides better and more efficient results. The exact process is repeated for the remaining gestures, namely ring, slope and unknown. With the data available for the above gestures, we can use it to train a model. The available data is split into two sets, namely training data and test data.

The major challenge while preparing data can be the presence of outliers or anomalies with unexpected values. This can be overcome by feeding our model with more data so that the model is efficient.[WS20b]

The different steps involved in preparing data can be as follows according to [WS20b]

Understanding the problem The problem, which in our case is the collection of data consisting of gestures. The main problem is divided into many parts for easy data preparation.

Preparation of data The step deals with converting the raw data into machine learning language that the compiler can interpret.

Evaluation of data This step deals with evaluating the model after the collection of the necessary data. This data is then checked for correctness, and analysis is done according to the accuracy of the output obtained.

Finalizing the data The model is tested with various parameters and the data is validated. If the results obtained are according to our requirements, the resulting dataset is finalized.

The same steps are repeated for various gestures and the final database is created.

Preparation of Datasets for the Magic Wand

Since the amount of data required for the creation of a database is very small, the database will be prepared by us. The first set of data that we are trying to capture the movement is for the gesture "W". The steps involved in capturing a W are mentioned below. [WS20b]

Figure 3.1.: **Wing Gesture [WS20b]**

- The device is first moved down and to the right.
- Then the device is moved up and to the right.
- The device is then moved down and to the right.
- The device is then moved up and to the right again.

Shows a sample of real data captured during the "wing" gesture, measured in milli-Gs. The process involved in capturing a ring is as follows. [WS20b]

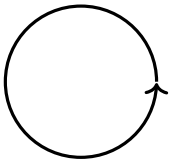


Figure 3.2.: **Ring Gesture [WS20b]**

- Trace a clockwise circle using the wand.
- Aim again to take around a second to perform the gesture.

The steps involved in waving the slope gesture are mentioned below [WS20b]:

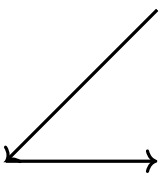


Figure 3.3.: **Slope Gesture [WS20b]**

- The device is first moved down and to the left.
- Then the device is moved to the right.
- Finally you should get a corner of the triangle as shown in the figure.

The process is repeated until around 15 readings are captured by the wand for Data Preparation and Transformation.

In order to consider the unknown readings, we will be carrying out another set of procedures to feed with unknown data. This would help the wand recognize any unknown gesture and give the output that it is false. [WS20b]

All readings are recorded in a unique text file for each gesture. The files `.txt` [Dat] contain raw accelerometer data that would later be transformed into machine learning language that should be interpreted by the compiler. [WS20b]

Since no existing databases exist for the model, a new database will be created.The database is a .txt file which contains raw data from the accelerometer readings recognized by the board Arduino Nano 33 BLE Sense. Since the way a person records the gesture varies from one to another, the gestures of three people are recorded as we are a group of three members. In order to the database to have a vast amount of data, each person will record at least 50 trials for each gesture. In order for the database to be more accurate data from both hands of the three users were used. This would improve the efficiency and effectiveness of the database making it more concrete. It is also possible to add further data to our model, improving it over time. The size of the database thus created will be petite, less than 2 MB. Obtaining sufficient data is one of the major concerns in a machine learning project and the amount of data involved in our data set is very small. The data thus created will be further used in the next step “Data Preparation and Transformation” where this raw data is further processed and the outliers are identified and removed if necessary.

3.4. Data Quantity

Edge Computing

The placement of computer and storage resources at the point where data produced is known as edge computing. This computes and stores the data source at the network edge, which is optimal. In other words, instead of sending raw data to a main data centre for processing and analysis, this work is done right where the data is generated. Edge computing is used to handle discrete tasks like determining if someone answered "yes" and responding appropriately. Instead of comparing the data with that on the web, the audio analysis is done on the edge. This drastically decreases expenses and complexity while also reducing the risk of data breaches.[Big21]

Edge computing has gained traction as a viable solution to a variety of issues related to transporting the massive amounts of data that today’s businesses generate and consume. It’s not just a matter of quantity, it’s also a matter of time; applications increasingly rely on processing and responses that are time-sensitive.The analysis and analysis of data produced by connected devices is made possible by edge computing, a crucial element of the Internet of Things (IoT).[STH24] claims that edge computing enables data collecting, processing, and analysis at the network’s edge, facilitating in-the-moment decision-making and minimizing the need to send data to a central point for processing. Edge computing reduces processing times for data, allowing businesses to react swiftly to shifting operational circumstances.

3.5. Data Quality

Creating a magic wand with board Arduino Nano 33 BLE Sense involves integrating various sensors components and programming logic to ensure data quality for a successful project you need to consider the reliability and accuracy of the data collected from sensors here are some key considerations[Dai].

Sensor Calibration

Sensor calibration calibrate sensors such as accelerometers gyroscopes or magnetometers to ensure accurate readings understand the sensor s specifications and calibrate them accordingly to reduce errors in measurements

Sensor Fusion

Sensor fusion use sensor fusion techniques to combine data from multiple sensors this can improve the accuracy and reliability of the data implement sensor fusion algorithms to obtain a more comprehensive understanding of the wand s orientation and movement

Wireless Communication Reliability

Wireless communication reliability if your magic wand involves wireless communication ensure the bluetooth low energy ble connection is stable and reliable implement error checking mechanisms to handle potential data transmission issues [GOP12]

Code Optimization

Write efficient and optimized code to minimize delays and improve the real-time response of the magic wand. Optimize algorithms to reduce processing time and enhance overall system performance.

3.6. Data Relevance

Real-Time Machine Learning

Real Time can be considered as a way of training the model by making it run through live data constantly in order to improve the model. This contradicts the traditional way of machine learning when machine learning engineers were dependent already existent data inputs for creating the model. A method by which we can implement real time learning in to a machine learning model is by continuously feeding it with a data stream and improving the model over time. This is achieved by using an event driven architecture. Event driven architecture is a modern design and development approach that centers about the events occurring in the model. Event-driven architectures create, detect, consume, and react to events.[STH24]. By using a scalable event driven architecture, large number of events in real time can be responded, to which are suitable for loosely coupled softwares(For eg: web services). They also work well with unpredictable and non linear events by making it versatile. [STH24]

3.7. Outliers

Outliers refer to exceptional or irregular data points that deviate significantly from the expected sensor readings or user interactions. These anomalies can arise from various sources such as sudden movements, extreme sensor values, interference, or unexpected user inputs.outliers in sensor data might occur if the accelerometer or gyroscope

registers an unusually rapid or erratic movement that does not align with typical usage patterns. Similarly, outliers in user interaction may manifest as unintended button presses or variations in touch sensor sensitivity.
Detecting and handling outliers is crucial for ensuring the accuracy and reliability of the magic wand's functionality. Strategies may involve implementing filtering algorithms, calibration techniques, and debouncing mechanisms to mitigate the impact of outliers on sensor readings and user inputs.[MO19]

3.8. Anomalies

Creating a magic wand with Arduino Nano 33 BLE involves considering potential anomalies or unexpected behaviors that might arise during the development and usage of the wand. The major challenge while preparing data can be the presence of outliers or anomalies where there are unexpected values. This can be overcome by feeding our model with more data so that the model is efficient.[WS20b]

Sensor Anomalies

- Accelerometer/Gyroscope Drift - Sensors like accelerometers and gyroscopes may experience drift over time, leading to inaccuracies in orientation tracking. Calibration techniques can help mitigate this issue.[Cho+21]
- Magnetic Interference - Magnetometers can be affected by nearby magnetic fields, leading to anomalies in readings. Shielding or compensation algorithms may be required.

Power Anomalies

- Unexpected Power Drain - Identify and address any unexpected power consumption patterns. Implement power-efficient code and periodically monitor power usage to detect anomalies.
- Battery Voltage Fluctuations - Voltage fluctuations in the power supply can affect the stability of the Arduino Nano 33 BLE. Implement voltage regulation and monitoring to handle such anomalies.

Firmware and Software Anomalies

- Memory Issues - Anomalies may occur due to memory limitations on the Arduino Nano 33 BLE. Monitor and optimize memory usage to avoid crashes or unexpected behavior.
- Firmware Bugs - Identify and address potential bugs in your firmware. Regular testing, debugging, and code reviews can help mitigate software anomalies.

4. Hardware Description

4.1. Board Arduino Nano 33 BLE Sense

The Arduino Nano 33 BLE Sense board serves as a prime example of edge computing. It integrates multiple sensors alongside BLE connectivity and wireless communication features [Raj19], making it suitable for a wide range of IoT and wearable applications. Equipped with sensors for temperature, pressure, magnetometry, acceleration, and gyroscopic measurements [Raj19], the board also includes a microphone and a proximity sensor. Its built-in BLE capabilities enable seamless interaction with other BLE-compatible devices, such as smartphones, facilitating remote data collection and control [**Bagur:2023**]. At Magic Wand, our focus lies in leveraging the Accelerometer, Gyroscope, and Magnetometer sensors. The Arduino Nano 33 BLE Sense stands out as a versatile board, ideal for applications requiring sensor integration and wireless communication in IoT and wearable technologies [Raj19].
The Arduino Nano 33 BLE Sense is a compact, microcontroller-based development board designed to operate at a maximum of 3.3V. It is essential to avoid exceeding this voltage on its digital and analog pins [Ard21]. The board features a Bluetooth Low Energy (BLE) module, making it suitable for IoT applications [**Bagur:2023**]. Powered by an nRF52840 processor, it boasts a 64 MHz clock speed, 256 KB of SRAM, and 1 MB of flash memory. Additionally, it includes 14 digital I/O pins, eight of which are analog input pins for connecting external components and sensors. The board's power consumption is remarkably low, drawing only 10 mA per I/O pin [Ard21].
The Arduino Nano 33 BLE Sense supports high-level programming languages similar to C/C++ [**Emeritus:2023**]. Programming for microcontrollers, including Arduino, is done on a host computer. Code is written and compiled using Arduino's Integrated Development Environment (IDE) before being uploaded to the microcontroller for execution. To connect the board, a micro-USB cable is used, linking the board to a laptop. Once powered, a green LED next to the micro-USB port indicates a successful connection.
Additionally, Arduino offers several versions of the Nano 33 BLE board. This includes the Nano 33 BLE Sense and Nano 33 BLE Sense Lite. The Lite version lacks the HTS221 temperature and humidity sensor, instead featuring the LPS22HB pressure sensor, which can measure temperature but not humidity [Ard22]. Since our project focuses on the IMU sensor for gesture recognition, both versions are suitable for use. The following descriptions and hardware tests are based primarily on the Arduino Nano 33 BLE Sense.

4.2. Interfaces

4.2.1. Board Arduino Nano 33 BLE Sense: Components Overview

The Arduino Nano 33 BLE Sense is equipped with several embedded sensors, including humidity, temperature, barometric pressure, proximity, and a microphone. These built-in components allow the board to be utilized for various practical applications without the need for additional circuitry [Ard21]. It supports standard communication protocols such as UART, I2C, and SPI, enabling seamless interaction with external circuits and sensors for data exchange. The board includes a micro-USB port for connecting to a laptop or desktop, which serves as both a power supply and a medium

Figure 4.1.: **Top View of Board Arduino Nano 33 BLE Sense** [Ard23]

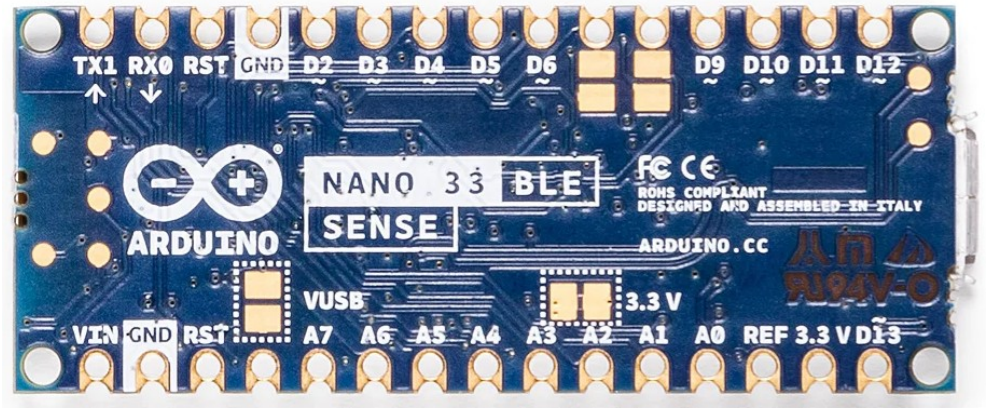


Figure 4.2.: **Bottom View of Board Arduino Nano 33 BLE Sense** [Ard23]

for file or data transfer via serial communication. While operating the board, it is crucial to ensure that no more than 3.3V is applied to its pins, as exceeding this voltage can result in permanent damage [Ard21].

Below is a detailed description of each sensor embedded on the board:

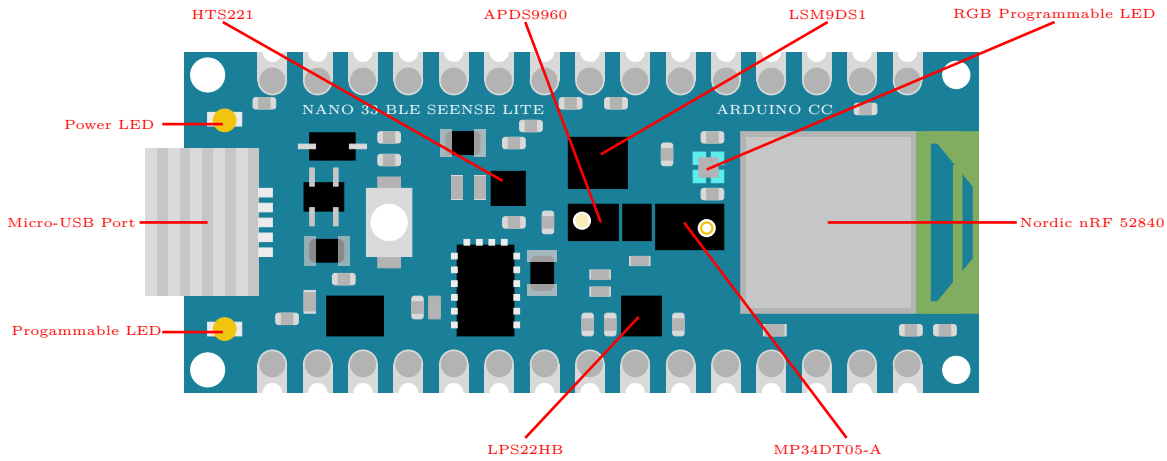


Figure 4.3.

- LSM9DS1 - Inertial Measurement Unit (IMU) features a 3D accelerometer, gyroscope and magnetometer and allows you to detect orientation, motion or vibrations in your project [**Alushi:2023**].
- APDS-9960 - The APDS-9960 chip allows for measuring digital proximity and ambient light as well as for detecting RGB colors and gestures [Ava15].
- LPS22HB - The LPS22HB picks up on barometric pressure and allows for a 24-bit pressure data output between 260 and 1260 hPa. This data can also be processed to calculate the height above sea level of the current location [**Stm:2017**].
- HTS221 - The HTS221 capacitive digital sensor measures relative humidity and temperature. It has a temperature accuracy of ± 0.5 °C (between 15-40 °C) and is perfectly suited to detect ambient temperature [**Stm:2023**].
- MP34DT05 - The MP34DT05 microphone allows you to capture and analyze sound in real-time and can be used to create a voice interface for your project[**Stm:2021**].
- USB port - USB port allows you to connect Arduino Nano 33 BLE sense to your machine.
- LEDs - There are 3 different LEDs that can be accessed on the Nano BLE Sense: Rot-Grün-Blau (RGB)(Programmable LED), the built-in LED(Programmable LED) and the power LED

LEDs function in Board Arduino Nano 33 BLE Sense

Apart from the Power LED indicate the board is powered, there are two LEDs in the board: Programmable LED(orange) and the RGB Programmable LED. The orange LED sparkles when it is connected to the computer. The RGB LED can be used during the creation of the actions that we perform. For example, in our project, each gesture can use one color to represent a specific gesture, indicating as the recognizing function.

4.3. Arduino Nano 33 BLE Pin Configuration

The **Arduino Nano 33 BLE** is an advanced version of the Arduino Nano board that is based on a powerful processor, the nRF52840. The following is the pin configuration of the board:

4.3.1. Pin Configuration

Digital Pins:

- The board has **14 digital I/O pins** that receive only two values: HIGH or LOW.
- These pins can function as input or output based on the requirement.
- When the pins receive 5V, they are in a HIGH state; when they receive 0V, they are in a LOW state.

Analog Pins:

- The board has a total of **8 analog pins** (A0–A7).
- These pins measure analog voltage ranging between 0 to 5V and can get any value as opposed to digital pins, which only receive HIGH or LOW values.

PWM Pins:

- All digital pins can be used as PWM pins.
- These pins generate analog results using digital means.

SPI Pins:

- The board supports the **Serial Peripheral Interface (SPI)** communication protocol.
- SPI is used to communicate between the controller and peripheral devices such as shift registers and sensors.
- Two pins, **MISO** (Master Input Slave Output) and **MOSI** (Master Output Slave Input), are used for SPI communication.

I2C Pins:

- The board supports the **I2C communication protocol**, a two-wire protocol.
- It includes two pins: **SDL** and **SCL**.

UART Pins:

- The board features the **UART communication protocol** for serial communication.
- It includes two pins: **Rx** (receiving pin) and **Tx** (transmitting pin).

External Interrupts Pins:

- All digital pins can be used as external interrupt pins.
- This feature allows the main running program to be interrupted and handle emergency instructions.

LED at Pin 13 and AREF Pin:

- There is an **LED connected to pin 13** of the board.
- The **AREF pin** is used as a reference voltage for input voltage.

4.4. Data Quality in Hardware Description

In the hardware description of our Magic Wand project with Arduino Nano 33 BLE Sense, we consider the following aspects related to data quality:

1. **Sensor Accuracy:** The sensors embedded in the Arduino Nano 33 BLE Sense board deliver precise measurements of environmental parameters such as motion, orientation, temperature, humidity, and pressure [**Dhow:2024**].

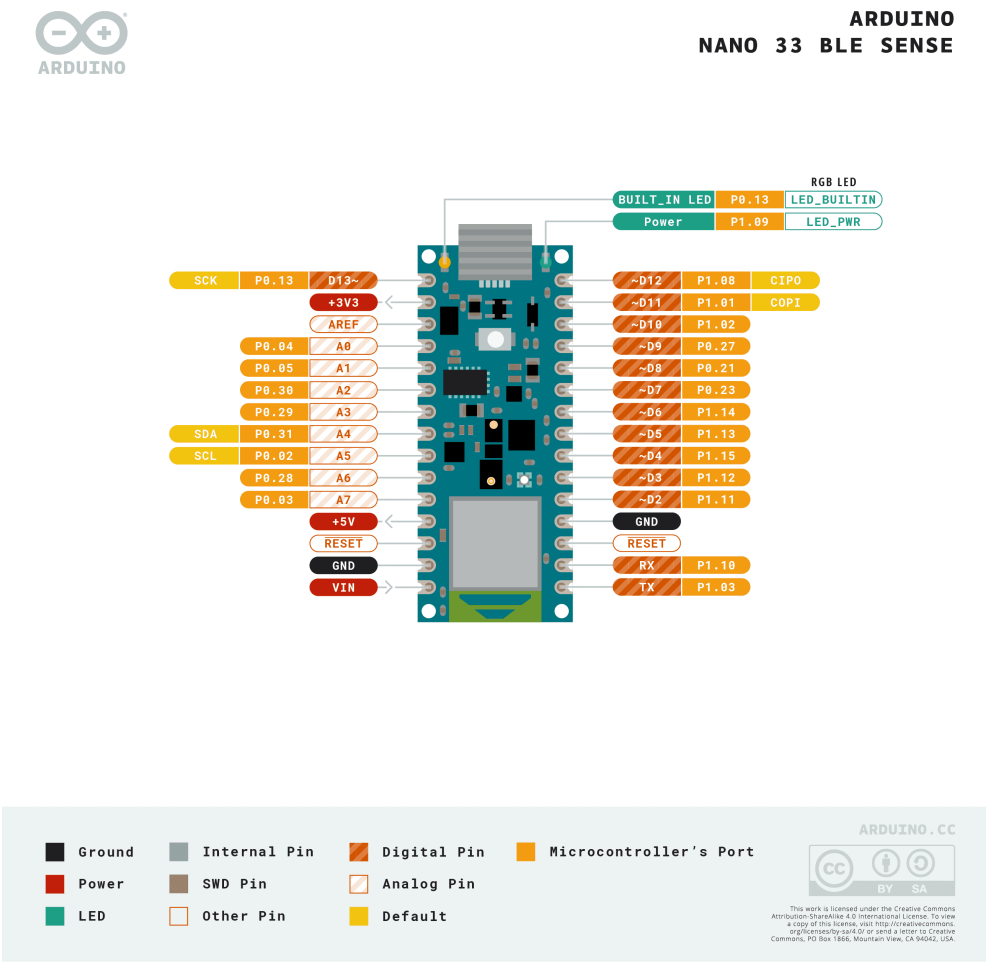


Figure 4.4.: **Arduino Nano 33 BLE Pin Configuration** [Ard23]

2. **Resolution:** Each sensor has a defined resolution, representing the smallest detectable input change. For instance, the accelerometer's resolution is <insert value>, allowing detection of subtle motion variations.

3. **Sampling Rate:**The board supports high sampling rates, enabling real-time data acquisition at <insert frequency> Hz. This is essential for applications requiring continuous and responsive data collection.

4. **Noise Level:** Sensor data may exhibit noise due to factors such as environmental conditions, sensor imperfections, or electromagnetic interference. The board minimizes noise using calibration and filtering techniques, ensuring high data fidelity.

5. **Calibration:** Calibration processes are applied to enhance the accuracy and reliability of sensor readings. These procedures involve fine-tuning sensor settings and using correction factors to address systematic errors [**Edm:2015**].

6. **Data Transmission:** Sensor data is transmitted from the Arduino Nano 33 BLE Sense board to other devices using wireless communication protocols such as Bluetooth Low Energy (BLE). Data transmission is efficient, ensuring minimal latency and reliable delivery.

7. **Data Integrity:** To maintain data integrity during transmission and processing, error detection and correction mechanisms are implemented. These measures help identify and resolve issues like data loss or corruption [Goo24].

8. **Power Consumption:** The Arduino Nano 33 BLE Sense is designed for low power consumption, making it ideal for battery-powered applications. Its optimized power management extends battery life without compromising continuous data collection.

By addressing these aspects of data quality in the hardware description, we ensure the reliability and accuracy of sensor data used in our Magic Wand project.

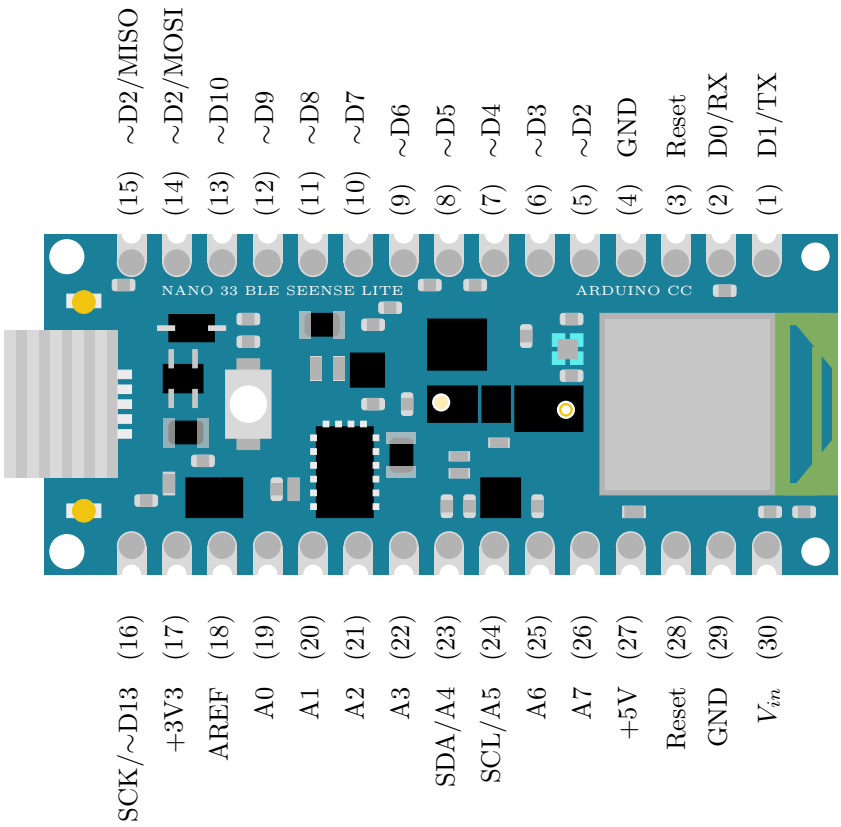


Figure 4.5.: Pin assignment of the Arduino Nano 33 BLE Sense; note that the orange built-in LED is connected to pin D13 and the power LED to pin D1. The built-in RGB LED occupies pins D18, D19 and D20.

4.4.1. Specifications of Arduino Nano 33 BLE Sense

NINA B306 Module

1. Processor:
   - 64 MHz Arm® Cortex-M4F (with FPU)
   - 1 MB Flash + 256 KB RAM

2. Bluetooth® 5 multiprotocol radio:
   - 2 Mbps
   - +8 dBm TX power

- -95 dBm sensitivity
- 4.8 mA in TX (0 dBm)
- 4.6 mA in RX (1 Mbps)
- Integrated balun with 50 Ohm single-ended output
- IEEE 802.15.4 radio support

3. Peripherals:
   - 12 Mbps USB
   - NFC-A tag
   - Arm CryptoCell CC310 security subsystem
   - QSPI/SPI/TWI/I2S/PDM/QDEC
   - 32 MHz SPI
   - Quad SPI interface 32 MHz
   - 12-bit 200 ksps ADC
   - 128 bit AES/ECB/CCM/AAR co-processor

LSM9DS1 (9 axis IMU)

[WS20a]

1. 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
2. ±2/±4/±8/±16 g linear acceleration full scale
3. ±4/±8/±12/±16 gauss magnetic full scale
4. ±245/±500/±2000 dps angular rate full scale
5. 16-bit data output

4.5. Data quantity in Hardware Description

1. **Sensor Data**: The Arduino Nano 33 BLE Sense board is equipped with a variety of sensors, including an accelerometer, a gyroscope, and a magnetometer. These sensors generate a large amount of data as you wave the magic wand. For instance, in one experiment, a window size of 2 seconds was used, which means 200 rows of accelerometer data or 600 values of x, y, and z acceleration axis were fed into the model. [**Miller:2022**]
2. **Data Processing Capacity**: The Arduino Nano 33 BLE Sense board has a 32-bit ARM Cortex-M4 CPU running at 64 MHz, which allows it to process a significant amount of sensor data in real-time.
3. **Memory**: The board has 1MB of flash memory and 256KB of SRAM. This memory is used to store the program code and handle runtime operations, including storing sensor data and machine learning model parameters.
4. **Machine Learning Model**: The size of the machine learning model used in the project also affects the quantity of data. The model needs to be small enough to fit into the board's memory along with the program code.
5. **Data Transmission**: The board also has a built-in Bluetooth Low Energy module, which can be used to transmit sensor data to another device for further processing. [Goo24]

4.6. Constraints

Arduino Nano 33 BLE Sense

- Operating Voltage: 3.3V
- Power Consumption:
  - Maximum 15mA in low power mode
  - Maximum 60mA in active mode
- Operating Temperature Range: -40°C to 85°C
- Memory Constraints: 1 MB Flash + 256 KB RAM
- Communication Interfaces: USB, Bluetooth 5, NFC-A, SPI, I2C, QSPI, etc.[**Arduino:2015**]

Sensors

- Accelerometer:
  - Measurement Range: ±8g
  - Operating Temperature Range: -40°C to 85°C
- Gyroscope:
  - Measurement Range: ±2000 dps
  - Operating Temperature Range: -40°C to 85°C [Ard21]

Actuators

- RGB LEDs:
  - Power Requirements: Voltage, Current
  - Operating Temperature Range
- Buzzer/Speaker:
  - Voltage, Current, Sound Output Levels

Power Supply

- Input Voltage Range: Specify the acceptable input voltage range.
- Power Consumption: Estimate the overall power consumption of the system.

Physical Constraints

- Size and Dimensions: Ensure compatibility with the project enclosure or housing.
- Mounting Requirements: Specify any specific mounting requirements for the components.

Environmental Constraints

- Environmental Protection: Ensure components are suitable for the intended environmental conditions (e.g., moisture resistance, dust resistance).
- Operating Conditions: Specify any limitations or special considerations for operating in certain environments.[Ard21]

4.7. Dimensions of the Arduino Nano 33 BLE Sense

The Arduino Nano 33 BLE Sense is a highly compact development board, measuring just 45mm x 18mm. Its small form factor makes it particularly well-suited for wearable devices and other space-constrained applications. Despite its size, the board is equipped with a wide range of sensors and features, offering versatility for various projects.

It is important to note that these dimensions apply to the board without headers. If you are using a version with pre-soldered headers or attaching additional components, the overall dimensions of your hardware setup may change. For precise measurements, always consult the specifications of your specific board and components [Ard23].

4.8. Sensor Accelerometer, Gyroscope, and Magnetometer LSM9DS1

Magic Wand gesture detection is mainly based on the sensor LSM9DS1 at the board. It is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor. A tiny device called an Inertial Measurement Unit (IMU) in the sensor is used to detect the motions physically [**Alushi:2023**]. It has several components, including magnetometer, gyroscope, and accelerometer. By monitoring acceleration and angular velocity changes, the IMU sensor is critical for identifying an object's orientation and movement in real-time [**Alushi:2023**].

Example Code for Arduino Nano 33 BLE Sense

Below is a simple example demonstrating how to use the built-in microphone on the Arduino Nano 33 BLE Sense:

```
1000  // Include the PDM library
      #include <PDM.h>
1002
      // Buffer to store the microphone data
1004  short sampleBuffer[256];

1006  // Variable to store the sound level
      int soundLevel = 0;
1008
      // Callback function for PDM data
1010  void onPDMdata() {
        // Read the PDM data
1012    int bytesAvailable = PDM.available();
        PDM.read(sampleBuffer, bytesAvailable);
1014
        // Calculate the sound level
1016    soundLevel = 0;
        for (int i = 0; i < bytesAvailable / 2; i++) {
1018      soundLevel += abs(sampleBuffer[i]);
        }
1020    soundLevel /= bytesAvailable / 2;
      }
1022
      void setup() {
1024    // Initialize serial communication
        Serial.begin(9600);
1026    while (!Serial);

1028    // Initialize PDM with a sample rate of 16 kHz and 16-bit
      resolution
        PDM.begin(1, 16000);
1030    PDM.onReceive(onPDMdata);
      }
1032
      void loop() {
1034    // Print the sound level to the serial monitor
        Serial.println(soundLevel);
1036    delay(100);
      }
1038
```

Listing 4.1.: Simple example using of the builtin microphone of the Arduino Nano 33 BLE Sense

```
1000  // Include the ArduinoBLE library
      #include <ArduinoBLE.h>
1002
      // Create a BLE service
1004  BLEService batteryService("1101");

1006  // Create a BLE characteristic
      BLEUnsignedCharCharacteristic batteryLevelChar("2101", BLERead |
      BLENotify);
1008
      void setup() {
1010    // Initialize serial communication
        Serial.begin(9600);
1012    while (!Serial);

1014    // Set up the built-in LED pin
        pinMode(LED_BUILTIN, OUTPUT);
1016
        // Initialize BLE
1018    if (!BLE.begin()) {
          Serial.println("Starting BLE failed!");
1020      while (1);
        }
1022
        // Set the BLE local name and advertised service
1024    BLE.setLocalName("BatteryMonitor");
        BLE.setAdvertisedService(batteryService);
1026
        // Add characteristic to the service
1028    batteryService.addCharacteristic(batteryLevelChar);

1030    // Add the service and start advertising
        BLE.addService(batteryService);
1032    BLE.advertise();

1034    Serial.println("Bluetooth device active, waiting for connections
      ...");
      }
1036
      void loop() {
1038    // Wait for a BLE central to connect
        BLEDevice central = BLE.central();
1040
        if (central) {
1042      Serial.print("Connected to central: ");
          Serial.println(central.address());
1044
          // Turn on the built-in LED
1046      digitalWrite(LED_BUILTIN, HIGH);

1048      // Loop while the central device is connected
          while (central.connected()) {
1050        int battery = analogRead(A0);
            int batteryLevel = map(battery, 0, 1023, 0, 100);
1052
            Serial.print("Battery Level is now: ");
1054        Serial.println(batteryLevel);

1056        // Update the characteristic value
            batteryLevelChar.writeValue(batteryLevel);
1058
            delay(200);
1060      }

1062      // Turn off the built-in LED
          digitalWrite(LED_BUILTIN, LOW);
1064      Serial.print("Disconnected from central: ");
          Serial.println(central.address());
1066    }
      }
1068
```

Listing 4.2.: Example of a BLE Battery Monitor using Arduino Nano 33 BLE Sense

4.9. Inertial Measurement Unit (IMU)

4.9.1. Description

An Inertial Measurement Unit (IMU) is an electronic system that measures movement across multiple axes using three primary sensors: an accelerometer, a gyroscope, and a magnetometer.[QG17] These sensors work together to provide data on acceleration, rotational motion, and magnetic fields, making IMUs essential for a wide range of applications such as navigation, fitness tracking, and robotics.[QG17]
General IMUs have become increasingly common in microcontroller projects, with some boards, such as the Arduino Nano 33 BLE Sense, featuring integrated IMUs for seamless development of motion-sensitive applications.[Zho+20]

4.9.2. Specific Sensors

Continuous Operation ("Always-On" Mode) The LSM9DS1 supports an "Always-On" mode, ensuring continuous operation even when the main system is in a low-power state.[Zho+20] This is critical for uninterrupted motion monitoring in devices like the Magic Wand, where gestures must be tracked instantly. With a power consumption of just 0.55 mA in high-performance mode, it provides precise and reliable motion data while preserving battery life.[Zho+20]
Tilt Detection Using the accelerometer, the IMU can detect orientation changes with minimal power usage. Tilt detection is particularly useful for identifying subtle shifts in position, enhancing the responsiveness of gesture-based controls.[Zho+20]
Significant Motion Detection The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[Zho+20]
The LSM9DS1 is a versatile and energy-efficient IMU with advanced sensing capabilities, supporting a range of applications where motion detection, orientation tracking, and gesture recognition are essential. Its compact design, low power consumption, and robust environmental tolerance make it a reliable choice for portable, battery-powered devices like the Magic Wand.[Zho+20]
An IMU consists of three sensors that measure an object's orientation, position, vibration and movement in 3D space in real-time. These sensors are typically arranged in a pattern, including a tri-axial accelerometer, gyroscope, and magnetometer[Ahm+13]. An accelerometer is used to measure the change in velocity of a moving or vibrating object[Ahm+13]. Gyroscope measures the angular rotation[Ahm+13]. A Magnetometer is used to measure yaw angle rotation, calibrating to the gyroscope data to improve the big drift issue[Ahm+13].
The accelerometer functions by gauging the acceleration (Ax, Ay, Az), which denotes the rate of acceleration change over time. The acceleration values along each axis are conventionally expressed in units of meters per second squared (m/s^2) [**Vernier:2023**]. To illustrate, if the LSM9DS1 records an acceleration of 15 m/s^2 along the x-axis, this signifies that the object under observation is experiencing a velocity increment of 15 meters per second every second in the x-direction.
A gyroscope, designed to quantify angular speed (ωx, ωy, ωz), serves as an indicator of the object's orientation change rate along each axis. Angular velocity along each axis is commonly expressed in degrees per second ($\circ$/s) [**Zhuang:2020**]. For instance, when the LSM9DS1 records an angular velocity of 100$\circ$/s along the z-axis, it signifies that the measured object is undergoing rotation at a rate of 100 degrees per second around the z-axis. Refer to the accompanying figure for visualization.
In the context of a magnetometer, the x, y, and z axes (Mx, My, Mz) typically delineate the three-dimensional space in which the magnetic field is being assessed. The x-axis typically corresponds to the horizontal component of the magnetic field, the y-axis

signifies the vertical component, and the z-axis reflects the magnetic field strength [**Kostiainen:2023**]. This three-dimensional measurement provides a comprehensive understanding of the magnetic field in the surrounding space.

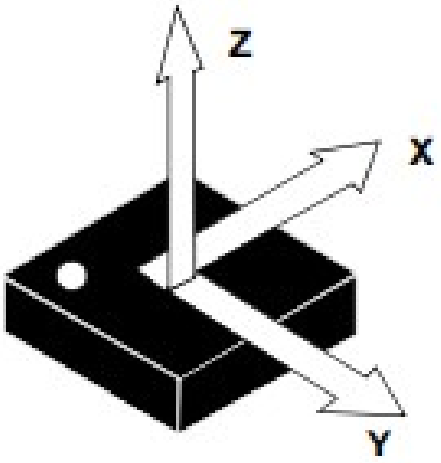


Figure 4.6.: **Accelerometer Accerlerations Directions** [**Stm:2015**]

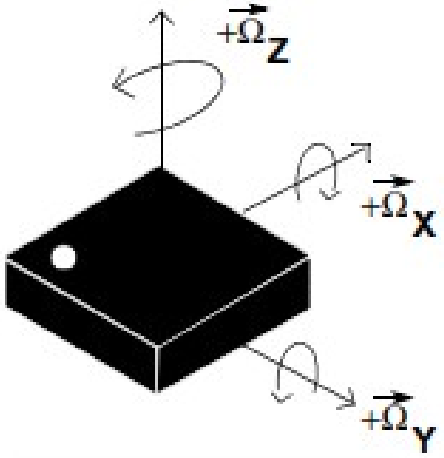


Figure 4.7.: **Gyroscope Angular Directions** [**Stm:2015**]

IMUs can be classified into two main categories based on the type of sensors used: Microelectromechanical Systems (MEMS) IMUs and Fiber Optic Gyro (FOG) IMUs. MEMS IMUs use small mechanical sensors that are etched onto a silicon chip, while FOG IMUs use optical fibers to measure angular velocity. MEMS IMUs are typically smaller and less expensive but are also less accurate and have a shorter lifespan compared to FOG IMUs[**Deppe:2017**].
IMU is sent to a microcontroller or computer, and software is written to read the data from the sensors and perform the necessary calculations. There are also many libraries and software packages available that can simplify the process of working with IMUs. IMUs are commonly used in a variety of applications, Industry Quality Control, Medical Rehabilitation, Robotics, Navigation Systems, Sports Learning and Augmented Reality Systems since they are essential to accurately measure the movement and orientation of the device for further function development [Ahm+13]. In general, they are used

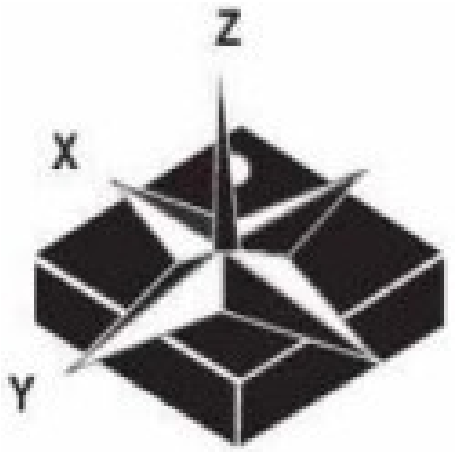


Figure 4.8.: **Magnetometer Directions**
[**Stm:2015**]

for:
Orientation tracking: IMUs can follow the orientation of an object in space by using a combination of acceleration and angular velocity sensors. The data from these sensors can be used to calculate the object's orientation. Robotics technology is one of the regions that require linear and angular data for the movements so robots can perform daily routines and advanced tasks similar to humans [Ahm+13].
Motion sensing: Detecting motion and acceleration changes is also one function that IMUs can achieve. Ryan et al. suggested integrating a miniaturized IMU sensor into the ball, comprising a tri-axial gyroscope and tri-axial accelerometer. Their initial experiment employed ball kinematics knowledge to estimate the drift error arising from measurements [**McGinnis:2011**]. Subsequent research efforts enhanced data accuracy by cross-referencing the measurements with a high-speed motion analysis system consisting of 10 cameras (VICON) [**McGinnis:2012**].
Navigation System: IMUs are marked as an upgrade in the Global Positioning System (GPS) to navigate by combining data from the acceleration and angular velocity sensors with information for better data accuracy. Also, IMU can provide locations where no GPS signals are detected as an alternative option in the car and aircraft in case of emergency [Ahm+13].
However, IMUs face a significant challenge due to the susceptibility of the underlying gyroscopes and accelerometers to measurement errors. This poses a fundamental issue for all IMUs as gyroscopes inherently exhibit drift, and accelerometers may introduce inaccuracies, resulting in misestimations of orientation concerning gravity. Despite incorporating minor measurement errors, the guidance system continually integrates the calculated position results with the original position information (refer to trajectory calculation). Although individual errors may be minor, their persistence across positions leads to cumulative effects known as "drift." Over time, this accumulation causes a widening disparity between the system's perceived and actual locations, and it becomes impossible to eliminate these errors [Har23]. Therefore, drift remains a fundamental challenge for any IMUs.

4.9.3. Specifications

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels.
- $\pm 2/\pm 4/\pm 8/\pm 16$ g linear acceleration full scale.
- $\pm 4/\pm 8/\pm 12/\pm 16$ gauss magnetic full scale.

- $\pm 245/\pm 500/\pm 2000$ dps angular rate full scale.
- 16-bit data output.

3D digital linear acceleration sensor: Measures linear motion with a full scale of ±2g/±4g/±8g/±16g.
3D digital angular rate sensor: Tracks rotational motion with an angular rate range of ±245/±500/±2000 degrees per second (dps).
3D digital magnetic sensor: Detects magnetic fields with a full scale of ±4/±8/±12/±16 gauss.
This compact and versatile IMU includes both I²C and SPI serial interfaces and supports a power-down mode for smart power management. It is available in a plastic land grid array (LGA) package and operates reliably across a wide temperature range from -40 °C to +85 °C.

4.9.4. Libraries

A library refers to a collection of pre-written code that can be used by developers to perform specific tasks or functions without having to write the code from scratch. Libraries are designed to make the development process easier and more efficient by providing pre-built solutions to common programming challenges.

Wire.h

`Wire.h` is a library in Arduino that allows for communication between I2C devices. I2C stands for Inter-Integrated Circuit, which is a synchronous serial communication protocol used for connecting microcontrollers to peripheral devices. `Wire.h` provides functions for initializing the I2C bus, sending and receiving data over the bus, and managing multiple devices on the same bus. With this library, developers can easily connect multiple I2C devices, such as sensors or displays, to an Arduino board [**Ardc; Ari21**].

Kalman.h

`Kalman.h` is a library that implements the Kalman filter algorithm. The Kalman filter is a mathematical technique used to estimate the state of a system based on incomplete measurements. It is commonly used in control systems, robotics, and navigation applications to improve the accuracy of sensor measurements and reduce errors. `Kalman.h` provides a simple interface for developers to implement the Kalman filter in their Arduino projects [**Ard19; Fet21**].

Arduino_LSM6DSOX.h

`Arduino_LSM6DSOX.h` is a library that provides access to the LSM6DSOX accelerometer and gyroscope sensor on the Arduino Mbed OS Nicla Board. The LSM6DSOX sensor is a 6-axis sensor that can measure both linear acceleration and angular velocity. `Arduino_LSM6DSOX.h` provides functions for initializing the sensor, reading data from the sensor, and configuring the sensor parameters. With this library, developers can easily integrate the LSM6DSOX sensor into their Arduino projects and use the sensor data for various applications, such as gesture recognition or orientation detection [**Lib21**].

LSM6DSOXSensor.h

`LSM6DSOXSensor.h` is a library that provides an interface for interacting with the LSM6DSOX sensor. The LSM6DSOX is a 6-axis inertial measurement unit (IMU)

sensor that combines a 3-axis accelerometer and a 3-axis gyroscope in a single chip. It is commonly used in applications that require motion sensing and orientation tracking, such as robotics, drones, wearable devices, and Internet of Things (IoT) devices. The `LSM6DSOXSensor.h` library allows developers to easily interact with the LSM6DSOX sensor by providing functions and classes for configuring the sensor, reading raw sensor data, and performing sensor fusion to obtain calibrated accelerometer and gyroscope data, as well as derived data such as orientation, linear acceleration, and angular velocity. The library abstracts the low-level communication with the sensor, providing a higher-level API that simplifies the process of working with the sensor.

4.9.5. Calibration of Sensors

There are various methods to calibrate the sensors involved. It is necessary to know whether the sensor is balanced since the time between each calibration needs to be explicitly defined. Regular calibration should be done, especially when strange outputs are noticed. Some methods of calibration are briefed below:

Low and high limit method

The low and high limit method involves recording the minimum and maximum values on all three axes of a sensor by performing simple scratches or movements to determine their absolute extremes. The sensor is subjected to circular rotations along each axis multiple times [**Edm:2015**]. The center point is calculated as the midpoint between these recorded extremes. Increasing the number of rotations improves the chances of identifying the absolute peak values. Ideally, the center point should be close to zero, indicating minimal sensor offset. However, deviations may signal a hard iron offset, often caused by distortions from the Earth's magnetic field [**Edm:2015**]. This method assumes minimal soft iron distortion, which would otherwise alter the sensor readings. The absence of significant soft iron interference is typically confirmed by observing rounded outlines in the resulting data graph.
It is crucial to recognize that the low and high limit method requires recalibration periodically to maintain performance, as sensor components can drift or degrade over time [**Edm:2015**]. For devices powered by primary batteries, recalibration is particularly important after every battery replacement. This is because the battery often acts as a significant source of magnetic disturbance, and newly installed batteries may introduce different interference compared to the previous ones [**Edm:2015**].

FreeIMU Calibration Application Magnetometer

The **FreeIMU Calibration Application Magnetometer method** involves pre-processing raw magnetometer data by applying axis-specific gain correction to convert it into nanoTesla. The conversion formula for each axis is as follows:

$$Xm\text{-nanoTesla} = \text{rawCompass.m.x} \times \left(\frac{100000.0}{1100.0}\right)$$

$$Ym\text{-nanoTesla} = \text{rawCompass.m.y} \times \left(\frac{100000.0}{1100.0}\right)$$

$$Zm\text{-nanoTesla} = \text{rawCompass.m.z} \times \left(\frac{100000.0}{980.0}\right)$$

These scaling factors (e.g., $\frac{100000.0}{1100.0}$) should be replaced with sensor-specific values to ensure accurate conversion. The processed data is then saved in a file named `Mag-raw.txt`, which can be opened using the **Magneto program**. Magneto generates twelve calibration parameters to correct for various sensor errors, including bias, hard iron distortion, scale factor errors, soft iron distortion, and misalignment [**Edm:2015**].

Additionally, this method can be applied to accelerometers. By pre-processing raw accelerometer data while accounting for bit depth and G sensitivity, the data can be converted into milliGalileo (mGal). A gravitational field "norm" value of 1000 mGal can also be used to refine calibration [**Edm:2015**]. This approach ensures improved accuracy and reliability of sensor measurements for both magnetometers and accelerometers.

4.9.6. Code

```
1000  // Include the necessary libraries
      #include <Wire.h>
1002  #include <MPU6050.h>

1004  // Create an MPU6050 object
      MPU6050 mpu;
1006
      // Setup function
1008  void setup() {
        // Initialize I2C communication and Serial communication
1010    Wire.begin();
        Serial.begin(9600);
1012
        // Initialize the MPU6050
1014    mpu.initialize();

1016    // Check if the MPU6050 is connected
        if (mpu.testConnection()) {
1018      Serial.println("MPU6050 connection successful");
        } else {
1020      Serial.println("MPU6050 connection failed");
        }
1022  }

1024  // Loop function
      void loop() {
1026    // Declare variables for accelerometer and gyroscope data
        int16_t ax, ay, az;
1028    int16_t gx, gy, gz;

1030    // Get the motion data from the MPU6050
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
1032
        // Print the accelerometer and gyroscope data
1034    Serial.print("a/g:\t");
        Serial.print(ax); Serial.print("\t");
1036    Serial.print(ay); Serial.print("\t");
        Serial.print(az); Serial.print("\t");
1038    Serial.print(gx); Serial.print("\t");
        Serial.print(gy); Serial.println(gz);
1040
        // Delay for 100ms before the next reading
1042    delay(100);
      }
1044
```

Listing 4.3.: Example of interfacing MPU6050 with Arduino for motion data

4.9.7. Applications

The IMU sensor on the Arduino Nano 33 BLE Sense can be used in the following applications:

- **Gesture Recognition**: Use accelerometer and gyroscope data to detect tilt, shake, or rotation gestures.
- **Fitness Tracker**: Monitor motion and orientation to calculate steps, measure speed, or track physical activities.
- **Robot Navigation**: IMU data can help track robot movement and determine orientation changes.
- **Virtual Reality**: IMUs provide orientation data for head tracking in VR applications.

4.9.8. Tests

Tilt Measurement Test

- Use the accelerometer data to calculate the tilt angle of the sensor relative to the ground.
- Compare the calculated tilt angle with a known reference (e.g., a protractor) to verify accuracy.

Motion Tracking Test

- Use both accelerometer and gyroscope data to track the sensor's motion in 3D space.
- Compare the tracked motion with a known reference (e.g., a motion capture system) to verify accuracy.

Environmental Tests

- **Temperature Stability Test**: Place the sensor in different temperature environments (e.g., cold, room temperature, hot) and record the readings. Verify that the sensor maintains accuracy across different temperatures. Any significant deviation may require temperature compensation.
- **Vibration Test**: Subject the sensor to different levels of vibration and record the readings. Verify that the sensor maintains accuracy under vibration. This is particularly important for applications like drones or vehicles.

4.9.9. Further Readings

- LSM9DS1 Datasheet - STMicroelectronics: LSM9DS1 Datasheet
- IMU Testing and Calibration: IMU Testing and Calibration

4.10. USB Cable

4.10.1. USB Type A Male to Micro-B Male

Professional USB 2.0 Type A Male to Micro-B Male Cable for High-Performance Commercial AV and IT Applications

- Supports data rates of up to **480Mbps**.
- Robust PVC housing with gold-plated contacts and nickel-coated connector sleeves.
- 2-fold shielded cable with corrosion-resistant, tinned copper conductor.
- **USB 2.0**, 480Mbps.

4.10.2. Cable Lines Concept

Cable Lines stands for the concept of contemporary, wired connectivity solutions from **Lindy**. The **Anthra Line USB 2.0 Type A Male to Micro-B Male Cables** from the Cable Line concept are the professional choice when it comes to realizing connections for the highest resolutions in commercial AV and IT applications.
The Anthra Line USB 2.0 cables feature:

- **2-fold shielding** and tinned copper conductors for the highest and lossless transmission performance.
- Permanent corrosion resistance and maximum reliability guaranteed by high--quality, gold-plated contacts and nickel-plated connector sleeves.

4.10.3. Key Features

- Data transfer speeds of up to **480Mbps** enable fast and smooth transfer of large volumes of data.

4.10.4. Specifications

General

- **Type:** USB 2.0 Cable
- **Execution:** Straight
- **Color:** Black
- **Material:** Plastic

Connections / Interfaces

- **Connection Input:** A-connector
- **Connection Output:** Micro-B connector

Metrics

- **Cable Length:** 0.20 m

Other

- **Specification:** USB 2.0
- **Manufacturer:** LINDY
- **Manufacturer's Article Number:** 36730
- **Tare:** 0.019 kg
- **RoHS:** Compliant
- **EAN / GTIN:** 4002888367301

4.11. Sticky Tape

tesa® WALLPAPER is a thin but tear-resistant masking tape that has been specially developed for use on very sensitive surfaces such as paper wallpaper or fine plaster. The wallpaper tape is easy to apply, allows precise work with clean paint edges, and can be removed within seven days without leaving any residue. Since the masking tape is suitable for all types of paint, it is the ideal basis for renovation work where you want to protect sensitive surfaces indoors.

Figure 4.9.: **USB**

4.11.1. Features

- Specially designed for sensitive surfaces such as paper wallpaper.
- Masking tape can be removed without leaving any residue for up to seven days.
- Solvent-free and primarily made from renewable raw materials.
- Suitable for all types of paint.

4.11.2. Dimensions

- **Width:** 25 mm
- **Length:** 25 m
- **Quantity:** 1 Roll

Metrics

- **Length:** 25 m
- **Width:** 25 mm

Other

- **Specification:** For sensitive substrates

Packaging

- **Packaging:** 1 roll

Manufacturer

- **Manufacturer:** TESA
- **Manufacturer's Article Number:** 56260-00000-03
- **Tare:** 0.0787 kg
- **RoHS:** Compliant
- **EAN / GTIN:** 4042448149992

Figure 4.10.: **Tapes**

5. Sensor/Actor

5.1. Accelerometer

5.1.1. Description

The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[Zho+20]
The LSM9DS1 is a versatile and energy-efficient IMU with advanced sensing capabilities, supporting a range of applications where motion detection, orientation tracking, and gesture recognition are essential. Its compact design, low power consumption, and robust environmental tolerance make it a reliable choice for portable, battery-powered devices like the Magic Wand.[Zho+20]
An IMU consists of three sensors that measure an object's orientation, position, vibration and movement in 3D space in real-time. These sensors are typically arranged in a pattern, including a tri-axial accelerometer, gyroscope, and magnetometer[Ahm+13]. An accelerometer is used to measure the change in velocity of a moving or vibrating object[Ahm+13]. Gyroscope measures the angular rotation[Ahm+13]. A Magnetometer is used to measure yaw angle rotation, calibrating to the gyroscope data to improve the big drift issue[Ahm+13].
The accelerometer functions by gauging the acceleration (Ax, Ay, Az), which denotes the rate of acceleration change over time. The acceleration values along each axis are conventionally expressed in units of meters per second squared (m/s^2) [**Vernier:2023**]. To illustrate, if the LSM9DS1 records an acceleration of 15 m/s^2 along the x-axis, this signifies that the object under observation is experiencing a velocity increment of 15 meters per second every second in the x-direction.

5.1.2. Specific Sensors

Tilt Detection Using the accelerometer, the IMU can detect orientation changes with minimal power usage. Tilt detection is particularly useful for identifying subtle shifts in position, enhancing the responsiveness of gesture-based controls.[Zho+20]
Significant Motion Detection The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[Zho+20]
The LSM9DS1 is a versatile and energy-efficient IMU with advanced sensing capabilities, supporting a range of applications where motion detection, orientation tracking, and gesture recognition are essential. Its compact design, low power consumption, and robust environmental tolerance make it a reliable choice for portable, battery-powered devices like the Magic Wand.[Zho+20]
An IMU consists of three sensors that measure an object's orientation, position, vibration and movement in 3D space in real-time. These sensors are typically arranged in a pattern, including a tri-axial accelerometer, gyroscope, and magnetometer[Ahm+13]. An accelerometer is used to measure the change in velocity of a moving or vibrating object[Ahm+13]. Gyroscope measures the angular rotation[Ahm+13]. A Magnetometer is used to measure yaw angle rotation, calibrating to the gyroscope data to improve the big drift issue[Ahm+13].

The accelerometer functions by gauging the acceleration (Ax, Ay, Az), which denotes the rate of acceleration change over time. The acceleration values along each axis are conventionally expressed in units of meters per second squared (m/s^2) [**Vernier:2023**]. To illustrate, if the LSM9DS1 records an acceleration of 15 m/s^2 along the x-axis, this signifies that the object under observation is experiencing a velocity increment of 15 meters per second every second in the x-direction.

5.1.3. Specification

5.1.4. Library

Details about libraries used to interface with accelerometers (e.g., Adafruit Sensor Library, Arduino libraries). Installation process and setup. Features provided by these libraries (e.g., data acquisition, filtering, scaling).

```
// Include the necessary libraries
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>
```

5.1.5. Functions

```
// Check if acceleration data is available
if (IMU.accelerationAvailable()) {
 // Read the acceleration values
 IMU.readAcceleration(x, y, z);

 // Print the acceleration values
 Serial.print("AccX: ");
 Serial.print(x);
 Serial.print(", AccY: ");
 Serial.print(y);
 Serial.print(", AccZ: ");
 Serial.println(z);
}
```

5.1.6. Calibration

Steps for calibrating an accelerometer:

- Importance of calibration for accelerometer sensors.
- Methods for calibrating accelerometers (e.g., zeroing, scaling, temperature compensation).
- Tools and techniques for accelerometer calibration (e.g., software tools, calibration kits).

5.1.7. Simple Code

```
 #include <Wire.h>
 #include <Adafruit_Sensor.h>
 #include <Adafruit_LSM9DS1.h>

 // Create LSM9DS1 object
 Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();
```

```
void setup() {
// Start the Serial Monitor
Serial.begin(115200);

// Initialize the LSM9DS1
if (!lsm.begin()) {
 Serial.println("Failed to initialize LSM9DS1! Check wiring.");
 while (1);
}

// Set accelerometer range (default is ±2g)
lsm.setupAccel(lsm.LSM9DS1_ACCELRANGE_2G);
// Options: 2G, 4G, 8G, 16G

// Optional: Set accelerometer data rate (default is 119 Hz)
lsm.setupAccelDataRate(lsm.LSM9DS1_ACCELDATARATE_119HZ);
}

void loop() {
// Read accelerometer data
sensors_event_t accelEvent;
lsm.getEvent(&accelEvent, NULL, NULL);
// Get accelerometer event data only

// Print accelerometer readings (in m/s²)
Serial.print("Accel X: ");
Serial.print(accelEvent.acceleration.x);
Serial.print(" m/s², Y: ");
Serial.print(accelEvent.acceleration.y);
Serial.print(" m/s², Z: ");
Serial.print(accelEvent.acceleration.z);
Serial.println(" m/s²");

// Delay to make output more readable
delay(100);
}
```

5.1.8. Applications

Practical uses of accelerometers:

- In smartphones (e.g., screen orientation, motion detection).
- Automotive industry (e.g., airbag deployment, stability control).
- Robotics (e.g., navigation, balance).
- Wearable devices and fitness trackers.
- Industrial applications (e.g., vibration monitoring, machine diagnostics).

5.1.9. Tests

Techniques to test an accelerometer's functionality: This is the C++ code for testing the accelerometer on the Arduino Nano 33 BLE Sense using the `Adafruit_LSM9DS1`

library.

Listing 5.1: Testing the accelerometer

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>

// Create an instance of the LSM9DS1 sensor
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

void setup() {
    // Start serial communication
    Serial.begin(115200);

    // Initialize the LSM9DS1 sensor
    if (!lsm.begin()) {
        Serial.println("Could not find a valid LSM9DS1 sensor, check wiring!"
            ↪ );
        while (1);
    }

    Serial.println("LSM9DS1 test initialized.");
}

void loop() {
    // Read accelerometer data
    sensors_event_t event;
    lsm.getEvent(&event);

    // Print accelerometer values
    Serial.print("X: ");
    Serial.print(event.acceleration.x);
    Serial.print(" Y: ");
    Serial.print(event.acceleration.y);
    Serial.print(" Z: ");
    Serial.println(event.acceleration.z);

    // Delay before the next reading
    delay(1000);
}
```

Figure 5.1.: C++ code for testing the accelerometer on the Arduino Nano 33 BLE Sense.

This is the Python code for reading the accelerometer data from the Arduino via serial communication. The `pyserial` library is required for this script.

5.1.10. Further Readings

5.2. Gyroscope

5.2.1. Description

The STMicroelectronics LSM9DS1 gyroscope is a precision instrument for measuring angular velocity around the x, y, and z axes. With adaptable measurement ranges of ±245, ±500, and ±2000 degrees per second (dps), the gyroscope is highly versatile for various applications such as inertial navigation, robotics, and drone stabilization.[STM24] Key Features:

- Sampling Rates: The gyroscope offers adjustable sampling rates, allowing it to operate at frequencies of 14.9 Hz, 59.5 Hz, 119 Hz, 238 Hz, 476 Hz, or 952 Hz.[STM24] This flexibility ensures compatibility with a broad range of motion measurement requirements.

Listing 5.2: Accelerometer testing in pyserial

```
import serial
import time

# Replace with the correct port for your system (e.g., 'COM3' on Windows or
    ↪ '/dev/ttyACM0' on Linux)
arduino_port = '/dev/ttyACM0'
baud_rate = 115200

# Establish connection to the Arduino
arduino = serial.Serial(arduino_port, baud_rate, timeout=1)
time.sleep(2)  # Wait for Arduino to initialize

# Function to read accelerometer data from the Arduino
def read_accelerometer():
while True:
line = arduino.readline().decode('utf-8').strip()
if line:
print(line)

# Start reading accelerometer data
read_accelerometer()
```

Figure 5.2.: Python code for reading accelerometer data from the Arduino Nano 33 BLE Sense.

- Voltage and Power Consumption: The gyroscope operates within a voltage range of 1.71 V to 3.6 V, making it suitable for different system configurations.[STM24] Its low power consumption—between 1 mA to 2 mA—enhances its suitability for portable, battery-operated devices.[STM24]

- Compact Design: The LSM9DS1 integrates seamlessly into compact systems, offering engineers and designers a robust yet space-efficient solution.[Mak24]

- Operation Principle: The gyroscope operates based on Coriolis acceleration, a phenomenon observed when a vibrating object moves in a rotating reference frame. Piezoelectric crystals are used to detect changes in angular velocity by converting inertial forces into electrical signals.

- Applications: The LSM9DS1 gyroscope is designed for:

Inertial Measurement Units (IMUs): Combined with accelerometers and magnetometers, it forms a 9-DOF motion tracking system.[Ahm+13] Drone Stabilization: Essential for real-time attitude adjustments.[Ahm+13] Navigation Systems: Key in applications like gyrocompasses or attitude heading reference systems.[Ahm+13]

$$\text{Angular velocity} = \left(\frac{\text{Gyroscope axis raw data}}{65536} \times \text{full scale Gyroscope range} \right) \frac{^\circ}{\text{s}}$$

For example, if the gyroscope's raw data along the X axis is 16384 and the range is $\pm 250°/\text{s}$, the calculation for angular velocity along the X axis would be:

$$\text{Angular velocity along the X axis} = \left(\frac{16384}{65536} \times 500 \right) \frac{^\circ}{\text{s}} = 125 \frac{^\circ}{\text{s}}$$

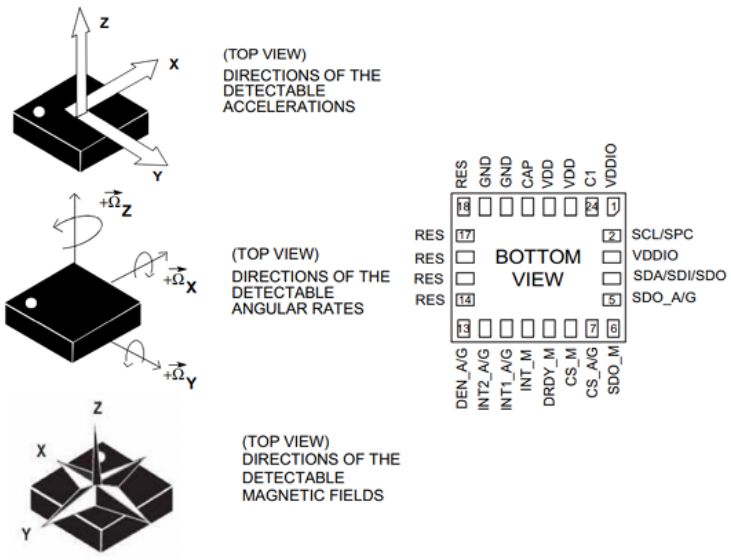


Figure 5.3.: **Gyroscope**

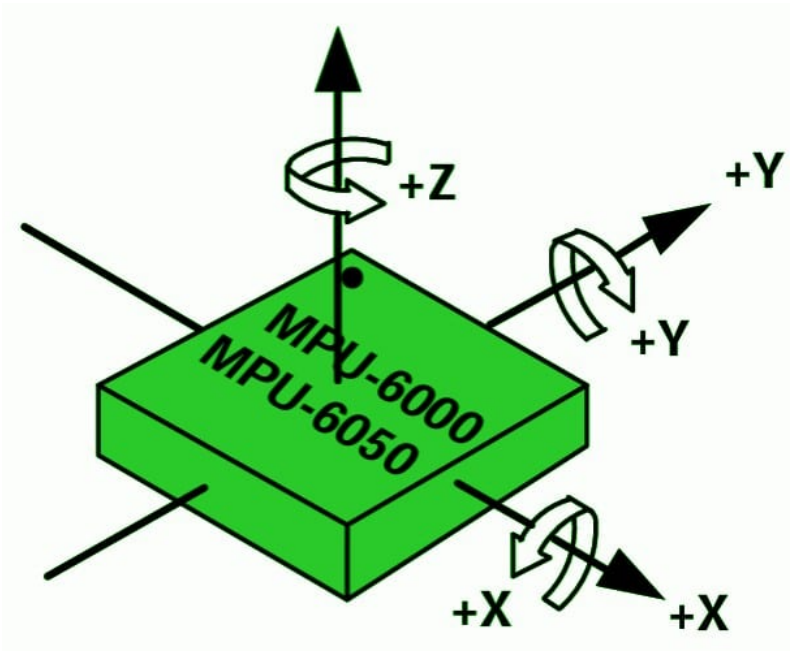


Figure 5.4.: **Orientation Axes of Gyroscope**

5.2.2. Specification

Gyroscope Specifications of MPU6050:

Full scale range	FS_SEL	Range
	0	±250°/s
	1	±500°/s
	2	±1000°/s
	3	±2000°/s
Sensitivity Scale Factor Tolerance		±3%
Gyroscope start-up time		30 ms
Output data rate		4 to 8000 Hz

5.2.3. Library

Description To meet the project's needs, three essential libraries must be integrated, each serving a specific purpose that ensures smooth operation and functionality of the system. Here's a breakdown of each library's role:

1. Wire.h: This is an Arduino standard library used for I2C communication, a protocol that allows devices to communicate with each other using only two wires, reducing the complexity of wiring in systems that need to connect multiple devices. I2C is widely used for connecting sensors, displays, and other peripherals to a microcontroller. The 'Wire.h' library simplifies interactions with I2C devices by handling the low-level details of communication. Developers can initialize the I2C bus, send and receive data, and manage multiple I2C devices on the same bus seamlessly. This is crucial for systems that need to manage several sensors or external modules, as it facilitates smooth and efficient data transfer.[Pas+17]

2. Kalman.h: The Kalman filter is an advanced mathematical algorithm used to process noisy sensor data and provide more accurate estimates of a system's state. The 'Kalman.h' library allows developers to easily implement this filter into Arduino-based projects. It is particularly useful in applications where precise data is essential, such as robotics, control systems, and navigation. By filtering out noise and accounting for uncertainties in sensor readings, the Kalman filter improves the accuracy of measurements like position, velocity, and orientation, which are crucial in motion tracking, sensor fusion, or stabilizing systems like drones and robots.[Pas+17]

3. Arduino_LSM9DS1.h: This library is specifically designed to interface with the LSM9DS1 IMU (Inertial Measurement Unit) sensor, developed by STMicroelectronics. The LSM9DS1 combines three essential motion sensors in one package: an accelerometer, a gyroscope, and a magnetometer. The 'Arduino_LSM9DS1.h' library allows developers to easily access data from these sensors, such as acceleration, angular velocity, and magnetic field strength. Additionally, the library offers functions to configure sensor parameters like measurement ranges, sampling rates, and operating modes, enabling customization based on the needs of the application. This is particularly useful in systems that rely on accurate motion tracking or orientation sensing.[Pas+17]

Together, these libraries enable the integration of multiple sensors into a cohesive system, facilitating communication, data processing, and precise motion measurement.

5.2.4. Installation

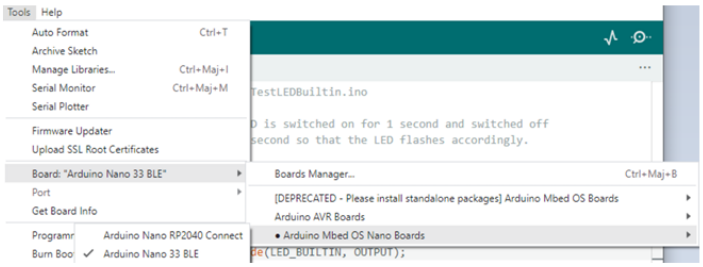


Figure 23.5.: Board card installation

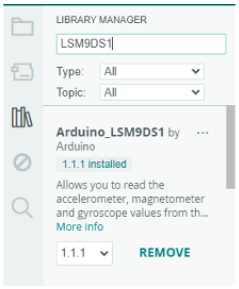


Figure 5.5.: **Installation of Libraries**

Figure 5.6.: **Installation of wire.h**

5.2.5. Functions

The LSM9DS1 library for the Arduino Nano 33 BLE Sense provides a set of functions to interact with the LSM9DS1 IMU (Inertial Measurement Unit) sensor. The functions available depend on the communication protocol being used, either I2C or SPI.[Pas+17]
Function IMU.readGyroscope() The function IMU.readGyroscope() is used to retrieve data from an Inertial Measurement Unit's (IMU) gyroscope. It returns the angular velocity in degrees per second (dps), providing information about the rotational movement detected by the IMU. This data is crucial for applications like robotics, motion tracking, and navigation systems, where precise orientation control and stabilization are required. The function typically returns x, y, and z values, representing the angular velocity along each of the three axes (x, y, and z). These values are floating-point numbers that describe the rotational velocity along each axis, enabling detailed tracking of changes in orientation. By regularly calling this function and analyzing the data over time, it becomes possible to monitor and respond to rotational movements accurately. For example, in robotics, these readings can be used to adjust a robot's position or orientation in real-time, while in drones, they assist with stabilization during flight.
Function IMU.gyroscopeAvailable() The function IMU.gyroscopeAvailable() is used to check whether new gyroscope data is available from the IMU. It returns a value of 1 if new data is ready to be retrieved, and 0 if no new data is available. This function is particularly useful in real-time applications, such as drone stabilization, virtual

```
1000  // Declare variables to store gyroscope data
      float x, y, z;
1002
      // Check if gyroscope data is available
1004  if (IMU.gyroscopeAvailable()) {
        // Read the gyroscope data into variables x, y, z
1006    IMU.readGyroscope(x, y, z);

1008    // Print the x-axis angular velocity
        Serial.print(x);
1010
        // Print a tab space for separation
1012    Serial.print('\t');

1014    // Print the y-axis angular velocity
        Serial.print(y);
1016
        // Print another tab space for separation
1018    Serial.print('\t');

1020    // Print the z-axis angular velocity and move to the next line
        Serial.println(z);
1022  }
```

Listing 5.1.: Sample Code of IMU.readGyroscope()

reality systems, or inertial navigation, where the system must determine whether to wait for updated gyroscope data or proceed with processing the available information.

For example, in a drone application, when the function returns 1, the system knows that new gyroscope data is available, and it can proceed to retrieve and use this information to adjust the drone's orientation or stabilize its motion. Conversely, a return value of 0 means that no new data has been received, prompting the system to wait until new data is available before continuing the process.

In the provided code example:
The program checks if new gyroscope data is available. If it is, the data is read and output to the serial monitor, showing the angular velocity along the x, y, and z axes. This function is integral for applications that require continuous monitoring of rotational movement, ensuring that the system can react promptly to changes in orientation.

Function IMU.gyroscopeSampleRate() The function IMU.gyroscopeSampleRate() is used to retrieve the rate at which the gyroscope integrated into the Inertial Measurement Unit (IMU) collects samples, typically expressed in Hertz (Hz). This sample rate indicates how frequently the gyroscope measures angular velocity, providing insights into its operational efficiency and performance.

Sample Rate: It is the frequency at which the gyroscope takes measurements. For example, a sample rate of 100 Hz means the gyroscope takes 100 readings per second.
Function Breakdown: IMU.gyroscopeSampleRate(): This function returns the sample rate of the gyroscope in Hertz (Hz). The Serial.print() functions then display the sample rate and additional information about the angular speed readings in degrees per second for the X, Y, and Z axes.

This function is especially useful in applications that require accurate and efficient motion tracking, such as drone stabilization, robotics, or virtual reality systems. Knowing the sample rate helps determine how often new data is available, which influences the system's ability to respond to changes in rotational movement.

```
1000  // Declare variables to store gyroscope data
      float x, y, z;
1002
      // Check if gyroscope data is available
1004  if (IMU.gyroscopeAvailable()) {
        // Read the gyroscope data into variables x, y, z
1006    IMU.readGyroscope(x, y, z);

1008    // Print the x-axis angular velocity
        Serial.print(x);
1010
        // Print a tab space for separation
1012    Serial.print('\t');

1014    // Print the y-axis angular velocity
        Serial.print(y);
1016
        // Print another tab space for separation
1018    Serial.print('\t');

1020    // Print the z-axis angular velocity and move to the next line
        Serial.println(z);
1022  }

1024
```

Listing 5.2.: Sample Code of IMU.gyroscopeAvailable()

```
1000  // Print the gyroscope sample rate
      Serial.print("Gyroscope sample rate= ");
1002  Serial.print(IMU.gyroscopeSampleRate());  // Print the gyroscope
        sample rate
      Serial.println("Hz");
1004  Serial.println();  // Print a blank line for separation

1006  // Print a label for angular speed in degrees/second
      Serial.println("Angular speed in degrees/second");
1008
      // Print the axis labels with tab separation
1010  Serial.println("X\tY\tZ");

1012
```

Listing 5.3.: Sample Code of IMU.gyroscopeSampleRate()

5.2.6. Calibration

There are various methods to calibrate the sensors involved. It is necessary to know whether the sensor is balanced since the time between each calibration needs to be explicitly defined. Regular calibration should be done, especially when strange outputs are noticed.
Some methods of calibration are briefed below:

- **Low and high limit method**
  The low and high limit method involves recording minimum and maximum values on all three axes using a simple scratch to determine their absolute values. The sensor undergoes circular rotations along each axis multiple times. The centre point is then identified between these extremes.[Gyr23] Increasing the

number of rotations enhances the likelihood of capturing the absolute peak. The center point will be close to zero if the sensor exhibits no offset. However, slight variations may indicate a hard iron offset attributed to distortion caused by the Earth's magnetic field.[Gyr23] This method assumes minimal soft iron distortion, evident from the rounded outlines in the graph.[Gyr23] It is important to note that this method necessitates capturing values each time to prevent performance degradation due to component drift and aging sensors. For devices relying on primary batteries, calibration becomes essential after each battery change, as the battery inevitably serves as the main source of magnetic disturbance, and new batteries may behave differently from their predecessors.[Gyr23]

- **Scale Factor and Non-Orthogonal Calibration**
  With given initial attitude derived from alignment procedure, the gyroscope measurement can be integrated to calculate the orientation information through Strapdown inertial navigation algorithm. However, the computed attitude will drift over time and the error is gradually accumulated because of the sensor error. In stationary or low dynamic condition, the accelerometer output can be used to estimate the orientation relative to horizontal plane (i.e., pitch and roll).[Gyr23] The attitude derived from accelerometer output is independent in different time epochs and not affected by accumulated error. Hence, based on the different sensors' complimentary error propagation characteristics, we can make use of the accelerometer-derived attitude as reference signal to evaluate the attitude error introduced during integration process, and consequently determine the gyroscope error.[Gyr23]

During the calibration process, the IMU is handheld by user and rotated along its axes slowly to avoid introducing external acceleration. The IMU orientation keeps varying during this procedure and the attitudes derived from different inertial sensors are compared to amend the attitude error and determine the sensor errors. A Kalman filter is designed to estimate the scale factor and non-orthogonal errors of gyroscope. The attitude error propagation equation, which includes sensor error, is utilized as the system dynamic model. The relationship between the accelerometer output and attitude error is modeled as the measurement equation.[Gyr23]

5.2.7. Simple Code

Example IMU: Accelerometer and Gyroscope

5.2.8. Simple Application

Gyroscopes in IMUs are essential for detecting and maintaining orientation and rotational motion in a wide range of devices:

- **Smartphones/Tablets**: They enable features like screen auto-rotation and motion controls for gaming and virtual reality (VR).[Pas+17]
- **Drones**: Gyroscopes stabilize flight by measuring angular velocity, allowing the drone to correct tilt and maintain stable orientation.[Pas+17]
- **Robotics**: In robots, gyroscopes assist in balancing and navigation by tracking rotational movements, ensuring accurate motion control.[Pas+17]
- **VR/AR**: In virtual and augmented reality systems, gyroscopes track head movement to provide an immersive experience by adjusting visuals based on real-time orientation.[Pas+17]

```
#include <Arduino_LSM9DS1.h>

void setup() {
Serial.begin(9600);

// Initialize the IMU
if (!IMU.begin()) {
  Serial.println("Failed to initialize IMU!");
  while (1); // Stop execution if initialization fails
}
}
void loop() {
float x, y, z;

// Check if new acceleration data is available
if (IMU.accelerationAvailable()) {
  IMU.readAcceleration(x, y, z);
  Serial.print("AccX: ");
  Serial.print(x);
  Serial.print(", AccY: ");
  Serial.print(y);
  Serial.print(", AccZ: ");
  Serial.println(z);
}
// Check if new gyroscope data is available
if (IMU.gyroscopeAvailable()) {
  IMU.readGyroscope(x, y, z);
  Serial.print("GyroX: ");
  Serial.print(x);
  Serial.print(", GyroY: ");
  Serial.print(y);
  Serial.print(", GyroZ: ");
  Serial.println(z);
}
delay(1000); // Wait for 1 second before reading again
}

```

Listing 5.4.: Example code for reading accelerometer and gyroscope data from the LSM9DS1 IMU

5.2.9. Testing

The following code reads the gyroscope data (X, Y, and Z axes) from the LSM9DS1 sensor on the Arduino Nano 33 BLE Sense and sends it to the Serial Monitor.

Listing 5.3: Arduino Code to Test the Gyroscope

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>

// Create an instance of the LSM9DS1 sensor
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

void setup() {
    // Start serial communication
    Serial.begin(115200);

    // Initialize the LSM9DS1 sensor
    if (!lsm.begin()) {
        Serial.println("Could not find a valid LSM9DS1 sensor, check wiring!");
        while (1);
    }

    Serial.println("LSM9DS1 Gyroscope test initialized.");
}

void loop() {
    // Read gyroscope data
    sensors_event_t event;
    lsm.getEvent(&event);

    // Print gyroscope values
    Serial.print("Gyroscope X: ");
    Serial.print(event.gyro.x);
    Serial.print(" Y: ");
    Serial.print(event.gyro.y);
    Serial.print(" Z: ");
    Serial.println(event.gyro.z);

    // Delay before the next reading
    delay(1000);
}
```

5.2.10. Python Code

The following Python script reads the gyroscope data sent by the Arduino and prints the values. Ensure you have the `pyserial` library installed.
Install the `pyserial` library:

Listing 5.4: Installing pyserial

```
pip install pyserial
```

Python Code:

Listing 5.5: Python Code to Read and Test Gyroscope Data

```
import serial
import time

# Set up the serial connection (replace with your actual port)
ser = serial.Serial('COM3', 115200)  # For Windows, replace COM3 with your port,
    ↪ for Mac/Linux, it may be /dev/ttyACM0 or /dev/ttyUSB0

# Allow some time for the Arduino to reset and start transmitting data
time.sleep(2)

# Loop to continuously read gyroscope data
while True:
try:
# Read a line of data from Arduino
```

```
    line = ser.readline().decode('utf-8').strip()

    # Only print the line if it contains the "Gyroscope" data
    if 'Gyroscope' in line:
    print(line)

    # Delay before the next reading
    time.sleep(1)

    except KeyboardInterrupt:
    print("Exiting...")
    break

    # Close the serial connection when done
    ser.close()
```

5.2.11. Further Readings

Schanda, Janos: Colorimetry: Understanding the CIE System.Wiley, 2007.[[Sch07]] Lukac, Rastislav and Plataniotis, Konstantinos N.: Color Image Processing: Methods and Applications.CTC Press,2018.[[LP18]]

5.3. Magnetometer

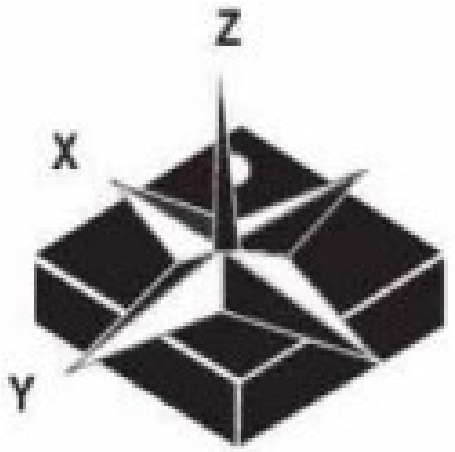


Figure 5.7.: **Magnetometer Directions** [**Stm:2015**]

5.3.1. Description

In the context of a magnetometer, the x, y, and z axes (Mx, My, Mz) typically delineate the three-dimensional space in which the magnetic field is being assessed. The x-axis typically corresponds to the horizontal component of the magnetic field, the y-axis signifies the vertical component, and the z-axis reflects the magnetic field strength [**Kostiainen:2023**]. This three-dimensional measurement provides a comprehensive understanding of the magnetic field in the surrounding space. A magnetometer is a sensor that measures the strength and direction of a magnetic field. It is commonly used to detect the Earth's magnetic field for navigation purposes, detect magnetic anomalies, or determine heading in devices like smartphones, drones, and robotics. Magnetometers are fundamental in compasses, GPS systems, and geological exploration (Ripka, 2001).

5.3.2. Specific Sensors

There are various types of magnetometers available, differing in precision, range, and applications. Here are some popular ones:

- HMC5883L: A widely used 3-axis digital magnetometer that provides low-cost magnetic field measurement [Hon12].
- LSM303DLHC: Combines an accelerometer and magnetometer, enabling tilt-compensated compass applications [Ada21].
- AK8963: A 3-axis magnetometer typically used with IMUs like the MPU9250 for high accuracy [Mic14].
- MAG3110: A small, low-power 3-axis magnetometer for embedded systems [Sem13].

5.3.3. Specification

General magnetometer specifications include:

- 3D digital magnetic sensor: Detects magnetic fields with a full scale of ±4/±8/±12/±16 gauss.
- Axes: Single-axis or 3-axis (most modern sensors are 3-axis).
- Sensitivity: Ability to detect weak magnetic fields (e.g., microteslas, nanoteslas).
- Range: Typically between ±8 to ±100 microteslas.
- Resolution: The smallest change in the magnetic field that the sensor can detect.
- Power Consumption: Important for portable systems.
- Interface: I2C, SPI, or analog output.

5.3.4. Library

Libraries simplify interfacing with magnetometer sensors, converting raw data into readable formats. Below are some libraries:
Arduino:

- HMC5883L: Adafruit_HMC5883_Unified.h (Adafruit, 2021b).
- LSM303: Adafruit_LSM303DLHC.h (Adafruit, 2021a).

Python:

- Use smbus or CircuitPython libraries for I2C sensors [Ada20].
- Example for Raspberry Pi: hmc5883l Python library [Hol19].

To install the library for Arduino:

```
// Include the necessary libraries
#include <Adafruit_Sensor.h>
#include <Adafruit_HMC5883_U.h>
```

5.3.5. Calibration

Basic calibration steps:

- Rotate the sensor in all directions.
- Plot the data to ensure it forms a circle.
- Apply corrections for centering the data.

5.3.6. Simple Code

```
1000  // Include the necessary libraries
      #include <Wire.h>
1002  #include <Adafruit_Sensor.h>
      #include <Adafruit_HMC5883_U.h>
1004  // Create an HMC5883 magnetometer object with a unique ID
      Adafruit_HMC5883_Unified mag = Adafruit_HMC5883_Unified(12345);
1006  // Setup function
      void setup(void) {
1008    // Start serial communication at 9600 baud rate
        Serial.begin(9600);
1010    // Initialize the magnetometer
        if (!mag.begin()) {
1012      Serial.println("No HMC5883L detected ... Check your wiring!");
          while (1);  // Halt the program if the sensor is not detected
1014    }

1016    // Set the magnetometer gain
        mag.setMagGain(HMC5883_MAGGAIN_1_3);
1018  }
      // Loop function
1020  void loop(void) {
        // Declare a variable to store magnetometer data
1022    sensors_event_t event;
        // Get the magnetometer event data
1024    mag.getEvent(&event);
        // Calculate the heading angle in radians
1026    float heading = atan2(event.magnetic.y, event.magnetic.x);
        if (heading < 0) heading += 2 * PI;
1028    // Convert the heading from radians to degrees
        float headingDegrees = heading * 180 / M_PI;
1030    // Print the heading in degrees
        Serial.println(headingDegrees);
1032    // Delay for 500ms before the next reading
        delay(500);
1034  }

1036
```

Listing 5.5.: Example code for reading magnetometer data

5.3.7. Applications

Magnetometers have diverse applications, including:

- Navigation: Electronic compasses in GPS, drones, and aircraft [She13].
- Robotics: Precise heading information for autonomous systems [Soc20].
- Consumer Devices: Smartphones, wearables for direction detection [Mic14].

- Geological Exploration: Detect magnetic anomalies for mineral exploration [Han+17].
- Space Exploration: Magnetic field mapping on planets.
- Security: Detection of ferromagnetic objects in metal detectors [Wil15].

5.3.8. Tests

The following C++ code reads and prints the magnetometer data (X, Y, Z values) from the LSM9DS1 sensor connected to the Arduino Nano 33 BLE Sense.

Listing 5.6: C++ Code for Arduino to Read Magnetometer Data

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>

// Create an instance of the LSM9DS1 sensor
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

void setup() {
    // Start serial communication
    Serial.begin(115200);

    // Initialize the LSM9DS1 sensor
    if (!lsm.begin()) {
        Serial.println("Could not find a valid LSM9DS1 sensor, check wiring!");
        while (1);
    }

    Serial.println("LSM9DS1 test initialized.");
}

void loop() {
    // Read magnetometer data
    sensors_event_t event;
    lsm.getEvent(&event, Adafruit_LSM9DS1::MAGNETOMETER);

    // Print magnetometer values
    Serial.print("Mag X: ");
    Serial.print(event.magnetic.x);
    Serial.print(" Y: ");
    Serial.print(event.magnetic.y);
    Serial.print(" Z: ");
    Serial.println(event.magnetic.z);

    // Delay before the next reading
    delay(1000);
}
```

This Python code reads the magnetometer data from the Arduino via serial communication.

Listing 5.7: Python Code to Read Magnetometer Data via Serial

```
import serial
import time

# Replace with the correct port for your system (e.g., 'COM3' on Windows or '/dev
    ↪ /ttyACM0' on Linux)
arduino_port = '/dev/ttyACM0'
baud_rate = 115200

# Establish connection to the Arduino
arduino = serial.Serial(arduino_port, baud_rate, timeout=1)
time.sleep(2)  # Wait for Arduino to initialize

# Function to read magnetometer data from the Arduino
def read_magnetometer():
while True:
line = arduino.readline().decode('utf-8').strip()
if line:
print(line)

# Start reading magnetometer data
read_magnetometer()
```

5.3.9. Further Readings

For more information, refer to:
Datasheets: Sensor-specific datasheets (e.g., HMC5883L, LSM303). Books:

- Ripka, A. (2001) Introduction to Magnetometers. New York: Wiley.
- Sherwood, T. (2013) Magnetic Sensors in Navigation Systems. Berlin: Springer.

Online Resources:

- Adafruit tutorials on magnetometers [Ada21].
- Research papers on magnetic anomaly detection [Han+17].

6. Domain System/Complete System

6.1. Description

The `Arduino Nano 33 BLE Sense` is a compact and low-power board ideal for IoT and AI applications, especially when space is a constraint. Its small size ($45mm \times 18mm$) and efficient power usage make it perfect for projects that require long battery life, operating at $3.3V$. This board comes with a variety of embedded sensors, making it versatile for a range of applications.[Gyr23] Below is a brief summary of the features on the board:

- **LEDs**: The board features three different LEDs for various purposes:
  - **RGB Programmable LED**: Allows you to program different colors and effects for visual feedback.
  - **Built-in Orange Programmable LED**: Can be used for status indication or customized signals.
  - **Power LED**: Indicates that the board is powered on.

- **Processor**: The board is powered by the `nRF52840` processor, a 32-bit ARM® Cortex™-M4 CPU running at 64 MHz, which is more powerful than many other Arduino boards, allowing it to handle more complex computations or real-time data processing.

- **USB-C Port**: The `Arduino Nano 33 BLE Sense` features a **USB-C port** for power and data transfer. This modern, reversible connector provides a reliable connection for both programming the board and powering it.

- **Power Button**: The board includes a **power button**, allowing you to manually power on or off the device without needing to disconnect it from the power source. This is useful for battery-powered projects, where conserving power or resetting the device is necessary.

The `Arduino Nano 33 BLE Sense` is ideal for low-power, compact devices that require real-time environmental sensing, motion tracking, or interactive capabilities, all within a small, portable form factor.[Pas+17]

6.2. Design

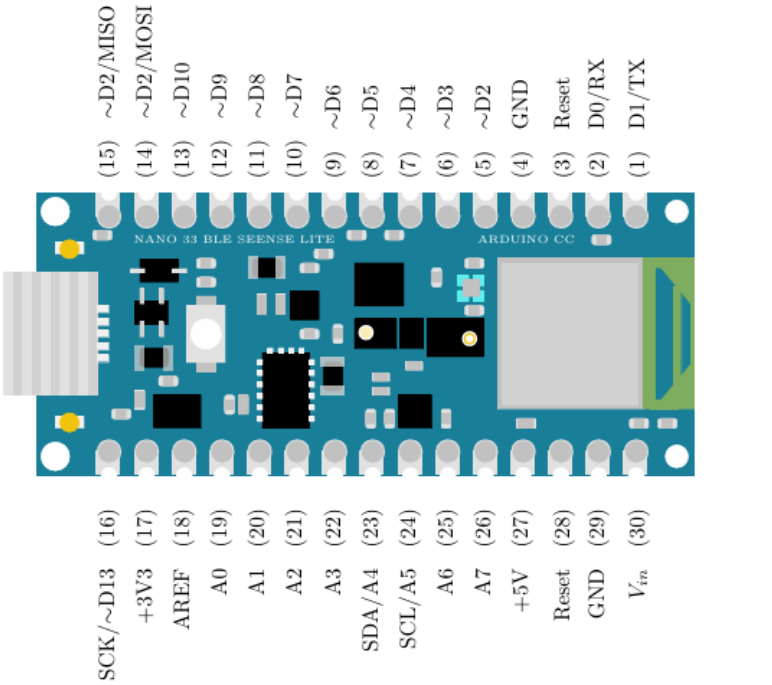


Figure 6.1.: **Pin Assignment of Arduino Nano 33 BLE Sense**

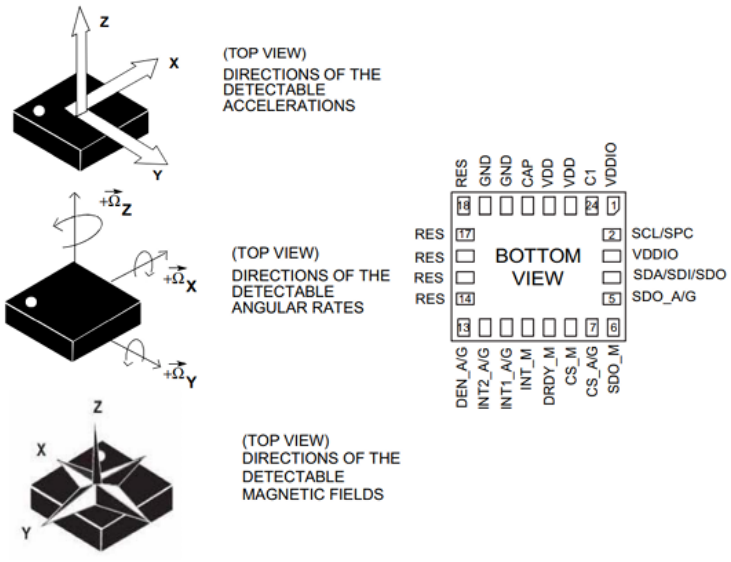


Figure 6.2.: **Pin Assignment of LSM9DS1**

Pin #	Name	Function
1	VDDIO(1)	Power supply for I/O pins
2	SCL/SPC	I²C serial clock (SCL) / SPI serial port clock (SPC)
3	VDDIO(2)	Power supply for I/O pins
4	SDA/SDI/SDO	I²C serial data (SDA) SPI serial data input (SDI) 3-wire interface serial data output (SDO)
5	SDO_A/G	SPI serial data output (SDO) for the accelerometer and gyroscope I²C least significant bit of the device address (SA0) for the accelerometer and gyroscope
6	SDO_M	SPI serial data output (SDO) for the magnetometer I²C least significant bit of the device address (SA0) for the magnetometer
7	CS_A/G	SPI enable I²C/SPI mode selection for the accelerometer and gyroscope (1: SPI idle mode / I²C communication enabled; 0: SPI communication mode / I²C disabled)
8	CS_M	SPI enable I²C/SPI mode selection for the magnetometer (1: SPI idle mode / I²C communication enabled; 0: SPI communication mode / I²C disabled)
9	DRDY_M	Magnetic sensor data ready
10	INT_M	Magnetic sensor interrupt
11	INT1_A/G	Accelerometer and gyroscope interrupt 1
12	INT2_A/G	Accelerometer and gyroscope interrupt 2
13	DEN_A/G	Accelerometer and gyroscope data enable
14	RES	Reserved. Connected to GND.
15	RES	Reserved. Connected to GND.
16	RES	Reserved. Connected to GND.
17	RES	Reserved. Connected to GND.
18	RES	Reserved. Connected to GND.
19	GND	0 V supply
20	GND	0 V supply
21	CAP	Connected to GND with ceramic capacitor(3)
22	VDD(4)	Power supply
23	VDD(5)	Power supply
24	C1	Capacitor connection (C1 = 100 nF)

Figure 6.3.: **Pin Description of LSM9DS1**

6.3. Hardware Interface

The Arduino Nano 33 BLE Sense is a small and powerful development board featuring various hardware interfaces for communication, control, and interaction. The following sections describe the available interfaces in detail:

1. Power and Input/Output Pins

- **Power Pin**: The board operates on 3.3V, which is the nominal voltage for most sensors on the board. The power input can be supplied either through the **USB-C** port or via the **VIN** pin (which can take an input voltage of 5V to 12V).
- **GND Pins**: The board provides multiple ground (GND) pins, which are essential for completing the electrical circuit when connecting external devices and sensors.
- **Digital I/O Pins**: The Arduino Nano 33 BLE Sense features 14 digital I/O pins, out of which 12 can be used as PWM (Pulse Width Modulation) outputs. These pins are capable of reading and writing digital signals, used to control LEDs, motors, or other digital devices.

- **Analog Input Pins**: There are 8 analog input pins (A0 to A7), which are used to measure analog voltages in the range of 0 to 3.3V, useful for connecting sensors such as potentiometers, light sensors, and others that output analog signals.

- **3.3V and 5V Output Pins**: These pins allow you to supply power to external devices. The 3.3V pin is useful for low-voltage components, while the 5V pin can be used for higher voltage devices.

- **Reset Pin**: The Reset pin is used to restart the microcontroller, typically used for debugging or reinitializing the device during a project.

2. USB-C Interface

- **USB-C Port**: The `Arduino Nano 33 BLE Sense` features a modern USB-C port that serves dual purposes:
  - **Programming Interface**: The USB-C connection allows you to upload your sketches (code) to the board via the Arduino IDE.
  - **Powering the Board**: The USB-C port provides a reliable power source to the board, typically at 5V, which is regulated down to 3.3V for the operation of the onboard sensors and components.

3. Communication Interfaces

- **Bluetooth Low Energy (BLE)**:
  - The Arduino Nano 33 BLE Sense includes Bluetooth Low Energy (BLE) support, powered by the nRF52840 chip. This allows for wireless communication with other BLE-enabled devices such as smartphones, tablets, and other microcontrollers.[STM24]
  - BLE allows for short-range communication and low energy consumption, making it ideal for IoT applications where the device needs to connect to other peripherals or networks without drawing much power.[STM24]

- **Serial Communication (UART)**:
  - The board supports UART communication, which is used for communication between the board and other devices such as sensors or serial interfaces. It uses the standard `TX` (transmit) and `RX` (receive) pins.

- **SPI and I2C Interfaces**:
  - The Arduino Nano 33 BLE Sense also supports SPI (Serial Peripheral Interface) and I2C (Inter-Integrated Circuit) protocols, which are widely used for communication with various sensors, actuators, and external modules.
  - **SPI Pins**: `MOSI`, `MISO`, `SCK`, and `CS`.
  - **I2C Pins**: `SCL` (Serial Clock Line) and `SDA` (Serial Data Line).

- **PWM Outputs**: The digital I/O pins (12 in total) can be configured as Pulse Width Modulation (PWM) outputs. This feature is useful for controlling devices such as motors or LEDs with varying intensity.

4. Onboard Sensors and Interfaces

The Arduino Nano 33 BLE Sense is equipped with a variety of sensors that interact directly with the microcontroller via internal buses or I2C/SPI interfaces. These sensors are pre-connected to the microcontroller, and data from them can be accessed directly in your program.

- **ADPS-9960 (Proximity, Light, RGB, and Gesture Sensor)**:
  - Connected via I2C, it provides proximity sensing, ambient light sensing, color recognition, and gesture detection. This sensor is perfect for interactive applications such as gesture-based controls.
- **LSM9DS1 (9-Axis IMU)**:
  - This sensor combines an accelerometer, gyroscope, and magnetometer, and it is typically accessed through an I2C or SPI interface for detecting motion, orientation, and magnetic fields. Ideal for wearable devices and motion sensing.
- **LPS22HB (Barometric Pressure Sensor)**:
  - This sensor, connected via I2C, measures atmospheric pressure, which is useful in weather stations and altitude sensing applications.
- **HTS221 (Humidity and Temperature Sensor)**:
  - Connected via I2C, it allows for accurate humidity and temperature measurements, useful for environmental monitoring and climate control systems.
- **MP34DT05 (Digital Microphone)**:
  - This microphone is used for capturing sound in real time and can be connected via I2S for high-quality audio signal processing. It is useful for voice recognition and audio detection applications.

5. LEDs and Indicators

- **RGB Programmable LED**:
  - A multi-color LED that can be programmed to display different colors, ideal for status indicators or visual feedback.
- **Built-in Orange Programmable LED**:
  - This LED can be used for status indication or custom visual alerts.
- **Power LED**:
  - A dedicated LED that indicates the board's power status. It turns on when the board is powered.

6. Reset and Debugging

- **Reset Pin**:
  - The Reset pin is used to manually restart the microcontroller. It can be triggered by an external circuit or button for a hardware reset.
- **Serial Debugging**:
  - The board supports serial communication, which allows you to send and receive data between the board and a connected computer for debugging and logging purposes.

6.4. Hardware Interfaces and Properties

The Arduino Nano 33 BLE Sense features a range of hardware interfaces, sensors, and components that are essential for various applications in IoT, AI, and embedded systems. Below are the properties of these hardware interfaces and sensors.

1. Power and Input/Output Pins

- **Power Pin**:
  - **Voltage**: 3.3V
  - **Current**: Can supply up to 150mA
  - **Purpose**: Supplies the power to the board and onboard sensors.
- **GND Pins**:
  - **Number**: Multiple GND pins are available for use.
  - **Purpose**: Used to complete the electrical circuit by providing the ground connection.
- **Digital I/O Pins**:
  - **Number**: 14 digital pins (D0 to D13)
  - **Voltage Level**: 3.3V logic (input and output)
  - **Max Current per Pin**: 7mA (recommended)
  - **PWM Capability**: 12 pins can be used for PWM output (D3, D5, D6, D9, D10, D11, D12, D13, D6, D7, D8, D9).
  - **Purpose**: Used for reading or controlling digital devices such as LEDs, switches, and sensors.
- **Analog Input Pins**:
  - **Number**: 8 analog pins (A0 to A7)
  - **Voltage Range**: 0V to 3.3V
  - **Resolution**: 12-bit (4096 discrete values)
  - **Purpose**: Used for reading analog signals from sensors (e.g., light sensors, temperature sensors).
- **3.3V and 5V Output Pins**:
  - **Voltage**: 3.3V or 5V
  - **Max Current for 3.3V Pin**: 150mA (regulated by the onboard voltage regulator)
  - **Purpose**: Used to power external components, such as sensors and actuators.
- **Reset Pin**:
  - **Purpose**: Resets the microcontroller to its initial state.
  - **Logic Level**: Active low (pulling this pin low resets the board).

2. USB-C Interface

- **USB-C Port**:
  - **Purpose**: Provides both power and data transfer between the board and the computer.
  - **Power**: Provides 5V, which is regulated to 3.3V for internal components.
  - **Data Transfer Rate**: Supports serial communication for programming and debugging.
  - **Reversible Connector**: Easier to plug in due to the reversible nature of the USB-C connector.

3. Communication Interfaces

- **Bluetooth Low Energy (BLE)**:
  - **Chipset**: nRF52840 (Nordic Semiconductor)
  - **Protocol**: Bluetooth 5.0, BLE (Bluetooth Low Energy)
  - **Range**: Up to 100 meters (depending on the environment)
  - **Purpose**: Used for wireless communication with compatible devices, such as smartphones or other microcontrollers.
- **Serial Communication (UART)**:
  - **Pins**: TX (Transmit), RX (Receive)
  - **Voltage Level**: 3.3V logic (TTL)
  - **Purpose**: Used for communication with other devices (e.g., sensors, displays) or for serial debugging.
- **SPI Interface**:
  - **Pins**: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Serial Clock), CS (Chip Select)
  - **Voltage Level**: 3.3V logic
  - **Purpose**: Allows high-speed communication with peripherals like sensors, displays, or memory chips.
- **I2C Interface**:
  - **Pins**: SDA (Serial Data), SCL (Serial Clock)
  - **Voltage Level**: 3.3V logic
  - **Purpose**: Used for communication with a wide range of sensors, actuators, and modules.
- **PWM Outputs**:
  - **Number**: 12 pins (D3, D5, D6, D9, D10, D11, D12, D13, D6, D7, D8, D9)
  - **Frequency Range**: 490 Hz to 1 kHz (dependent on the microcontroller and pin configuration)
  - **Resolution**: 8-bit (256 levels)
  - **Purpose**: Used for controlling the speed of motors, brightness of LEDs, and other devices requiring analog control.

4. Onboard Sensors and Properties

- **ADPS-9960 (Proximity, Light, RGB, and Gesture Sensor)**:
  - **Interface**: I2C
  - **Measurement Range**: Proximity sensing up to 10 cm, ambient light from 0 to 4000 lux, RGB color from 0 to 255, and gesture detection (up, down, left, right, forward, backward).
  - **Purpose**: Detects proximity, light, color, and gestures for interaction-based applications.
- **LSM9DS1 (9-Axis IMU)**:
  - **Interface**: I2C (SPI option)
  - **Accelerometer Range**: ±2g, ±4g, ±8g, ±16g
  - **Gyroscope Range**: ±245°/s, ±500°/s, ±2000°/s
  - **Magnetometer Range**: ±4 Gauss, ±8 Gauss, ±12 Gauss, ±16 Gauss
  - **Purpose**: Used for motion sensing, orientation tracking, and wearable devices.
- **LPS22HB (Barometric Pressure Sensor)**:
  - **Interface**: I2C
  - **Pressure Range**: 260 hPa to 1260 hPa
  - **Accuracy**: ±1 hPa
  - **Purpose**: Measures atmospheric pressure, ideal for altitude measurements and weather-related applications.
- **HTS221 (Humidity and Temperature Sensor)**:
  - **Interface**: I2C
  - **Humidity Range**: 0
  - **Temperature Range**: -40°C to 120°C
  - **Accuracy**: ±3
  - **Purpose**: Used for environmental monitoring, climate control, and weather stations.
- **MP34DT05 (Digital Microphone)**:
  - **Interface**: I2S (Inter-IC Sound)
  - **Frequency Range**: 20 Hz to 20 kHz
  - **Purpose**: Captures sound in real-time for audio sensing, speech recognition, and other audio applications.

5. LEDs and Indicators

- **RGB Programmable LED**:
  - **Color Range**: Full RGB spectrum
  - **Purpose**: Provides visual feedback for status, effects, and alerts.
- **Built-in Orange Programmable LED**:
  - **Purpose**: Used for status indication, such as power or error alerts.
- **Power LED**:
  - **Purpose**: Indicates when the board is powered on.

6. Reset and Debugging

- **Reset Pin**:
  - **Logic Level**: Active low (pull this pin low to reset the board)
  - **Purpose**: Resets the microcontroller to its initial state.
- **Serial Debugging**:
  - **Purpose**: Used for debugging applications via the serial monitor on the computer.

6.5. OS/Software Interface/Protocol

The Arduino Nano 33 BLE Sense integrates several software interfaces and protocols that enable communication with external devices, sensors, and peripherals. These software interfaces and protocols are essential for building complex applications in IoT, AI, and embedded systems.

1. Communication Protocols

- **Bluetooth Low Energy (BLE)**:
  - **Protocol**: Bluetooth 5.0, BLE (Bluetooth Low Energy)
  - **Library**: `ArduinoBLE`
  - **Purpose**: Enables wireless communication with compatible devices such as smartphones, tablets, or other Bluetooth-enabled devices.
  - **Features**:
    - Supports Peripheral, Central, and Broadcaster modes.
    - Low-power communication, ideal for battery-operated devices.
    - Suitable for short-range communication, typically up to 100 meters.
- **Serial Communication (UART)**:
  - **Protocol**: UART (Universal Asynchronous Receiver/Transmitter)
  - **Library**: `Serial`
  - **Purpose**: Used for communication with other devices (sensors, displays, etc.) or for debugging through a serial monitor.
  - **Features**:
    - Allows bi-directional data transfer between the board and connected devices.
    - Supports baud rates from 300 to 1,000,000 baud.
- **I2C (Inter-Integrated Circuit)**:
  - **Protocol**: I2C (Inter-Integrated Circuit)
  - **Library**: `Wire`
  - **Purpose**: Facilitates communication with a wide range of sensors and peripherals that support I2C.
  - **Features**:
    - Supports multiple devices on the same bus using unique device addresses.
    - Supports speeds of 100 kHz (standard mode) and 400 kHz (fast mode).

    - Supports master-slave communication.

- **SPI (Serial Peripheral Interface)**:
  - **Protocol**: SPI (Serial Peripheral Interface)
  - **Library**: `SPI`
  - **Purpose**: High-speed communication with external peripherals such as sensors, displays, and memory devices.
  - **Features**:
    - Uses four wires: MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Clock), and CS (Chip Select).
    - High-speed communication up to 10 Mbps or more.
    - Allows full-duplex data transfer.

- **PWM (Pulse Width Modulation)**:
  - **Protocol**: PWM (Pulse Width Modulation)
  - **Library**: `analogWrite`
  - **Purpose**: Used to simulate analog output, such as controlling the brightness of LEDs or speed of motors.
  - **Features**:
    - Controls the duty cycle of the signal from 0% to 100%.
    - 8-bit resolution (256 levels) for controlling devices.
    - Affects the average voltage output based on the duty cycle.

2. Sensor Libraries

- **ADPS-9960 (Proximity, Light, RGB, and Gesture Sensor)**:
  - **Library**: `Adafruit_APDS9960`
  - **Purpose**: Allows interaction with the ADPS-9960 sensor to measure proximity, light, color, and gestures.
  - **Features**:
    - Detects motion, proximity, and ambient light in real-time.
    - Can identify gestures like up, down, left, right, and forward.

- **LSM9DS1 (9-Axis IMU)**:
  - **Library**: `Adafruit_LSM9DS1`
  - **Purpose**: Allows interaction with the LSM9DS1 sensor for 3D motion sensing, including acceleration, rotation, and magnetic field strength.
  - **Features**:
    - Provides accelerometer, gyroscope, and magnetometer readings.
    - Supports multiple sensor ranges for both accelerometer and gyroscope.
    - Can be configured to use I2C or SPI communication.

- **LPS22HB (Barometric Pressure Sensor)**:
  - **Library**: `Adafruit_LPS22HB`
  - **Purpose**: Provides readings of atmospheric pressure, useful for altitude or weather applications.
  - **Features**:

    - Pressure range from 260 hPa to 1260 hPa.
    - High accuracy with ±1 hPa error margin.

- **HTS221 (Humidity and Temperature Sensor)**:
  - **Library**: `Adafruit_HTS221`
  - **Purpose**: Measures humidity and temperature for environmental monitoring applications.
  - **Features**:
    - Temperature accuracy of ±0.5°C and humidity accuracy of ±3%.
    - Humidity range of 0% to 100% RH.

- **MP34DT05 (Digital Microphone)**:
  - **Library**: `I2S`
  - **Purpose**: Allows real-time audio signal processing from the MP34DT05 microphone.
  - **Features**:
    - Supports high-fidelity audio capture with a frequency range of 20 Hz to 20 kHz.
    - I2S interface for digital audio output.

3. Software Environment and IDE

- **Arduino IDE**:
  - **Platform**: Windows, macOS, Linux
  - **Features**:
    - Integrated development environment for writing, compiling, and uploading code to Arduino boards.
    - Supports a wide range of libraries and pre-built examples for hardware components.

- **Arduino Mbed OS**:
  - **Purpose**: Supports development for ARM Cortex-M microcontrollers, including the nRF52840 chip on the Nano 33 BLE Sense.
  - **Features**:
    - Provides additional low-power features and real-time operating system (RTOS) capabilities.
    - Supports BLE communication, power management, and peripheral handling for advanced projects.

6.6. Installation

The installation process for the magic wand system using the Arduino Nano 33 BLE Sense includes hardware setup and final system integration.

1. **Connect the Board**: Use a USB-C cable to connect the Arduino Nano 33 BLE Sense to a computer for programming and power supply.

2. **Mount the Board**: Secure the board onto the wand structure using a protective enclosure or mounting brackets.

3. **Battery Installation**: Connect a 3.7V Li-Po battery for portable operation. Ensure proper polarity to avoid damage.

4. **Sensor Orientation**: Align the sensors correctly to match the intended axes (X, Y, Z). Mark the axes for easy reference during testing.

6.7. Configuration

1. Hardware Configuration

The Arduino Nano 33 BLE Sense is the core component of the system, with several integrated sensors and communication interfaces for interaction with the environment.

- **Arduino Nano 33 BLE Sense Board**:
  - **Microcontroller**: nRF52840 (ARM Cortex-M4 CPU, 64 MHz)
  - **Sensors**:
    - `ADPS-9960`: Proximity, ambient light, RGB, and gesture sensing.
    - `LSM9DS1`: 9-axis Inertial Measurement Unit (IMU), comprising accelerometer, gyroscope, and magnetometer.
    - `LPS22HB`: Barometric pressure sensor.
    - `HTS221`: Temperature and humidity sensor.
    - `MP34DT05`: Digital microphone for sound detection.
- **Communication Interfaces**:
  - Bluetooth Low Energy (BLE) for wireless communication.
  - I2C, SPI, and UART for sensor communication.
  - PWM for controlling actuators (e.g., LEDs, motors).
- **External Components** (Optional):
  - Actuators such as LEDs, vibration motors, or speakers for feedback.
  - Battery (Li-Po or similar) for portability and long operation.

2. System Configuration

The system configuration ensures that the Arduino Nano 33 BLE Sense can effectively manage sensor data and communicate with external devices while maintaining low power consumption.

- **Power Supply**:
  - Powered by a Li-Po battery or similar for portable operation.
  - The board's low-power consumption ensures extended battery life, suitable for long-term use in a portable setup.
- **Communication and Control**:
  - Bluetooth Low Energy (BLE) for wireless communication with external devices such as mobile apps or other BLE-enabled systems.
  - I2C for communication with most of the onboard sensors (e.g., ADPS-9960, LSM9DS1).
  - UART or SPI for debugging or additional sensor integration if needed.
  - PWM for controlling feedback devices (e.g., LED light intensity, motor speed).

- **Interaction Flow**:
  - User interactions are captured via gestures (ADPS-9960) and motion (LSM9DS1).
  - Feedback is provided through actuators like LEDs or motors.
  - Sensor data can be transmitted via BLE to a mobile device or IoT platform for further processing.
- **Connectivity**:
  - BLE enables the Arduino Nano 33 BLE Sense to communicate with smartphones, tablets, or other BLE devices for remote control or data exchange.

6.8. Data Quality

The quality of data processed by the Arduino Nano 33 BLE Sense is critical for accurate and reliable operation. Key dimensions are:

- **Accuracy**: Reflects true conditions; ensured by high-resolution sensors and calibration.
- **Precision**: Consistent measurements with minimal variance; aided by reliable communication protocols.
- **Timeliness**: Real-time data capture and transmission via high sampling rates and low-latency BLE communication.
- **Completeness**: Comprehensive data collection using multiple sensors and reliable protocols to avoid data loss.
- **Validity**: Data adheres to predefined formats and thresholds; supported by gesture recognition algorithms.
- **Consistency**: Uniform data across sessions ensured by sensor calibration and synchronized channels.
- **Reliability**: Robust sensors, low power consumption, and fault-tolerant design ensure dependable operation.

6.9. Data Quantity

The Arduino Nano 33 BLE Sense generates, processes, and transmits varying amounts of data depending on sensor usage and application requirements. The details are as follows:

1. Communication Data Throughput

- **BLE (Bluetooth Low Energy)**:
  - Maximum Throughput: 236 Kbps (practical).
  - Supports periodic data transmission or streaming.

2. Storage and Processing Requirements

- Memory Usage: Fits within 256 KB SRAM of the nRF52840 microcontroller.
- Real-time processing depends on CPU load and active sensors.

6.10. Constraints

The Arduino Nano 33 BLE Sense, as a compact and low-power device, faces several constraints in both hardware and system-level configurations. Key constraints are as follows:

1. Hardware Constraints

- **Power Supply**:
  - Operates at 3.3V, requiring efficient power management.
  - Limited battery capacity affects runtime in portable systems.
- **Processing Power**:
  - The 64 MHz nRF52840 processor may struggle with complex, real-time algorithms.
- **Memory**:
  - Limited to 256 KB SRAM and 1 MB Flash, constraining program size and data storage.
- **Sensor Accuracy**:
  - Sensors like LSM9DS1 require proper calibration to maintain accuracy.
  - Noise and interference can affect proximity, motion, and audio measurements.
- **Communication Range**:
  - BLE range is limited to 100 meters under ideal conditions, which decreases in high-interference environments.

2. System Constraints

- **Real-Time Performance**:
  - High sampling rates (e.g., 952 Hz for IMU) can overload the processor if multiple sensors are active simultaneously.
- **Data Throughput**:
  - BLE's practical bandwidth (236 Kbps) may limit streaming of high-frequency data, such as real-time audio.
- **Environment Sensitivity**:
  - Extreme environmental conditions (e.g., high humidity or temperature) can degrade sensor performance.
- **Latency**:
  - Gesture recognition and BLE communication introduce delays, impacting immediate responses in time-sensitive applications.

6.11. Dimensions

The Arduino Nano 33 BLE Sense is designed to be compact and lightweight, making it ideal for portable and space-constrained applications. The dimensions are as follows:

1. Physical Dimensions

- **Board Size**:
  - Length: **45 mm**.
  - Width: **18 mm**.
- **Thickness**: Approximately **3.5 mm** (excluding connectors).
- **Weight**: Approximately **5 grams**.

2. Connector Layout

- **Pin Count**: 15 pins on each side, totaling 30 pins.
- **USB Connector**: USB-C port for power and data transfer.
- **LED Indicators**:
  - RGB programmable LED.
  - Built-in orange programmable LED.
  - Power LED.

3. Hardware Dimensions

- **Power Supply Requirements**: Operates on **3.3V**, suitable for battery-powered systems.
- **Processing Core**: nRF52840, 64 MHz Cortex-M4.
- **Memory**: 256 KB SRAM and 1 MB Flash.
- **Communication**: BLE 5.0, UART, I2C, SPI.
- **Sensor Coverage**: Includes proximity, motion, environmental, and audio sensors.

6.12. Conclusion

The magic wand system, powered by the Arduino Nano 33 BLE Sense, showcases the effective use of compact and low-power hardware to enable advanced functionalities. The system utilizes the board's versatile sensors to achieve accurate motion tracking, environmental sensing, and gesture recognition.[Pas+17]
The domain system is designed to integrate real-time data processing, efficient power management, and BLE communication, ensuring seamless operation and portability. The hardware's small size, energy efficiency, and processing capabilities make it ideal for applications where space and power are limited.[Pas+17]
This successful combination of smart hardware and responsive system design demonstrates the potential for creating innovative and reliable solutions in IoT and AI-driven projects.[Pas+17]

Part III.

Methodology

7. Knowledge Discovery in Databases(KDD) Process

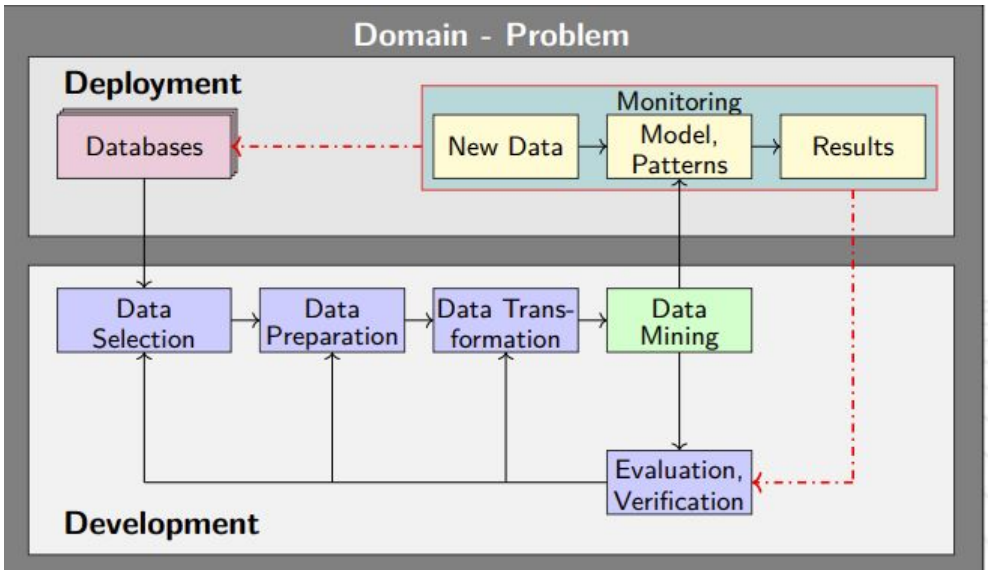


Figure 7.1.: **Process Workflow** [**Wings:2022**]

7.1. Introduction to KDD

Knowledge Discovery in Databases (KDD) is a fundamental step in the broader knowledge discovery process. It refers to the systematic extraction of meaningful patterns, trends, and insights from large datasets. At its core, KDD enables organizations and researchers to transform raw data into actionable knowledge. A key aspect of this process is data mining, which involves identifying interesting patterns within vast amounts of data.

By analyzing existing datasets, data mining facilitates the discovery of solutions to complex problems. This process involves leveraging historical data to build predictive models that anticipate future trends and behaviors. The ultimate objective of data mining is to uncover patterns that are not only valid and novel but also potentially useful and easily understandable. These insights often form the foundation for strategic decision-making across various industries [**Wings:2022**].

7.2. Types of Data Mining Tasks

Data mining tasks can generally be classified into two primary categories: **descriptive tasks** and **predictive tasks**.

7.2.1. Descriptive Tasks

Descriptive data mining focuses on analyzing datasets to uncover inherent patterns and summarize their properties. These tasks help characterize the general attributes of data, providing insights into the underlying structure of the database. For example, clustering techniques can group similar data points, offering a better understanding of the dataset's composition.

7.2.2. Predictive Tasks

Predictive data mining, on the other hand, involves using historical data to forecast unknown or future trends. These tasks rely on identifying relationships between variables to make informed predictions. For instance, regression and classification models can predict customer behavior, sales trends, or market dynamics [FAD18].

7.3. Applications of Data Mining

Data mining finds extensive applications across numerous industries, including:

- **Healthcare:** In medicine, data mining assists in analyzing patient records to predict disease progression, improve diagnostics, and tailor treatment plans. By identifying patterns in medical histories, healthcare providers can enhance patient care and operational efficiency.
- **Retail and Marketing:** Retailers use data mining to understand consumer behavior, optimize pricing strategies, and personalize marketing campaigns. Analyzing sales data helps identify purchasing patterns, leading to better inventory management and targeted promotions.
- **Finance:** In the financial sector, data mining plays a crucial role in fraud detection, credit scoring, and risk assessment. By analyzing transaction patterns, banks and financial institutions can mitigate risks and improve decision-making.
- **Telecommunications:** Telecommunications companies use data mining to analyze customer usage patterns, predict churn rates, and improve service quality. These insights enable companies to develop more effective customer retention strategies.
- **Scientific Research:** In science, data mining facilitates the analysis of large-scale datasets to uncover patterns and correlations that drive discoveries and innovations.

The insights derived from data mining empower organizations to improve profitability, enhance customer satisfaction, and optimize operations. The following sections delve deeper into specific data mining tasks and their methodologies.

7.4. KDD for Magic Wand Project

The integration of Knowledge Discovery in Databases (KDD) is a cornerstone in optimizing the design and performance of our Magic Wand project. This project leverages the Arduino Nano BLE 33, equipped with an accelerometer, to enable on-device gesture recognition. Through the application of KDD, meaningful insights are derived from the extensive dataset generated by the device's accelerometer, enabling the seamless recognition and classification of predefined gestures.

7.4.1. Understanding Descriptive and Predictive Tasks

The application of KDD in the Magic Wand project involves two primary categories of data mining tasks: descriptive and predictive.

- **Descriptive Tasks:** Descriptive tasks focus on analyzing accelerometer data to uncover inherent functional patterns. These patterns help characterize the general properties of hand movements, offering valuable insights into the gestures' nature and variations. For instance, clustering similar motion patterns can highlight commonalities in user behavior, aiding in feature engineering for the recognition model.
- **Predictive Tasks:** Predictive tasks utilize historical accelerometer data to anticipate future trends and behaviors. In the Magic Wand project, predictive modeling enables the system to identify and classify gestures in real time. By training the model on labeled datasets, it becomes capable of recognizing both simple and complex gestures with high accuracy [**Wings:2022**].

7.4.2. Applications of Data Mining in Gesture Recognition

Data mining, as a key component of KDD, empowers the Magic Wand project to extract meaningful patterns from raw accelerometer data. This capability is instrumental in recognizing valid and novel hand gestures. The process involves feature extraction, pattern identification, and classification to ensure precise gesture recognition. By enhancing the accuracy and responsiveness of the system, data mining significantly improves user experience and system reliability [FAD18].

7.4.3. Real-world Impact in Various Domains

The insights derived through KDD extend the Magic Wand project's potential applications across multiple industries:

- **Healthcare:** Gesture recognition systems powered by KDD can assist in physical therapy, enabling patients to perform prescribed exercises while the system monitors and evaluates their movements.
- **Retail and Marketing:** Retail environments can benefit from gesture-based interaction systems, such as virtual assistants or kiosks that respond to user gestures for navigation or selection.
- **Finance:** In banking and finance, gesture recognition can facilitate secure authentication mechanisms and improve accessibility for individuals with disabilities.
- **Telecommunications:** Gesture-based controls powered by data mining can enhance user interaction with smart devices, improving accessibility and functionality.
- **Science and Education:** Gesture recognition systems can be used in educational tools to provide interactive learning experiences, especially in virtual or augmented reality environments [FAD18].

7.4.4. CRISP-DM

The CRISP-DM process was developed by the means of the effort of a consortium initially composed with DaimlerChrysler, SPSS, and NCR. CRISP-DM stands for **Cross-Industry Standard Process for Data Mining**. It consists of a cycle that comprises six stages (Figure 2):

1. **Business understanding** – This initial phase focuses on understanding the project objectives and requirements from a business perspective, then converting this knowledge into a data mining problem definition and a preliminary plan designed to achieve the objectives.

2. **Data understanding** – The data understanding phase starts with an initial data collection and proceeds with activities in order to get familiar with the data, to identify data quality problems, to discover first insights into the data, or to detect interesting subsets to form hypotheses for hidden information.

3. **Data preparation** – The data preparation phase covers all activities to construct the final dataset from the initial raw data.

4. **Modeling** – In this phase, various modeling techniques are selected and applied, and their parameters are calibrated to optimal values.

5. **Evaluation** – At this stage, the model (or models) obtained is more thoroughly evaluated, and the steps executed to construct the model are reviewed to be certain it properly achieves the business objectives.

6. **Deployment** – Creation of the model is generally not the end of the project. Even if the purpose of the model is to increase knowledge of the data, the knowledge gained will need to be organized and presented in a way that the customer can use it.

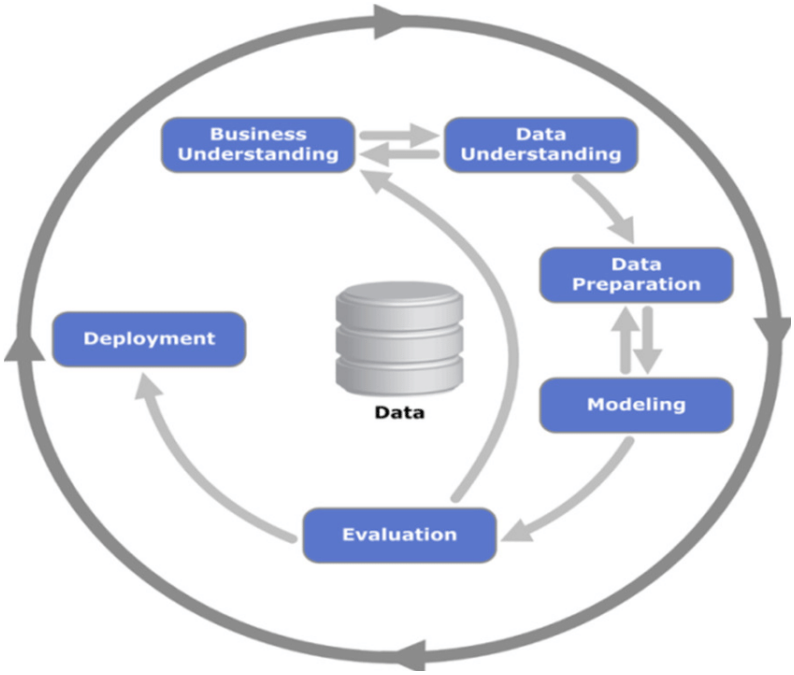


Figure 7.2.: The CRISP-DM life cycle (Chapman et al., 2000)

The sequence of the six stages is not rigid, as is schematized in Figure 2. CRISP-DM is extremely complete and documented. All its stages are duly organized, structured, and defined, allowing that a project could be easily understood or revised (Santos & Azevedo, 2005). Although the CRISP-DM process is independent from the DM chosen tool, it is linked to the SPSS Clementine software.

Aspect	KDD (Knowledge Discovery in Databases)	CRISP-DM (Cross-Industry Standard Process for Data Mining)
Origin and Scope	Focused on the overall process of extracting knowledge from databases.	Aimed at structuring data mining projects in a business or industrial setting.
Phases		
	1. Data Selection	1. Business Understanding
	2. Data Preprocessing	2. Data Understanding
	3. Data Transformation	3. Data Preparation
	4. Data Mining	4. Modeling
	5. Interpretation/Evaluation	5. Evaluation
		6. Deployment
Flexibility	High flexibility, emphasizes iterative exploration and hypothesis testing.	Prescriptive and structured, with an emphasis on well-documented steps for business clarity.
Tool Independence	Independent of tools and platforms.	Initially linked to SPSS Clementine (now IBM SPSS Modeler).
Target Audience	Academics and technical experts focused on novel insights and experimental exploration.	Business professionals requiring a structured, repeatable process for solving problems.
Documentation Emphasis	Less formalized; focuses on the iterative discovery of knowledge.	High emphasis on documentation for ease of project understanding and reproducibility.

Table 7.1.: Comparison of KDD and CRISP-DM

7.4.5. Comparison of KDD and CRISP-DM

7.4.6. Why KDD is Better for the Magic Wand Project

The magic wand project using the Arduino Nano 33 BLE Sense involves real-time sensor data processing and gesture recognition. Below are key reasons why KDD is better suited:

1. **Focus on Sensor Data Processing** The Arduino Nano 33 BLE Sense provides raw sensor data (e.g., accelerometer, gyroscope, magnetometer). KDD emphasizes *data preprocessing* and *transformation*, which are essential for converting raw data into meaningful features (e.g., gesture patterns). KDD's iterative nature allows for fine-tuning these transformations, crucial for a real-time system like the magic wand.

2. **Exploratory Nature of KDD** The magic wand project involves *exploring gestures* and creating patterns using embedded sensors. KDD allows repeated hypothesis testing and refinement, enabling the discovery of novel gesture mappings or better feature extraction techniques.

3. **Less Emphasis on Business Objectives** CRISP-DM's *business understanding phase* is less relevant for this project, which is *experimental and technical* rather

than business-driven. KDD focuses directly on the data and knowledge discovery, aligning well with the goals of the magic wand.

4. **Tool Independence** KDD is tool-agnostic and compatible with *open-source platforms* like Arduino. CRISP-DM, initially linked to SPSS Clementine (a proprietary tool), is less adaptable for hardware-based projects.

5. **Iterative Feedback** In KDD, the iterative cycle between preprocessing, transformation, and data mining is critical for adapting to sensor noise or calibration changes. This feedback loop is vital for optimizing gesture recognition in the magic wand project.

KDD is better suited for the *magic wand project using Arduino Nano 33 BLE Sense* because it provides the flexibility needed to work with raw sensor data, emphasizes preprocessing and transformation, and facilitates iterative exploration and optimization of gesture recognition models. These aspects align closely with the challenges and goals of the project, making KDD the ideal methodology.

7.4.7. ML Pipeline

7.4.8. 1. Data Selection

Objective: Identify and collect relevant sensor data.

- Use the embedded sensors (e.g., accelerometer, gyroscope, magnetometer) on the Arduino Nano 33 BLE Sense.
- Define the gestures to be recognized (e.g., wave, circle, swipe).
- Collect sensor data for each gesture, ensuring variation in speed, orientation, and user input.
- Store raw data for each axis (X, Y, Z) over time.

Tools: Arduino IDE for data logging, serial communication, or an SD card for storage.

7.4.9. 2. Data Preprocessing

Objective: Clean and prepare raw sensor data for analysis.

- **Data Cleaning**: Remove noise and outliers using filters (e.g., low-pass filter to remove high-frequency noise).
- **Data Resampling**: Normalize sampling rates for consistent data across all gestures.
- **Windowing**: Segment data into fixed-size time windows (e.g., 1-second windows) to capture meaningful patterns.
- **Scaling**: Normalize sensor values to a consistent range (e.g., [-1, 1] or [0, 1]).

Tools: Python (e.g., NumPy, Pandas), Arduino libraries (e.g., filters).

7.4.10. 3. Data Transformation

Objective: Extract features from the preprocessed data.

- Compute statistical features like mean, variance, standard deviation, energy, and entropy for each window.

- Calculate domain-specific features (e.g., Fast Fourier Transform (FFT) for frequency analysis).
- Combine features from multiple sensors (e.g., accelerometer + gyroscope).
- Encode gestures into labeled data (e.g., "gesture_1", "gesture_2").

Tools: Python (e.g., SciPy, sklearn), MATLAB (if needed).

7.4.11. 4. Data Mining (Modeling)

Objective: Train a machine learning model to classify gestures.

- **Split the dataset**: Use an 80-20 train-test split.
- **Model Selection**: Choose lightweight models suitable for edge devices, such as:
  - Decision Trees
  - Random Forest
  - k-Nearest Neighbors (k-NN)
  - Neural Networks (e.g., TensorFlow Lite for microcontrollers)
- **Training**: Optimize hyperparameters (e.g., tree depth, number of neighbors).
- **Validation**: Use cross-validation to ensure model generalization.

Tools: TensorFlow Lite, Edge Impulse, or sklearn.

7.4.12. 5. Interpretation/Evaluation

Objective: Evaluate model performance and refine it.

- **Metrics**:
  - Accuracy, precision, recall, and F1-score.
  - Confusion matrix to analyze gesture misclassifications.
  - Latency and memory usage to ensure real-time performance on the Arduino.
- **Steps**:
  - Test the model on unseen gesture data.
  - Evaluate performance under real-world conditions (e.g., different lighting or motion speeds).

Tools: Arduino Nano's onboard resources for testing, Python for analysis.

7.4.13. 6. Deployment

Objective: Deploy the trained model onto the Arduino Nano 33 BLE Sense.

- Convert the model into a format suitable for the Arduino (e.g., TensorFlow Lite format).
- Integrate the model with gesture recognition logic using Arduino libraries.
- Test gesture recognition in real-time and refine thresholds or logic as needed.
- Optimize for power consumption and memory.

Tools: TensorFlow Lite for Microcontrollers, Arduino IDE, Edge Impulse Studio.

7.4.14. Pipeline Flow Diagram

Sensor Data Collection → Preprocessing → Feature Extraction → Model Training → Evaluation → Deploymen

7.4.15. Conclusion

By employing the principles of KDD, the Magic Wand project effectively harnesses the potential of data mining to analyze and understand hand movements. This approach ensures accurate and real-time gesture recognition, paving the way for innovative applications across diverse industries.

8. Convolutional Neural Networks(CNN)

8.1. Introduction

Recent advancements in artificial intelligence, particularly in deep learning, have significantly impacted various cutting-edge technology applications, ranging from autonomous vehicles to creative fields like music and art generation. A central ambition within the scientific community is to enable computers to communicate and comprehend language in a manner akin to human interaction [Li+21]. As a subset of machine learning, deep learning stands out due to its focus on learning data representations rather than relying on predefined task-specific algorithms. This methodology empowers systems to build complex concepts by integrating information from simpler, fundamental elements.
A Convolutional Neural Network (CNN) has been selected to develop the model, building upon and enhancing the ideas proposed by Warden and Giménez [WS20a]. This choice makes CNN an ideal candidate for applications in voice recognition.
Convolutional Neural Networks (CNNs) are a specialized type of multi-layer neural network designed to identify visual patterns in images directly with minimal preprocessing. In artificial neural networks, a neuron serves as a transformative unit that processes inputs to produce outputs. The number of neurons utilized varies based on the specific task and can range from a few to several thousand. Artificial neurons are interconnected in various configurations to form a CNN. Within this structure, individual neurons receive inputs from others, with each input's weight determining its positive or negative influence.
The collective learning of the network enables it to perform critical computations required for tasks like object recognition and language understanding. These interconnected neurons form a feed-forward network, where outputs from one layer are passed to subsequent layers. This process continues until a final output is generated [Gu+18].

8.2. Description

Understanding the internal mechanisms of the model is not essential for its application; however, exploring these processes can be valuable for addressing potential issues and is intrinsically engaging. This section provides an overview of the model's predictive functionality. Within neural network architecture, a specific type excels in managing multidimensional tensors by leveraging the relationships between neighboring values—known as a Convolutional Neural Network (CNN). Although typically used for image processing, where inputs are 2D pixel grids, CNNs demonstrate exceptional adaptability in analyzing spectrogram data, highlighting their utility beyond traditional visual tasks [WS20a]. This section introduces the basic concepts of CNN. Furthermore, descriptions of crucial elements including the optimizer, loss function, and activation function.

8.2.1. CNN Components

Convolutional Neural Networks (CNNs) are a type of feedforward neural network that excel in automatically extracting features from data through their convolutional

architecture. Unlike traditional methods that rely on manual feature extraction, CNNs streamline this process by learning features directly from the data. Inspired by visual perception, CNNs employ kernels that act as artificial receptors, detecting various features and aligning artificial neurons with the behavior of biological neurons.
Activation functions, analogous to the biological threshold for signal transmission, simulate the electrical signaling between neurons. Meanwhile, loss functions and optimizers play a critical role in guiding the CNN to identify and learn the desired patterns effectively [Li+21].
Compared to fully connected (FC) networks (see Figure 8.1), CNNs offer several advantages:

- **Localized Connections:** In CNNs, neurons connect only to a subset of the previous layer rather than all neurons, reducing parameters and computational complexity. This structure helps focus on spatially relevant features and speeds up convergence.
- **Weight Sharing:** A single set of weights is reused across different locations in the input, capturing patterns like edges regardless of position. This reduces parameters and improves generalization.
- **Downsampling:** Pooling layers summarize data by retaining key features (e.g., max or average values), reducing spatial dimensions and computational cost while eliminating redundant information.

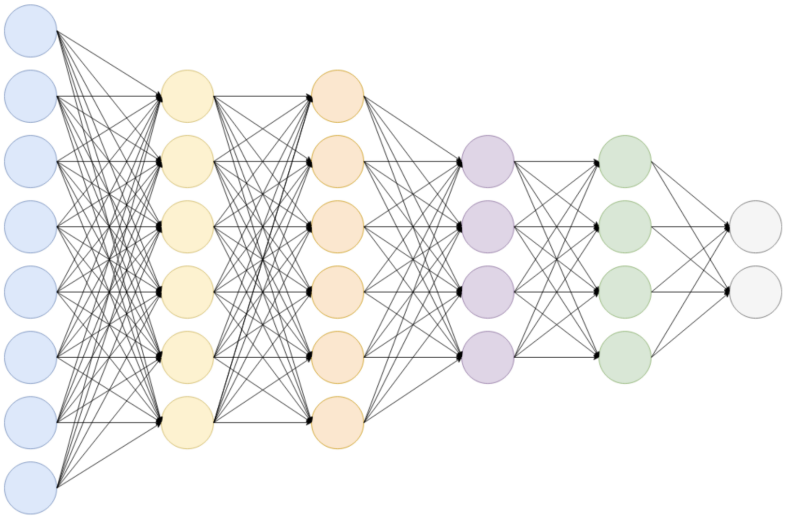


Figure 8.1.: CNN and FC layers [**Wings:2023**]

Convolution is a key step in the feature extraction process, generating feature maps. Padding, achieved by adding zero values to the input, helps prevent information loss at the edges of the input. Stride controls the density of the convolution operation. However, the feature maps produced can lead to overfitting, necessitating the use of pooling (either max pooling or average pooling) to eliminate redundant information. Figure 8.2 illustrates the overall CNN workflow. Below is a detailed explanation of padding, stride, and pooling [Li+21]:

- **Padding:** Padding refers to the addition of extra pixels around the edges of the input image before the convolution process. This is typically done by surrounding the image with rows and columns filled with zeros. The purpose of padding is to prevent the loss of information at the image borders during convolution.

- **Stride:** Stride refers to the step size, or the number of pixels the convolution filter moves across the image during the convolution operation. A larger stride results in smaller feature maps, which leads to a more compact representation.

- **Pooling:** Pooling is a down-sampling technique applied after convolution that reduces the size of the feature maps. It helps control the number of parameters in the network. Common pooling methods include max pooling and average pooling [**Wings:2023**].

    1. **Max Pooling:** Max pooling is a pooling operation where, for each region of the feature map, the highest value is selected. This preserves the most important features from that region, providing a compact summary of the spatial information.

    2. **Average Pooling:** Average pooling is another pooling method where the average value of each region in the feature map is computed. This technique reduces the spatial dimensions while preserving a smoother, less detailed version of the features.

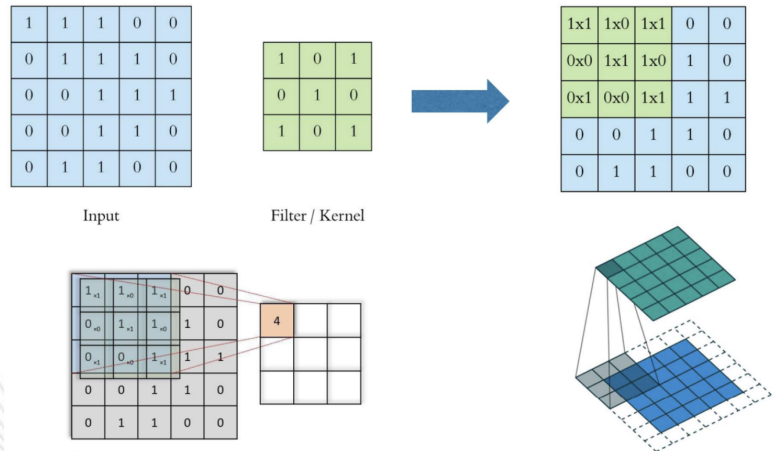
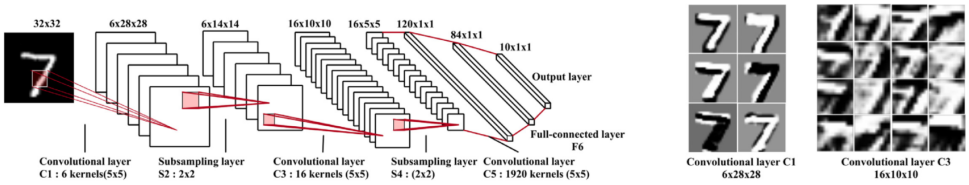


Figure 8.2.: Procedure of a two-dimensional CNN [**Wings:2023**]

The field of Convolutional Neural Networks (CNNs) includes numerous architectural variations, but their fundamental components remain consistent. For instance, the well-known LeNet-5 architecture consists of three primary layers: convolutional, pooling, and fully-connected layers [Gu+18]. The primary role of the convolutional layer is to extract meaningful feature representations from the input data.
As illustrated in Figure 8.3a, the convolutional layer comprises multiple convolutional kernels, each responsible for generating unique feature maps. Each neuron in a feature map is connected to a specific region of neurons in the preceding layer, referred to as its receptive field. The creation of a feature map involves convolving the input data with a learned kernel, followed by applying an element-wise nonlinear activation function. Importantly, each feature map is generated by sharing the kernel across all spatial regions of the input, with multiple distinct kernels utilized to produce the complete set of feature maps.
Mathematically, the feature value at location (i, j) in the kth feature map of the lth layer (denoted as $z^l_{i,j,k}$) is computed by the expression:

$$z^l_{i,j,k} = \mathbf{w}^{l\,T}_k \mathbf{x}^l_{i,j} + b^l_k \tag{8.1}$$



(a) LetNet-5 network (b) Learned features

Figure 8.3.: The architecture of the LeNet-5 network [Gu+18]. (a) The architecture of the LeNet-5 network, renowned for its effectiveness in digit classification tasks. (b) Displaying the features within the LeNet-5 network through visualizations, where each layer's feature maps are showcased in distinct blocks.

Here, $\mathbf{w}_k^l$ and b_k^l denote the weight vector and bias term of the kth filter in the lth layer, respectively, while $\mathbf{x}_{i,j}^l$ represents the input patch centered at location (i, j) in the lth layer. The weight-sharing mechanism, where the same kernel is used to generate the feature map across the input, helps reduce model complexity and improves the efficiency of network training.
The activation function $(a(\cdot))$ introduces nonlinearity into the CNN, which is essential for detecting complex, nonlinear patterns in multi-layer networks. The activation value $(a_{l,i,j,k})$ of the convolutional feature $z_{l,i,j,k}$ is calculated as:

$$a_{i,j,k}^l = a(z_{i,j,k}^l) \tag{8.2}$$

Common activation functions include sigmoid, tanh, and ReLU.The pooling layer, positioned between convolutional layers, aims to achieve shift-invariance by downsampling the feature maps' resolution. Each feature map in the pooling layer connects to its corresponding feature map in the preceding convolutional layer. The pooling operation, denoted as pool$(\cdot)$, is applied to each feature map $a_{m,n,k}^l$ with:

$$y_{i,j,k}^l = \text{pool}(a_{m,n,k}^l), \forall (m, n) \in \mathcal{R}_{ij} \tag{8.3}$$

In this context, $\mathcal{R}_{ij}$ represents a local neighborhood around the location (i, j). Common pooling techniques include average pooling and max pooling. As shown in Figure 8.3b for the digit 7, the learned feature maps progressively capture hierarchical patterns. Early layers extract low-level features, such as edges and curves, while deeper layers encode more abstract and high-level representations.
After the convolutional and pooling layers, one or more fully-connected layers are often used for high-level reasoning. These layers establish dense connections by linking all neurons from the previous layer to each neuron in the current layer, generating global semantic representations. A fully-connected layer can also be replaced by a 1×1 convolutional layer to achieve similar functionality.
The final layer in CNNs serves as the output layer. For classification tasks, the softmax function is frequently used. Alternatively, CNN features can be combined with Support Vector Machines (SVM) to address various classification challenges. Let the desired input-output relationships be represented as $\{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}); n \in [1, \ldots, N]\}$. The CNN parameters, denoted by θ (including weight vectors and bias terms), are optimized by minimizing a task-specific loss function:

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \ell(\theta; \mathbf{y}^{(n)}, \mathbf{o}^{(n)}) \tag{8.4}$$

Here, $\mathcal{L}$ represents the loss function, which quantifies the discrepancy between the predicted output $(\mathbf{o}^{(n)})$ and the actual target label $(\mathbf{y}^{(n)})$ for nth input data point

$\mathbf{x}^{(n)}$. The symbol θ denotes all the parameters of the CNN, including weight vectors and bias terms. The function $\ell(\theta; \mathbf{y}^{(n)}, \mathbf{o}^{(n)})$ is the individual loss for a specific data point, measuring the dissimilarity between the predicted output and the true label. The goal during training is to minimize the average loss over all data points, achieved through techniques such as stochastic gradient descent.
The training of a CNN involves global optimization, with the aim of finding the best set of parameters by minimizing the loss function. Stochastic gradient descent emerges as a common solution for optimizing CNN networks, providing an effective means of iterative parameter adjustment.

8.2.2. Activation Function

Convolutional Neural Networks (CNNs) utilize a variety of activation functions to capture complex features [Li+21]. Similar to the behavior of neurons in the human brain, the activation function acts as a mechanism that determines whether information is passed to the next neuron. In a multilayer neural network, activation functions are positioned between layers, as shown in Figure 8.4.
In Figure 8.4, x_i represents the input features, with n features simultaneously fed into neuron j. The term w_{ij} corresponds to the weight of the connection between input feature x_i and neuron j, b_j represents the bias (internal state) of neuron j, and y_j is the neuron's output. The activation function, denoted as $f(\cdot)$, may include options such as sigmoid, tanh, or rectified linear unit (ReLU), among others.
If no activation function or a linear activation function is applied, the network's output becomes a simple linear combination of the inputs, which restricts its learning ability. Introducing nonlinear activation functions, such as sigmoid or tanh, enables the network to model and fit more complex patterns in the data.

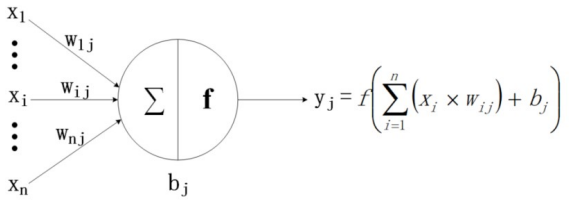
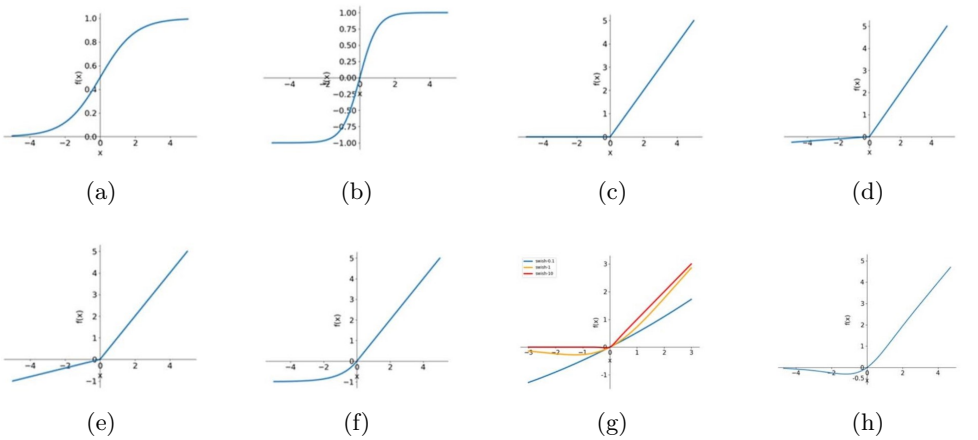


Figure 8.4.: Structure of an activation function [Li+21].

Some of the well-known actiovatoion functions are shown in the Figure 8.5. The sigmoid function maps a real number to (0, 1), suitable for binary classification. The tanh function maps to (-1, 1), aiding normalization. ReLU, with its advantages in learning speed, is preferred in deep networks. Leaky ReLU and PReLU address limitations of ReLU, reducing neuron inactivation. ELU improves convergence by having a negative output average.
Swish, proposed by Google, and mish, a novel activation function, demonstrate improved performance compared to ReLU and Swish in deeper models across diverse datasets. mish, in particular, exhibits superior gradient flow, accuracy, and generalization properties. Experimental results consistently support the effectiveness of Mish across standard architectures.

Guidelines for Activation Function Selection

Choosing the right activation function is critical for optimizing the performance of a neural network. Below are some general guidelines to assist in selecting an appropriate activation function:



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 8.5.: Diagrams of activation functions [Li+21]. (a) Sigmoid function. (b) Tanh function. (c) ReLU function. (d) Leaky ReLU function. (e) PReLU function. (f) ELU function. (g) Swish function. (h) Mish function.

1. For **binary classification** tasks, use the sigmoid function in the output layer, as it effectively maps outputs to probabilities between 0 and 1. For **multiclass classification**, the softmax function is more suitable as it normalizes outputs into a probability distribution across multiple classes.

2. While the sigmoid and tanh functions are effective for introducing nonlinearity, they may cause **gradient vanishing problems**, especially in deep networks. To mitigate this, activation functions like ReLU (Rectified Linear Unit) or leaky ReLU are recommended for hidden layers, as they provide faster convergence and avoid saturation.

3. When there is **uncertainty** about which activation function to use, ReLU or leaky ReLU can serve as reliable starting points, offering simplicity and consistent performance in most scenarios.

4. If a large number of neurons become **inactive** during training, resulting in the "dying ReLU" problem, consider alternatives such as leaky ReLU, Parametric ReLU (PReLU), or exponential linear unit (ELU). These activation functions ensure that inactive neurons can still contribute to the learning process.

5. To **speed up training**, set the negative slope of leaky ReLU to a small value like 0.02. This adjustment prevents neurons from becoming entirely inactive and facilitates faster convergence, particularly in challenging optimization landscapes.

Guidelines for Activation Function Selection

Choosing the right activation function is critical for optimizing the performance of a neural network. Below are some general guidelines to assist in selecting an appropriate activation function:

1. For **binary classification** tasks, use the sigmoid function in the output layer, as it effectively maps outputs to probabilities between 0 and 1. For **multiclass classification**, the softmax function is more suitable as it normalizes outputs into a probability distribution across multiple classes.

2. While the sigmoid and tanh functions are effective for introducing nonlinearity, they may cause **gradient vanishing problems**, especially in deep networks.

To mitigate this, activation functions like ReLU (Rectified Linear Unit) or leaky ReLU are recommended for hidden layers, as they provide faster convergence and avoid saturation.

3. When there is **uncertainty** about which activation function to use, ReLU or leaky ReLU can serve as reliable starting points, offering simplicity and consistent performance in most scenarios.

4. If a large number of neurons become **inactive** during training, resulting in the "dying ReLU" problem, consider alternatives such as leaky ReLU, Parametric ReLU (PReLU), or exponential linear unit (ELU). These activation functions ensure that inactive neurons can still contribute to the learning process.

5. To **speed up training**, set the negative slope of leaky ReLU to a small value like 0.02. This adjustment prevents neurons from becoming entirely inactive and facilitates faster convergence, particularly in challenging optimization landscapes.

Here's the expanded version with added details and the corresponding LaTeX code:

8.2.3. Optimizer

In CNNs, optimizing nonconvex functions is a critical step, often requiring robust optimizers to efficiently minimize the loss function within a reasonable timeframe [Li+21]. Nonconvex functions present complex landscapes with multiple local optima, making optimization challenging. These functions lack the convex property, where a line segment connecting any two points lies entirely within the function's domain, complicating the search for a global minimum. Effective optimization is crucial for improving model accuracy and convergence speed.

Gradient Descent

Gradient descent is a fundamental technique for training CNNs, with three common variants: batch gradient descent (BGD), stochastic gradient descent (SGD), and mini-batch gradient descent (MBGD).

- **Batch Gradient Descent (BGD):** BGD calculates the average gradient over the entire dataset for each update. While it provides a stable descent direction, it is computationally expensive and impractical for large datasets due to memory constraints.

- **Stochastic Gradient Descent (SGD):** SGD updates the model parameters using a single data sample per iteration, making it suitable for online learning. However, it can result in high variance and oscillatory convergence due to noisy updates.

- **Mini-Batch Gradient Descent (MBGD):** MBGD strikes a balance between BGD and SGD by computing updates using small batches of data. It reduces variance compared to SGD and is computationally efficient, making it a popular choice for modern deep learning tasks.

- **Gradient Descent Optimization Algorithms**

  Several advanced algorithms based on MBGD have been developed to address its limitations:

  – **Momentum:** Introduces a velocity term to accelerate convergence and reduce oscillations in the optimization path.

  - **Nesterov Accelerated Gradient (NAG):** Improves momentum by predicting the next position, allowing the optimizer to slow down when approaching steep slopes.
  - **Adagrad:** Adapts the learning rate for each parameter, making it effective for sparse datasets, though it suffers from a monotonically decreasing learning rate.
  - **Adadelta:** Addresses the diminishing learning rate issue in Adagrad by limiting updates to a window of recent gradients.
  - **RMSprop:** Modifies Adagrad by normalizing gradients using an exponentially decaying average, maintaining a consistent learning rate.
  - **Adam:** Combines the benefits of momentum and RMSprop, making it versatile and widely used in CNNs.
  - **Other Variants:** Algorithms like AdaMax, Nadam, and AMSGrad provide further enhancements to address specific issues like stability and overshooting.

- **Guidelines for Optimizer Selection:**
  1. Employ **Mini-Batch Gradient Descent (MBGD)** for a balance between computational efficiency and gradient stability.
  2. Recognize that optimizer performance depends on the dataset's characteristics. According to the **No Free Lunch Theorem**, no optimizer is universally superior, so selection should be based on the task requirements and data distribution.
  3. If convergence issues such as oscillation or divergence occur, consider reducing the learning rate to stabilize updates.

8.2.4. Feature Extraction

In feature extraction, convolutional layers (convolutional layer and pooling layer) are extracted from a trained CNN and a new classifier, i.e. a fully connected layer with new classifications, is added. Two problems arise in this process:

- The representation only contains information about the probability of occurrence of the class trained in the basic model.
- The fully connected layer contains no information about where the object is located.

Should the position of an object in the image matter, the fully connected layers are largely useless, as the later layers only generate abstract concepts. [Cho18]

8.2.5. AlexNet

The most popular CNNs for object detection and classification from image data are AlexNet, GoogleNet and ResNet50. [SJM18]
Gu et al. [Gu+18] refer to the development of AlexNet in 2012 as a breakthrough in large-scale image classification. Nevertheless, its architecture is only comlpex enough to keep its operation comprehensible. Therefore, it seems a good choice for first attempts at your own training.
AlexNet consists of five Convolutional Layers and three Fully Connected Layers. The number and size of the filters as well as their stride are shown in table 8.1 for the sake of clarity. The first and second convolutional layers are each followed by a response normalisation layer, also known as batch normalisation. This additional

layer is intended to mitigate the effect of unstable gradients through normalisation. After the response normalisation layers as well as after the fifth convolutional layer, a max-pooling layer follows. Overlapping pooling was chosen as the pooling method. This means that the individual sections to which pooling is applied overlap. The pooling area measures 3x3 and the stride 2 pixels. ReLu is applied to the output of each Convolutional Layer and each of the Fully Connected Layers. Each Fully Connected Layer consists of 4096 neurons. The output layer uses the softmax function to map the probabilities for the different categories via the output neurons. [KSH12; Ala08]

Convolutional Layer	Anzahl Kernel	Dimension	Stride
1	96	$11 \times 11 \times 3$	4
2	256	$5 \times 5 \times 48$	1
3	384	$3 \times 3 \times 256$	1
4	384	$3 \times 3 \times 192$	1
5	256	$3 \times 3 \times 192$	1

Table 8.1.: Convolutional Layer Specifications

In total, AlexNet consists of 650000 neurons. About 60 million parameters need to be trained. [KSH12]
This puts it about in the middle compared to the number of learnable parameters in other successful CNNs.
The input images for AlexNet need to be cropped to size 227×227. Three colour channels are used (see table 8.1), so no conversion to greyscale is necessary, but one usually normalises the data to mean 0 and standard deviation 1.[Ala08]

8.3. Applications

The utilization of Convolutional Neural Networks (CNN) in computer vision has made possible achievements that were once deemed impossible over the past centuries. These accomplishments include facial recognition, autonomous vehicles, self-service supermarkets, and intelligent medical treatments [Li+21].

8.3.1. 1-D CNN Applications

1. **Time Series Prediction:**
   - Time series prediction involves forecasting future values based on historical data. CNNs are particularly useful for detecting temporal patterns and trends in time series data.
   - Common applications include predicting electrocardiogram (ECG) signals to monitor heart health, weather forecasting for climate studies, and traffic flow prediction to optimize transportation systems.
   - **Example:** Predicting atrial fibrillation by analyzing short-term ECG recordings, aiding in early detection and treatment of cardiac conditions.
2. **Signal Identification:**
   - Signal identification uses CNNs to classify and differentiate input signals by extracting relevant features from the data.
   - This technique is widely applied in fields like medical diagnostics (e.g., identifying ECG abnormalities), structural engineering (e.g., detecting structural damage in buildings or bridges), and fault detection in industrial systems.

- **Example:** Classifying ECG signals to identify arrhythmias or anomalies in heartbeat patterns, enhancing diagnostic accuracy.

8.3.2. 2-D CNN Applications

- **Magic Wand Project:**
- The Magic Wand Project involves the use of artificial neural networks in interactive devices, where the magic wand serves as an intuitive input tool that enables users to perform various actions via hand gestures.
- While gesture recognition has been explored for many years, the integration of neural networks since the 2010s has greatly enhanced the precision and responsiveness of gesture-based interfaces.
- Unlike traditional rule-based systems, neural networks improve the wand's ability to interpret complex hand movements, offering more dynamic and accurate user interaction.
- A crucial aspect of the magic wand project is the real-time analysis of sensor data, necessitating efficient neural network architectures that can rapidly detect gesture patterns and convert them into actionable commands.
- Applications include:
  - *Interactive User Interfaces:* Enabling control of various functionalities through simple hand gestures.
  - *Gesture-based Gaming:* Allowing users to interact with virtual objects or control game characters by making specific gestures with the magic wand.
  - *Smart Home Control:* Using hand gestures to control lights, temperature, or appliances in smart homes.
- The use of artificial neural networks has been crucial in improving the interactivity and usability of the magic wand, making it a versatile and efficient tool.
- A key consideration in the design of this device is user privacy. Measures are implemented to ensure that sensor data is processed only when specific, predefined gestures (acting as "keywords") are detected by the neural network.
- This approach not only enhances efficiency by minimizing unnecessary data transmission but also addresses privacy concerns regarding continuous sensor monitoring, providing a secure and responsible foundation for the magic wand project [**Alushi:2024**].
- **Image Classification:**
  - Image classification is one of the most common applications of Convolutional Neural Networks (CNNs). The goal is to assign a label to an input image from a predefined set of categories.
  - CNNs achieve remarkable performance in image classification tasks by learning hierarchical feature representations from raw image pixels. Early layers of the network detect low-level features such as edges, corners, and textures, while deeper layers capture higher-level features like shapes, objects, or patterns.
  - Applications include:
    - *Medical Imaging:* Identifying diseases such as pneumonia, cancer, or diabetic retinopathy from X-rays, CT scans, and retinal images.

    - *Facial Recognition:* Matching faces to identities in security systems, social media platforms, and smartphones.
    - *Retail and E-commerce:* Automatically tagging products in images to improve search and recommendation systems.
  - Notable models such as AlexNet, VGGNet, and ResNet have set benchmarks for image classification tasks, showcasing the power of CNNs in this domain.

- **Object Detection:**
  - Object detection involves identifying and localizing objects within an image by drawing bounding boxes around them. Unlike image classification, which assigns a single label to an image, object detection provides detailed information about multiple objects and their locations.
  - CNNs use specialized architectures such as Region-Based CNNs (R-CNN), YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector) to achieve high accuracy and speed in object detection tasks.
  - Applications include:
    - *Autonomous Vehicles:* Detecting pedestrians, traffic signs, vehicles, and other road elements to ensure safe navigation.
    - *Surveillance Systems:* Identifying suspicious activities or objects in real-time video feeds.
    - *Retail Inventory Management:* Monitoring stock levels by identifying products on shelves.
    - *Agriculture:* Detecting diseases in plants or estimating crop yield by analyzing aerial images.
  - Advanced techniques such as multi-scale detection and anchor boxes improve the performance of CNNs in handling objects of varying sizes and aspect ratios within a single image.

8.3.3. Multidimensional CNN Applications

1. **Human Action Recognition:** [**Mardiyanto:2017**]
   - Human action recognition refers to the task of identifying actions or activities performed by humans in video sequences. This involves recognizing both spatial and temporal patterns in the video data.
   - 3D CNNs are particularly effective for extracting spatiotemporal features from video frames, as they can process both the spatial dimensions (height, width) and the temporal dimension (time) of the video. These networks are capable of capturing motion dynamics and interactions between objects and humans in video data.
   - In some systems, 2D CNN features (extracted from individual frames) are integrated with 3D CNN features (extracted from video sequences), enhancing the model's ability to capture complex actions and movements across time.
   - Applications include:
     - *Surveillance Systems:* Identifying suspicious or abnormal human activities in video surveillance footage.
     - *Human-Computer Interaction:* Enabling gesture-based controls and actions in virtual or augmented reality environments.
     - *Sports Analytics:* Analyzing athletes' actions and movements to assess performance and improve training.

2. **Object Recognition/Detection in 3D:**
    - Object recognition and detection in 3D involves identifying objects in 3D space based on their geometric and visual properties. This is a critical task for applications requiring understanding of the spatial relationships between objects.
    - 3D CNNs are applied to 3D data, such as point clouds, voxel grids, or RGBD images (which combine RGB images with depth information). These models capture both the shape and spatial distribution of objects in 3D space.
    - Notable approaches include the use of 3D ShapeNet and VoxNet for 3D object recognition tasks, where voxelized representations of 3D models are processed by CNNs.
    - Applications include:
        - *Autonomous Vehicles:* Recognizing and localizing obstacles (e.g., pedestrians, vehicles) in 3D space using depth cameras and LiDAR sensors.
        - *Medical Imaging:* Analyzing 3D medical images like CT scans and MRIs for diagnosing conditions such as tumors or fractures.
        - *Robotics:* Enabling robots to recognize and manipulate objects in 3D environments for tasks such as pick-and-place or assembly.
    - Advanced 3D object detection models can handle high-dimensional images, enabling the recognition of complex shapes and structures in various applications, from healthcare to autonomous navigation.

8.4. CNN for Magic Wand

Selecting a Convolutional Neural Network (CNN) architecture for the Magic Wand Project is a well-justified decision based on the unique characteristics of the input data, which consists of sensor readings or gesture patterns. In the context of the Magic Wand Project, the input data is typically represented as multidimensional arrays, reflecting the sensor outputs that capture the dynamic nature of the hand gestures. CNNs, specifically designed for handling multidimensional tensors, are well-suited for extracting spatially hierarchical features from such data structures [WS20a].
CNNs are traditionally renowned for their success in image processing tasks, where they excel at identifying patterns and objects in pixel-based data. However, their versatility extends beyond image data, making them highly effective for processing other forms of multidimensional vector inputs, such as those produced by the sensors in the magic wand system.
The data generated by the Magic Wand Project shares a key similarity with images, as it is often structured as two-dimensional grids, with each grid representing the values from different sensors at various time steps. This two-dimensional structure makes CNNs an ideal choice for analyzing gesture dynamics. By leveraging the convolutional layers, which can capture local dependencies in a two-dimensional space, CNNs can effectively learn and interpret the relationships within the input data, thus recognizing intricate gesture patterns with precision.
Furthermore, CNNs are capable of handling noise and variations in input, which is critical for gesture recognition systems where sensor data can be influenced by external factors such as lighting, angle, or user movement. The hierarchical feature extraction capability of CNNs allows the model to discern subtle patterns in the sensor data, making it possible to detect complex gestures accurately.
By employing CNNs, the Magic Wand Project benefits from a powerful tool for feature extraction, enabling the neural network to classify and interpret gesture patterns

in real-time. This ensures the system is both responsive and capable of providing a seamless user experience, driving the project's success in creating interactive and intuitive gesture-based interfaces [Xu+22].

8.5. Hyperparameters

Tuning activation functions, loss functions, optimizers, and other hyperparameters is crucial for the model's success. It is widely recognized that there is no universal set of hyperparameters that guarantees optimal results for every task. Hence, the process of hyperparameter tuning in the Magic Wand Project requires a combination of expertise and established guidelines to achieve the best model performance [Li+21].

- **Learning Rate:** The learning rate determines the step size at which the weights of the neural network are updated during training. In the context of the magic wand project, selecting an appropriate learning rate is essential for maintaining a balance between fast convergence and model stability. A learning rate that is too high may cause the model to converge too quickly, possibly missing the optimal solution, while a rate that is too low may lead to slow learning or getting stuck in suboptimal solutions.

- **Epoch:** The epoch refers to the number of complete passes through the entire dataset of gesture samples during training. In the magic wand project, adjusting the number of epochs is important to avoid both underfitting and overfitting. A lower number of epochs may result in underfitting, where the model doesn't learn the gesture patterns well enough, while too many epochs may lead to overfitting, where the model becomes too specialized to the training data and performs poorly on new, unseen data. Monitoring the gap between training and validation set accuracy can help determine the optimal number of epochs.

- **Mini-Batch Size:** The mini-batch size refers to the number of gesture samples processed in each iteration during training. The choice of batch size influences both the convergence rate and the stability of the training process. A smaller batch size may lead to noisy gradient estimates, but it could help generalize better, while a larger batch size offers more stable gradient updates but requires more memory. In the context of the Magic Wand Project, the batch size should be selected according to the computational limits of the device (e.g., Arduino Nano), balancing training efficiency and the accuracy of gesture recognition.

- **Number of Conv Layers:** The number of convolutional layers defines the depth of the neural network and its ability to learn different levels of features from the gesture data. Deeper networks with more convolutional layers can learn more complex and abstract features. However, increasing the number of layers also raises the risk of overfitting, especially if the dataset is small. The architecture of the convolutional layers should be carefully tuned to ensure that the network can capture both low-level features (such as edges) and high-level representations (such as hand gestures).

- **Conv Kernel Size:** The convolution kernel size, which determines the spatial extent of the convolutional filter, plays an important role in extracting features from the input data. For gesture recognition, kernel sizes such as 3x3 or 1x1 are commonly used. A larger kernel size captures more spatial context, but it can increase the computational load and potentially blur finer details, while a smaller kernel size captures more localized patterns. The choice of kernel size depends on the nature of the gesture data and the desired feature extraction.

- **Number of Filters (Kernels):** The number of filters in each convolutional layer determines how many feature maps are generated and reflects the depth of the network. More filters allow the network to capture a broader range of features from the gesture data. However, increasing the number of filters also increases the computational cost and memory requirements. Typically, the number of filters increases with the depth of the network to capture more complex features as the data moves through successive layers.

- **Activation Function:** The activation function introduces non-linearity to the neural network, enabling it to learn complex, non-linear relationships in the data. For gesture recognition in the Magic Wand Project, popular activation functions include ReLU (Rectified Linear Unit) and Sigmoid. ReLU is often preferred in convolutional layers because it helps prevent vanishing gradients and allows for faster training. Sigmoid functions may be used in output layers when dealing with classification tasks. The choice of activation function should align with the problem's characteristics and the nature of the input data.

8.6. Requirements

- **Input Format:** To ensure that the CNN is capable of processing the sensor data generated by the magic wand, the input format should be carefully tailored to the specific characteristics of the data. The sensor data might include accelerometer, gyroscope, or other motion-related readings, which need to be preprocessed to align with the input format of the CNN. Since the Arduino Nano has limited input capabilities, efficient data handling is crucial, potentially requiring data compression or dimensionality reduction techniques to fit within the constraints of the device [WS20a].

- **Real-time Processing:** Real-time processing is essential for ensuring that gestures are recognized and responded to without significant delay. The CNN architecture should be optimized for fast processing, focusing on reducing the latency between gesture input and output action. This can involve simplifying the network architecture, utilizing efficient convolutional operations, and optimizing the network layers to process data quickly. Considering the limited processing capabilities of the Arduino Nano, the model should also prioritize computationally efficient operations [MO19].

- **Energy Efficiency:** Energy consumption is a critical factor, especially in battery-powered or low-power devices like the Arduino Nano. The CNN should be designed to minimize power usage during gesture recognition while maintaining the accuracy and speed of recognition. Techniques such as pruning, quantization, and using lightweight architectures like MobileNets or EfficientNet can help reduce the energy consumption of the CNN without compromising its performance. This ensures that the device operates effectively within the power limits of the hardware.

- **Output Interface:** The output format generated by the CNN needs to be compatible with the capabilities of the magic wand and the Arduino Nano. It should provide outputs in a format that can be easily translated into actionable commands for controlling devices or triggering specific actions. Since the Arduino Nano has limited output options, it is important to design a streamlined output that can be directly mapped to the corresponding actions without requiring excessive processing or additional interfaces.

- **Noise Robustness:** In real-world applications, sensor readings can be subject to noise due to environmental factors, motion artifacts, or hardware imperfections.

To address this, the CNN should be robust to noisy inputs, implementing preprocessing steps such as data smoothing, noise filtering, or sensor fusion. Additionally, the network architecture could incorporate noise-robust techniques such as dropout, batch normalization, or robust loss functions to improve the model's performance in variable conditions [MO19].

- **Inference Speed:** To provide a smooth and responsive user experience, the CNN should be optimized for fast inference. This requires minimizing the computation time required for the model to classify input gestures. Techniques such as reducing the number of layers, simplifying convolutions, and using low-cost activation functions can help speed up inference. Given the limited processing power of the Arduino Nano, it's essential to minimize the network's computational load while ensuring that gesture recognition remains swift and accurate.

- **Training Considerations:** Training the CNN effectively requires a manageable amount of gesture data, considering the limited data collection capabilities of the magic wand and Arduino Nano. Data augmentation techniques, such as rotation, scaling, and noise injection, should be utilized to artificially expand the training dataset, improving the model's ability to generalize. Additionally, transfer learning or pre-trained models can be employed to reduce the amount of required data while achieving good performance in recognizing gestures.

- **Postprocessing Techniques:** After the CNN processes the gesture data, postprocessing methods should be implemented to refine the model's predictions. These techniques can include consensus-based recognition, which aggregates predictions over time to improve accuracy, and temporal filtering, which smooths predictions to account for motion dynamics. Such methods can help eliminate noise and ensure that the recognized gesture is accurate and consistent in real-world scenarios.

- **Compatibility:** The final CNN model must be compatible with the hardware specifications of the Arduino Nano, both in terms of memory capacity and computational power. The model's architecture and parameters should be carefully chosen to ensure that the network can be deployed on the device without exceeding its memory or processing limits. Lightweight CNN architectures or specialized frameworks such as TensorFlow Lite can help in achieving this goal by offering optimized versions of the model that are tailored for low-resource devices.

8.7. Input

Understanding the complexities of input data is essential for the effective construction and training of Convolutional Neural Networks (CNNs). The structure of input data plays a pivotal role, as it is represented as a multi-dimensional array, also referred to as a tensor [Xu+22]. This tensor forms the basis for feature extraction and hierarchical learning within CNNs, which is crucial for tasks like image classification and gesture recognition in projects such as the magic wand.

8.7.1. Input Data Specifications

Dimensions for Image Data

The input data in a Convolutional Neural Network (CNN) is structured as a multi-dimensional array, often called a tensor. This tensor is key for feeding raw input into the network and for enabling subsequent feature extraction. In the case of image data, this

tensor has dimensions that correspond to height, width, and color channels. For example, in the RGB color model, the tensor is typically represented as $(height, width, 3)$, where 3 refers to the three color channels: Red, Green, and Blue. The size of the image, such as 100×100 pixels, results in a tensor of shape $100 \times 100 \times 3$, where each element in the array corresponds to the intensity of a specific color channel at a given pixel.
Grayscale images differ in that they contain only one channel, representing the intensity of the pixel in terms of brightness. In an 8-bit grayscale image, the pixel intensity values range from 0 (black) to 255 (white). This results in a two-dimensional matrix representing the image, where each element signifies the brightness of a pixel. Grayscale images are often normalized to a range between 0 and 1, improving computational stability during model training.
The structure of the input tensor determines how the neural network processes and interprets the raw data, enabling it to recognize patterns such as edges, shapes, and other visual features.

Input Shape Specifications

In the case of image data, the input tensor has a shape denoted as $(height, width, channels)$, where:

- **Height:** The vertical dimension of the image.
- **Width:** The horizontal dimension of the image.
- **Channels:** The number of color channels (e.g., 3 for RGB images).

For example, a typical RGB image with a resolution of 100×100 pixels would have the input shape $(100, 100, 3)$. In the case of grayscale images, the tensor shape would be $(height, width, 1)$, where the single channel represents the intensity of each pixel.
In the code provided in Section 8.10, the actual dimensions for the input images will be substituted for the placeholders ‘height‘, ‘width‘, and ‘channels‘. The actual dimensions of the input data will depend on the specific problem being solved and the format of the input data.

8.7.2. Input Data for Magic Wand

In the context of the Magic Wand Project, the input data consists of time-series accelerometer readings, which capture the motion of the device during different gestures. The accelerometer data from each axis (X, Y, Z) is typically represented as a series of values over time. The changes in these values over time help identify specific gestures.
Adjacent accelerometer readings provide insights into the device’s motion. For example, if the acceleration on one axis rapidly changes from zero to positive and then returns to zero, it indicates that the device has started moving in that direction. These movements form the building blocks for recognizing different gestures.
Figure 8.6 provides an example of accelerometer values captured along one axis of the device. This data is used to detect and analyze the motion associated with each gesture.
Each gesture is typically composed of a sequence of motions, with specific patterns in the accelerometer readings. For example, a "wing" gesture involves alternating upward and downward movements, along with changes in the horizontal direction. Figure 8.7 shows an example of accelerometer data during a "wing" gesture, measured in milli-Gs. The accelerometer data consists of changes along the X, Y, and Z axes. By analyzing the relationship between these axes, a CNN can learn to identify specific gestures. For instance, the Z-axis acceleration might indicate up-and-down motion, while the X-axis

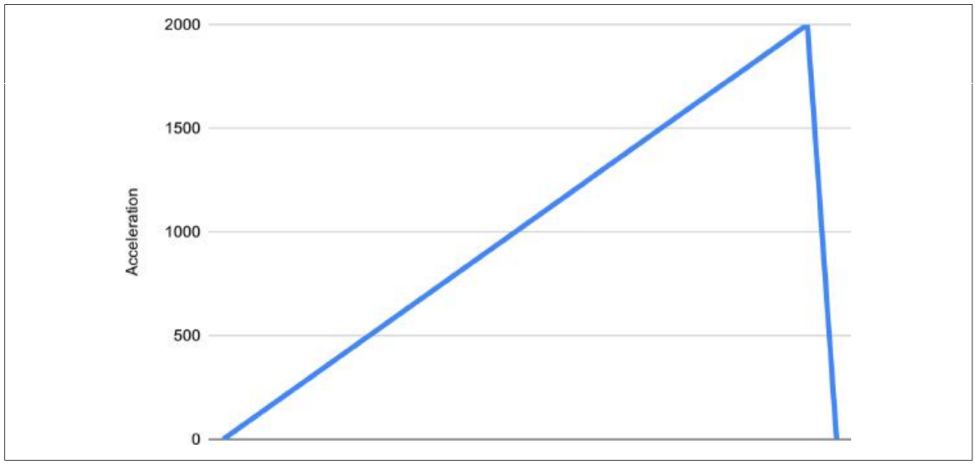


Figure 8.6.: **Accelerometer values for a single axis of a device being moved** [WS20a]

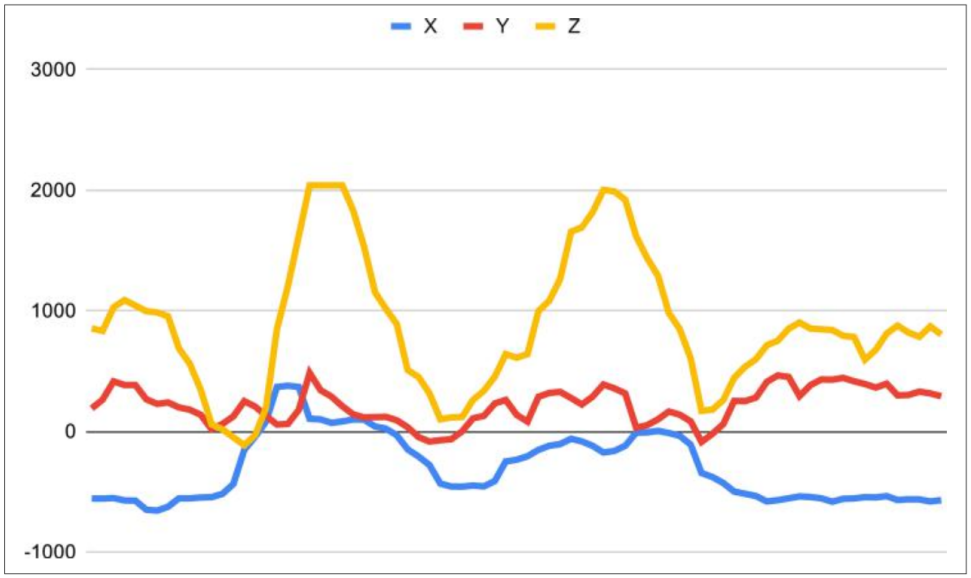


Figure 8.7.: **Accelerometer values during the “wing” gesture** [WS20a]

acceleration can represent horizontal movement. The combination of these patterns helps identify the gesture being performed.

To achieve this, the CNN employs multiple layers of filters. Each filter learns to detect specific features within the data. For example, a filter in the first layer of the network might learn to detect upward acceleration, while a second layer might combine these detections to identify more complex features, such as the "wing" shape of the gesture. As the data progresses through the layers, simple patterns are combined into more complex structures, enabling the network to recognize gestures with high accuracy.

This process allows the CNN to learn from the noisy and raw input data, gradually transforming it into a high-level symbolic representation. At the final layers of the network, the symbolic representation is analyzed to predict the specific gesture being performed.

8.8. Conclusion

By using time-series accelerometer data as input, combined with the power of Convolutional Neural Networks, the Magic Wand Project can achieve accurate gesture recognition. The ability of CNNs to learn hierarchical feature representations from multidimensional input data makes them particularly effective for tasks involving motion and gesture analysis.

8.9. Output

The layer is configured with a "softmax" activation function 8.9.2, which results in the layer's output being a set of probabilities that sum to 1. This output is what we see in the model's output tensor 8.8 8.9 8.10.
This type of model architecture—a combination of convolutional and fully connected layers—is very useful in classifying time-series sensor data like the measurements we obtain from our accelerometer. The model learns to identify the high-level features that represent the "fingerprint" of a particular class of input. It's small, runs fast, and doesn't take long to train. This architecture will be a valuable tool in your belt as an embedded machine learning engineer [WS20a].

Figure 8.8.: **Wing Gesture**

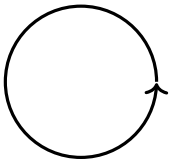


Figure 8.9.: **Ring Gesture**

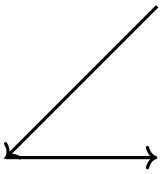


Figure 8.10.: **Slope Gesture**

8.9.1. Regression

In a regression task, the CNN output is a continuous numerical value. The network is designed to predict a specific numerical outcome, and the final layer often uses an activation function, such as linear, that allows for unbounded output. The training process involves minimizing the difference between the predicted value and the actual target value, as determined by the chosen loss function [Xu+22].

8.9.2. Output of the Model for Magic Wand

The CNN functions as a classifier, producing class probabilities as its output. The final outcome is determined by the softmax layer 8.9.2, resulting in a set of numerical

values corresponding to different gesture categories. For example, if the magic wand project involves recognizing gestures like "swirl," "tap," "circle," and "wave," the model might output probabilities such as `[0.05, 0.70, 0.10, 0.15]`.
Each number in the output array represents the model's confidence score for a specific gesture category, and the category with the highest score is considered the model's prediction for the given input gesture. In the provided example, the model predicts that the gesture category "tap" is the most likely result with a confidence score of 0.70.
To enhance the reliability of gesture recognition, postprocessing techniques can be applied. Methods like score averaging or consensus-based recognition are commonly used. Averaging scores over multiple runs contributes to a more stable and consistent output, improving the model's ability to adapt to variations in gesture patterns and environmental conditions [WS20a].
Upon successful recognition of a gesture, the magic wand's command responder can utilize the device's output capabilities, triggering actions such as changing LED colors, producing sounds, or initiating specific functions. The two-dimensional structure of the model's output, aligning with the first dimension as a wrapper and the second holding probabilities for each gesture class, is designed to suit the requirements of the embedded hardware implementation on Arduino Nano 33 BLE Sense [**Alushi:2024**].

The Softmax Function

The softmax function is commonly used in Convolutional Neural Networks (CNNs) and other machine learning models, particularly in the output layer, to convert a vector of raw scores or logits into probabilities. It essentially normalizes the input values into a probability distribution that sums to 1 [**Sewak:2018**].
In the context of CNNs, the softmax function is often applied to the output layer when the network is used for classification tasks. Here's the formula for the softmax function [**Wings:2023**]:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{8.5}$$

Where x_i is the raw score or logit for class i, K is the total number of classes, and e is the base of the natural logarithm (Euler's number). An example of applying the Softmax function is shown in Figure 8.11.

Figure 8.11.: Example of applying the softmax function [**Sewak:2018**]

8.10. Python Example Code

The identification of regional patterns within an image is facilitated by the convolutional layer. Following the convolutional layer, the max pooling layer is employed to diminish dimensionality. This section illustrates image classification by using a code provided by Sewak et al [**Sewak:2018**].
It is crucial to initially standardize all images to a uniform size. The initial convolution layer necessitates an additional parameter, `input.shape()`. The focus here is on training a Convolutional Neural Network (CNN) for image classification using the CIFAR-10 database. CIFAR-10 comprises 60,000 color images, each of size 32×32.

These images are categorized into 10 classes, with 6,000 images per category, namely airplane, automobile, bird, cat, dog, deer, frog, horse, ship, and truck.

8.10.1. Imports

In the Listing 8.1 are the necessary libraries and modules required for working with neural networks, image data, and visualization. The versions are shown in the Table 8.2.

- Python version: 3.9.

Table 8.2.: Versions of Libraries

Library	Version
Numpy	1.23.5
Matplotlib	3.7.1
Keras	2.12.0

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import cifar10
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator
```

Listing 8.1.: Importing necessary libraries and modules.

8.10.2. Load CIFAR-10 Dataset

In Listing 8.2 the CIFAR-10 dataset is loaded.CIFAR-10 is a dataset of 50,000 32×32 color training images and 10,000 test images.

```
(xTrain, yTrain), (xTest, yTest) = cifar10.load_data()
```

Listing 8.2.: Loading and preparing the CIFAR-10 dataset.

Data Preprocessing

In the Listing 8.3 the image data is normalized and one-hot encoding is performed on the labels. The training set is split into training and validation sets.

8.10.3. Augmented Image Generator

In the Listing 8.4 image data generators for training and validation is created and configured. These generators will perform data augmentation, such as shifting and flipping, to increase the diversity of the training set.

8.10.4. Plot the First Nine Images of Cifar-10

Listing 8.5 loads the cifar-10 dataset and plots the first nine images as shown in Figure 8.12.

```
# rescale [0,255] -> [0,1]
xTrain = xTrain.astype('float32') / 255

# one-hot encode the labels
numClasses = len(np.unique(yTrain))
yTrain = np_utils.to_categorical(yTrain, numClasses)
yTest = np_utils.to_categorical(yTest, numClasses)

(xTrain, xValid) = xTrain[5000:], xTrain[:5000]
(yTrain, yValid) = yTrain[5000:], yTrain[:5000]
```

Listing 8.3.: Preprocessing data: normalization, one-hot encoding, and splitting into training and validation sets

```
datagenTrain = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

datagenValid = ImageDataGenerator(
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

# Fitting augmented image generator on data
datagenTrain.fit(xTrain)
datagenValid.fit(xValid)
```

Listing 8.4.: Configuring image data generators for augmentation and fitting them on training and validation data

```
plt.figure(figsize=(10, 10))
for i in range(10,18):
    plt.subplot(330 + 1 + i)
    plt.imshow(xTrain[i])
plt.tight_layout()  # Adjust layout for better visualization
plt.savefig('CIFAR10NineImages.png')  # Save the figure
plt.show()
```

Listing 8.5.: Loading and visualizing the first nine images from the CIFAR-10 dataset

8.10.5. CNN Model Definition

A Convolutional Neural Network (CNN) model is defined in Listing 8.6 using Keras with convolutional layers, max pooling, dropout for regularization, and dense layers.

```
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(10, activation='softmax'))

model.summary()
```

Listing 8.6.: Defining a Convolutional Neural Network (CNN) model using Keras

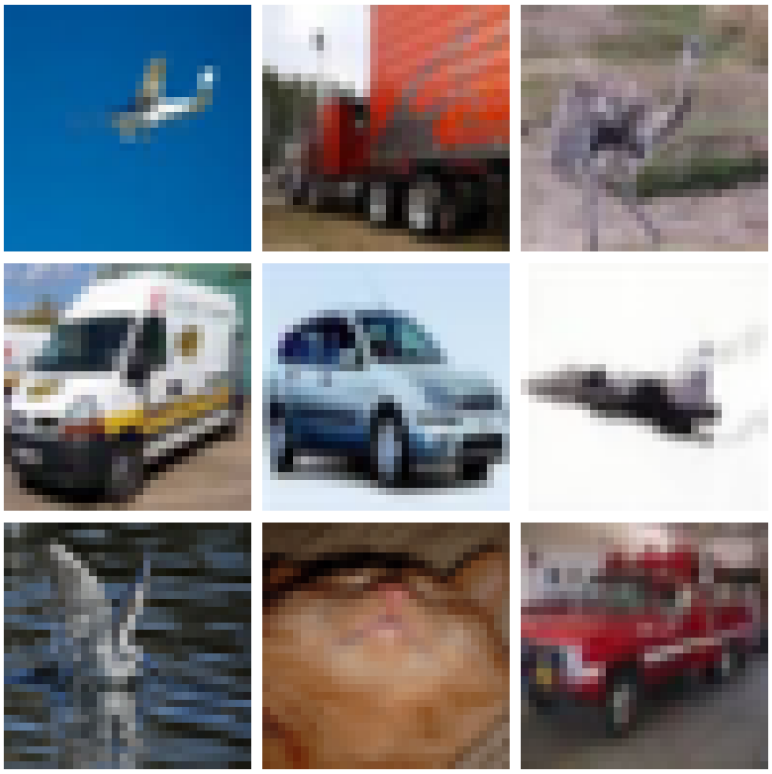


Figure 8.12.: First nine images from the CIFAR-10 dataset.

8.10.6. Compile the Model

In Listing 8.7 the model is compiled with the specified loss function, optimizer, and evaluation metric.

```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['
    ↪ accuracy'])
```

Listing 8.7.: Compiling the CNN model with specified loss function, optimizer, and metric

8.10.7. Train the Model with Augmented Data

In Listing 8.8 the model is trained using the augmented data generators. This involves calling the `fit_generator` function instead of `fit` and providing the data generators for training and validation sets.

8.10.8. Plotting the Loss and Accuracy Curves

In Listing 8.9 the accuracy and loss curves are plotted. The results are shown in the Figure 8.13. The observed trends in the training and validation metrics, with a downward trend in both training and validation loss and an upward trend in accuracy, indicate positive learning and generalization behavior of the neural network. The somewhat unconventional scenario of validation loss being below training loss and validation accuracy exceeding training accuracy could be influenced by effective data augmentation, contributing to the model's robustness. These trends collectively suggest

```
hist = model.fit_generator(
    datagenTrain.flow(xTrain, yTrain, batch_size=32),
    steps_per_epoch=len(xTrain) // 32,
    epochs=10,
    validation_data=datagenValid.flow(xValid, yValid, batch_size=32),
    validation_steps=len(xValid) // 32,
    callbacks=[checkpointer],
    verbose=2,
    shuffle=True
)
```

Listing 8.8.: Training the CNN model with augmented data using data generators

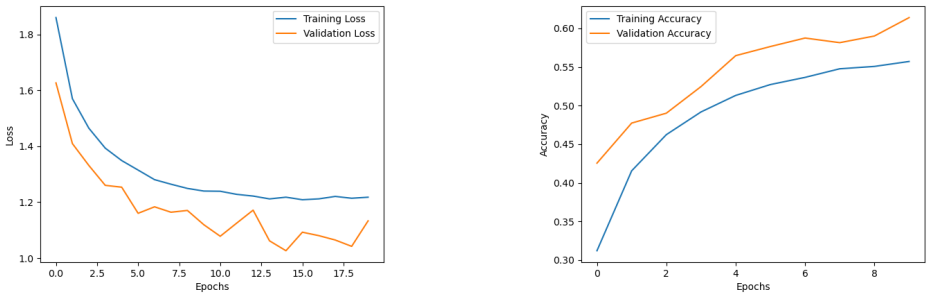
that the model is not overfitting the training data and is likely to generalize well to unseen data, highlighting successful training and potential for further improvement.

```
trainingLoss = hist.history['loss']
trainingAccuracy = hist.history['accuracy']
validationLoss = hist.history['val_loss']
validationAccuracy = hist.history['val_accuracy']

# Plotting and saving the loss curve
plt.plot(trainingLoss, label='Training Loss')
plt.plot(validationLoss, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('lossPlot.png')
plt.show()

# Plotting and saving the accuracy curve
plt.plot(trainingAccuracy, label='Training Accuracy')
plt.plot(validationAccuracy, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.savefig('accuracyPlot.png')
plt.show()
```

Listing 8.9.: Plotting the loss and accuracy curves



(a) (b)

Figure 8.13.: (**??**) Training and validation loss trends over epochs. (**??**) Training and validation accuracy trends over epochs.

Part IV.

Development

9. Development Environment

9.1. Arduino IDE Description

It is an open source official Arduino software which used for editing, uploading and compiling codes in to the Arduino module. It is a cross-platform software which is available for Operating Systems like Windows, Linux, macOS. It runs on Java platform and supports a range of Arduino modules. It supports C and C++ languages. The microcontrollers present on the Arduino boards are programmed which accepts the information in the form of code. The program written in the IDE is called a sketch which will generate a Hex file which is then transferred and uploaded in the controller. The IDE environment is made up of two parts: an editor and a compiler. The editor is used to write the required code, while the compiler is used to compile and upload the code to the Arduino Module.[**fezari:2018**] The Menu bar has options such as File in which there are many options including Opening a new file or existing, Examples-in which we can find sketches for different applications like Blink, Fade etc. There is an error console at the bottom of the screen for displaying errors.
The 6 buttons are present on top of the screen are as follows:

Figure 9.1.: **Menu button.**

- The check mark is used to verify your code. Click this once you have written your code.
- The arrow uploads your code to the Arduino to run.
- The dotted paper will create a new file.
- The upward arrow is used to open an existing Arduino project.
- The downward arrow is used to save the current file.
- The far right button is a serial monitor, which is useful for sending data from the Arduino to the PC for debugging purposes.

9.1.1. Installation

To install the Arduino IDE, we need to download the latest version from the Arduino webpage `https://www.arduino.cc/en/software`. We can select the version based on the operating system we are using. Here we are installing Arduino 1.8..15 for a Windows 10 operating system. The set up file name is arduino-1.8.15-windows.exe and the size of it is 1,17,470 KB. we can specify the path according to our needs. Here the path is set as `C:/Program Files (x86)/Arduino`.

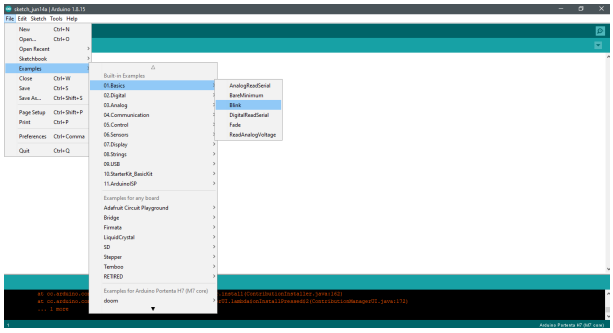


Figure 9.2.: **Menu bar options.**

After the download is done, open the setup file and proceed to install. Select all the components in the dialog box and click Next.

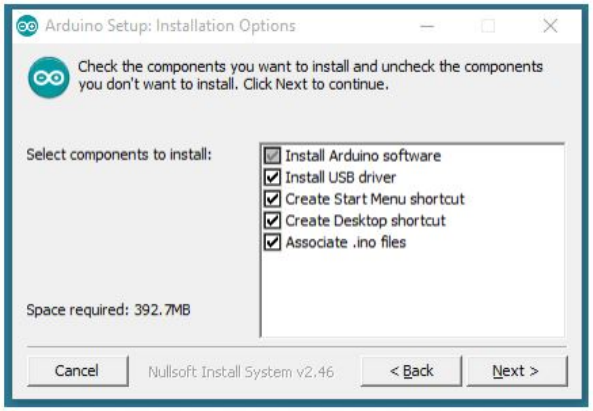


Figure 9.3.: **Arduino Setup Installation options.**

Select the destination folder and click Install

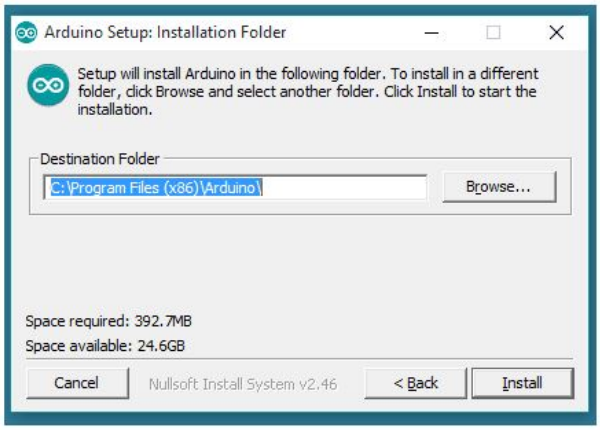


Figure 9.4.: **Arduino Setup Installation Folder.**

Once the installation is done, open the Arduino IDE and a default sketch appears on the screen as shows.

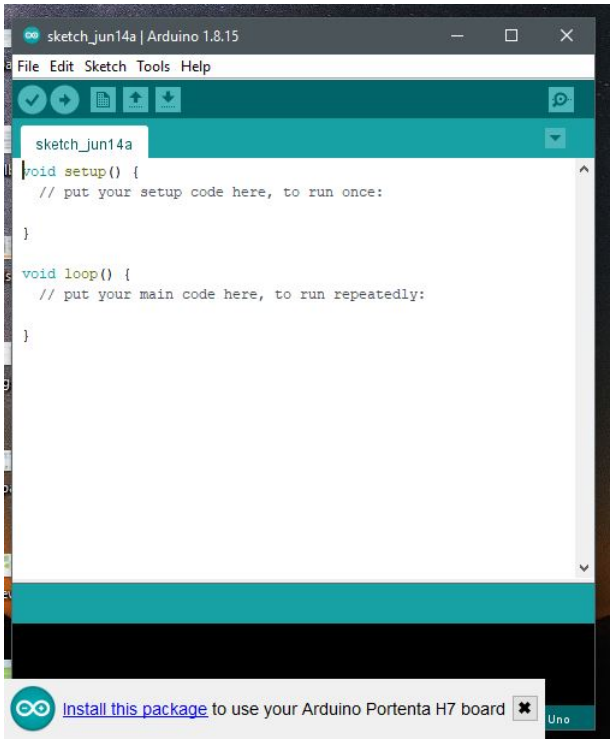


Figure 9.5.: **Arduino Sketch.**

It can be seen from the above figure that the basic arduino sketch has two parts. The first part is the function `void setup()` which returns void and we do the intiliaztion such as the output LED color, specifying the core etc. The second part is the function `void loop()` where we define functions which are to be performed through out the loop. These codes are placed between paranthesis {} and each function has a return type, here it has void return type.

9.1.2. Arduino IDE on PC

Installation

Arduino Nano 33 BLE Sense uses the Arduino software integrated development environment (IDE) for programming, which is the most widely used and common (IDE) for all arduino boards that can be run online and offline. This is a open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. There are various version of software which is supported for each operating system (OS) e.g: mac, linux, and windows. Arduino community also provide us to start coding online and save our sketches in the cloud, this online arduino editor is most up-to-date version of the IDE includes all libraries and also supports new Arduino boards. For getting access to these software packages go to the following link `https://www.arduino.cc/en/software` and get more up to date inforamtion, because every single day there are some updates occurs which is available on the link mention above. These software can be used with any Arduino board, the most recent offline arduino IDE 1.8.15 can be seen in Figure,9.6. it is also supportive for all operating systems.

Figure 9.6.: **Arduino Creat Agent Installation.**

9.1.3. Configuration

Configuration for the Arduino Nano 33 BLE Sense

To program the Arduino Nano 33 BLE Sense in offline state, we need to install one of the latest arduino IDE on our desktop. After installation, for getting access to the Arduino nano 33 ble sense board, we need to make configuration in our IDE. By opening the IDE, go to tool which can be seen on the uper left corner in IDE, in the tool there is an option for managed board. At this point we need to write our board name in the search which is Arduino Nano 33 BLE Sense as shown in figure,9.7.Select the Arduino Mbed OS Boards and install it. The Mbed OS nano board supports also other nano family boards including Arduino nano 33 ble sense, after installing simply connect the Arduino Nano 33 BLE Sense to the computer via USB cable.

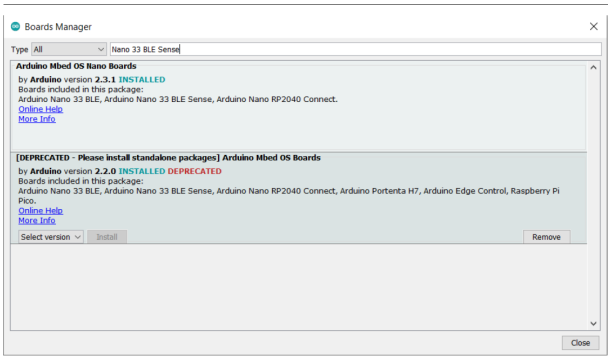


Figure 9.7.: **Arduino Mbed OS Nano Boards Installation.**

9.1.4. Setup

There are set of examples which are build in Arduino (IDE) for the testing purpose, for checking all the configuration and setting up the board we can open one of the basic LED blink example first as shown in the figure. 9.8.

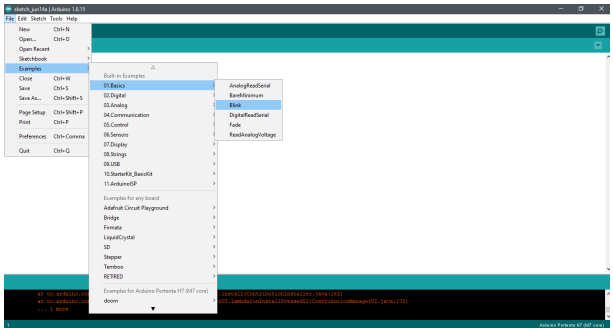


Figure 9.8.: **LED-Example Test.**

This LED-blink example support all the arduino boards, for the checking purposes just need to run this basic example on any arduino embed board and it will blink the LED on our Arduino board after pre-set miliseconds. In the same example folder, there are also number of build in usefull example written in Arduino IDE for embedded boards. These examples are very usefull for getting the basic knowledge about the board and programming.

9.1.5. constraints

There are some pre-requisite steps need to follow either we need to run the build in example or run by our own written program. By operating the Arduino board with Laptop with the help of USB connection, need to open the Arduino IDE on desktop, it appears a blank arduino environment page just a Void setup and void loop written on it. At this step we need to go to the tool-Arduino board and select the connected board which is Arduino Nano 33 Ble Sense as shown in the figure 9.9

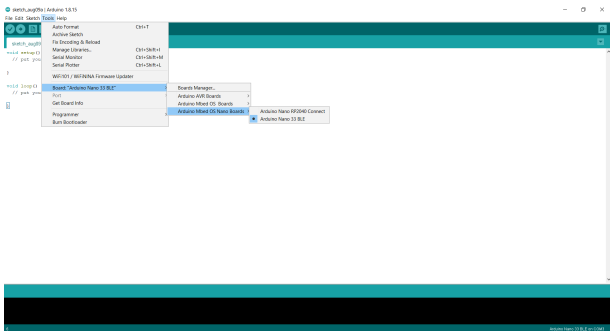


Figure 9.9.: **Select the Connected board -here Arduino Nano 33 BLE Sense.**

Select the Appropriate Port

By selecting the Arduino nano 33 BLE sense board, next we need to check the connected port. For doing this, we need to set our arduino borad in Boot setup by clicking the white reset button on arduino as show in figure 9.10

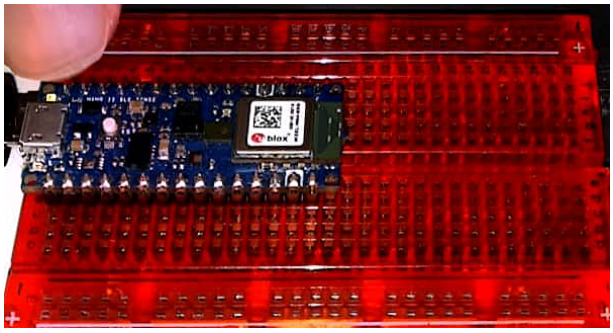


Figure 9.10.: **Arduino Nano 33 BLE Sense Reset Button.**

By clicking the white reset button, the arduino borad will be in boot setup and make sure to check the orange LED glows as shown in the figure9.11

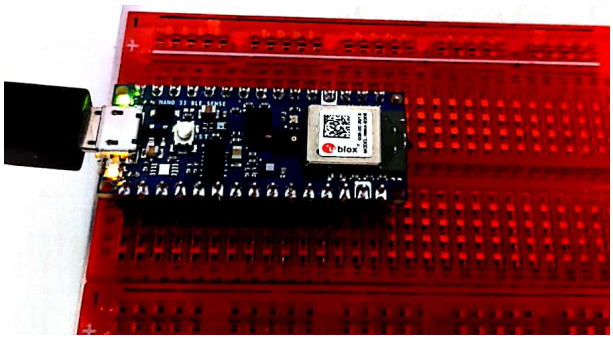


Figure 9.11.: **Arduino Nano 33 BLE Sense Orange LED Glow.**

After successfully applying the above mention step, next we need to select the connected port before upload the program. For this, go to tool select arduino port and make sure to check it available port for uploading the program as shown in figure9.12

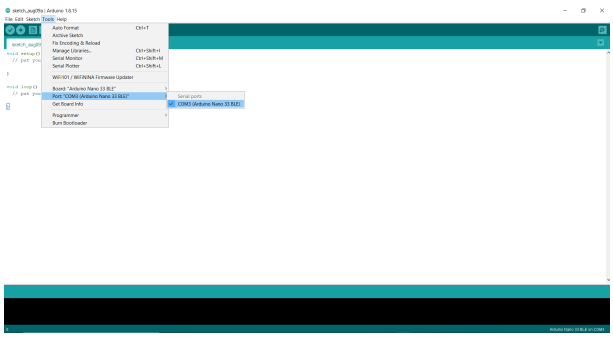


Figure 9.12.: **Select Available Port for Uploading Arduino Sketch.**

Upload Code in Arduino Board

By making sure to select the appropriate port, it's time to upload the Arduino program. There are five icons (verify, upload, new, open, save) below the file section, before uploading the program the best practice is to verify the program first, it show us if there are any error or warning in the program exist or not. By successfully verifying the program we can safely upload the program by click the upload button in the top below the file section as shown in figure.9.13

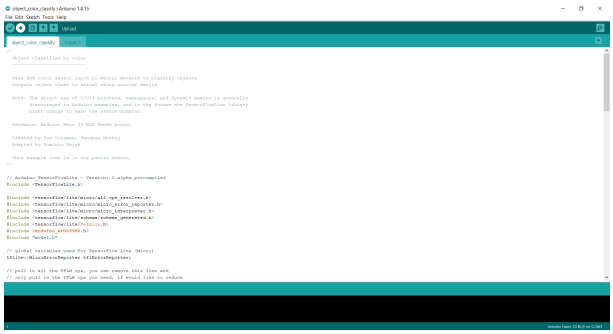


Figure 9.13.: **Upload the Program in Arduino board.**

After uploading, the code will compile and if there is any issue in our program it will pop up in the bottom black window as well. After successfully uploading and compiling the code in Arduino board, it also require to change the port again as we did it previously. Go to tool select arduino port and make sure to check the port again as shown in figure9.14 by getting output in the serial monitor.

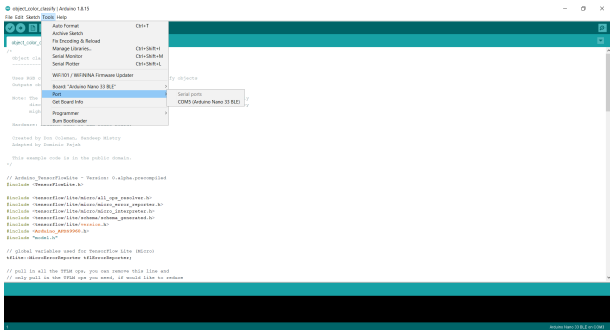


Figure 9.14.: **Setting the Port.**

9.1.6. Conclusions

Output Window (Serial Monitor)

Serial Monitor is the another window on the Arduino IDE, which shows the Input/Output of our program and results appear on it as per the required output. For getting access to Serial monitor, we need to go extreme right in the Arduino IDE, the small circle pop up when we reach it is the serial monitor as show in the figure.9.15

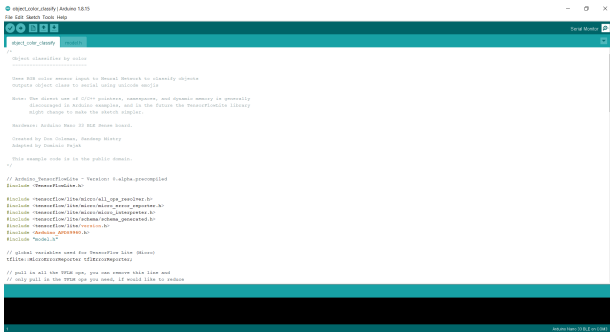


Figure 9.15.: **Serial Monitor Icon.**

The Final results, all the variables, input, sensor values are shown in the serial monitor the (Output Window) as shown in the figure9.16 by clicking the serial monitor button.

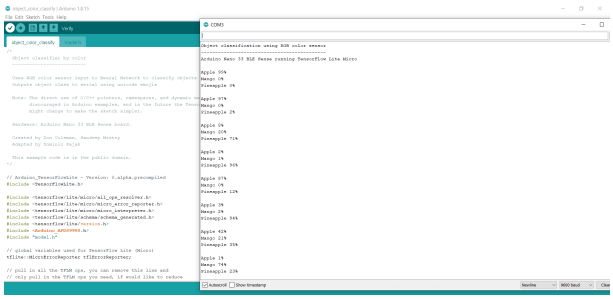


Figure 9.16.: **Output Window.**

9.2. TensorFlow

TensorFlow (tf) is an end-to-end, open-source platform popularly used for the quick implementation of machine learning algorithms. A rich ecosystem of tools, libraries, and community resources has made it extremely popular among machine learning researchers and practitioners to develop and deploy various machine learning algorithms with greater efficiency and flexibility. TensorFlow has become popular in recent times for the quick development of complex deep neural network architectures for both experimentation and developing production-ready software.[**Kha:2021**]
TensorFlow was originally developed by Google within their Machine Intelligence Research Organization to conduct various machine learning and neural networks related research. The initial version was released in 2015 under the Apache License 2.0. TensorFlow 2.0, the newest stable version, was released in 2019. TensorFlow is highly flexible. It supports a wide variety of programming languages, including Python, C++, and Java. Moreover, it can run on CPU, GPU, and TPU for a faster processing of large machine learning applications. TensorFlow is available in Linux, macOS X, and Windows platforms. It also supports TensorFlow Lite, a highly optimized lighter version of the original TensorFlow that is available on mobile computing platforms such as Android, iOS-based smartphones, and Linux-based single board computers like Raspberry Pi. TensorFlow Lite models can be further optimized using a few standard APIs to run on microcontroller units. Hence, TensorFlow is heavily used in TinyML applications. Visit the official TensorFlow website for more details. To summarize, some of the key features of TensorFlow are as follows:

- It is open-source
- Efficiently works with multi-dimensional data
- Provides a higher level of abstraction, which reduces the code length for the developer
- Supports various platforms and architectures
- Highly scalable and provides greater flexibility for quick prototyping

9.2.1. Installation

TensorFlow, along with all its dependencies, is already installed in Colab. So, you just need to import the libraries to write codes without any package installation. TensorFlow can be imported by typing the following command in the code cell.

Listing 9.1: Import TensorFlow and Check Version

```
# Import TensorFlow
import tensorflow as tf

# Check TensorFlow version
tf._version_
```

Once TensorFlow is imported, you can check the version by typing the command **tf.__version__**. At the time of writing this book, the TensorFlow version in Colab is 2.12.0, which may change with time. TensorFlow 2, or TF 2, is the newest version which is significantly different compared to the previous version TF 1.x. In this book, we will use TF2 for all our programming. However, TF 2 provides a backward compatibility module to use TF 1.x. The eager execution mode of TF 2 makes it easier to create a machine learning architecture with lesser lines of code. [Goo24]

9.2.2. constraints

Once we are done with loading and pre-processing of the data, we can define neural network architecture.[**Flow:2021**]
A Keras model has the following four stags:

1. Defining the model: We define a Sequential model and add the necessary layers.
2. Compiling the model: We configure the model for training by defining the loss function to be minimized, the optimizer that minimizes the loss function, and a performance metric to internally evaluate the performance.
3. Fitting the model: Here, we fit the model on the actual training data for a given number of epochs to update the training parameters. We will get the model at the end of training.
4. Evaluation: Once you have the model, you can evaluate on unseen test data

9.2.3. Conclusions

One of the biggest benefits of using TensorFlow is the level of abstraction. It takes care of the major details of most of the underlying algorithms in a machine learning or a deep learning application, for example, backpropagation. Hence, writing programs in TensorFlow is super easy as you mostly need to focus on your application logic. TensorFlow applications can run on almost any target environment, such as local desktops, remote servers running Windows, Linux, macOS X, smartphone devices running Android and iOS, or Linux-based single board computers. TensorFlow contains additional libraries to convert a machine learning model into C++ equivalent libraries to run on selected microcontrollers. Hence, you can use TensorFlow to create your TinyML applications. Throughout this book, we will primarily use Keras to implement the machine learning models. Keras is a high-level set of Python Application Programming Interface (API) in TensorFlow, which is popularly used in the rapid prototyping of various neural network models. Fortunately, both TensorFlow and Keras are readily available in Colab. So, you can easily start writing your program in Colab without installing any extra libraries. In this chapter, we will primarily focus on creating end-to-end neural network models using TensorFlow. The later chapters will focus on optimizing the neural networks to create deployable TinyML applications.

9.3. TensorFlow Lite

To meet these lower size requirements for mobile platforms, in 2017 Google started a companion project to mainline TensorFlow called TensorFlow Lite. This library

is aimed at running neural network models efficiently and easily on mobile devices. To reduce the size and complexity of the framework, it drops features that are less cons mon on these platforms. For example, it doesn't support training, just running infer ence on models that were previously trained on a cloud platform. It also doesn't support the full range of data types (such as double) available in mainline Tensor Flow Additionally, some less-used operations aren't present, like tf.depth_to_space You can find the latest compatibility information on the TensorFlow website.
In return for these trade-offs, TensorFlow Lite can fit within just a few hundred kilobytes, making it much easier to fit into a size-constrained application. It also has highly optimized libraries for Arm Cortex-A-series CPUs, along with support for Android's Neural Network API for accelerators, and GPUs through OpenGL Another key advantage is that it has good support for 8-bit quantization of networks Because a model might have millions of parameters, the 75% size reduction from 32- bit floats to 8-bit integers alone makes it worthwhile, but there are also specialized code paths that allow inference to run much faster on the smaller data type.[Goo24]

9.3.1. Installation

Listing 9.2: Load and Run TensorFlow Lite Model

```
import tensorflow as tf
import numpy as np

# Load the TensorFlow Lite model.
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

# Get the input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Create some input data.
input_data = np.array([[1.0, 2.0, 3.0]])

# Set the input tensor.
interpreter.set_tensor(input_details[0]["index"], input_data)

# Perform inference.
interpreter.invoke()

# Get the output tensor.
output_data = interpreter.get_tensor(output_details[0]["index"])

# Print the output.
print(output_data)
```

9.3.2. constraints

TensorFlow Lite for Microcontrollers is designed for the specific constraints of microcontroller development.[Goo24] If you are working on more powerful devices (for example, an embedded Linux device like the Raspberry Pi), the standard TensorFlow Lite framework might be easier to integrate The following limitations should be considered:

- Support for a limited subset of TensorFlow operations.
- Support for a limited set of devices.
- Low-level C++ API requiring manual memory management.
- On device training is not supported

9.3.3. Conclusions

TensorFlow Lite is a lightweight version of TensorFlow, designed specifically for mobile and embedded devices. In the TinyML magic wand project using Arduino Nano, TensorFlow Lite is used to run a pre-trained machine learning model on the Arduino Nano board. The magic wand project involves using a gesture recognition model to detect different hand movements and control a small toy wand. The Arduino Nano board is used to collect sensor data from an accelerometer and gyro sensor, which is then fed into the machine learning model running on the board. The model uses TensorFlow Lite to make predictions based on the sensor data and determine what hand gesture is being made. Based on the prediction, the wand can be controlled to perform different actions. The use of TensorFlow Lite in this project allows the machine learning model to run efficiently on the limited resources of the Arduino Nano board. By using a lightweight model and optimizing the code for the microcontroller architecture, it is possible to perform gesture recognition on a low-power, low-cost device like the Arduino Nano. This makes the project more accessible and affordable for hobbyists and students who want to experiment with TinyML applications.[**Magicwand:2022**]

9.4. TensorFlow Lite for Microcontrollers

9.4.1. Introduction

TensorFlow Lite has been widely adopted by mobile developers, but its engineering trade-offs didn't meet the requirements of all platforms. The team noticed that there were a lot of Google and external products that could benefit from machine learning being build on embedded platforms, on which the existing TensorFlow Lite library wouldn't fit. Again, the biggest constraint was binary size. For these environments even a few hundred kilobytes was too large; they needed something that would fit within 20 KB or less. A lot of the dependencies that mobile developers take for granted. like the C Standard Library, weren't present either, so no code that relied on these libraries could be used. A lot of the requirements were very similar, though. Inference was the primary use case, quantized networks were important for performance, and having a code base that was simple enough for developers to explore and modify was a priority.[Goo24]

9.4.2. Work Flow

1. Train a model:
    - Generate a small TensorFlow model that can fit your target device and contains supported operations.
    - Convert to a TensorFlow Lite model using the TensorFlow Lite converter.
    - Convert to a C byte array using standard tools to store it in a read-only program memory on device.
2. Run inference on device using the C++ library and process the results.

9.4.3. Installation

Listing 9.3: Run Inference with TensorFlow Lite Micro

```
#include "model.h"
#include "tensorflow/lite/micro/kernels/micro_ops.h"
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/version.h"
```

```
// Create an instance of the error reporter
static tflite::MicroErrorReporter micro_error_reporter;

// Define the tensor arena
static constexpr int kTensorArenaSize = 2 * 1024;
static uint8_t tensor_arena[kTensorArenaSize];

// Create an instance of the interpreter
static tflite::MicroInterpreter static_interpreter(
model, tflite::MicroOpResolver::GetDefault(), tensor_arena,
kTensorArenaSize, &micro_error_reporter);

// Allocate memory for the input and output tensors
TfLiteTensor* input = static_interpreter.input(0);
TfLiteTensor* output = static_interpreter.output(0);

// Perform inference
static_interpreter.Invoke();

// Print the output
for (int i = 0; i < output->bytes; i++) {
    printf("%f ", output->data.f[i]);
}
```

9.4.4. constraints

Running in embedded environments imposed a lot of constraints on how the code could be written, so it identified some key requirements for the library:

1. **No operating system dependencies**

   A machine learning model is fundamentally a mathematical black box where numbers are fed in, and numbers are returned as the results. Access to the rest of the system shouldn't be necessary to perform these operations, so it's possible to write a machine learning framework without calls to the underlying operating system. Some of the targeted platforms don't have an OS at all, and avoiding any references to files or devices in the basic code made it possible to port to those chips.[WS20b]

2. **No standard Cor C++ library dependencies at linker time**

   This is a bit subtler than the OS requirement, but the team was aiming to deploy on devices that might have only a few tens of kilobytes of memory to store a pro- gram, so the binary size was very important. Even apparently simple functions like sprintf() can easily take up 20 KB by themselves, so the team aimed to avoid anything that had to be pulled in from the library archives that hold the implementations of the C and C++ standard libraries. This was tricky because there's no well-defined boundary between header-only dependencies (like stdint.h, which holds the sizes of data types) and linker-time parts of the standard libraries (such as many string functions or sprintf().

3. **No floating point hardware expected**

   Many embedded platforms don't have support for floating-point arithmetic in hardware, so the code had to avoid any performance-critical uses of floats. This meant focusing on models with 8-bit integer parameters, and using 8-bit arith metic within operations (though for compatibility the framework also supports float ops if they're needed). [**Ard:2021**]

4. **No dynamic memory allocation**

   A lot of applications using microcontrollers need to run continuously for months or years. If the main loop of a program is allocating and deallocating memory using malloc()/new and free()/delete, it's very difficult to guarantee that the

heap won't eventually end up in a fragmented state, causing an allocation failure and a crash. There's also very little memory available on most embedded systems, so upfront planning of this limited resource is more important than on other platforms, and without an OS there might not even be a heap and allocation rou- tines. This means that embedded applications often avoid using dynamic memory allocation entirely. Because the library was designed to be used by those applications, it needed do the same.

In practice the framework asks the calling application to pass in a small, fixed-size arena that the framework can use for temporary allocations (like activation buffers) at initialization time. If the arena is too small, the library will return an error immediately and the client will need to recompile with a larger arena. Otherwise, the calls to perform inference happen with no further memory allocations, so they can be made repeatedly with no risk of heap fragmentation or memory errors.[WS20b]

9.4.5. Conclusions

In the TinyML magic wand project using Arduino Nano, TensorFlow Lite Micro is used to run a machine learning model on the Arduino Nano board to recognize gestures made with the wand. The model is trained to recognize a specific set of gestures (e.g., waving the wand up and down, left and right) and can trigger different actions based on the recognized gesture (e.g., turn on/off lights, play music). Using TensorFlow Lite Micro on the Arduino Nano enables the wand to perform gesture recognition locally on the device, without the need for an internet connection or a more powerful computer. This makes the wand more portable and flexible, and can be used in a wider range of environments and applications. Overall, TensorFlow Lite Micro enables the development of TinyML (Tiny Machine Learning) applications that run on microcontrollers and other embedded devices, making it possible to bring machine learning capabilities to a wide range of IoT and edge devices.[**Pico:2019**]
Our application will be deployed on Arduino Nano 33 BLE Sense, which is a recommended microcontroller board for TinyML applications. The application we are going to implement is fairly simple. The microcontroller will read the linearly varying electrical voltage provided by an external potentiometer as its input, convert the voltage values as per a halfwave sinusoid function using a simple neural network, and controls the brightness of an LED accordingly. Remember, due to its resource constraints, you cannot even train a small neural network on a microcontroller. We will train the neural network in Colab using TensorFlow, The model will be converted into a TFLite model. This part of the project will be implemented in Python. We will then convert the model into an equivalent C/C++ library for microcontrollers. Next, we will write an inference application for Arduino using a programming language that is very similar to C/C++.

9.5. Python

The Arduino IDE is written only in C++ language and is not supported the other programming languages. Some of the Module I used specially for executing the Gesture detection part of this project is only support python language e.g., MediaPipe and OpenCV. MediaPipe and OpenCV module are the most important part for gesture detection, for detecting the landmarks on hand the supported function is written in python. Without the MediaPipe Module, the hand landmarks technique is not possible to implement. At the moment, there is no such module or library who can support directly MediaPipe Module in Arduino IDE and C++, because MediaPipe Module is written in python.[**Alk:2019**]

9.5.1. Installation

Listing 9.4: Load and Run TensorFlow Lite Model

```
import tensorflow as tf
import numpy as np

# Load the TensorFlow Lite model.
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

# Get the input and output tensors.
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Create some input data.
input_data = np.array([[1.0, 2.0, 3.0]])

# Set the input tensor.
interpreter.set_tensor(input_details[0]["index"], input_data)

# Perform inference.
interpreter.invoke()

# Get the output tensor.
output_data = interpreter.get_tensor(output_details[0]["index"])

# Print the output.
print(output_data)
```

9.5.2. constraints

Python, although versatile, has constraints in the context of the magic wand project with Arduino Nano 33 BLE. Real-time operations pose a challenge due to Python's lack of native support, potentially affecting precise timing for gesture recognition. Additionally, Python's interpreted nature may result in slower execution, impacting sensor data processing and model inference speed. Memory usage can be a concern on resource-constrained microcontrollers, like Arduino Nano. While Python is not inherently optimized for the lightweight requirements of such devices, developers can mitigate these constraints by employing efficient coding practices and considering alternatives for critical real-time tasks. [**Pyt:2012**]

9.5.3. Conclusion

The use of Python in the software aspect of the magic wand project enhances flexibility, allowing for efficient data processing, machine learning model training, and seamless communication with the Arduino Nano 33 BLE.[**Alk:2019**]

9.6. PyCharm

PyCharm, a robust Python integrated development environment (IDE), plays a pivotal role in the software development process for the magic wand project with Arduino Nano. Leveraging PyCharm's advanced code editing and navigation features, the development team benefits from an efficient and organized coding experience. The IDE's support for project management, version control integration with Git, and seamless debugging and profiling tools contribute to a streamlined development workflow. PyCharm's capabilities extend to virtual environment support, ensuring effective dependency management for the machine learning aspects of the project. Additionally, PyCharm facilitates integration with continuous integration (CI) systems, enabling automated testing and build processes. Overall, PyCharm proves indispensable in enhancing

code quality, collaboration, and the overall efficiency of the machine learning software development on the Arduino Nano platform.[Jet24]

9.6.1. Setup

To install PyCharm, begin by visiting the official PyCharm website and downloading the appropriate version for your needs—Community (free) or Professional (paid). Run the installer, typically an executable file on Windows, and follow the on-screen instructions. Choose the installation location, edition, and any additional settings. If you opt for the Professional edition, activate a license during the installation or choose to evaluate it for a trial period. Once installed, run PyCharm, configure a Python interpreter, and start coding. It's advisable to consult the official PyCharm documentation for detailed and platform-specific instructions. [**Set:2023**]

9.6.2. constraints

PyCharm, while a versatile and powerful integrated development environment (IDE) for Python, presents certain constraints that users should consider. One notable constraint is its resource intensiveness, potentially causing performance issues on machines with limited RAM or processing power, especially in larger projects. Additionally, the learning curve can be steep for new users due to the extensive feature set. Cost can be a constraint for users requiring advanced features in the professional version. Compatibility issues may arise with specific Python libraries or frameworks, and users should stay updated to avoid such constraints. While PyCharm is specialized for Python, its support for other languages may be limited. Plugin dependencies, heavy initial indexing for large projects, and constraints related to remote development or extremely complex projects further warrant consideration. Awareness of these constraints empowers users to make informed decisions based on their development needs and project characteristics.[**Jet:2023**]

9.6.3. Conclusion

PyCharm serves as an integral component of the development toolkit for the magic wand project with Arduino Nano. Its comprehensive set of features enhances the coding experience, fosters collaboration, and contributes to the overall efficiency of the software development process.[Jet24]

9.7. Github

The Github repository for the "Magic Wand with Arduino Nano 33 BLE" project comprises key components essential for its development and usage. The heart of the project lies in the Arduino code, intelligently crafted to collect sensor data, interact seamlessly with TensorFlow Lite, and execute actions based on machine learning predictions. This codebase is housed within the dedicated arduino code directory, providing a clear organization of the software responsible for the wand's functionality.

9.8. Data Base

9.9. Data Characteristics

1. **Structure:**

   The JSON structure is organized into *strokes*, each containing an *index* and an array of *stroke points* with X-Y coordinates. This structured format is conducive

to representing sequential information, allowing efficient processing and analysis of stroke data.

Listing 9.5: Example of gesture data with sensor readings

```
{
    "gesture_data": [
    {
        "label": "W",
        "sensor_readings": [
        {"acceleration_x": 0.23, "acceleration_y": -0.15, "acceleration_z":
            ↪  0.98, ...},
        {"acceleration_x": 0.21, "acceleration_y": -0.18, "acceleration_z":
            ↪  0.95, ...},
        ...
        ]
    },
    {
        "label": "O",
        "sensor_readings": [
        {"acceleration_x": 0.14, "acceleration_y": 0.22, "acceleration_z":
            ↪ 0.93, ...},
        {"acceleration_x": 0.12, "acceleration_y": 0.20, "acceleration_z":
            ↪ 0.91, ...},
        ...
        ]
    },
    {
        "label": "L",
        "sensor_readings": [
        {"acceleration_x": -0.10, "acceleration_y": -0.25, "acceleration_z"
            ↪ : 0.88, ...},
        {"acceleration_x": -0.12, "acceleration_y": -0.28, "acceleration_z"
            ↪ : 0.85, ...},
        ...
        ]
    },
    ...
    ]
}
```

2. **Size:**

   The dataset comprises a total of 200 labeled instances, providing a robust foundation for training and evaluation. Each gesture type (W, O, L) is well-represented, with over 70 instances for each, ensuring balanced class distribution and supporting reliable model performance.

3. **Format:**

   The dataset is stored in the JSON format, a flexible and widely adopted standard for data representation. Each JSON entry contains a hierarchical structure, including stroke indices, an array of stroke points, and corresponding X-Y coordinates for each point. This structured organization facilitates efficient parsing and analysis, making it ideal for sequential data representation and machine learning applications.

4. **Anomalies:**

   Efforts were made to ensure clear and deliberate wand movements during gesture performances to minimize the occurrence of anomalies. A manual review process was applied during the labeling phase to detect and correct any mislabeled or ambiguous instances, thereby enhancing the overall quality and reliability of the dataset.

5. **Measurement and Screen Size:**

   - **Accelerometer Measurements:**

Motion data was captured in three dimensions (X, Y, Z) using the accelerometer on the Arduino Nano 33 BLE Sense board. Each gesture was performed within a duration of 1 to 2 seconds, ensuring consistency in the data collection process. The accelerometer's sensitivity enabled precise tracking of gesture dynamics.

- **Screen Size:**

  A laptop screen with dimensions of 25x30 cm was used for real-time visualization and labeling of recorded gestures. The user interface provided a robust platform for reviewing, labeling, and correcting gesture data, ensuring accurate annotations and facilitating seamless data management.

6. **Origin:**

   The dataset for the Magic Wand project originates from the integration of real-world gestures with advanced technology, enabled by the Magic Wand website. Our team utilized the Magic Wand Capture sketch, which was uploaded onto the Arduino Nano 33 BLE Sense board, to collect motion data. The user-friendly interface of the website facilitated seamless recording, reviewing, and labeling of unique gestures. This collaborative effort ensured the creation of a diverse and representative dataset, encompassing authentic real-world scenarios and variations among users. The Magic Wand prototype, combined with the Arduino Nano 33 BLE Sense board, served as a crucial tool for capturing nuanced gesture variations. The structured methodology provided by the website significantly contributed to refining the dataset, aligning with the project's objectives and enhancing its utility for practical machine learning applications.

The key benefit of expanding a dataset with more input data is that it allows the machine learning model to improve in terms of accuracy, efficiency, and performance. By including additional data, the model can better learn the patterns and characteristics of the gestures. This approach is applied to the remaining gestures, such as ring, slope, and unknown, following the same process [**Wings:2023**]. With the data collected for these gestures, we can split the dataset into two parts: one for training the model and another for testing its performance.
A significant challenge in preparing the data is the presence of outliers or anomalies, which are values that deviate significantly from the expected range. These outliers can negatively impact the model's performance. However, one way to address this challenge is by increasing the volume of data to provide a more robust and diverse training set, making the model more resilient to such anomalies. Below are some common types of outliers that may be encountered during data preparation [MO19]:

- **Noise in Accelerometer Readings:**

  Environmental interference or external factors during data collection may introduce noise into the accelerometer readings. Outliers can appear as sudden spikes or drops in sensor values that do not correspond to genuine gestures. These are typically considered noise and can be minimized by increasing data points and ensuring proper sensor calibration.

- **Abnormal Gesture Patterns:**

  Outliers may arise when users perform gestures that deviate from the expected behavior or in an unconventional manner. These abnormal gestures can negatively affect the training process and may distort the model's ability to correctly classify gestures.

- **Sensor Malfunctions:**

  Outliers can also be a result of sensor malfunctions or inaccuracies in the Arduino Nano 33 BLE Sense device. Issues like sudden jumps, constant offsets, or erratic

sensor behavior are typically indicative of hardware problems and should be addressed by either recalibrating the sensors or discarding faulty readings.

- **Inconsistent Data Recording:**

  Variations in the way users record gestures—such as differences in speed, amplitude, or gesture duration—can lead to inconsistencies in the dataset. It is important to standardize the recording process and eliminate these variations to ensure that the model receives consistent and reliable data for training.

- **User-Specific Outliers:**

  Each individual may exhibit unique movement patterns or unintentional variations in how they perform gestures. These user-specific outliers can affect the model's ability to generalize across different users. Identifying and addressing these variations, such as normalizing gesture data for different users, is essential for creating a model that works well across a diverse group of individuals.

- **Data Transmission Errors:**

  Errors or interruptions during the data transmission from the Arduino Nano to the recording system can result in outliers, such as missing or corrupted data points. These transmission errors can be detected and rectified by performing error-checking and data validation techniques during preprocessing to ensure that the dataset is complete and reliable.

In practice, outliers can be handled in several ways. They can either be removed from the dataset if they are deemed irrelevant or incorrect, or they can be corrected if a reasonable method of rectification is available. Handling outliers effectively ensures that the model can learn from clean, reliable data, improving its accuracy and performance. Additionally, identifying and addressing outliers early in the data preparation process helps create a more robust and generalizable model.

9.10. Data Transformation and Data Mining

The data mining step is the phase where the model is developed and trained to recognize patterns and make predictions. However, before beginning this process, it is crucial to select a suitable algorithm that aligns with the project's objectives and the dataset's characteristics. Factors such as the type of data, computational constraints, and desired output play a key role in this decision. As detailed in the previous section, the dataset is split into training and testing subsets to ensure a fair evaluation of the model's performance. This division allows for effective testing and validation, enabling the identification of areas for improvement and fine-tuning of the model parameters. Additionally, cross-validation techniques can be employed to enhance reliability and minimize overfitting, ensuring that the model generalizes well to new data [WS20a].

Training the Model

To begin our project, we first need to customize one of the example applications included in the SparkFun Edge Board Support Package (BSP) to accommodate the input of our captured dataset. As a prerequisite, follow SparkFun's "Using SparkFun Edge Board with Ambiq Apollo3 SDK" guide to configure the Ambiq SDK and the SparkFun Edge BSP. Once the initial setup is complete, modifications can be made to the example code to handle the dataset appropriately [Lai22].
After adapting the code, the program will be prepared to process the dataset as input. The next step involves building the modified application and flashing it onto the SparkFun Edge device. This ensures the program is ready for testing and data collection.

Table 9.1.: **CNN sequence to classify gestures**

Layer (type)	Output Shape	Param#
conv2d(Conv2D)	(None, 128, 3, 8)	104
max_pooling2d(MaxPooling2D)	(None, 42, 1, 8)	0
dropout(Dropout)	(None, 42, 1, 8)	0
conv2d_1(Conv2D)	(None, 42, 1, 16)	528
max_pooling2d_1(MaxPooling2D)	(None, 42, 1, 16)	0
dropout_1(Dropout)	(None, 42, 1, 16)	0
flatten(Flatten)	(None, 224)	0
dense(Dense)	(None, 16)	3600
dropout_2(Dropout)	(None, 16)	0
dense_1(Dense)	(None, 4)	68

In the subsequent phase, the three group members will perform the designated motions to construct the dataset. To achieve this, open a terminal window and execute the following command:

```
script output.txt
```

In the terminal interface, connect to the "115200" device. Once connected, real-time measurements from the accelerometer will be displayed on the screen. These readings will also be saved in a file named `output.txt` for further processing. This process ensures an organized and accurate dataset, essential for training and validating the model. Additionally, these steps enable efficient data collection and synchronization, facilitating seamless integration into the project workflow [WS20a].

The text file will store data formatted to meet the training set's requirements. To build a high-quality dataset, each gesture is repeated multiple times to ensure sufficient variation before exiting the program. Once one group member completes this process, the next member records a similar file named `output.txt`. The logging of accelerometer data can be stopped by pressing the button labeled "14" [**Wings:2023**].

To maintain clarity, the output.txt files are renamed according to the individuals who performed the gestures. This naming convention helps differentiate datasets and ensures organized testing and validation [WS20a].

Additionally, data for the "unknown" category is collected and incorporated into the dataset. This step enables the model to classify gestures outside the predefined set as "unknown." With this inclusive data, the training process improves progressively, leading to enhanced validation accuracy over time.

The following steps are used to train the model:

- **Loading TensorBoard:** Set up TensorBoard to monitor the training process and visualize metrics.
- **Running Training Code:** Initiate the training process using scripts executed in PyCharm.
- **Data Augmentation:** Execute the data_augmentation script to enhance the training dataset by introducing variations in acceleration values, providing the model with more diverse input data.
- **Monitoring Output:** Observe the output values displayed on the screen, which include metrics such as the memory size of the model and training progress.

9.10.1. Model

In this project, the model processes a sequence of 128 three-axis accelerometer readings, corresponding to approximately five seconds of motion, and outputs an array of four probabilities: one for each predefined gesture and one for "unknown." Convolutional Neural Networks (CNNs) are employed due to their ability to capture patterns and relationships within adjacent data points effectively [WS20a]. The multi-layered CNN is designed to learn and recognize each gesture by analyzing its fundamental components. For example, it may learn to identify simple up-and-down movements and further understand how combining these with specific z- and y-axis motions forms a "wing" gesture [WS20a].

A CNN achieves this by employing a series of filters organized in hierarchical layers, where each filter is trained to recognize specific data features. In the initial layer, filters may detect basic structures like an upward acceleration. The identified features are then passed to the next layer, which combines them into more complex structures. For instance, in the "wing" gesture, the "W" shape could be identified by a sequence of four alternating upward and downward accelerations.

This hierarchical structure enables the CNN to progressively build an understanding of intricate patterns and gestures, resulting in robust and precise classification based on the input accelerometer readings.

For data acquisition, Inertial Measurement Unit (IMU) signals are utilized 9.17. The IMU is an electronic component that integrates the accelerometer. The IMU object is derived from the Arduino LSM9DS1 library, which facilitates seamless data collection and processing. This integration ensures efficient and reliable gesture data acquisition, forming the basis for effective training and testing of the CNN model.

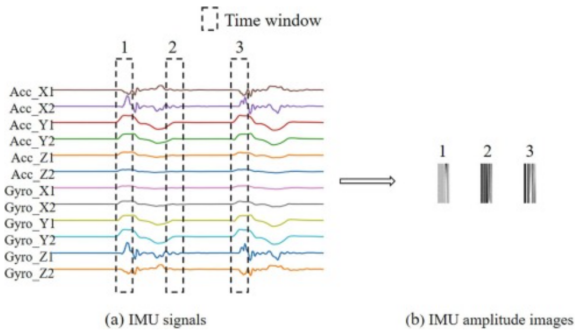


Figure 9.17.: **IMU signals [Xu+22]**

The convolutional layer is the first component of the network to receive the raw accelerometer data as input. This data is structured into a specific shape, defined by the ‘input_shape‘ argument. The shape is ‘(seqlength, 3, 1)‘, where ‘seqlength‘ denotes the total number of accelerometer readings provided, which is 128 by default. Each reading comprises three values corresponding to the x, y, and z axes of motion [Xu+22].

This input format ensures that the network can accurately process the multidimensional nature of the accelerometer data, facilitating the extraction of relevant features for subsequent layers in the model.

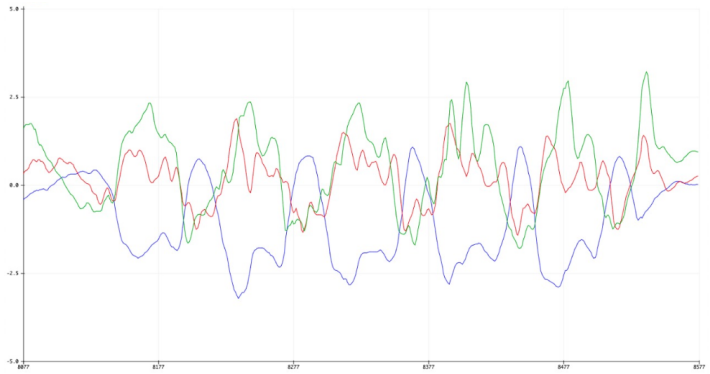


Figure 9.18.: **IMU Accelerometer Graph [Wings:2023]**

The convolutional layer is responsible for processing raw input data and identifying foundational features that subsequent layers can analyze and interpret. This is achieved through the use of the ‘Conv2D()‘ function, which specifies the parameters for feature extraction. The key parameter is the window size, defined in this instance as ‘(4, 3)‘. This configuration implies that the convolutional filter examines four consecutive accelerometer readings across all three axes.
By encompassing four consecutive measurements, each filter effectively captures and analyzes a brief snapshot of time. This allows the model to detect and represent variations in acceleration over time. The process of feature extraction using these filters is visually depicted in 9.19 [**Wings:2023**].
This capability to identify time-dependent changes forms the basis for understanding and classifying gestures, as the extracted features are subsequently passed through the network for further processing and interpretation.

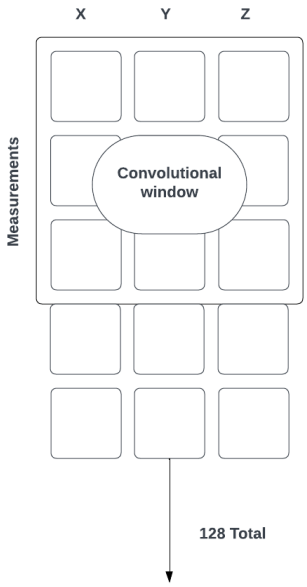


Figure 9.19.: **A convolution window overlaid on the data**

The padding argument defines how the filter window moves across the data during convolution operations. When set to "same," the layer’s output dimensions remain

consistent with the input, maintaining a length of 128 and a width of 3. Each movement of the filter window generates one output value, and with the "same" padding, the window slides across the width three times and down the length 128 times. Once the convolution window completes its traversal, the data is transformed into eight feature maps using the filters. These feature maps are passed to the next layer, MaxPool2D. The MaxPool2D layer processes the (128, 3, 8) tensor output from the convolutional layer and reduces it to a smaller (42, 1, 8) tensor. This reduction is achieved by sliding a window across the data and selecting the largest value within each window, which is then passed to the output. The size of the sliding window is specified as (3, 3). By default, the window shifts to ensure it processes only new, non-overlapping data. Figure **??** demonstrates this process [WS20a].
The primary objective of a CNN is to condense a large, intricate input tensor into a smaller, simpler representation. The MaxPool2D layer contributes to this goal by summarizing the first convolutional layer's output into a concentrated, high-level abstraction of the most relevant information. This abstraction helps eliminate irrelevant details, retaining only the most significant features required to identify the gesture.
Following the pooling operation, the data passes through a Dropout layer. Dropout is a regularization method designed to mitigate overfitting by introducing noise. It randomly excludes certain data points between layers, forcing the neural network to adapt to variability and improving its robustness. This layer is active during training but inactive during inference, allowing all data to flow through at that stage [WS20a]. The Dropout layer further refines the input, distilling it into a multidimensional tensor with a shape of (14, 1, 16), representing the critical features of the input data. This process can be repeated by adding more convolutional and pooling layers, with the number of layers acting as a hyperparameter that can be adjusted. In this model, two convolutional layers were deemed sufficient for achieving the desired performance. Figure 9.20 illustrates the CNN's layer sequence.

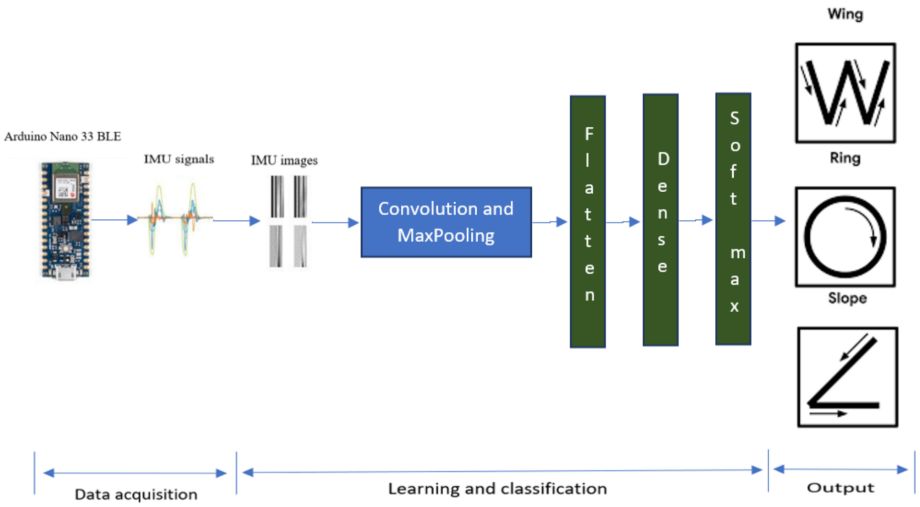


Figure 9.20.: **CNN sequence to classify Wing,Ring and Slope [Wings:2023]**

We begin by flattening the multidimensional data from the convolutional layers and feeding it into a Dense layer, also called a fully connected layer, to identify the major features in the input. The Flatten layer transforms a tensor with shape $(14, 1, 16)$ into a one-dimensional tensor with shape (224). This step condenses the multidimensional data into a single dimension, simplifying further processing.
The resulting flattened tensor is then fed into a Dense layer with 16 neurons. Each

neuron in this layer is connected to every input, enabling the model to analyze all features simultaneously and learn the relationships among various combinations of inputs. The Dense layer generates a compressed representation of the input data, summarized into 16 key features [WS20a].
Next, these 16 values are reduced further to represent the four gesture classes: Wing, Ring, Slope, and Unknown. The final Dense layer contains four neurons, each corresponding to one class. These neurons are connected to all 16 outputs from the previous layer. During training, this layer learns the patterns and relationships that identify each gesture class. The layer uses a "softmax" activation function, producing output probabilities that sum to 1. This provides a probabilistic classification of the input data into the four classes.

Gesture Prediction and Validation

Once the model produces an output tensor containing gesture probabilities, a function called `PredictGesture()` ensures accurate classification by minimizing false positives. This function performs two key tasks:

1. **Threshold Check**: It verifies that the probability of the detected gesture meets a predefined minimum threshold.
2. **Inference Count Check**: It ensures the gesture is consistently detected across a required number of inferences to confirm its validity.

The number of inferences required varies based on the gesture, as each gesture takes a different amount of time to perform. These thresholds and inference requirements are specified in the `constants.cc` file [WS20a].
The `PredictGesture()` function returns a numeric value indicating the detected gesture:

- **0**: Wing
- **1**: Ring
- **2**: Slope

Workflow in `PredictGesture()`

1. The function receives the prediction scores from the main program.
2. It calculates a `maxPredictionScore` and identifies the corresponding `maxPredictionIndex`.
3. These values are compared with the `kNoGesture` and `kDetectionThreshold` constants to determine whether a valid gesture has been identified.
4. If a gesture is detected, its numeric value (equal to `maxPredictionIndex`) is assigned to the `foundGesture` variable.
5. The `foundGesture` value is returned to the main function and passed to the `output_handler()` function, which displays the recognized gesture.

This combination of convolutional and fully connected layers, paired with a robust validation function, ensures that the model accurately classifies gestures while minimizing false positives. This architecture is particularly effective for analyzing time-series sensor data, such as accelerometer readings, and enables reliable gesture recognition [WS20a].

10. Documentation Development

10.1. Structure, Idea and Flow Chart

- Data Collection and Preprocessing:

  Real-world gestures are recorded and stored in JSON format. The strokes are loaded, visualized, and augmented for diverse training.

- Image Generation and Augmentation:

  Strokes are converted into images, and augmentation techniques are applied to create a robust dataset.

- Dataset Splitting:

  The dataset is divided into training, validation, and test sets.

- Model Creation and Training:

  A CNN model is defined and compiled. The model is trained using the training dataset with checkpoints saved.

- Model Evaluation:

  The trained model is evaluated on the test dataset for accuracy metrics.

- Model Conversion and Quantization:

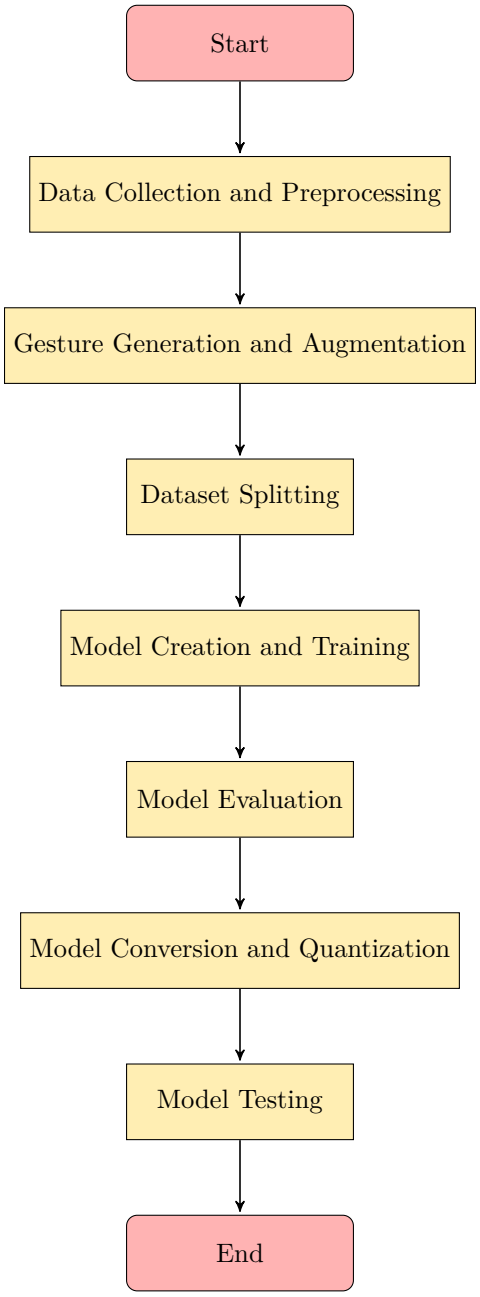
  The model is converted to TensorFlow Lite format. Quantization is applied to reduce model size.

- Model Testing:

  TensorFlow Lite models are tested on the test dataset.

- Conversion to C Source File:

  The quantized TensorFlow Lite model is converted to a C source file for deployment.



The flowchart outlines the comprehensive process involved in training a machine learning model for the Magic Wand project. It commences with the "Start" node, where the journey begins. The initial step, "Data Collection and Preprocessing," involves capturing real-world gestures using the Arduino Nano 33 BLE Sense board and subsequently preprocessing the data. The following stage, "Gesture Generation and Augmentation," transforms the recorded strokes into images and applies augmentation techniques to enhance dataset diversity.

The process then proceeds to "Dataset Splitting," where the dataset is partitioned into training, validation, and test sets. The core of the operation lies in "Model Creation and Training," where a Convolutional Neural Network (CNN) model is crafted and trained using the designated training dataset. Following this, the model's performance is evaluated on the test dataset through "Model Evaluation."

The subsequent steps, "Model Conversion and Quantization," involve converting the

trained model into TensorFlow Lite format and applying quantization to optimize the model's size. The optimized model is then put to the test in "Model Testing," where TensorFlow Lite models undergo testing on the dedicated test dataset. Finally, the process concludes with the "End" node.
This flowchart provides a visual representation of the intricate steps undertaken, offering a clear and organized overview of the Magic Wand project's data training pipeline. From initial data collection to the deployment-ready TensorFlow Lite models, the flowchart encapsulates the entire journey in a structured and systematic manner.

10.2. ML Pipeline

A Machine Learning (ML) pipeline for the Magic Wand project involves a systematic sequence of steps to develop, train, and deploy a machine learning model for gesture recognition [**Cong:2022**]. Here's a detailed explanation of each phase along with the significance and advantages of using an ML pipeline:

1. **Problem Definition:**
   - Objective: Recognize gestures made with the Arduino Nano BLE 33 sensor as specific commands (e.g., gestures to control a device).
   - Input: Time-series data from the sensor capturing gesture information.
   - Output: A prediction of the recognized gesture.
2. **Data Collection:**
   - Hardware: Arduino Nano BLE 33 sensor.
   - Sensors: Accelerometer, gyroscope, or any other relevant sensors on the Arduino Nano BLE 33 [**Mardiyanto:2017**].
   - Data: Collect labeled time-series data for each gesture to create a training dataset.
3. **Data Preprocessing:**
   - Normalization: Normalize sensor readings to a common scale.
   - Feature Extraction: Extract relevant features from the time-series data (e.g., peaks, slopes) [**Cong:2022**].
   - Label Encoding: Convert categorical labels (gestures) into numerical format.
4. **Dataset Splitting:**

   Train-Validation-Test Split: Split the dataset into training, validation, and test sets.
5. **Model Selection:**
   - Algorithm: Choose a suitable ML algorithm for time-series gesture recognition (e.g., LSTM, CNN, Random Forest).
   - Model Architecture: Design the architecture of the chosen model.
6. **Model Training:**
   - Training: Train the model using the training dataset.
   - Validation: Validate the model using the validation dataset to prevent overfitting.
   - Hyperparameter Tuning: Optimize hyperparameters for better performance.
7. **Model Evaluation:**

- Test Set Evaluation: Evaluate the trained model using the test dataset.
- Metrics: Choose appropriate metrics (e.g., accuracy, precision, recall) for evaluation [Zho+20].
- Convert the model to TensorFlow Lite format and apply quantization to reduce model size.
- Test TensorFlow Lite models on the test dataset. Convert the quantized TensorFlow Lite model to a C source file for microcontroller deployment [Goo24].

8. **Deployment to Arduino Nano BLE 33:**
   - Conversion: Convert the trained model to a format compatible with the Arduino Nano BLE 33.
   - Integration: Integrate the model into the Arduino Nano BLE 33 environment.
   - Real-time Inference: Implement code for real-time inference on the Arduino Nano BLE 33.
9. **Testing and Debugging:**
   - Device Testing: Test the complete system with the Arduino Nano BLE 33 and ensure accurate gesture recognition.
   - Debugging: Address any issues related to sensor data, model inference, or device integration.

     `Tests`

     `Errorhanddler`
10. **Optimization:**
    - Size Optimization: Optimize the model for size and efficiency suitable for the Arduino Nano BLE 33.
    - Power Optimization: Optimize power consumption for prolonged device usage [**Cong:2022**].
11. **Documentation:**
    - Code Documentation: Provide detailed comments in the code for better understanding.
    - Project Report: Create a comprehensive report documenting the entire ML pipeline, including challenges faced and solutions.

10.2.1. Why Use an ML Pipeline

1. Structured Development:

   ML pipelines provide a systematic and organized approach to model development, ensuring clarity and maintainability.

2. Reproducibility:

   The pipeline allows for easy reproduction of experiments, facilitating collaboration and research reproducibility.

3. Hyperparameter Tuning:

   Enables systematic optimization of model hyperparameters for better performance [**Cong:2022**].

4. Ease of Debugging:

   A step-by-step approach makes it simpler to identify and rectify issues at each stage.

5. Efficient Resource Utilization:

   ML pipelines enable efficient use of computational resources by breaking down complex tasks into manageable steps.

6. Scalability:

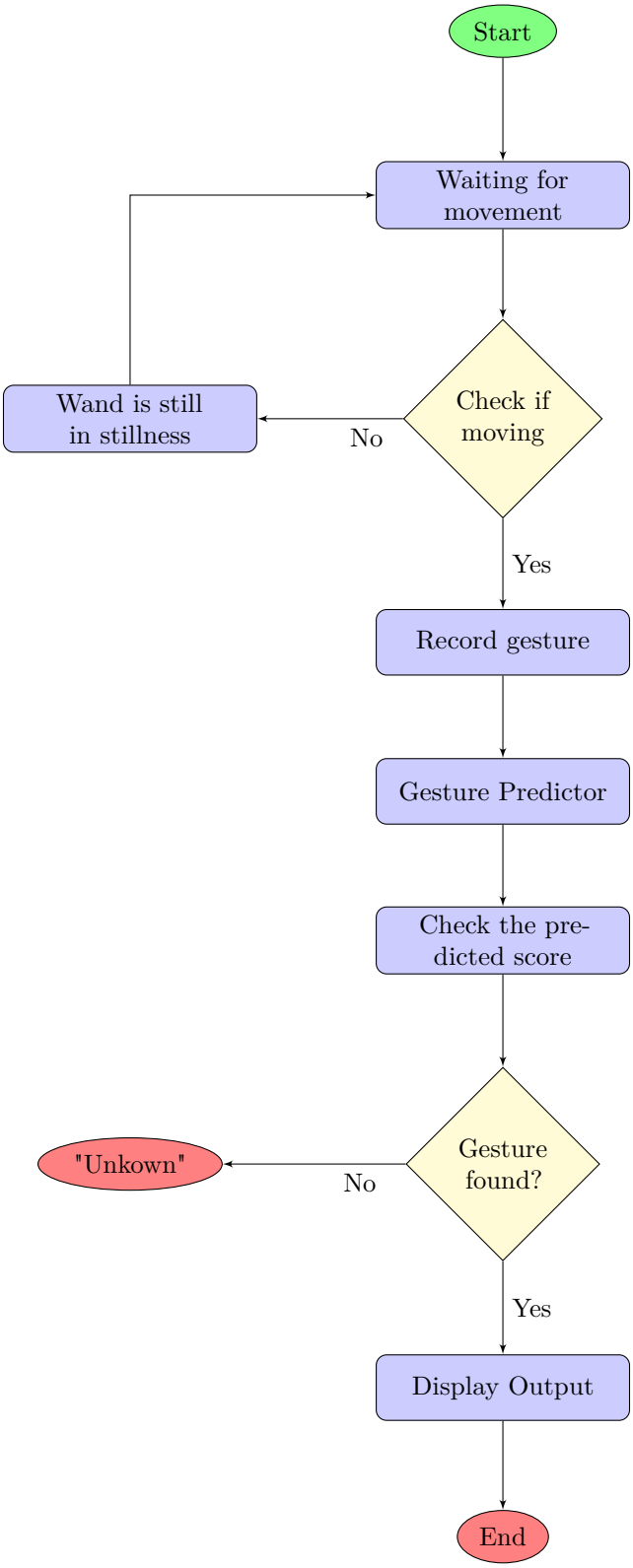
   A well-defined pipeline is scalable, allowing for the incorporation of additional data and improvements without a complete overhaul.

7. Deployment Readiness:

   Ensures that the model is not only accurate but also optimized and ready for deployment on edge devices.

An ML pipeline for the Magic Wand project streamlines the development process, enhances model performance, and ensures the successful deployment of the gesture recognition system [Zho+20].

10.2.2. Magic Wand Flowchart



The flow chart depicts a basic flow of the program. Initially the wand is still , once the wand starts moving, the program recognizes the movement followed by recording and

predicting the gesture. Then the most probable gesture to the one that was waved by the user is displayed in the serial monitor window.

11. Development to Development

11.1. Development to Deployment

11.1.1. 1. Development Phase

Define Problem

- Recognize gestures (e.g., "wing," "ring," "slope") using an accelerometer.
- Minimize model size due to memory constraints of the Arduino Nano 33 BLE Sense.

Data Collection

- Collect gesture data using the board's sensors (accelerometer, gyroscope).
- Label data corresponding to gestures, including "unknown" gestures.

Data Preprocessing

- Filter out noise using smoothing algorithms.
- Normalize accelerometer values to ensure consistency across datasets.

Model Design

- Use TensorFlow to design a CNN model for gesture recognition.
- Choose lightweight architectures suitable for TinyML.

Model Training

- Split the dataset into training, validation, and test sets.
- Optimize the model using quantization, pruning, or knowledge distillation to fit edge device constraints.

11.1.2. 2. Tools and Libraries

11.1.3. 1. Hardware

Arduino Nano 33 BLE Sense

- Sensors: Accelerometer, gyroscope, magnetometer, temperature, pressure.
- Features: Bluetooth Low Energy (BLE) for communication, small form factor.
- Use Case: Collect gesture data, run TinyML models, and communicate results via BLE.
- **Documentation Link**: `https://www.arduino.cc/en/Guide/NANO33BLESense`

Peripheral Tools

- USB Cable: For data transfer and power supply.
- Sticky Tape: To secure the Arduino to the wand (physical setup).
- External LEDs or buzzers (optional): For additional feedback mechanisms.

11.1.4. 2. Development Environment

Arduino IDE

- Version: Latest stable release.
- Use Case: Write and upload C++ code to the Arduino Nano 33 BLE Sense.
- Extensions: Add board-specific libraries like `Arduino_TensorFlowLite`.
- Installation: Available for Windows, Mac, and Linux.
- **Download Link**: `https://www.arduino.cc/en/software`

Python

- Version: 3.8 or later.
- Use Case: Data preprocessing, model development, and testing.
- Popular IDEs: PyCharm, Visual Studio Code, or Jupyter Notebook.

11.1.5. 3. Libraries for Data Preprocessing

- **NumPy**:
  - Use Case: Perform numerical operations on accelerometer data, such as normalization and smoothing.
  - Installation: `pip install numpy`
- **Pandas**:
  - Use Case: Organize sensor data into data frames for easy manipulation and analysis.
  - Installation: `pip install pandas`
- **Matplotlib**:
  - Use Case: Visualize gesture patterns (e.g., acceleration over time).
  - Installation: `pip install matplotlib`

11.1.6. 4. Machine Learning Libraries

- **TensorFlow**:
  - Version: TensorFlow 2.x.
  - Use Case: Model creation, training, and optimization.
  - Installation: `pip install tensorflow`
- **TensorFlow Lite Converter**:
  - Use Case: Convert TensorFlow models to a format suitable for microcontrollers.

  - Integrated with TensorFlow.

- **TensorFlow Lite for Microcontrollers**:
  - Use Case: Load and run the `.tflite` model on the Arduino Nano 33 BLE Sense.
  - Integrated into Arduino libraries.

11.1.7. 5. Deployment Libraries

- **Arduino_TensorFlowLite**:
  - Use Case: Interface TensorFlow Lite models with the Arduino Nano 33 BLE Sense.
  - Installation: Install via Arduino Library Manager.

- **Arduino_LSM9DS1**:
  - Use Case: Access accelerometer, gyroscope, and magnetometer data.
  - Installation: Available in the Arduino IDE.

11.1.8. 6. Optional Libraries

- **Scikit-learn**:
  - Use Case: Preprocessing tools like scaling, splitting datasets, and evaluating performance metrics.
  - Installation: `pip install scikit-learn`

- **Seaborn**:
  - Use Case: Enhanced data visualizations for exploring gesture patterns.
  - Installation: `pip install seaborn`

- **Keras**:
  - Use Case: High-level API for rapid prototyping of the CNN model.
  - Installation: `pip install keras`

11.1.9. 7. Version Control and Collaboration

- **Git**:
  - Use Case: Track code changes and collaborate with team members.
  - Integration: Platforms like GitHub or GitLab.

- **Google Colab (Optional)**:
  - Use Case: Train and test models in a cloud-based Jupyter environment.

11.1.10. 8. Debugging and Profiling Tools

- **Serial Monitor (Arduino IDE)**:
  - Use Case: Debug real-time sensor readings and verify program behavior.

- **TensorBoard**:
  - Use Case: Monitor model training metrics like loss and accuracy.
  - Command to launch: `tensorboard -logdir=logs/`

11.1.11. 3. File Structure

Here's the directory structure in LaTeX using plain text and removing Unicode symbols:
latex Copy code

```
MagicWandProject/
|-- data/
|   |-- raw/            # Raw sensor data
|   |-- processed/       # Preprocessed data
|   |-- labels.csv       # Gesture labels
|-- models/
|   |-- cnn_model.h5      # Saved Keras model
|   |-- cnn_model.tflite   # TensorFlow Lite model
|-- src/
|   |-- preprocess.py     # Data preprocessing scripts
|   |-- train.py         # Model training script
|   |-- deploy/
|      |-- main.ino      # Arduino C++ script
|      |-- tflite_integration.ino  # TFLite integration code
|-- requirements.txt      # Required Python libraries
|-- README.md           # Project description
```

11.1.12. Saving and Loading Models

Saving the Model

The following Python code demonstrates how to save a model in the HDF5 format and convert it to TensorFlow Lite using 'TFLiteConverter'.

Listing 11.1: Saving and Converting a Model to TensorFlow Lite

```
import tensorflow as tf

# Save the model in HDF5 format
model.save('models/cnn_model.h5')

# Convert to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_saved_model('models/cnn_model.h5')
tflite_model = converter.convert()

# Save the converted model
with open('models/cnn_model.tflite', 'wb') as f:
f.write(tflite_model)
```

Loading the Model

Listing 11.2: Loading a Saved Model for Evaluation

```
import tensorflow as tf

# Load the model for evaluation
model = tf.keras.models.load_model('models/cnn_model.h5')
```

12. Deployment

Following the KDD process, this chapter intends to explain the major components for the implementation in the Board Arduino nano 33 BLE Sense to detect specific gestures. This chapter will primarily be substantiated by previously covered topics, the deployment phase. Along with these concepts, the behavior of the board and system are also verified.

12.1. Application Description

The Magic Wand is project leveraging gesture recognition to provide intuitive and seamless control over devices. It uses the Arduino Nano 33 BLE Sense with built-in sensors (accelerometer, gyroscope, and microphone) and a TensorFlow Lite gesture recognition model.[Dai] This system interprets user gestures and translates them into commands, unlocking a wide range of applications, such as:

- **Home Automation:** Control lights, appliances, and other IoT devices.
- **Gaming and AR/VR:** Enhance interactive experiences through gesture-based controls.
- **Assistive Technology:** Empower individuals with disabilities to interact with devices.
- **Healthcare:** Enable gesture-based monitoring and control in rehabilitation.
- **Education:** Teach gesture recognition and machine learning fundamentals in classrooms.[Dai]

12.2. Structure

The deployment structure of the Magic Wand involves several interconnected components:

12.2.1. Hardware

- **Arduino Nano 33 BLE Sense:** Equipped with sensors for data collection.
- **Sensors:**
  - The accelerometer enables significant motion detection (SMD), which recognizes large-scale movements. SMD can be used to activate specific device functions, such as waking the system from sleep mode or triggering predefined actions based on significant gestures.[Zho+20]
  - The STMicroelectronics LSM9DS1 gyroscope is a precision instrument for measuring angular velocity around the x, y, and z axes. With adaptable measurement ranges of ±245, ±500, and ±2000 degrees per second (dps), the gyroscope is highly versatile for various applications such as inertial navigation, robotics, and drone stabilization.[STM24]

  - a magnetometer, the x, y, and z axes (Mx, My, Mz) typically delineate the three-dimensional space in which the magnetic field is being assessed. The x-axis typically corresponds to the horizontal component of the magnetic field, the y-axis signifies the vertical component, and the z-axis reflects the magnetic field strength [**Kostiainen:2023**].

12.2.2. Software

- TensorFlow Lite provides one of the most popular model optimization techniques is called quantization.[**tensorflowlite:2025**]

- A trained gesture recognition model deployed on the Arduino Nano.

12.2.3. Data Processing

- Captures real-time sensor data.

- Preprocesses data for gesture recognition.

12.2.4. Output/Interface

- Visual feedback via serial monitor.

- Actuation of connected devices or systems.

12.3. Idea

The Magic Wand translates motion data into meaningful actions by:

1. Capturing real-time motion signals through sensors.[**Anh:2009**]

2. Preprocessing data to extract relevant features.

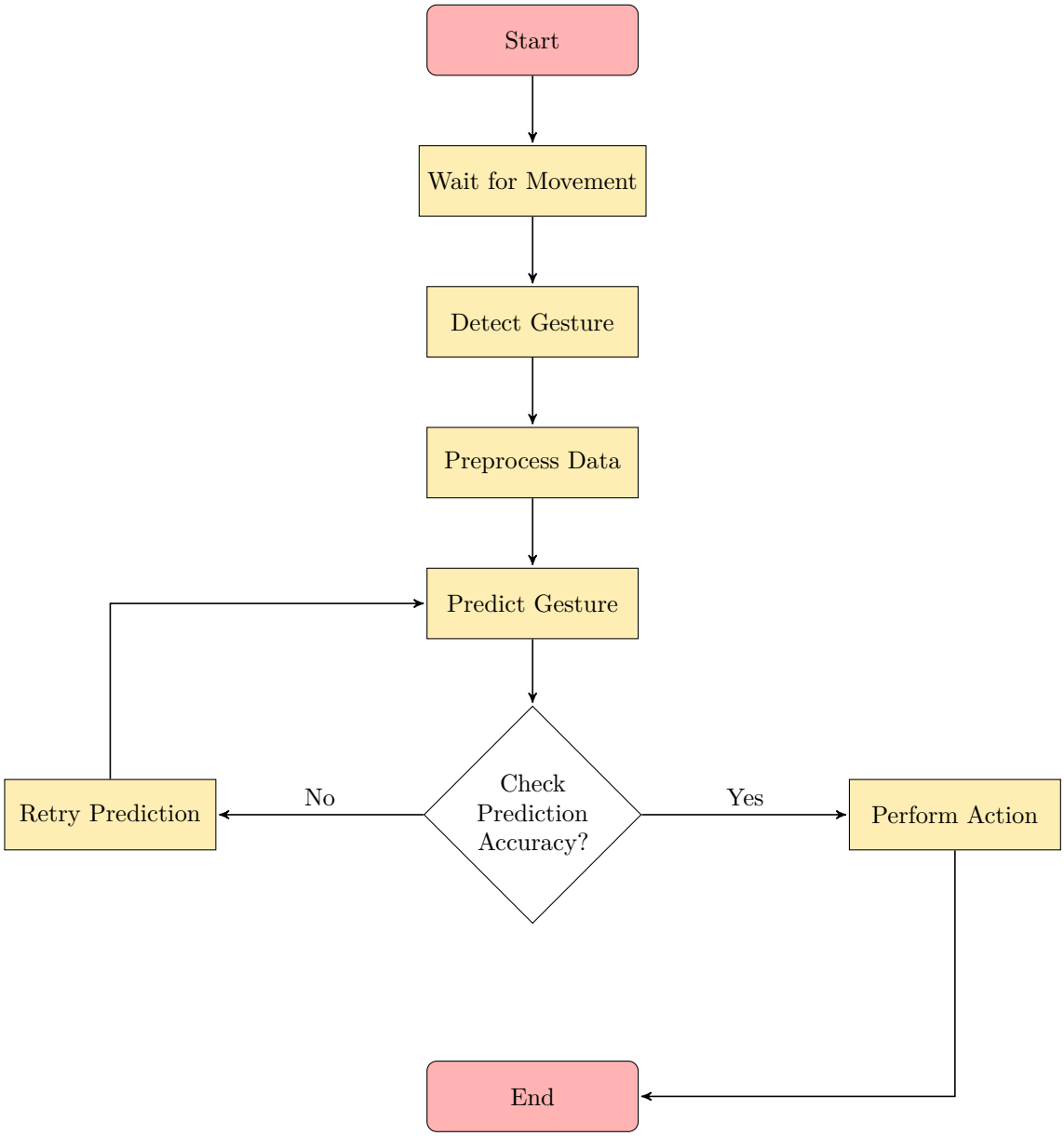
3. Feeding preprocessed data into a pre-trained ML model for classification.

4. Producing accurate gesture predictions.

5. Translating predictions into actionable commands for device control or feedback.

12.4. Flow Chart

12.4.1. Overall System Flow Chart



12.5. ML Pipeline

The ML pipeline for deploying the Magic Wand involves the following steps:

12.5.1. Step 1: Problem Definition

- Define the goal: Recognize hand gestures as specific commands.
- Input: Real-time sensor data from accelerometer and gyroscope.
- Output: Predicted gestures.

12.5.2. Step 2: Data Collection

- Collect raw sensor data for gestures using the Arduino Nano 33 BLE Sense.
- Store data in a structured format (e.g., JSON or CSV) with labeled gestures.

12.5.3. Step 3: Data Preprocessing

- Normalize sensor data to a consistent range.
- Extract meaningful features such as velocity and angular motion.
- Encode gesture labels numerically for model training.

12.5.4. Step 4: Dataset Splitting

- Divide the data into:
  - Training set: 70%
  - Validation set: 20%
  - Test set: 10%

12.5.5. Step 5: Model Design

- Choose a lightweight ML model such as a Convolutional Neural Network (CNN) or Recurrent Neural Network (RNN) optimized for time-series data.

12.5.6. Step 6: Model Training

- Train the model using the training dataset.
- Validate the model periodically to avoid overfitting.
- Save checkpoints for the best model.

12.5.7. Step 7: Model Evaluation

- Test the model on unseen data (test set).
- Evaluate accuracy, precision, recall, and F1-score.

12.5.8. Step 8: Model Optimization

- Convert the model to TensorFlow Lite format.
- Apply quantization techniques to reduce size and improve inference speed.

12.5.9. Step 9: Deployment

- Convert the quantized TensorFlow Lite model into a C header file.
- Integrate the model into the Arduino IDE.
- Upload the model to the Arduino Nano 33 BLE Sense.

12.5.10. Step 10: Testing and Debugging

- Test the system for real-time gesture recognition.
- Debug issues with sensor data, model inference, or device integration.

12.5.11. Step 11: Deployment to Real-World Application

- Use the Magic Wand for controlling devices, providing feedback, or integrating into larger systems.

12.6. TensorFlow Lite (TFLite)

12.6.1. Description

TensorFlow Lite provides one of the most popular model optimization techniques is called quantization. Quantization used to reduce the precision of the model's parameters such as weights and activation outputs into 8-bit integers.[**tensorflowlite:2025**] By default, all weights parameters are 32-bit floating-point numbers. It enables to greatly reduce the model size as 8-bit integers occupy less memory than 32-bit floating-point numbers. Although these 8-bit representations can be less precise so it turns outs little degradation in the model accuracy. TFLite supports techniques like 8-bit quantization, making models smaller and faster to run, and includes optimizations for CPUs, GPUs, and specialized accelerators.[**tensorflowlite:2025**]

12.6.2. Code Example

Listing 12.1: Running Inference with a TensorFlow Lite Model in Python

```python
import tensorflow as tf
import numpy as np

# Load the TensorFlow Lite model
interpreter = tf.lite.Interpreter(model_path="model.tflite")
interpreter.allocate_tensors()

# Get input and output tensors
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

# Create input data
input_data = np.array([[1.0, 2.0, 3.0]])

# Set the input tensor
interpreter.set_tensor(input_details[0]['index'], input_data)

# Run inference
interpreter.invoke()

# Retrieve the output
output_data = interpreter.get_tensor(output_details[0]['index'])
print("Model Output:", output_data)
```

12.6.3. Use of TensorFlow Lite (TFLite) in the Magic Wand Project

Quantization can take place during model training or after model training. Here we are referring to the quantization during model training as Quantization-aware training.[**tensorflowlite:2025**] LiteRT now supports converting weights to 8-bit precision as part of model conversion from TensorFlow GraphDefs to LiteRT's flat buffer format. Dynamic range quantization achieves a 4× reduction in model size. In addition, TensorFlow Lite (TFLite) supports on-the-fly quantization and dequantization of activations to allow for:

- Using quantized kernels for faster implementation when available.[**tensorflowlite:2025**]
- Mixing of floating-point kernels with quantized kernels for different parts of the graph.

The activations are always stored in floating point. For operations that support quantized kernels, the activations are quantized to 8-bit precision dynamically prior to processing and are dequantized to floating point precision after processing. Depending on the model being converted, this can provide a speedup over pure floating-point computation.

In contrast to quantization-aware training, the weights are quantized *post-training*, and the activations are quantized dynamically at inference in this method. Therefore, the model weights are not retrained to compensate for quantization-induced errors. It is important to check the accuracy of the quantized model to ensure that the degradation is acceptable.[**tensorflowlite:2025**]

- **Model Training and Conversion:** TFLite is utilized during the development phase to convert a pre-trained TensorFlow model (e.g., a gesture recognition model trained on accelerometer data) into a lightweight format. The conversion process includes techniques such as quantization, where model weights and activations are reduced from 32-bit floating-point numbers to 8-bit integers. This step ensures that the model is optimized for memory usage and inference speed, making it suitable for the microcontroller.[**tensorflowlite:2025**]
- **Testing and Validation:** TFLite is used to validate the model on a desktop or mobile device by running inference on example inputs (e.g., accelerometer data) and verifying the output classifications (e.g., *wing*, *ring*, or *slope*). This testing ensures that the model performs accurately before deployment.
- **Optimization for Edge Deployment:** Using TFLite tools, the model is fine-tuned and optimized to run efficiently on edge devices like the Arduino Nano 33 BLE Sense. Profiling and optimizations ensure that the deployed model meets the hardware constraints.[**tensorflowlite:2025**]

a TensorFlow model to train a simple Convolutional Neural Network (CNN) on the CIFAR-10 image dataset. We then evaluate the model's accuracy and save it for further quantization.

12.6.4. TensorFlow Model Code

In order to quantize model, we need a trained TensorFlow model. So, let's train a simple CNN model on cifar10 image dataset from scratch. And will compare model accuracy of original TensorFlow model and the converted model with quantization.[**tensorflowlite:2025**]

Listing 12.2: TensorFlow CNN Model for CIFAR-10

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
import numpy as np

# Check TensorFlow version
tf.__version__

# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
# Normalize the data to range [0, 1]
x_train, x_test = x_train / 255.0, x_test / 255.0

# Define a simple CNN model
model = models.Sequential([
layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.MaxPooling2D((2, 2)),
layers.Conv2D(64, (3, 3), activation='relu'),
layers.Flatten(),
layers.Dense(64, activation='relu'),
layers.Dense(10)  # Output layer for 10 classes
])
# Compile the model
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
```

```
# Train the model
history = model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test)
    ↪ )

# Evaluate model accuracy
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"Original Model Accuracy: {test_acc:.4f}")

# Save the model
model.save("cifar10_cnn.h5")
```

Float16 quantization reduces the model size by quantizing the model's weight parameters to float16 bit floating-point numbers for a minimal impact on accuracy and latency. This quantization technique significantly reduces the model size by half.[**tensorflowlite:2025**]
We add float16 quantization of weights while convert model into TensorFlow Lite. First set the optimizations flag to default optimizations that quantize all fixed parameters such as weights. Then specify float16 is the supported type on the target platform: By default converted model still considered input and output as a float data type. This quantization method only quantized weight parameters. However, activations are still stored in floating-point.

Listing 12.3: TensorFlow Quantization for Model Conversion

```
import tensorflow as tf

# Load the trained model
model = tf.keras.models.load_model("cifar10_cnn.h5")

### Float16 Quantization ###
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Set the optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Set float16 as the supported type on the target platform
converter.target_spec.supported_types = [tf.float16]

# Convert and save the Float16 quantized model
tflite_model_fp16 = converter.convert()
with open("model_fp16.tflite", "wb") as f:
f.write(tflite_model_fp16)

print("Float16 Quantized Model Saved: model_fp16.tflite")

### Dynamic Range Quantization ###
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Set the optimization mode for dynamic range quantization
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Convert and save the dynamically quantized model
tflite_model_dynamic = converter.convert()
with open("model_dynamic.tflite", "wb") as f:
f.write(tflite_model_dynamic)

print("Dynamic Range Quantized Model Saved: model_dynamic.tflite")
```

12.6.5. Key Differences Between Float16 and Dynamic Range Quantization

Integer quantization Microcontroller devices, Edge TPU performs an integer-based operation. So above generated TFLite model won't compatible with integer-only hardware. To execute the TensorFlow model on integer-only hardware, we need to quantize all model parameters, input and output tensor to an integer.[**tensorflowlite:2025**]

Feature	Float16 Quantization	Dynamic Range Quantization
Weight Precision	16-bit floating point	8-bit integer
Activation Storage	Float32	Float32
Inference Speed	Slightly improved	Faster than Float16
Model Size Reduction	~2x	~4x

Table 12.1.: Comparison between Float16 and Dynamic Range Quantization

The post-training integer quantization is an optimization technique that converts both model's weights and activation outputs from 32-bit floating-point numbers to the nearest 8-bit fixed-point numbers. It also quantizes a model's input/output data. That tends to smaller model size and increased inference speed which is most suitable to deploy TensorFlow model on low-powered devices such as microcontrollers. Sometimes, it is also called full integer quantization as it converts all model parameters such as weights and activations into 8-bit integer numbers.[**tensorflowlite:2025**]
To quantize the variable data such as a model's input/output and intermediates between layers, we need to provide a RepresentativeDataset by supplying a set of input data in a generator function. This enables the converter to estimate a dynamic range for all the variable data.[**tensorflowlite:2025**]
Here, integer-based quantization must be required integer input and output tensor for compatibility. By default, the TensorFlow Lite Converter assign the model input and output tensor in a float. Applying a full integer quantization technique to convert a model into TFLite:

Listing 12.4: Full Integer Quantization for TensorFlow Model

```
import tensorflow as tf
import numpy as np

# Load the trained model
model = tf.keras.models.load_model("cifar10_cnn.h5")

# Load and preprocess CIFAR-10 dataset for representative dataset generation
(x_train, _), (_, _) = tf.keras.datasets.cifar10.load_data()
x_train = x_train.astype(np.float32) / 255.0  # Normalize

# Function to generate a representative dataset
def representative_data_gen():
for input_value in tf.data.Dataset.from_tensor_slices(x_train).batch(1).take(100)
    ↪ :
yield [input_value]

### Full Integer Quantization ###
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Set optimization mode
converter.optimizations = [tf.lite.Optimize.DEFAULT]

# Pass the representative dataset to help with activation quantization
converter.representative_dataset = representative_data_gen

# Restrict supported operations to INT8
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]

# Ensure input/output tensors are in INT8 format
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8

# Convert and save the quantized model
tflite_model_int8 = converter.convert()
with open("model_int8.tflite", "wb") as f:
f.write(tflite_model_int8)

print("Full Integer Quantized Model Saved: model_int8.tflite")
```

Feature	Full Integer Quantization (INT8)
Weight Precision	8-bit integer
Activation Storage	8-bit integer
Inference Speed	Highest (suitable for Edge TPU & microcontrollers)
Model Size Reduction	~4x smaller than FP32

Table 12.2.: Key Benefits of Full Integer Quantization (INT8)

Full Integer Quantization offers significant advantages in terms of reduced model size and faster inference, making it ideal for deployment on low-power microcontrollers and Edge TPU accelerators. This method quantizes all parameters (weights and activations) to 8-bit integers, optimizing the model for edge devices.[**tensorflowlite:2025**]

12.7. TensorFlow Lite Micro (TFLite Micro)

12.7.1. Description

LiteRT for Microcontrollers is designed to run machine learning models on microcontrollers and other devices with only a few kilobytes of memory.[**litert:2023**] The core runtime just fits in 16 KB on an Arm Cortex M3 and can run many basic models. It doesn't require operating system support, any standard C or C++ libraries, or dynamic memory allocation.[**litert:2023**]

12.7.2. Code Example

Listing 12.5: Integrating TensorFlow Lite with LSM9DS1 Sensor for Gesture Detection

```
#include <TensorFlowLite.h>
#include <Wire.h>
#include <LSM9DS1.h>

// Include the generated model C array
extern "C" {
    #include "model.cc"
}

// Initialize objects
LSM9DS1 imu;
tflite::MicroInterpreter* interpreter;
tflite::Model* model;
tflite::MicroAllocator* allocator;

// TensorFlow Lite setup
void setup() {
    Serial.begin(9600);
    Wire.begin();

    if (!imu.begin()) {
        Serial.println("Failed to initialize IMU sensor!");
        while (1);
    }

    model = tflite::GetModel(model_tflite);
    interpreter = tflite::MicroInterpreter(model, allocator, tflite::
        ↪ kTensorArenaSize);

    Serial.println("Setup complete!");
}

void loop() {
    imu.readAccel();
    float input_data[] = {imu.accelX(), imu.accelY(), imu.accelZ()};

    for (int i = 0; i < 3; i++) {
```

```
            interpreter->input(0)->data.f[i] = input_data[i];
        }

        interpreter->Invoke();
        float* output = interpreter->output(0)->data.f;

        Serial.println(output[0] > 0.5 ? "Gesture Class: 1" : "Gesture Class: 0");
        delay(500);
    }
```

12.7.3. Use of TensorFlow Lite Micro (TFLite Micro) in the Magic Wand Project

TensorFlow Lite Micro (TFLite Micro) is designed specifically for running machine learning models on microcontrollers.[**litert:2023**] Its use in the Magic Wand project is detailed below:

- **Real-Time Inference on Arduino Nano 33 BLE Sense:** TFLite Micro enables the deployment of the lightweight TFLite model on the Arduino Nano 33 BLE Sense. The model is converted into a `C` array during deployment and integrated into the Arduino firmware. Real-time sensor data from the LSM9DS1 (accelerometer, gyroscope, and magnetometer) is processed by the model for gesture classification.[**litert:2023**]
- **Memory and Computational Efficiency:** TFLite Micro is optimized for devices with limited memory (e.g., 256 KB RAM) and processing power. It employs static memory allocation and int8 quantization to ensure smooth execution of the model.[**litert:2023**]
- **Gesture Recognition Flow:** The system follows these steps:
  1. **Input:** Real-time accelerometer data is captured in three dimensions (X, Y, Z).
  2. **Preprocessing:** The data is normalized and formatted to match the input requirements of the model.
  3. **Inference:** The TFLite Micro interpreter processes the input through the model, classifying the gesture as *wing*, *ring*, *slope*, or *unknown*.
  4. **Output:** Based on the classification, the Arduino controls LEDs or sends visual feedback via the serial monitor.
- **Interactivity and Deployment:** TFLite Micro ensures real-time interactivity, allowing the Arduino Nano 33 BLE Sense to respond immediately to gestures. The deployment process involves embedding the model and the TFLite Micro runtime into the Arduino program, creating a portable and efficient system.

Supported Platforms
LiteRT for Microcontrollers is written in C++ 17 and requires a 32-bit platform. It has been tested extensively with many processors based on the Arm Cortex-M Series architecture, and has been ported to other architectures including ESP32. The framework is available as an Arduino library. It can also generate projects for development environments such as Mbed. It is open source and can be included in any C++ 17 project.[**litert:2023**]
The following development boards are supported:

- Arduino Nano 33 BLE Sense
- SparkFun Edge

- STM32F746 Discovery kit
- Adafruit EdgeBadge
- Adafruit LiteRT for Microcontrollers Kit
- Adafruit Circuit Playground Bluefruit
- Espressif ESP32-DevKitC
- Espressif ESP-EYE
- Wio Terminal: ATSAMD51
- Himax WE-I Plus EVB Endpoint AI Development Board
- Synopsys DesignWare ARC EM Software Development Platform
- Sony Spresense

Explore the Examples
Each example application is on GitHub and has a `README.md` file that explains how it can be deployed to its supported platforms.[**litert:2023**] Some examples also have end-to-end tutorials using a specific platform, as given below:

- **Hello World** - Demonstrates the absolute basics of using LiteRT for Microcontrollers.
- **Tutorial using any supported device** - *Micro speech* - Captures audio with a microphone to detect the words "yes" and "no".
- **Tutorial using SparkFun Edge**
- **Person detection** - Captures camera data with an image sensor to detect the presence or absence of a person.

Workflow
The following steps are required to deploy and run a TensorFlow model on a microcontroller:

1. **Train a model:** Generate a small TensorFlow model that can fit your target device and contains supported operations.
2. **Convert to a LiteRT model:** Use the LiteRT converter to convert the model.
3. **Convert to a C byte array:** Use standard tools to store the model in a read-only program memory on the device.
4. **Run inference on the device:** Use the C++ library to run inference and process the results.[**litert:2023**]

Limitations
LiteRT for Microcontrollers is designed for the specific constraints of microcontroller development. If you are working on more powerful devices (for example, an embedded Linux device like the Raspberry Pi), the standard LiteRT framework might be easier to integrate.[**litert:2023**]
The following limitations should be considered:

- Support for a limited subset of TensorFlow operations.
- Support for a limited set of devices.
- Low-level C++ API requiring manual memory management.
- On-device training is not supported.

12.7.4. Comparison of TFLite and TFLite Micro in the Project

Aspect	TFLite	TFLite Micro
Purpose	Optimizes models for mobile/edge devices.	Executes models on microcontrollers.
Use Case in Project	Model conversion, quantization, and testing on desktop.	Real-time inference on Arduino Nano 33 BLE Sense.
Key Features	High-performance library with additional overhead.	Extremely lightweight and microcontroller-friendly.
Programming Context	Python for training and testing.	C++ for microcontroller deployment.

Table 12.3.: Comparison of TFLite and TFLite Micro in the Magic Wand Project.

12.8. Gesture Recognition

As already extensively detailed, the project's main objective is to recognize the gestures. To fully understand its implications and usage surroundings, explaining how the project's scope can be translated into multiple and more complex implementations is necessary. Currently, the board will target three gestures which are trained in the data model and give feedback on the response of these gestures.

12.8.1. Software

In the Previous Software description9.1, Installation steps have been explained comprehensively. In the following codes, some important libraries are shown, which will be required to build the software part of the project. The main packages are from the LSM9DS1 for gesture output detection and tensorflow lite for Microcontrollers to run the trained model**??**.

Listing 12.6: C++ Code for Initializing TensorFlow Lite on Arduino

```
#include <Arduino_LSM9DS1.h>
#include <TensorFlowLite.h>
#include "tensorflow/lite/micro/micro_error_reporter.h"
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/micro_mutable_op_resolver.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"
```

Here is the recap of the Tensorflow lite Micro packages used in the Arduino:

- micro_mutable_op_resolver.h provides the operations used by the interpreter to run the model.
- micro_error_reporter.h outputs debug information.
- micro_interpreter.h contains code to load and run models.
- schema_generated.h contains the schema for the TensorFlow Lite FlatBuffer model file format.
- version.h provides versioning information for the TensorFlow Lite schema.

Upon uploading the modified sketch and trained model through Tensorflow Lite Micro and the Arduino IDE, the board Arduino Nano 33 BLE Sense should exhibit gesture recognition capabilities. The recognized gestures, including wings, rings, and slope, will be displayed in the Serial Monitor of the Arduino editor during the project stage.

12.9. Configuration

After successfully fitting and uploading the model onto the Arduino Nano 33 BLE Sense board, the device operates under typical environmental conditions. The project's success depends on clearly defining the environment and conditions necessary for proper functionality. Understanding the module's surroundings is crucial. In this application, the Arduino Nano 33 BLE Sense board is connected directly to the computer via a Micro-USB cable, as explained in the Hardware section 4.2.1. This connection powers the device during operation and facilitates data exchange with the user, along with essential calibration parameters for image capturing.

12.9.1. Code Structure in Program

In our program, it consists 12 major files. These files are coded under the Arduino IDE Platform.

12.9.2. Magic_Wand.ino - isMoving()

Listing 12.7: C++ Function to Detect Movement Based on Accelerometer Data

```
bool IsMoving() {
    // Look at the most recent accelerometer values.
    const float* input_data = model_input->data.f;
    const float last_x = input_data[input_length - 3];
    const float last_y = input_data[input_length - 2];
    const float last_z = input_data[input_length - 1];

    // Figure out the total amount of acceleration being felt by the device.
    const float last_x_squared = last_x * last_x;
    const float last_y_squared = last_y * last_y;
    const float last_z_squared = last_z * last_z;
    const float acceleration_magnitude =
    sqrtf(last_x_squared + last_y_squared + last_z_squared);

    // Acceleration is in milli-Gs, so normal gravity is 1,000 units.
    const float gravity = 1000.0f;

    // Subtract out gravity to get the actual movement magnitude.
    const float movement = acceleration_magnitude - gravity;

    // How much acceleration is needed before it's considered movement.
    const float movement_threshold = 40.0f;
    const bool is_moving = (movement > movement_threshold);

    return is_moving;
}
```

Here is one of the main functions in the program main file `Magic_Wand.ino`. The above code determines whether a device is in motion based on accelerometer data. The function extracts the latest accelerometer values for the x, y, and z axes, calculates the total acceleration magnitude, and then subtracts the assumed normal gravity (1,000 units) to obtain the actual movement magnitude. The threshold for considering movement is set at 40.0 units, and the function returns true if the calculated movement magnitude exceeds this threshold, indicating that the device is in motion; otherwise, it returns false. It is essential to note that the threshold value may be application-specific and may need adjustment based on the particular context in which this code is employed.

12.9.3. Magic_Wand.ino - RecognizeGestures()

Listing 12.8: Gesture Recognition Using a Pretrained Model in C++

```
// This is the regular function we run to recognize gestures from a pretrained
```

```
    // model.
    void RecognizeGestures() {
        const bool is_moving = IsMoving();

        // Static state used to control the capturing process.
        static int counter = 0;
        static enum {
            ePendingStillness,
            eInStillness,
            ePendingMovement,
            eRecordingGesture
        } state = ePendingStillness;
        static int still_found_time;
        static int gesture_start_time;
        // State machine that controls gathering user input.
        switch (state) {
            case ePendingStillness: {
                if (!is_moving) {
                    still_found_time = counter;
                    state = eInStillness;
                }
            } break;

            case eInStillness: {
                if (is_moving) {
                    state = ePendingStillness;
                } else {
                    const int duration = counter - still_found_time;
                    if (duration > 25) {
                        state = ePendingMovement;
                    }
                }
            } break;

            case ePendingMovement: {
                if (is_moving) {
                    state = eRecordingGesture;
                    gesture_start_time = counter;
                }
            } break;

            case eRecordingGesture: {
                const int recording_time = 128;
                if ((counter - gesture_start_time) > recording_time) {
                    // Run inference, and report any error.
                    TfLiteStatus invoke_status = interpreter->Invoke();
                    if (invoke_status != kTfLiteOk) {
                        TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed on index:
                            ↪  %d\n",
                        begin_index);
                        return;
                    }

                    const float* prediction_scores = interpreter->output(0)->data.f;
                    const int found_gesture = PredictGesture(prediction_scores);

                    // Produce an output
                    HandleOutput(error_reporter, found_gesture);

                    state = ePendingStillness;
                }
            } break;

            default: {
                TF_LITE_REPORT_ERROR(error_reporter, "Logic error - unknown state");
            } break;
        }

        // Increment the timing counter.
        ++counter;
    }
```

The above section would like to define a function named "RecognizeGestures" that implements a state machine to capture and recognize gestures based on accelerometer data using a pretrained model.

- **Initialization of State and Counter:**

  The function begins by initializing some static variables to control the capturing process. A counter keeps track of the number of times the function has been called, and the state is an enumeration representing the different states in the gesture recognition process.

- **Gesture Recognition State Machine:**

  The core of the function is a state machine that manages the gesture recognition process. It includes the following functions:

  – ePendingStillness: Waits for the device to become still.

  – eInStillness: Monitors the duration of stillness and transitions to the pending movement state if the stillness persists.

  – ePendingMovement: Waits for the device to start moving.

  – eRecordingGesture: Records accelerometer data for a specified duration, runs inference using a TensorFlow Lite model and produces an output based on the recognized gesture.

- **State Transitions:** The function transitions between states based on the current state and the status of motion detection (is_moving):

  – From ePendingStillness to eInStillness if the device becomes still.

  – From eInStillness to ePendingMovement if stillness persists for a specified duration.

  – From ePendingMovement to eRecordingGesture if the device starts moving.

- **Gesture Recording and Inference:** When in the eRecordingGesture state, the function records accelerometer data for a fixed duration (recording_time) and then performs inference using a TensorFlow Lite model (interpreter->Invoke()). The result is processed using the PredictGesture function.

- **Output Handling:** The recognized gesture is processed further using the HandleOutput function, and the state is transitioned back to ePendingStillness to wait for the next gesture.

- **Error Handling:** The code includes error handling, reporting an error if the state machine encounters an unknown state.

- **Counter Increment:** The counter is incremented at the end of each function call to keep track of the overall timing.

12.9.4. Magic_Wand.ino - CaptureGestureData()

Listing 12.9: Gesture Data Capture State Machine in C++

```
void CaptureGestureData() {
    const bool is_moving = IsMoving();

    // Static state used to control the capturing process.
    static int counter = 0;
    static int gesture_count = 0;
    static enum {
        eStarting,
        ePendingStillness,
        eInStillness,
        ePendingMovement,
        eRecordingGesture
    } state = eStarting;
    static int still_found_time;
    static int gesture_start_time;
```

```
        static const char* next_gesture = nullptr;
        // State machine that controls gathering user input.
        switch (state) {
            case eStarting: {
                if (!next_gesture || (strcmp(next_gesture, "other") == 0)) {
                    next_gesture = "wing";
                } else if (strcmp(next_gesture, "wing") == 0) {
                    next_gesture = "ring";
                } else if (strcmp(next_gesture, "ring") == 0) {
                    next_gesture = "slope";
                } else {
                    next_gesture = "other";
                }
                TF_LITE_REPORT_ERROR(error_reporter, "# Hold the wand still");
                state = ePendingStillness;
            } break;
        }
    }
```

The CaptureGestureData() function is to capture and process gesture data. This is crucial for the gesture recognition system within the interactive device. Key to its functionality are static variables and a boolean is_moving variable, which together track the device's motion and gesture states. All these states have all mentioned earlier in the RecognizeGestures() function 12.9.3. The key difference is that it gathers training data and does not predict any gesture or record them.

12.9.5. Magic_Wand.ino - loop()

Listing 12.10: Main Loop for Gesture Recognition or Data Capture

```
        void loop() {
            // Attempt to read new data from the accelerometer.
            bool got_data =
            ReadAccelerometer(error_reporter, model_input->data.f, input_length);
            const bool should_capture_data = false;
            if (should_capture_data) {
                CaptureGestureData();
            } else {
                RecognizeGestures();
            }
        }
```

The loop() function serves as the central operational loop for the program. It reads new data from the accelerometer to perform real-time motion tracking. The boolean got_data indicates the success of this data acquisition. Intriguingly, the function includes a conditional should_capture_data, which, although set to false, suggests a dual-mode operation: one for capturing gesture data CaptureGestureData() and another for recognizing gestures RecognizeGestures(). Setting as false value can use the program in the prediction mode (i.e. RecognizeGestures()).

12.9.6. Arduino_acceleometer_handler

Listing 12.11: IMU Data Sampling and Buffer Management in C++

```
            while (IMU.accelerationAvailable()) {
                float x, y, z;
                // Read each sample, removing it from the device's FIFO buffer
                if (!IMU.readAcceleration(x, y, z)) {
                    TF_LITE_REPORT_ERROR(error_reporter, "Failed to read data");
                    break;
                }
                // Throw away this sample unless it's the nth
                if (sample_skip_counter != sample_every_n) {
                    sample_skip_counter += 1;
                    continue;
                }
                const float norm_x = -z;
```

```
            const float norm_y = y;
            const float norm_z = x;
            save_data[begin_index++] = norm_x * 1000;
            save_data[begin_index++] = norm_y * 1000;
            save_data[begin_index++] = norm_z * 1000;
            // Since we took a sample, reset the skip counter
            sample_skip_counter = 1;
            // If we reached the end of the circular buffer, reset
            if (begin_index >= 600) {
                begin_index = 0;
            }
            new_data = true;
        }

        // Skip this round if data is not ready yet
        if (!new_data) {
            return false;
        }

        // Check if we are ready for prediction or still pending more initial
            ↪ data
        if (pending_initial_data && begin_index >= 200) {
            pending_initial_data = false;
        }

        // Return if we don't have enough data
        if (pending_initial_data) {
            return false;
        }

        // Copy the requested number of bytes to the provided input tensor
        for (int i = 0; i < length; ++i) {
            int ring_array_index = begin_index + i - length;
            if (ring_array_index < 0) {
                ring_array_index += 600;
            }
            input[i] = save_data[ring_array_index];
        }
        return true;
    }
```

The Provided code offers a detailed illustration of efficient data acquisition, filtering, and preparation methodologies. Central to this process is the continuous monitoring of acceleration data availability, as evidenced by the while loop querying IMU.accelerationAvailable(). Upon successful data retrieval, indicated by IMU.readAcceleration(x, y, z), the code employs a strategic sampling approach, selectively processing every nth data point. This method, controlled by sample_skip_counter, effectively manages the data rate, a crucial aspect in scenarios with limited processing capabilities or specific data requirements. The transformation and normalization of the data into a different coordinate system, as seen in the assignment of norm_x, norm_y, and norm_z, further exemplify the tailored preprocessing necessary for subsequent analytical stages.
The implementation of a circular buffer, as indicated by the handling of begin_index and the buffer size of 600, showcases an efficient data storage and management strategy, ensuring continuous data flow without overrunning memory limits. This approach is particularly relevant in real-time systems where older data is periodically replaced by newer inputs. The readiness of the data for further processing is meticulously managed through flags like new_data and pending_initial_data, ensuring that the system only proceeds with sufficient data accumulation.

12.9.7. Gesture_Predictor

Listing 12.12: Gesture Prediction Based on Model Output Scores

```
    // Return the result of the last prediction
    // 0: wing("W"), 1: ring("O"), 2: slope("angle"), 3: unknown
    int PredictGesture(const float* prediction_scores) {
        int max_prediction_index = -1;
```

```
            float max_prediction_score = 0.0f;
            for (int i = 0; i < kGestureCount; i++) {
                const float prediction_score = prediction_scores[i];
                if ((max_prediction_index == -1) ||
                (prediction_score > max_prediction_score)) {
                    max_prediction_score = prediction_score;
                    max_prediction_index = i;
                }
            }

            int found_gesture;
            if ((max_prediction_index == kNoGesture) ||
            (max_prediction_score < kDetectionThreshold)) {
                found_gesture = kNoGesture;
            } else {
                found_gesture = max_prediction_index;
            }

            return found_gesture;
        }
```

The above code analyzes and predicts gestures based on an array of prediction scores. Each score in the array corresponds to a different gesture, and the function's role is to determine which gesture has the highest likelihood of being correct. It operates by iterating through the array of scores, returning numeric values 0, 1, 2 and 3 for Wing, Ring, Slope and unknown respectively and maintaining a record of the highest score encountered and its associated index. This is achieved through a for-loop that compares each score with the current maximum; if a higher score is found, the function updates the maximum score and its corresponding index.
Initially the "prediction scores" obtained from the main is passed on to the function in which a "maxPredictionScore" and "maxPrediction Index". These values above are then compared to "kNoGesture" and "kDetectionThrehsold", then the value of found gesture variable is found which is equal to max prediction index.

12.9.8. Arduino_Output_Handler

Listing 12.13: LED Control Based on Gesture Recognition

```
const int ledPinRed = 22;
const int ledPinGreen = 23;
const int ledPinBlue = 24;

void LightUpRGB(int kind) {
    digitalWrite(ledPinRed, HIGH);
    digitalWrite(ledPinGreen, HIGH);
    digitalWrite(ledPinBlue, HIGH);

    switch (kind) {
        case 0: // W
        digitalWrite(ledPinRed, LOW);
        break;
        case 1: // O
        digitalWrite(ledPinGreen, LOW);
        break;
        case 2: // L
        digitalWrite(ledPinBlue, LOW);
        break;
        default:
        break;
    }
}

void HandleOutput(tflite::ErrorReporter* error_reporter, int kind) {
    // The first time this method runs, set up our LED
    static bool is_initialized = false;
    if (!is_initialized) {
        pinMode(LED_BUILTIN, OUTPUT);
        pinMode(ledPinRed, OUTPUT);
        pinMode(ledPinGreen, OUTPUT);
        pinMode(ledPinBlue, OUTPUT);
```

```
            is_initialized = true;
        }
        // Toggle the LED every time an inference is performed
        static int count = 0;
        ++count;
        if (count & 1) {
            digitalWrite(LED_BUILTIN, HIGH);
        } else {
            digitalWrite(LED_BUILTIN, LOW);
        }

        LightUpRGB(kind);
    }
```

- **Variable Declarations:** "ledPinRed, ledPinGreen, ledPinBlue:" These are constants representing the pin numbers to which the red, green, and blue LEDs are connected. The values 22, 23, and 24 indicate the specific GPIO (General Purpose Input/Output) pins used.

- **Function LightUpRGB:** This function takes an integer kind as a parameter and controls the RGB LEDs based on the value of kind. Initially, it turns all LEDs (red, green, blue) on. The switch statement then determines which LED to turn off based on the value of kind (0 for red, 1 for green, 2 for blue).

- **HandleOutput:** This function is designed to handle the output based on an input kind. It initializes the LED pins only once, as indicated by the is_initialized static variable. It toggles the built-in LED on the microcontroller on each call to indicate that an inference (or some processing) has been performed. Finally, it calls LightUpRGB(kind) to light up the appropriate LED based on the kind value.

Overall, this function serves as a state machine that captures accelerometer data, recognizes gestures using a pretrained TensorFlow Lite model, and handles the output based on the recognized gestures. The specific logic and parameters, such as stillness duration and recording time, can be adjusted based on the requirements of the application.

12.9.9. Upload the code to the board

Lastly, referring back to the Arduino IDE Configuration part9.1.5, the program will be verified and uploaded to the board through the Arduino IDE.

12.10. Tests

It is clear that after all the previous steps, some parameters need to be defined for the project functionality. This subsection's objective is to present the result from the implementation on one part: SW. It is important to specify this feature to understand how the system will be under corroboration in the coming steps.

12.10.1. Common things to consider during Deployment test

The reliability of the neural network to recognize the digit increases with the increase in the training data. The neural network learns to abstract to the general rule and recognizes various aspects of the gestures, such as the speed of the movement and other factors.

12.10.2. Software tests

After creating Python programs for loading and processing images, it should be tested that the intended results are achieved. Also, after fitting the model, the performance of the CNN needs to be checked to see whether it is returning the desired results. Moreover, some negative test scenarios should be tested with the help of gestures apart from the wings, rings and slope.

12.11. Monitoring stage during deployment

After the test, outputs from the board may not be accurate enough. At this moment, we should decide to modify the data set or add fresh training data. Therefore, we should monitor the model's behavior to observe and record it. Alternatively, whenever a database suffers modifications due to environment changes, it is called Monitoring. The data set for this project is unchanged mainly because it only trains three gestures, and the remaining gestures are marked as unknown if they do not belong to the three trained gestures. In order to ensure that the model is robust enough since its first development, trained models may be changed to create several versions.
The goal of this stage is to ensure that the knowledge extracted from the data is being used to make informed decisions and improve the system's overall performance. During this stage, the following activities are performed:

- **Performance monitoring:** This involves keeping a close eye on the effectiveness of the model created to interpret the data from the meters in order to make sure they are producing precise and pertinent findings.
- **Maintenance:** This entails maintaining the models and the database with fresh data and ensuring their functionality. In this particular case, since the database is "numbers," this is a very unusual case since numbers are universal.
- **Evaluation:** Finding any odd or unexpected trends in the meter data, such as sudden spikes or dips in use, is a part of this process.
- **Feedback:** This involves setting up automated alerts or notifications to let consumers know when particular criteria are satisfied, such as when the use reaches a specific threshold or a leak is discovered.
- **Continuous improvement:** To enhance the performance of the models and the whole KDD process, this entails regularly monitoring the system and making the appropriate adjustments.

The KDD process can be enhanced over time by continuously monitoring the models and the data to spot any flaws, inconsistencies, or outliers in the data. It also gives the system's users valuable insights to make better-educated decisions.

13. Software Tests

The documentation for the magic wand project encompasses detailed explanations of each module, function, and class. It provides insights into the purpose, functionality, and importance of different components. The documentation also includes test classes and methods to ensure the correctness and reliability of the implemented functionalities. The clear organization of the project into modules promotes modularity and code maintainability.
This comprehensive documentation serves as a valuable resource for project understanding, collaboration, and future development. It enhances transparency, allowing developers and stakeholders to grasp the project's architecture and functionality effectively.

13.1. Functions

- **time_wrapping(original_data, factor, num_frames)**

  The time_wrapping function takes in an original_data list, a time-wrapping factor, and the number of resulting num_frames. It performs a time-wrapping operation on the input data, creating a modified dataset with a different temporal structure. This function is utilized in the data augmentation module to introduce temporal variations in the input data, contributing to the robustness of the trained model.

- **augment_data(original_data, original_label)**

  The augment_data function performs data augmentation by generating variations of the input original_data and corresponding original_label. It creates a more diverse dataset by applying operations such as time-wrapping. The augmented data is essential for improving the model's ability to generalize to various input scenarios.

- **setUp()**

  The setUp method is part of the test classes and is executed before each test case. It initializes necessary components for testing, such as data loaders or test-specific variables, ensuring a consistent and controlled environment for testing.

- **test_time_wrapping()**

  The test_time_wrapping method is a unit test for the time_wrapping function. It checks whether the time-wrapping operation produces the expected number of frames and maintains the structure of the input data. This test ensures the correctness of the data augmentation technique.

- **test_augment_data()**

  The test_augment_data method is a unit test for the augment_data function. It verifies that the augmented data has the expected length, type, and relationship with the original data and labels. This test is crucial to ensure the effectiveness of the data augmentation process.

13.2. Classes

- **TestAugmentation**

  The TestAugmentation class is a unittest class containing test methods for the data augmentation module. It tests the functionality of the data augmentation techniques, including time-wrapping and overall data augmentation.

- **TestLoad**

  The TestLoad class is a unittest class for testing the data loading module. It checks the correct initialization of data loaders and ensures that the loaded data has the expected structure and length.

- **TestPrepare**

  The TestPrepare class is a unittest class focusing on the correctness of the dataset preparation process. It tests functions related to preparing and writing data, including generating negative data to enhance dataset diversity.

- **TestSplit**

  The TestSplit class is a unittest class that validates the dataset splitting functionality. It ensures that the dataset is correctly split into training, validation, and testing sets according to specified ratios.

- **TestTrain**

  The TestTrain class is a unittest class dedicated to testing the model training process. It covers loading data, building neural network models (CNN and LSTM), and reshaping input data to ensure compatibility with the models.

13.3. Test Files

13.3.1. Unit Tests

One of the principles of **agile** development (although not exactly being the case for our study) is that testing should be tightly integrated with development, and programmers should write tests for their own code [**Ousterhout:2018**].

- unit tests
  - most often written by developers
  - small and focused
  - are often run in conjunction with a test coverage tool[1]
    - Coverage.py
    - pytest-cov

When writing new code or modifying existing code, it is essential to update the corresponding unit tests to ensure continued code functionality and test coverage [**Ousterhout:2018**].
Unit tests facilitate refactoring [**Ousterhout:2018**]. Without a test suite

- It would be dangerous to make major structural changes to a system.
- bugs will go undetected until the new code is deployed.
  - much more expensive to find and fix

With a good set of tests, developers can be more confident when refactoring because the test suite will find most bugs that are introduced [**Ousterhout:2018**].

[1] ensures that every line of code in the application is tested.

13.3.2. `dataaugmentationtest.py`

This test code is responsible for verifying the functionality of the data augmentation process. Data augmentation involves creating variations of the existing dataset to enhance the model's ability to generalize. In the context of the magic wand project, it likely applies transformations to the gesture data such as rotation, scaling, or translation to create a more robust training set.
To ensure that the data augmentation techniques are correctly implemented and producing the expected variations in the input data. This helps in improving the model's performance by exposing it to diverse examples of each gesture.
`dataaugmentationtest.py`

- This test module validates the functionality of the data augmentation module used in the magic wand project. It specifically tests the **time_wrapping** and **augment_data** functions.
- **test_time_wrapping**: Tests if the time wrapping function (time_wrapping) produces the expected output dimensions and values.
- **test_augment_data**: Checks if the data augmentation function (**augment_data**) appropriately increases the dataset size and maintains the correct structure.

13.3.3. `dataloadtest.py`

This test code focuses on the loading of the dataset. It checks whether the dataset, which consists of gesture data (W, L, O), can be successfully loaded into the program. This includes reading image files, processing them, and organizing them for training or testing.
To validate the functionality of the data loading process, ensuring that the code can access and organize the dataset correctly before feeding it into the machine learning model.
`dataloadtest.py`

- This module focuses on testing the data loading functionality of the project using the **DataLoader** class. It ensures that the training, validation, and test datasets are loaded correctly.
- **test_get_data:** Validates the structure and lengths of the loaded training, validation, and test datasets.
- **test_pad:** Tests the padding functionality, ensuring it pads sequences correctly.
- **test_format:** Verifies that the data is formatted as expected, and TensorFlow datasets are created.

13.3.4. `datapreparetest.py`

The purpose of this test code is to verify the preparation of data before the training process. This may involve tasks such as normalization, resizing, or any other preprocessing steps required to make the data suitable for training.
To confirm that the data is preprocessed correctly and is in the desired format for training the machine learning model.
`datapreparetest.py`

- This test module ensures the correctness of the dataset preparation process, including the creation of negative data and writing data to files.

- **test_prepare_data:** Checks if the original data is prepared correctly and matches the expected format.
- **test_generate_negative:** Verifies the generation of negative data.
- **test_write_data:** Tests if the data is correctly written to files and can be read back successfully.

13.3.5. `datasplittest.py`

This test code is likely responsible for splitting the dataset into training, validation, and testing sets. It checks if the division is done correctly and ensures that each subset contains a representative distribution of the gestures.
To guarantee that the dataset is appropriately divided, which is crucial for assessing the model's performance during training and testing phases.
`datasplittest.py`

- This test module validates the dataset splitting functionality, ensuring that the data is divided into training, validation, and test sets as intended.
- **test_read_data:** Checks if the data is read correctly.
- **test_split_data:** Tests different scenarios of data splitting and verifies the lengths of the resulting datasets.

13.3.6. `traintest.py`

The primary purpose of this test code is to validate the training process of the machine learning model. It includes setting up the model, feeding it with the training data, and monitoring its performance during training epochs.
To ensure that the training process is functioning as expected, allowing the model to learn from the training data and adjust its parameters to make accurate predictions for the gestures.
`traintest.py`

- This module focuses on testing the training process of the machine learning model. It includes building the neural network models, loading data, and reshaping the input.
- test_load_data: Validates the loading of training, validation, and test data as TensorFlow datasets.
- test_build_net: Tests the construction of CNN and LSTM models, ensuring the correct output shapes.

  test_reshape_function: Checks if the reshape function is correctly transforming the input data.

These test codes collectively contribute to the robustness and reliability of the magic wand project by validating different components of the machine learning pipeline.

13.4. Pytest Automation for Magic Wand Project Testing

Pytest is a testing framework that simplifies the process of writing and executing test code in Python. It allows you to create test functions, organize them into test modules, and run tests efficiently. Below is an overview of how you can use Pytest to automate the testing of your Magic Wand project.

13.4.1. Install Pytest

Listing 13.1: Installing Pytest Using pip

```
pip install pytest
```

13.4.2. Folder Structure

Organize your test files into a folder named tests in your project directory. The file names should start with test to be discovered by pytest.

```
|-- magic_wand_project
|-- Modules/
|   |-- dataaugmentation.py
|   |-- dataload.py
|   |-- dataprepare.py
|   |-- datasplit.py
|   |-- train.py
|-- Tests/
|   |-- dataaugmentationtest.py
|   |-- dataloadtest.py
|   |-- datapreparetest.py
|   |-- datasplittest.py
|   |-- traintest.py
|-- handleErrors/
|   |-- errorHandler.py
```

13.4.3. Writing Test Functions

In each of your test files (e.g., dataaugmentationtest.py, dataloadtest.py, etc.), write test functions using the pytest naming convention (starting with test).

Listing 13.2: Example of Test Functions for Data Augmentation

```
# Example: test_dataaugmentation.py

def test_time_wrapping():
# Test logic ...

def test_augment_data():
# Test logic ...
```

13.4.4. Run Tests with Pytest

Open a terminal in the project directory and run:
pytest
pytest will automatically discover and execute all the test functions in the tests directory.

- **Pytest Command-Line Options**
  - Run specific test file:

    Listing 13.3: Running a Test for Data Augmentation with Pytest

    ```
    pytest tests/dataaugmentationtest.py
    ```

  - Run specific test function in a file:

Listing 13.4: Running a Specific Test with Pytest

```
pytest tests/dataaugmentationtest.py::
    ↪ test_time_wrapping
```

– Show detailed output:

Listing 13.5: Running Pytest with Verbose Output

```
pytest -v
```

- **Automation Script**

  To run all the tests together, you can create a simple automation script. For example, create a file named run_tests.sh:

Listing 13.6: Bash Script for Running Pytest in a Specific Directory

```
#!/bin/bash

# Navigate to the project directory
cd Documents/MagicWand/
ML23-06-Magic-Wand-with-an-Arduino-Nano-33-BLE-sense/
Sourcecode/Code/Datatraining

# Activate virtual environment if needed
# source venv/bin/activate

# Run pytest
pytest
```

- **Make the script executable**

Listing 13.7: Setting Executable Permission and Running the Script

```
chmod +x run_tests.sh

# Run the script
./run_tests.sh
```

13.4.5. Execution

The result of running `pytest` in the `Command Prompt` in the directory of the test files is shown in Figure 13.1.

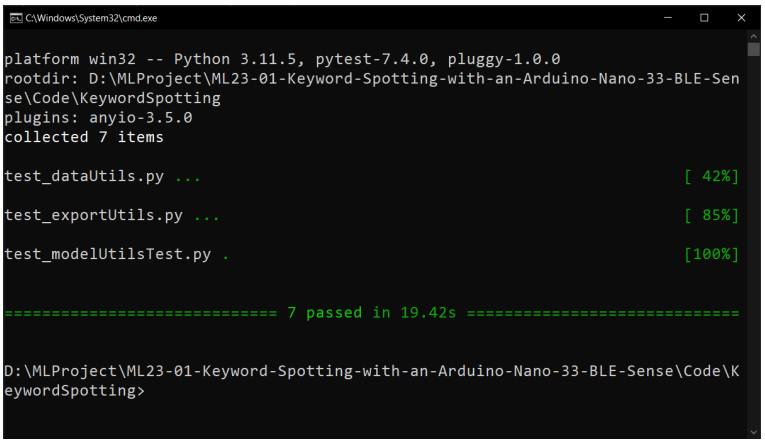


Figure 13.1.: Result of running `pytest` in the direcotry of the test files in `Command Prompt`

```
collected 7 items
```

This line indicates that `pytest` found and collected a total of 7 test items. These items represent individual test files or modules in the project. Note that the name of the files has been changed, starting with "`test_`" so that `pytest` could automatically find them.

```
test_dataUtils.py ...                [ 42%]
test_exportUtils.py ...               [ 85%]
test_modelUtilst.py .                [100%]
```

Each line corresponds to the progress of test execution for a specific test file. Dots (`...` and `.`) represent successful tests. For the case of `...`, three tests were successful, each dot representing a successful test. The percentage values ([42%], [85%], [100%]) indicate the progress through the entire test suite.

```
7 passed in 28.32s
```

This final summary line provides an overview of the test results. It states that out of the 7 tests executed, all 7 passed successfully. The total execution time for all tests was 28.32 seconds. The [100%] success rate indicates that all the tests you ran have passed.

14. Bill of Materials

14.1. Material List and Hardware Bill of Materials

Component list required for the Project are :

- Arduino Nano 33 BLE Lite Sense Board
- USB Cable
- Laptop
- Mouse
- Wand/ Stick
- Sticky Tape

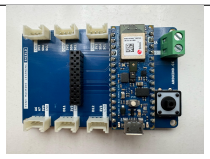
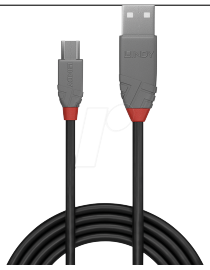
Hardware	Description	Quantity	Link	Price
	Arduino Nano board with 33 BLE Lite support and integrated sensors	1	Arduino	36,60 €
	USB Cable 3.0 fast synchronisation	1	USBCable	2,50 €
	Sticky tape	1	Tape	4,49 €
	Wand	1	Wand	3,18 €
	HP 816F9EA Notebook/Laptop	1	Laptop	629,00 €
	Mouse	1	Mouse	10,95 €

Table 14.1.: Hardware Bill of Materials

The hardware bill of materials table 14.1 outlines the essential components required for the successful execution of the project Magic Wand, each serving a specific purpose in enabling functionality and interaction.

15. Software Requirement and Bill Of Materials

15.1. Software Requirement and Bill Of Materials

Software	Version	License	Dependencies	Libraries
Arduino IDE	2.1.2	Open-source (available under General Public License)	Java Runtime Environment (JRE) to run	ArduinoBLE and ArduinoLSM9DS1
Python	3.11.3	Open-source (Python Software Foundation License)	doesn't have strict dependencies	NumPy, Pandas and Matplotlib
Tensor Flow	2.15.0	Open-source (Apache License 2.0)	Hardware libraries	TensorFlow Lite Micro
Pycharm	2023.1.2 (Community Edition)	Open-source (Apache License 2.0)	Java Runtime Environment	Python Interpreter and Git Integration

Table 15.1.: Software Requirements

Table **??**, titled Software Requirements, outlines the essential tools and environments required for the effective development and implementation of the project. All the listed software is compatible with Windows, macOS (Mac), and Linux operating systems.

1. **Arduino IDE on PC**

   The Arduino IDE serves as the primary programming interface for the Arduino Nano 33 BLE Sense Board. This open-source, official Arduino software enables users to edit, upload, and compile code for Arduino modules. Built on the Java platform, the IDE supports various Arduino modules and is compatible with both C and C++ programming languages. The Arduino Nano 33 BLE Sense relies on the Arduino IDE, which is the most commonly used development environment for Arduino boards and can be utilized both online and offline [**Arduino:2015**]. Different versions of the software are available for macOS, Linux, and Windows operating systems. Users can access these software packages through the following link: `https://www.arduino.cc/en/software`. Notably, during installation, it is crucial to ensure that the necessary drivers for the Arduino Nano 33 BLE Sense Board are also installed [FAD18].

2. **Python**

   The project's coding requirements are fulfilled using Python, a highly versatile programming language distributed under the Python Software Foundation License. To streamline dependency management, it is advisable to establish a virtual environment [**Python:2021**].

   Table 15.2 outlines the key Python packages utilized in the project, detailing their versions, licenses, and dependencies. The "Dependencies" column specifies

the required dependencies for each package, including any version constraints. For a comprehensive list of the packages, refer to the `Requirements.txt` file.

3. **Tensorflow**

   TensorFlow, an open-source library developed by Google, is primarily designed for deep learning but also supports machine learning applications. It processes data in multi-dimensional arrays, making it highly efficient for managing large datasets. TensorFlow may have hardware dependencies, and utilizing GPU support requires specific configurations. The library provides APIs for Python and C++ and can also be integrated with R and Java. One of TensorFlow's key advantages is its compatibility with both GPUs and CPUs, enabling versatile performance optimization. Licensed under the Apache License 2.0, users should carefully review hardware compatibility and configuration requirements as detailed in TensorFlow's documentation [**Goldsborough:2016**].

Package	**Version**	**License**	**Dependencies**
Python	2.9	Open Source (PSF)	-
tensorflow	2.15.0	Open Source (Apache 2.0)	numpy (>=1.16.6) gast (=0.3.3) keras-preprocessing (=1.1.0) keras-applications (=1.0.8) tensorboard (=2.6.0) termcolor (>=1.1.0) wrapt (>=1.11.1) absl-py (>=0.7.0) grpcio (>=1.8.6) astunparse (=1.6.3) google-pasta (=0.2.0) h5py (>=2.9.0) opt-einsum (=3.3.0) protobuf (>=3.12.2)
numpy	1.26.2	Open Source (BSD)	-
Matplotlib	3.8.2	Open Source (PSF)	cycler (>=0.10.0) kiwisolver (>=1.0.1) numpy (>=1.15) Pillow (>=6.2.0) pyparsing (>=2.0.3) six (>=1.5)
Pandas	2.1.4	Open Source (BSD)	numpy (>=1.16.6) python-dateutil (>=2.7.3) pytz (>=2015.7)
six	1.16.0	MIT	-

Table 15.2.: Python packages and dependencies

- **Python 3.9:**
  - Python is the programming language in which your project is written. Python 3.9 is a stable and widely used version with numerous improvements and features compared to Python 2. Using the latest version

is recommended for compatibility, security, and access to the latest language features.

- **TensorFlow 2.7.0:**
  - TensorFlow is a popular open-source machine learning library. In your project, it's likely used for developing and training machine learning models for keyword spotting. TensorFlow provides tools for building neural networks and processing audio data, which are essential for speech-related tasks.
    - Note that `TensorFlow Lite` is used to facilitate the deployment of machine learning models on edge devices with limited resources. The framework provides tools like the TensorFlow Lite Converter, which optimizes models for memory-constrained devices, and supports quantization to reduce model size and improve execution speed on CPUs [**Goldsborough:2016**].
- **NumPy 1.26.2:**
  - NumPy is a fundamental library for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays. In your project, NumPy may be used for efficient handling and manipulation of numerical data, which is common in machine learning tasks.
- **Matplotlib 3.8.2:**
  - Matplotlib is a plotting library for creating visualizations in Python. In the context of your project, it may be used for visualizing data, training progress, or results. For instance, you might use Matplotlib to create graphs or spectrograms of audio data.
- **six 1.16.0:**
  - The `six` library is used for writing code that is compatible with both Python 2 and Python 3. However, since you are using Python 3.9, the need for `six` might be limited. It's possible that it's used in parts of the codebase that were originally designed to be compatible with Python 2.
- **Pandas 1.3.2:**
  - Pandas is a powerful data manipulation and analysis library. It provides data structures like DataFrames that are useful for handling structured data. In your project, Pandas might be used for organizing and preprocessing data before training machine learning models.

4. **Pycharm**

   PyCharm offers a robust development environment for Python programming. It requires a valid Java Runtime Environment (JRE) and can be installed by downloading it from the JetBrains website. The Community Edition is free, open-source, and distributed under the Apache License 2.0. Proper configuration of PyCharm to use the Python interpreter from the project's virtual environment is crucial for optimal functionality [**JetBrains:2023**].

Collectively, these software components constitute a comprehensive and well-integrated toolset for the successful realization of the project.

Hardware	Description	Quantity
	Laptop for software and final output	1
	Wireless bluetooth mouse	1

Table 15.3.: Supplements

The supplementary hardware table 15.3 enhances the project by incorporating devices that support user interaction and software management.
The laptop serves as the central hub for software development and the primary output display. It plays a pivotal role in programming the Arduino Nano, debugging, and real-time performance monitoring. Additionally, a wireless Bluetooth mouse complements the laptop by providing precise control and navigation, ensuring a seamless and user-friendly experience during interactions with the system.
`requirements.txt`

Part V.

Verification - Evaluation - Conclusion

16. Monitoring

16.1. Monitoring

Monitoring an ML pipeline ensures its continuous performance, reliability, and compliance. This document provides a comprehensive plan covering data acquisition, pipeline updates, testing, privacy, robustness, and end-to-end processes.The "Monitoring" part of the Magic Wand undertaking involves the continuous observation and analysis of sensor data obtained from the Arduino Nano 33 BLE Sense board. This data includes inputs from various onboard sensors, such as the accelerometer, gyroscope, temperature sensor, microphone, and potentially additional external sensors connected to the board.

16.2. Plan for Monitoring

16.2.1. Real-Time Monitoring

- **System Metrics:** Monitor CPU, memory, and latency.
- **Model Metrics:** Track accuracy, precision, recall, and F1-score.
- **Data Metrics:** Validate schema, detect missing values, and outliers.

16.2.2. Historical Analysis

Store logs and metrics to analyze long-term trends.

16.2.3. Alerting Mechanisms

Integrate alert systems using tools like PagerDuty or Slack to notify when thresholds are breached.

16.3. Incorporating New Data

16.3.1. Data Acquisition

Automate data ingestion using APIs or streaming systems like Kafka.

16.3.2. Incremental Updates

Schedule periodic or event-triggered data updates.

16.3.3. Data Storage

Use version-controlled repositories such as DVC for reproducibility.

16.4. Data Update in ML Pipeline

16.4.1. Triggering Retraining

Retrain based on time schedules or data drift detection.

16.4.2. Automation

Leverage CI/CD tools (e.g., Kubeflow, MLflow) for automating retraining and deployment.

16.4.3. Validation

Deploy shadow models to validate performance on live data.

16.5. Checks and Tests

16.5.1. Data Checks

- Schema Validation: Ensure correct formats.
- Anomaly Detection: Identify outliers using statistical methods.

16.5.2. Model Checks

- Unit Tests: Test individual pipeline components.
- Integration Tests: Verify end-to-end functionality.

16.5.3. Monitoring Drift

Track changes in data distribution and concept drift using tools like `evidently`.

16.6. Code Functions

16.6.1. Data Drift Detection

Listing 16.1: Detecting Data Drift Using the Kolmogorov-Smirnov Test

```
from scipy.stats import ks_2samp

def detect_data_drift(reference_data, new_data, feature):
stat, p_value = ks_2samp(reference_data[feature], new_data[feature])
return p_value < 0.05  # True if drift detected
```

16.6.2. Model Performance Monitoring

Listing 16.2: Evaluating Model Performance Using Common Metrics

```
from sklearn.metrics import accuracy_score, precision_score, recall_score

def evaluate_model_performance(model, X_test, y_test):
predictions = model.predict(X_test)
metrics = {
    'accuracy': accuracy_score(y_test, predictions),
    'precision': precision_score(y_test, predictions, average='weighted'),
    'recall': recall_score(y_test, predictions, average='weighted')
}
return metrics
```

16.6.3. Automating Retraining

Listing 16.3: Retraining a Model Using an External Python Script

```
import subprocess

def retrain_model(script_path):
result = subprocess.run(["python", script_path], capture_output=True, text=True)
if result.returncode == 0:
print("Retraining successful.")
else:
print(f"Retraining failed: {result.stderr}")
```

16.7. Privacy Considerations

16.7.1. Privacy-by-Design

- Anonymize sensitive data by removing Personally Identifiable Information (PII).
- Use pseudonymization and encryption.

16.7.2. Differential Privacy

Use techniques like TensorFlow Privacy to add noise and safeguard individual data.

16.8. Robustness Strategies

16.8.1. Adversarial Testing

Simulate edge cases like corrupted data using libraries like `cleverhans`.

16.8.2. Fault Tolerance

Implement retry mechanisms and fallbacks for critical pipeline components.

16.9. End-to-End Process

16.9.1. Monitoring Dashboard

Develop dashboards using tools like Grafana or Power BI for centralized metric visualization.

16.9.2. Logging and Alerts

Use logging frameworks (e.g., ELK stack) and define thresholds for automated notifications.

16.9.3. Iterative Improvement

Analyze logs and metrics to refine pipeline components continuously.
This plan outlines a holistic approach to monitoring ML pipelines, ensuring robustness, reliability, and compliance while enabling adaptability in dynamic environments.

17. Evaluation and Verification

Once we get the model ready, it is important to test the model with a different dataset. This helps us confirm our model will work fine with gestures from other persons as well. To test the model we need to now call the Keras's model.evaluate() function.[WS20a] We can also use a confusion matrix to check the performance of the model. An example of a confusion matrix according to the [WS20a] is shown below which is calculated by tf.math.confusion_matrix() function:

$$\text{tf.Tensor}\left(\begin{bmatrix} 75 & 3 & 0 & 4 \\ 0 & 69 & 0 & 15 \\ 0 & 0 & 85 & 3 \\ 0 & 0 & 1 & 129 \end{bmatrix}, \text{shape} = (4,4), \text{dtype} = \text{int32}\right)$$

The confusion matrix function assesses the accuracy of the predicted class for each input in the test dataset by comparing it to the actual class. It reveals the model's limitations, pinpointing areas that need improvement. By leveraging this feedback, we can adjust the dataset and retrain the model, leading to continuous performance enhancement as the model learns from its mistakes [WS20a].

18. Results

After considerable effort in building a comprehensive dataset and training the model, the Magic Wand project, powered by the Arduino Nano BLE 33 sensor, demonstrated promising results during implementation. The gesture recognition system accurately identified and responded to predefined gestures, specifically W (Wing), O (Ring), and L (Slope). This successful integration of TinyML with the Arduino platform highlights the potential for real-time, edge-based gesture recognition in interactive applications, marking a significant step forward in utilizing machine learning for intuitive control systems.

- **Gesture Recognition:**

  The system demonstrated a strong capability in recognizing predefined gestures: W (Wing) 18.1, O (Ring) 18.3, and L (Slope) 18.5. By adding an "Unknown" category for unrecognized movements, the project gained greater flexibility and adaptability to different gestures.

- **LED Feedback:**

  Real-time visual feedback through LED lights in red **??**, green **??**, blue **??**, and white **??** reinforced the accuracy of gesture recognition. This feedback not only improved the user experience but also acted as a clear indicator of system performance.

- **Challenges Overcome:**

  The development journey wasn't without its hurdles. Initially, the model struggled with precise gesture recognition. However, these issues were addressed by consistently refining data collection methods, improving model training processes, and optimizing the code for better performance.

- **Iterative Refinement:**

  The project's success can largely be attributed to its iterative approach. Through repeated adjustments to the dataset, careful model tuning, and code improvements, the project achieved significant progress in gesture recognition accuracy.

- **Accomplishments:**

  Despite facing early challenges, the Magic Wand project is now a successful example of integrating hardware with machine learning to enable gesture-based interactions. The project demonstrates the potential for real-time, accurate gesture recognition using TinyML.

- **Visual Representation:**

  Enclosed are images that capture moments when recognized gestures are triggered, providing a visual insight into the successful implementation of the project.

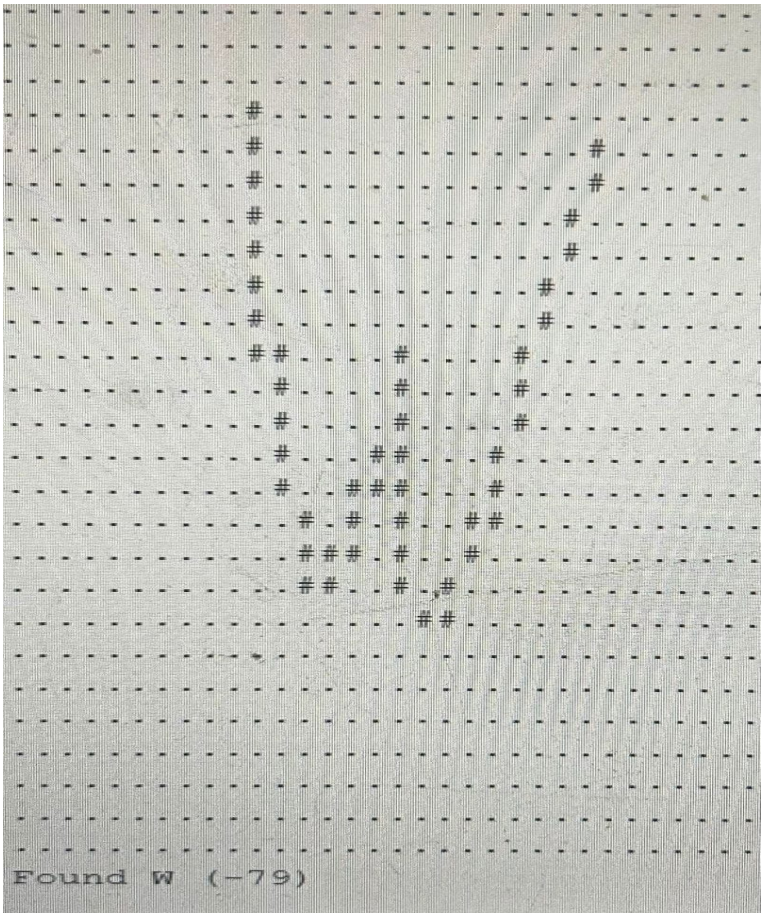


Figure 18.1.: Gesture Output for WING **W** on Output Terminal

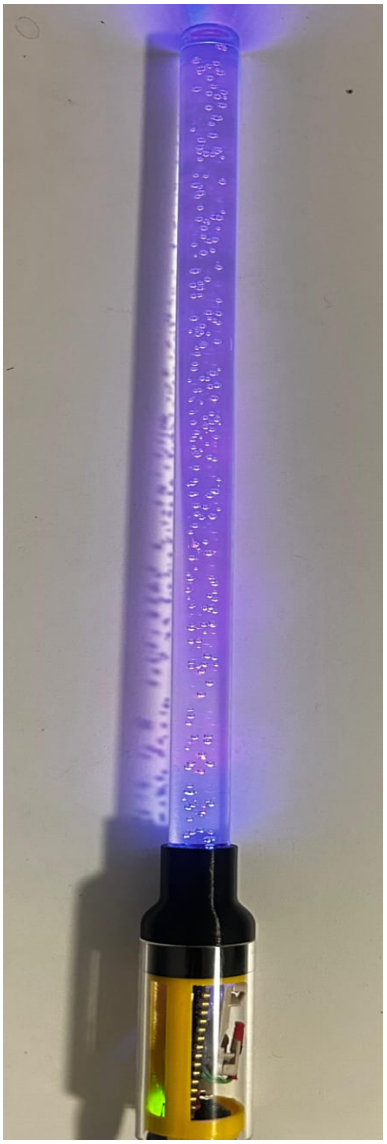


Figure 18.2.: RED Colour Output for WING **W** on Magic Wand

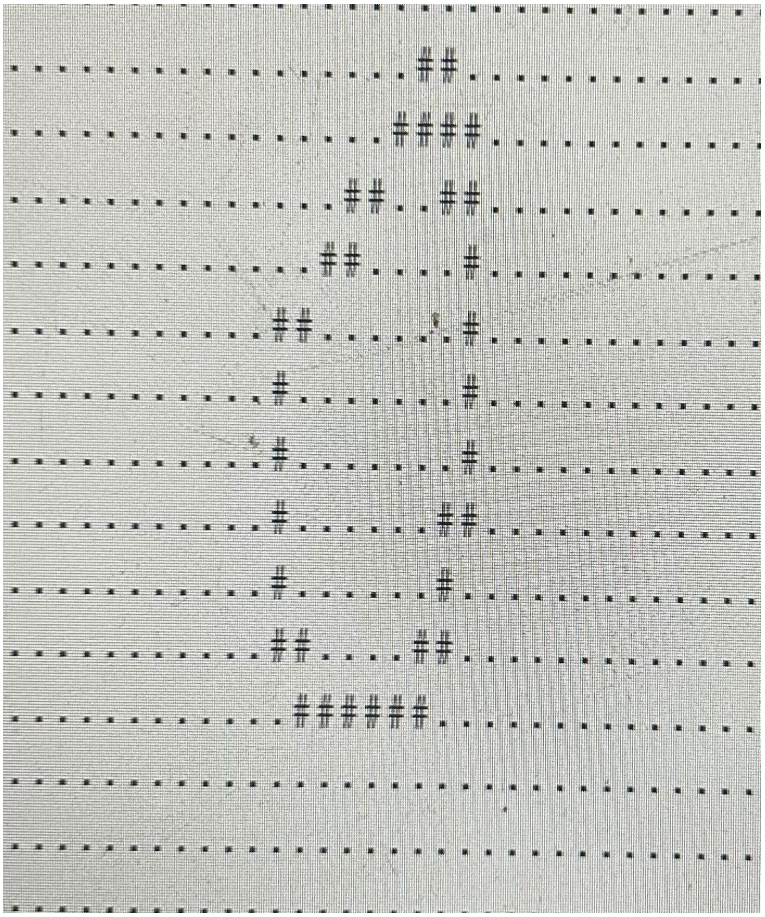


Figure 18.3.: Gesture Output for RING **O** on Output Terminal

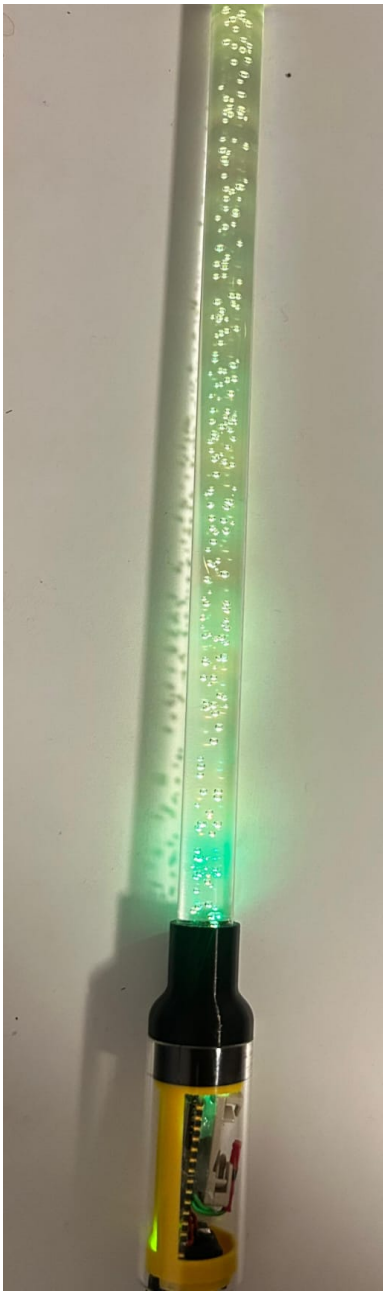


Figure 18.4.: GREEN Colour Output for RING **O** on Magic Wand

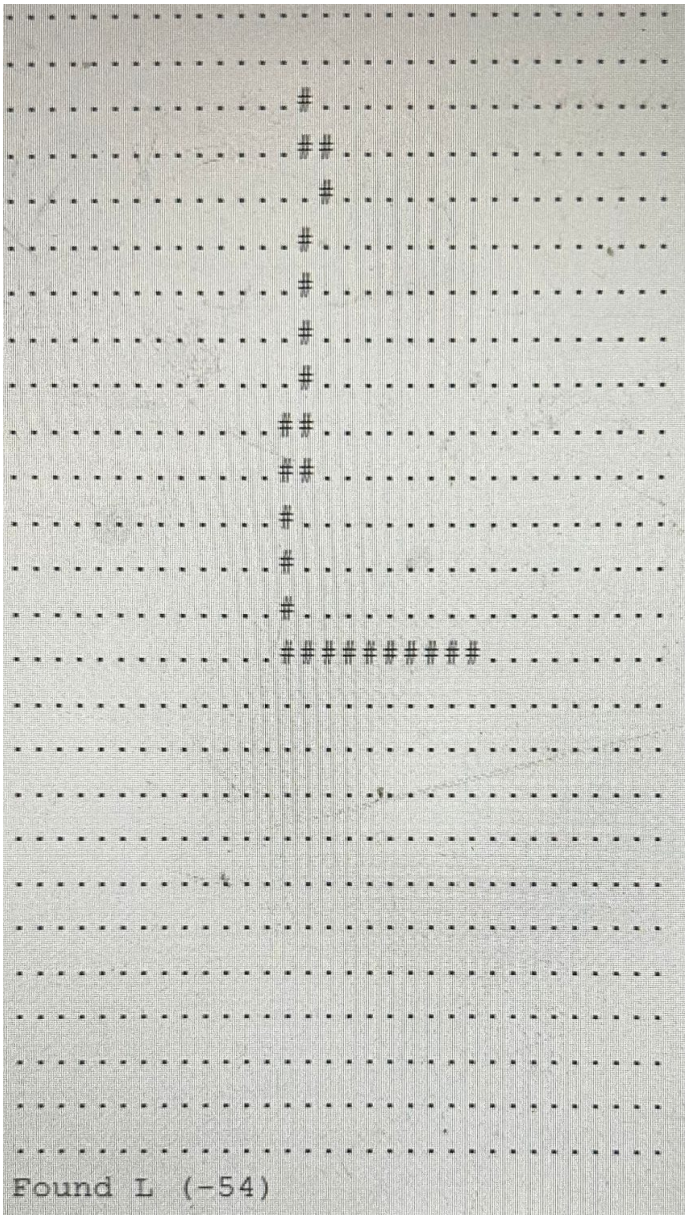


Figure 18.5.: Gesture Output for SLOPE **L** on Output Terminal

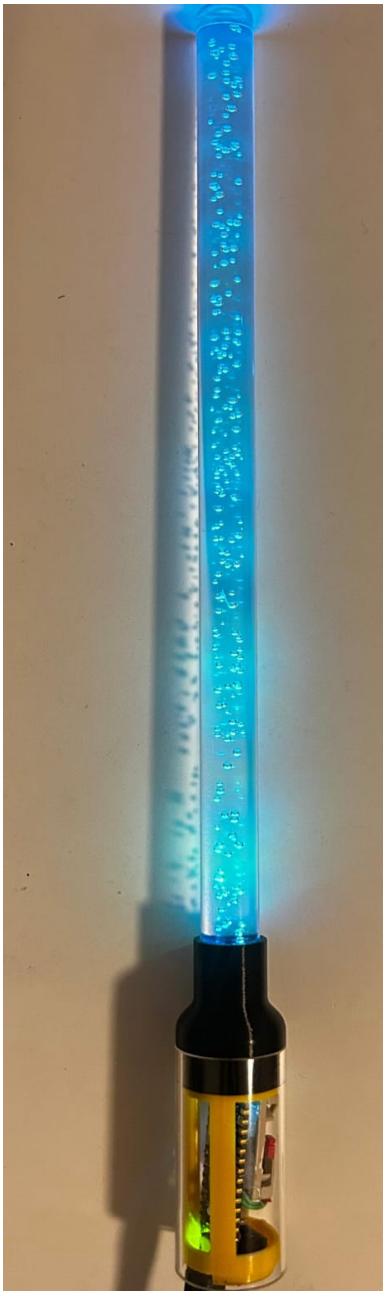


Figure 18.6.: BLUE Colour Output for SLOPE **L** on Magic Wand

19. Conclusion

In conclusion, the integration of Tiny Machine Learning (TinyML) with the Arduino Nano 33 BLE Sense has paved the way for the development of a unique and interactive "Magic Wand." This project signifies the synergy between machine learning advancements and the versatile capabilities of edge devices in the realm of the Internet of Things (IoT). The ability to process data locally, demonstrated through gesture recognition on the Arduino board, showcases the potential of edge computing in responding instantly to user inputs.
The implementation involves TensorFlow Lite, which contributes to the success of gesture recognition by utilizing a Neural Network model. The incorporation of a heuristic function adds an element of customization, allowing users to define and refine gesture knowledge without extensive programming skills. The visual output of recognized gestures and the LED response underscore the real-time interaction achieved through this integration.
Despite the challenges encountered during the planning process, such as managing the data model size within the Arduino Nano 33 BLE Sense's memory limitations and training for accuracy, the project's success demonstrates the feasibility of deploying machine learning applications on resource-constrained platforms.

1. **Accuracy Enhancement:** Focus on refining the algorithms or machine learning models to improve the accuracy and reliability of gesture recognition, minimizing both false positives and negatives. Furthermore, the script has been updated to integrate an LED output as an additional result of the gesture recognition, providing an alternate output view alongside the Arduino IDE serial output monitor.

2. **Enrichment of the testing procedures:** The testing process now specifies the exact aspects being tested (e.g., accuracy of motion detection and responsiveness of LEDs). We have documented the hardware setup, including all necessary connections and the test environment. Clear, step-by-step instructions for conducting the tests are provided, including relevant code snippets or commands. Expected outcomes are clarified, with specific values or behaviors that indicate successful testing.

3. **Optimized the code listing:** The complete code used for testing has been significantly improved, with key sections explained in detail beneath the code. This enhances readability and provides comprehensive guidance for understanding the experiments, ensuring that the process is accessible and well-documented for future reference.

4. **Enhancement of the test results:** The results from the hardware are documented thoroughly, including images of the serial monitor outputs and visual indicators used during testing to demonstrate that the code is functioning as expected. The experiments are interpreted, confirming the reliability of the tested components and providing insights into the overall system performance.

5. **Software Description Enrichment:** A sensor calibration code has been added to the software suite. Calibration is critical for ensuring accurate sensor readings by correcting any inherent hardware deviations or inaccuracies. This step not only enhances the precision of our data collection but also establishes a higher

standard for data integrity and reliability, which is essential for subsequent analyses and applications.

19.1. Additions

1. **User Feedback:** Enhance the user interaction by incorporating auditory or haptic feedback during the gesture recognition process. This improvement will provide users with a more immersive and responsive experience, making the system more intuitive to use.
2. **Expansion of Gesture Library:** Extend the current gesture library to include more complex and diverse motions, enabling the recognition of a wider range of gestures. This expansion will allow users to perform more nuanced actions, enhancing the versatility and functionality of the system.
3. **Energy Efficiency Improvements:** Introduce energy-saving features to optimize the device's battery life, which is especially critical for portable or wearable applications. This includes techniques such as power-down modes and efficient processing to reduce overall power consumption.
4. **Environmental Adaptability:** Increase the system's robustness to varying environmental conditions, such as different lighting, backgrounds, and noise levels, to maintain consistent gesture recognition performance. This will make the device more reliable in real-world settings where conditions often change.
5. **Community Engagement:** Create an online platform where users can share custom gestures and application ideas. This will foster a collaborative community around the project, enabling others to contribute, improve, and personalize their gesture-based experiences.

19.2. To-Do List

For the next phase of the Magic Wand project using the Arduino Nano 33 BLE Sense, we plan to tackle the following tasks:

1. **Design of Enclosure:**
   - Design and develop a custom enclosure to house the Arduino Nano 33 BLE Sense and the associated components.
   - Ensure the enclosure is ergonomic and facilitates easy interaction with the wand while providing sufficient protection for the internal components.
2. **Power Supply Management:**
   - Calculate the power requirements of the entire system and select an appropriate power supply that meets those needs.
   - Evaluate the possibility of making the project battery-powered and implement an efficient power management system to prolong battery life during use.
3. **Stability of Connections:**
   - Test the connection stability between the Arduino Nano 33 BLE Sense and any peripheral components to ensure reliable communication throughout the system.
   - Resolve any issues related to unstable or loose connections, ensuring smooth and uninterrupted functionality.

4. **Physical Attachment and Comfort:**
    - Securely attach the Arduino Nano 33 BLE Sense to the wand, considering factors such as weight distribution, balance, and ease of use.
    - Test the attachment to ensure that it can withstand typical handling and use without any risk of detachment or damage.
5. **Testing and Calibration:**
    - Calibrate all sensors and components to ensure that the gesture recognition system is accurate and reliable.
    - Conduct comprehensive testing of the entire hardware setup to identify and address any potential issues before final deployment.
6. **Comprehensive Documentation:**
    - Develop thorough documentation for the hardware setup, including assembly instructions, troubleshooting guides, and maintenance tips to ensure ease of use.
    - Include clear instructions on how to replace batteries (if applicable) and perform other necessary maintenance tasks to keep the system running smoothly.
7. **Safety and Compliance:**
    - Ensure that the project complies with relevant safety standards and industry regulations.
    - Introduce safety features, such as emergency shutdown procedures or safeguards, if applicable, to ensure user safety during operation.
8. **Packaging and Presentation:**
    - Design attractive and protective packaging for the Magic Wand that enhances its presentation and ensures it remains secure during shipping.
    - Consider branding and labeling options to create a professional and polished look for the final product.

19.3. Unanswered Points

1. **Device Durability:** There are still questions surrounding the long-term durability and reliability of the device under continuous use. The impact of regular usage on its components, especially over time, needs further investigation.
2. **User Customization:** The degree of customization available to users for personal preferences, such as customizing gestures and feedback mechanisms, is yet to be fully defined. For instance, our data model has been optimized for users who can perform gestures steadily and at a regular speed, but the system's performance may be impacted by users who have more erratic hand movements. This needs further analysis to address the recognition accuracy for such cases.
3. **Data Privacy and Security:** There is a need for clarification on how user data, particularly gesture inputs, are handled and protected against unauthorized access. This includes defining the measures in place to secure sensitive information.
4. **Cross-Platform Compatibility:** Uncertainties remain regarding the system's compatibility with various operating systems and devices. Further testing is required to ensure smooth integration without encountering compatibility issues across multiple platforms.

5. **Real-World Usability:** We need to explore the system's effectiveness in a variety of real-world scenarios and environments. Its performance in different lighting conditions, user settings, and diverse interaction styles has yet to be fully tested.

6. **Expansion of Gesture Recognition:** While additional gestures are being considered for inclusion in the data model to enhance recognition capabilities, the potential challenges around the model's capacity to handle diverse and complex gestures need further investigation. This includes examining the limits of gesture resistance and whether the model can effectively handle additional types.

19.4. Next Steps

In the next phase, we aim to address the unanswered points and expand the scope of the project. Key actions include gathering diverse user data and testing new, more complex gestures for broader use cases.

1. **Field Testing:** Conduct extensive field testing to assess the system's performance in real-world environments, gathering valuable user feedback to guide further improvements.

2. **Model Refinement:** Refine the machine learning models using real-world data collected from testing, aiming to enhance the accuracy and robustness of gesture recognition.

3. **Expand Application Ecosystem:** Develop and integrate new applications that take full advantage of the Magic Wand's gesture recognition capabilities, exploring a wider array of use cases and possibilities.

4. **Partnerships and Collaborations:** Seek potential partnerships with technology companies and academic institutions to explore innovative technologies and broaden the project's impact and reach.

5. **Sustainability Initiatives:** Implement strategies to ensure the sustainability of the project, such as incorporating energy-efficient design practices and using recyclable or eco-friendly materials.

19.5. Future Work

Although our current model is trained to recognize only three distinct gestures, there is considerable potential for expanding its functionality to recognize a broader range of gestures, based on evolving requirements.
Future developments of the project may include the following:

1. **LED Blinking for Specific Gestures:** Implement a feature where the LED attached to the Magic Wand blinks in response to a specific gesture. For instance, a particular gesture could trigger a distinct blinking pattern (e.g., slow blinking for "Wing" gesture, fast blinking for "Ring" gesture). This visual feedback can enhance the user experience, making the system's response more intuitive and noticeable. The LED blinking can also serve as a form of confirmation, ensuring the user that the gesture was recognized.

2. **Integration of External AI Software:** We plan to explore the possibility of using advanced AI software platforms to further train the model, thereby improving the accuracy of gesture recognition. Additionally, these platforms could help optimize the size of the data model, making it more efficient while retaining high performance.

3. **Wireless Communication Using BLE:** Leveraging the Bluetooth Low Energy (BLE) capability of the Arduino Nano 33 BLE Sense, we aim to enable wireless communication between the Magic Wand and a computer. This would enhance the system's flexibility, allowing for remote control or interaction without the need for physical connections.

4. **Advancements in TinyML:** As TinyML technologies continue to evolve, we anticipate greater advancements that can significantly improve the performance of our system. With future improvements in TinyML hardware, such as larger local memory storage and even smaller physical sizes, the capabilities of the Magic Wand could be further enhanced, enabling more sophisticated operations with minimal power consumption.

Part VI.

Packages

20. Libraries/Packages List

20.1. Introduction

Data science has become a crucial component across industries such as finance and healthcare, transforming how businesses operate. The rapid increase in data generation has created a demand for tools capable of managing large and complex datasets effectively. This has driven the creation of numerous data science packages that support data manipulation, visualization, and machine learning.
These tools are indispensable for data scientists, enabling efficient data analysis, model building, and deriving insights from complex datasets. This report highlights some of the most widely used and popular packages, including **pandas**, **numpy**, **matplotlib**, **datetime**, **Sklearn**, **keras**, **lightgbm**, and **tensorflow**. These open-source tools are user-friendly, extensively documented, and accessible to both beginners and experienced professionals.

20.2. Numpy

NumPy is a versatile Python library that forms the backbone of scientific computing and data analysis. It introduces a multidimensional array object and provides a wide range of functions and tools for array operations. NumPy's efficient array structure enables the storage and manipulation of large datasets, facilitating fast numerical computations and mathematical operations.
With features like linear algebra, Fourier transforms, and random number generation, NumPy is a critical tool for scientific and data-intensive tasks [McK12]. Its ease of use, high performance, and seamless integration with other scientific libraries make it a preferred choice for professionals and researchers in disciplines such as physics, engineering, finance, and machine learning.

Listing 20.1: Example of importing NumPy

```
import numpy as np
```

20.3. Pandas

The name "pandas" is derived from the econometrics term "panel data," which refers to datasets that contain observations over multiple time periods for the same individuals. Additionally, the name is a creative acronym that combines "Python data analysis" with the concept of panel data [[McK12]].
Pandas is a highly powerful and flexible library for data manipulation and analysis, specifically designed for working with structured data, such as tabular datasets. It provides a wide array of functions to clean, merge, transform, and analyze data efficiently. Built on top of the numpy library, pandas offers two primary data structures: Series (for one-dimensional data) and DataFrame (for two-dimensional data). These structures allow for intuitive and fast data manipulation, making pandas an essential tool for anyone working with data in Python, whether for exploratory analysis, data preprocessing, or complex data transformations.

Listing 20.2: Commands to update or install a specific version of NumPy using conda

```
conda update numpy
conda install numpy=1.21.0
```

20.4. Keras

Keras is a high-level, user-friendly Python library designed to streamline the development of deep learning models. It is built to work seamlessly with back-end frameworks such as TensorFlow, Theano, or CNTK, providing a simple and efficient interface for building complex neural networks. Keras abstracts away the intricate mathematical details of tensors and their operations, allowing developers to focus on the design and architecture of neural network layers without getting bogged down in low-level implementation.
Keras offers two main APIs for model creation: the **Sequential API** and the **Functional API**. The Sequential API is particularly popular due to its simplicity, enabling the construction of a linear stack of layers where each layer is added one after the other. This approach is especially beneficial for beginners or for projects where a straightforward architecture is sufficient. On the other hand, the Functional API provides more flexibility and is ideal for creating more complex models, such as those with multiple inputs, outputs, or shared layers. Overall, Keras simplifies the development of deep learning applications, allowing users to harness the power of deep learning without dealing directly with the complexities of the underlying back-end frameworks.

20.5. TensorFlow

TensorFlow is an open-source machine learning framework developed by the Google Brain team, designed to facilitate the development and deployment of machine learning models. Keras, on the other hand, is an open-source, high-level neural network API written in Python, which runs on top of TensorFlow. While TensorFlow serves as the primary machine learning framework, Keras provides a user-friendly, high-level interface that simplifies the process of building, training, and evaluating neural networks. The Keras layers module is particularly crucial, as it allows for the easy definition and stacking of layers within a neural network, enabling efficient model design and experimentation [Goo24].

Listing 20.3: Importing TensorFlow and Keras modules

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

20.6. Matplotlib

Matplotlib is a versatile and powerful Python library designed for creating high-quality 2D plots and visualizations. It allows users to generate a wide variety of graph types, including line plots, bar charts, pie charts, histograms, and more. Whether for interactive or static plotting, Matplotlib offers both options and can export visualizations to multiple file formats such as PNG, PDF, and SVG. The library's extensive customization capabilities enable users to fine-tune elements such as labels, colors, and axis scales, ensuring that graphs meet specific presentation requirements. Known for its flexibility and ease of use, Matplotlib is widely adopted in both academic and professional environments, making it a standard tool for data visualization in Python [**Hunter:2007**].

Listing 20.4: Importing the Matplotlib library for plotting

```
import matplotlib.pyplot as plt
```

20.7. Sklearn

Scikit-learn, commonly referred to as sklearn, is a popular and highly regarded open-source Python library for machine learning. It offers a comprehensive suite of tools and algorithms that facilitate a wide range of tasks including data analysis, preprocessing, feature extraction, and model selection. Built on top of core scientific libraries such as NumPy, SciPy, and matplotlib, scikit-learn provides an easy-to-use and efficient interface for implementing machine learning workflows. The library includes a broad spectrum of algorithms for tasks like regression, classification, clustering, and dimensionality reduction, along with capabilities for model evaluation and selection. Thanks to its simplicity, versatility, and strong community support, scikit-learn has become a go-to choice in both academic research and industry applications, making it one of the most widely adopted and powerful tools in the Python ecosystem for machine learning.

20.8. Imagedataset

A utility function in TensorFlow/Keras designed to load and preprocess image datasets directly from a specified directory. This function automates the process of reading images, performing necessary preprocessing steps like resizing, normalization, and data augmentation, to ensure the images are in the right format for training a machine learning model. By using this function, users can efficiently prepare their image data, reducing manual effort and minimizing potential errors. It streamlines the setup process, allowing for smoother integration into the training pipeline and ensuring the data is optimized for model performance.

20.9. google.colab and IPython

Google Colaboratory, often referred to as Colab, is a cloud-based platform that enables users to write and execute Python code in an interactive environment directly from their browser. It is especially useful for tasks involving machine learning, data analysis, and other computational activities, offering free access to powerful resources like GPUs and TPUs. On the other hand, IPython (Interactive Python) is an enhanced interactive shell that allows users to execute Python code and display outputs in a command-line interface. While Colab serves as a collaborative space for running code in the cloud, IPython is commonly used within notebooks to enable advanced features like inline visualization, such as displaying images or interactive plots, enhancing the user experience during data exploration or analysis.

Listing 20.5: Importing modules from Google Colab and IPython for file handling and displaying images

```
from google.colab import files
from IPython.display import Image, display
```

=

21. Numpy

21.1. Introduction

NumPy extends Python's functionality by adding support for large, multidimensional arrays and matrix operations. This library enables the creation and manipulation of complex structures such as masked arrays and matrices, while offering a comprehensive set of mathematical functions for efficient array processing. These functions support various operations, including algebraic and logical computations, array reshaping, sorting, selection, input/output handling, discrete Fourier transforms, basic linear algebra, statistical functions, and random number generation. [Har23]
Developed by Travis Oliphant in 2005, NumPy (short for Numerical Python) is an essential open-source library that plays a significant role in scientific computing and data analysis. Its primary feature, the ndarray (N-dimensional array), simplifies the management and processing of large, homogeneous datasets. Additionally, NumPy provides vital tools for linear algebra, Fourier transforms, and matrix operations, making it an integral component of many scientific computing frameworks and applications. [**NumPy:2024**]

21.2. Description

As a prominent library in the Python ecosystem, NumPy delivers strong support for large-scale, multidimensional arrays and matrices. It enhances Python by enabling efficient element-wise operations through broadcasting and offers a wide array of functions for essential mathematical tasks like linear algebra. Additionally, NumPy provides interfaces that allow code written in languages such as C, C++, and Fortran to be integrated, which is critical for performance-sensitive computing applications. [Har23]
The power of NumPy lies in its optimized management of numerical data and its ability to execute various mathematical operations. The library excels in linear algebra and integrates smoothly with other computational libraries, offering substantial performance gains when processing large datasets. These features make NumPy an essential tool for scientific computing, data analysis, and machine learning projects.
Key Capabilities of NumPy:

1. **ndarray** NumPy overcomes the limitations of Python lists with its ndarray object, which improves data storage efficiency. While lists can store multiple data types, ndarrays ensure that each column contains a single data type, which enhances computational performance [**NumPy:2024**].

2. **Array Creation and Manipulation** NumPy provides users with the ability to create arrays of varying dimensions and predefined values using utility functions such as numpy.zeros(), numpy.ones(), and numpy.random. It also supports a wide range of array manipulation techniques, including slicing, indexing, merging, and segmentation, making complex data operations easier [**GeeksforGeeks:2017**].

3. **Mathematical Functions and Operations** The library includes an extensive set of mathematical functions optimized for array operations. These functions

cover essential arithmetic, trigonometric, exponential, and logarithmic operations, enabling efficient calculations across arrays without needing to use loops [**GeeksforGeeks:2017**].

4. **Linear Algebra** A key feature of NumPy is its robust support for linear algebra, offering operations for vectors and matrices, including matrix multiplication (numpy.dot()), inversion (numpy.linalg.inv()), eigenvalue calculation (numpy.linalg.eig()), and singular value decomposition (numpy.linalg.svd) [**GeeksforGeeks:2017**].

5. **Random Number Generation** The numpy.random module provides advanced functionality for generating random numbers and arrays that follow specific probability distributions, which is essential for statistical modeling and simulations [**GeeksforGeeks:2017**].

6. **Integration with Other Libraries** NumPy acts as the core component for many other scientific computing libraries, such as SciPy, Matplotlib, pandas, and scikit-learn, facilitating seamless data interoperability and supporting efficient analysis and visualization workflows [Raj19].

21.3. Manual

21.3.1. Installation Instructions

NumPy is compatible with both Python 2 and Python 3 versions, though support varies by version.

- For Python 2, it is advisable to use NumPy versions up to 1.16.5 as these versions maintain compatibility with this Python version.
- For Python 3, specifically versions 3.5 and later, users should employ NumPy 1.17.0 or newer versions to ensure full compatibility and access to the latest features.
- To verify the installation of Python on your machine, you can execute the following command in your terminal or command prompt:

Listing 21.1: Example command to check Python version

```
# Check the Python version
python --version
```

Installing NumPy via pip
NumPy can be installed using pip, the Python package manager, which is included by default with most Python installations. To install NumPy, run the following command:

Listing 21.2: Example command to install NumPy using pip

```
# Install NumPy using pip
pip install numpy
```

This command retrieves the latest version of NumPy from the Python Package Index (PyPI) and installs it.

Installing NumPy via Conda
If you are using the Conda package manager, it is recommended to install NumPy within a separate environment to avoid conflicts with other packages. You can follow these steps:

Listing 21.3: Example commands to create a Conda environment and install NumPy

```
# Creating a new environment named 'my-env'
conda create -n my-env

# Activating the environment (use the appropriate command based on your operating
    ↪ system)
conda activate my-env  # Windows
source activate my-env # Linux and macOS

# Optionally add Conda-forge as a channel
conda config --env --add channels conda-forge

# Install NumPy in the active environment
conda install numpy
```

21.3.2. Verifying Installation: Import NumPy

To confirm that NumPy has been installed properly, open a Python interpreter or create a new Python script and input the following commands. This will import NumPy and display the installed version, verifying that it is ready for use:

Listing 21.4: Example code to print the installed NumPy version

```
import numpy as np
print("NumPy version:", np.__version__)
```

This script checks the version of NumPy currently installed on your system, such as **numpy - 1.24.1**, and displays it.

21.3.3. Key Attributes of NumPy

NumPy's utility is defined by several core attributes that facilitate efficient data management and operations [**NumPy:2024**]:

- **shape**: This attribute provides the dimensions of an array, indicating the size along each axis as a tuple.
- **dtype**: This describes the data type of the array's elements, supporting types like integers, floats, or booleans, which aids in the consistent handling of data.
- **ndim**: This denotes the number of dimensions or axes of the array.
- **size**: It reflects the total count of elements in the array, computed as the product of the array's dimensions.
- **itemsize**: This shows the memory size in bytes for each element in the array, illustrating the storage efficiency.
- **nbytes**: This calculates the total amount of memory (in bytes) the array uses, which is the product of the size and the itemsize.
- **data**: Represents a buffer containing the actual data of the array, allowing direct access to the array's raw data.

21.3.4. Practical Examples of Attributes

Listing 21.5: Example code demonstrating NumPy array operations

```
import numpy as np

# Defining data types for structured data
dtypes = {'store_nbr': np.dtype('int64'),
    'item_nbr': np.dtype('int64'),
```

```
        'unit_sales': np.dtype('float64'),
        'onpromotion': np.dtype('O')}

# Check for unique data
unique_data = np.unique(data['variable'], return_counts=True)

# Reshape and manage data
data_array = np.arange(15).reshape(3, 5)
reshaped_array = data_array.reshape(data_array.shape[0], -1)

# General array creation
array_example = np.array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9],
[10, 11, 12, 13, 14]])

# Accessing array attributes
print("Shape:", data_array.shape)
print("Number of dimensions:", data_array.ndim)
print("Data type:", data_array.dtype.name)
print("Size of each element:", data_array.itemsize)
print("Total size of array:", data_array.size)

# Example of array operation
unique_values, counts = np.unique(data_array, return_counts=True)
print("Unique values and counts:", np.array((unique_values, counts)))
```

21.4. Examples

21.4.1. Linear Algebra Fundamentals

NumPy is equipped with a vast selection of functions designed for high-level linear algebra operations. The following example illustrates how to solve a system of linear equations represented by Ax = B:

Listing 21.6: Example code to solve a system of linear equations using NumPy

```
import numpy as np

# Defining coefficients matrix A and constant vector B
A = np.array([[3, 1, -2], [1, -1, 4], [2, 0, 3]])
B = np.array([5, 6, 4])

# Computing solution vector x
x = np.linalg.solve(A, B)

print("Solution vector x:", x)
```

21.4.2. Advanced Array Operations

NumPy excels in performing vectorized operations across arrays efficiently and swiftly, surpassing traditional Python techniques.

Element-wise Operations

Operations such as addition and multiplication can be applied element-wise between arrays of identical shapes by using (a + b and a * b) formula, which is critical for mathematical and scientific computations.

Scalar Operations

NumPy enables operations between arrays and scalars seamlessly for example (5 * a). This functionality is integral for modifying data scales and normalizations efficiently.

```
    # Setup two example NumPy arrays
    a = np.array([1, 2, 3, 4])
    b = np.array([10, 20, 30, 40])

    # Demonstrating various operations
    print("Element-wise addition:", a + b)
    print("Element-wise multiplication:", a * b)
    print("Scalar multiplication:", 5 * a)
    print("Elements greater than 5:", a > 5)
    print("Sum of elements in a:", np.sum(a))

```

Listing 21.1.: Example code demonstrating basic NumPy array operations

Boolean Operations

NumPy facilitates direct array comparisons, producing boolean arrays ideal for conditional filtering. For example, a>5 checks if elements in array a are greater than 5.

Summation Functions

The library provides methods like np.sum() to perform summations across arrays rapidly, an essential tool in data analysis.

21.4.3. Advanced Broadcasting Techniques

NumPy's broadcasting feature is pivotal for performing operations on arrays of different sizes, by extending the smaller array across the larger one to align their shapes. This capability is essential for efficiently applying scalar operations across entire arrays or combining arrays of differing dimensions without the explicit replication of data.

Listing 21.7: Example code demonstrating broadcasting in NumPy

```
import numpy as np

# Example with a scalar and an array
a = np.array([1, 2, 3])
b = 2
print("Broadcasting with scalar addition:", a + b)

# Example with 2D and 1D arrays
A = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([1, 0, 1])
print("Complex broadcasting with 2D and 1D arrays:\n", A + b)
```

21.4.4. Reshaping and Flattening

Manipulating the structure of arrays is a common task in data processing, especially in contexts like machine learning where data shape directly impacts model behavior. NumPy offers several functions to reshape and flatten arrays, facilitating the reorganization of data structures.

Reshaping Arrays

Changing the shape of an array to fit specific requirements without altering the data is performed with the reshape method.

Flattening Arrays

Converting a multi-dimensional array into a 1D array is frequently necessary for processes that require a linear sequence of elements.

Listing 21.8: Example code demonstrating array reshaping and flattening in NumPy

```
import numpy as np

# Reshaping an array into a 3x3 matrix
a = np.arange(9).reshape(3, 3)
print("Reshaped to 3x3 array:\n", a)

# Flattening the array
flat_a = a.flatten()
print("Flattened array:", flat_a)

# Reshaping to 1x9 for a different view
reshaped_a = a.reshape(1, 9)
print("Reshaped to 1x9 array:\n", reshaped_a)
```

21.4.5. Stacking and Splitting Arrays

NumPy simplifies the process of combining multiple arrays into one and dividing a single array into multiple parts, crucial for data preparation and segmentation tasks.

Stacking Arrays

Both vertical and horizontal stacking are common operations that combine different datasets into a single array.

Splitting Arrays

Dividing data into manageable or required portions is often necessary in cross-validation workflows or during data analysis.

Listing 21.9: Example code demonstrating array stacking and splitting in NumPy

```
import numpy as np

# Vertical and horizontal stacking examples
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
v_stack = np.vstack((a, b))
h_stack = np.hstack((a, b))
print("Vertically stacked:\n", v_stack)
print("Horizontally stacked:", h_stack)

# Splitting an array into three parts
c = np.arange(9)
split_c = np.split(c, 3)
print("Equally split array:", split_c)
```

21.4.6. Managing NumPy Versions

Understanding the compatibility between NumPy and Python versions is crucial for maintaining stable applications. Here's an overview of how NumPy aligns with different Python environments:

- For Python 2 users, it's recommended to use NumPy versions up to 1.16.5, as these are the last releases to support Python 2.
- For Python 3, especially versions 3.5 and later, NumPy 1.17.0 or newer should be used to ensure full functionality and support.

- To verify the Python version on your machine, execute the following in your command prompt or terminal:

Listing 21.10: Command to check the installed Python version

```
python --version
```

To keep NumPy up-to-date or to install a specific version, use the package manager that was originally used for installation, typically pip for Python installations or conda for Anaconda distributions:

21.4.7. Upgrading NumPy

Utilize pip, Python's package installer, to update NumPy to the most recent version available from the Python Package Index (PyPI) or to install a specific version as needed.

Listing 21.11: Commands to upgrade or install a specific version of NumPy using pip

```
pip install --upgrade numpy
pip install numpy==1.21.0
```

Upgrading NumPy with conda
If you are using Anaconda, manage your NumPy installation using the conda command line. This approach is beneficial for handling dependencies more effectively than pip.

Listing 21.12: Commands to update or install a specific version of NumPy using conda

```
conda update numpy
conda install numpy=1.21.0
```

Maintaining compatibility with your project's dependencies during upgrades is vital. Consider the following:

- **Maintaining a Requirements File**

  Keep a 'requirements.txt' file updated to manage package versions within your project environment. After any upgrades, generate a new requirements file and thoroughly test your project to ensure all components function as expected without conflicts or deprecated issues.

Listing 21.13: Command to export installed packages to a requirements.txt file

```
pip freeze > requirements.txt
```

21.4.8. File Interaction with NumPy

NumPy offers robust functionality for file operations, allowing arrays to be saved to and loaded from the disk efficiently.
Storing and Retrieving Arrays
NumPy utilizes ".npy" as the standard file format for saving single arrays, which optimizes space and preserves data integrity. When dealing with multiple arrays, the ".npz" format is preferable as it encapsulates several arrays into one file without loss of information.

Listing 21.14: Example code to save and load arrays in .npy and .npz formats

```
# Saving a single array to a .npy file
import numpy as np
a = np.array([1, 2, 3, 4, 5])
np.save('my_array.npy', a)
```

```
# Loading the array from a .npy file
b = np.load('my_array.npy')
print("Loaded array:", b)

# Saving multiple arrays into a .npz file
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([7, 8, 9])
np.savez('my_arrays.npz', array1=a, array2=b)

# Retrieving arrays from a .npz file
data = np.load('my_arrays.npz')
print("First array:", data['array1'])
print("Second array:", data['array2'])
```

Handling Custom File Formats
NumPy's 'genfromtxt' function provides a versatile solution for importing data from structured text files, like CSVs. This function is especially useful for data that may contain missing values, requiring conversion between different data types or extraction based on specific criteria.

Listing 21.15: Example code to import and export data using NumPy with CSV files

```
# Importing data from a CSV file
import numpy as np
data = np.genfromtxt('my_data.csv', delimiter=',')

# Exporting an array to a CSV file
np.savetxt('array.csv', data, delimiter=',')
```

21.4.9. Error Handling in NumPy

NumPy provides robust tools for managing errors and exceptions that arise during numerical computations, particularly with floating-point operations. Here's how NumPy handles some common numerical errors:

21.4.10. Handling Floating-Point Errors

Overflow Error in Exponential Functions
Calculating the exponential of a large value with 'np.exp(1000)' triggers an overflow error because the result exceeds the range that can be represented by the floating-point type in NumPy.
Invalid Operation Error
An invalid operation error occurs with 'np.sqrt(-1)', as taking the square root of a negative number is not valid in the realm of real numbers, thus raising a FloatingPointError.
Division by Zero Error
Attempting to divide by zero using 'np.divide()' where the denominator is zero leads to a divide by zero error, which NumPy handles by raising a FloatingPointError.
Managing Runtime Warnings
When configured to 'raise' errors for invalid operations, such as division by zero, NumPy will elevate these issues to FloatingPointError exceptions. This is crucial for debugging and ensuring that numerical calculations adhere to expected standards of accuracy and reliability.

Listing 21.16: Example code demonstrating handling floating-point errors in NumPy

```
import numpy as np
np.seterr(all='raise')  # Set to raise exceptions for all types of floating-
    ↪ point errors

try:
# Attempting an operation that might cause an error
np.divide(1, 0)
except FloatingPointError as e:
```

```
        print("Caught an exception:", e)
```

21.5. Example - files

NumPy can interact with files primarily through loading and saving arrays to and from disk.

Saving and Loading Array

The standard file formats that can save array using the ".npy" format to store the array. If we would like to save multiple arrays in one file, ".npz" will be used instead of ".npy". For a single array, we can use the "save" function to store the data. For multiple arrays, we use the "savez" function to store the multiple data.

Listing 21.17: Example code for saving and loading NumPy arrays in .npy and .npz formats

```
# For single array, ".npy" format would be applied
# Example 1:
import numpy as np
a = np.array([1, 2, 3, 4, 5])
np.save('my_array.npy', a)

# Loading an array from a ".npy" file:
b = np.load('my_array.npy')
print(b)

# Example 2:
# Saving multiple arrays into a single file in NumPy .npz format:
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([7, 8, 9])
np.savez('my_arrays.npz', array1=a, array2=b)

# Loading multiple arrays from a .npz file:
data = np.load('my_arrays.npz')
print(data['array1'])
print(data['array2'])
```

Saving and Loading for custom file formats

NumPy's genfromtxt function is a flexible and powerful way to import data from text files, like CSV, into NumPy arrays. It allows for handling missing values, converting data types, and selecting specific columns.

Listing 21.18: Example code for loading and saving data using NumPy

```
import numpy as np
# Basic usage to load data from a CSV file
data = np.genfromtxt('my_data.csv', delimiter=',')

# Save an array to a text file, such as CSV
np.savetxt('array.csv', data, delimiter=',')
```

21.6. Further Reading

Diving deeper into NumPy and enhancing your numerical computing skills can be significantly aided by engaging with diverse educational materials. The following resources have been carefully selected to provide comprehensive learning opportunities:

21.6.1. Neural Network for Beginners: Build Deep Neural Networks and Develop Strong Fundamentals using Python's NumPy, and Matplotlib

Authored by Sebastian Klaas, this introductory guide focuses on the foundational principles and hands-on experience necessary for building neural networks. Utilizing

NumPy for computations and Matplotlib for visualization, it provides a clear path to understanding the mathematical foundations of neural networks, including key concepts like forward propagation, backpropagation, and optimization[**Sewak:2018**].

21.6.2. Effective Computation in Physics: Field Guide to Research with Python

Targeted not just at physicists but any researchers using computational methods, this book provides a robust introduction to scientific computing with Python. Covering key libraries such as NumPy, SciPy, and Matplotlib, it extends into areas like parallel computing and database management. The practical, example-driven approach makes it an excellent resource for applying Python to solve real-world scientific problems [**McKinney:2011**].

21.6.3. Python for Data Analysis

Written by Wes McKinney, the creator of pandas, this book is invaluable for those looking to understand how NumPy fits into the larger picture of data analysis with Python. Covering everything from data preparation to complex analyses, it provides a solid foundation in using NumPy and pandas together to handle, process, and analyze data effectively.

21.6.4. NumPy Official Website

As the primary resource for all things NumPy, the official documentation offers exhaustive material suitable for learners at all levels. From beginner tutorials to advanced feature guides and release notes, it's the definitive source for up-to-date and accurate information on how to utilize NumPy to its fullest potential [**NumPy:2024**].

21.6.5. NumPy Beginner's Guide - Third Edition

Ideal for newcomers to NumPy or scientific computing in general, Ivan Idris's guide is updated with the latest features of NumPy and offers a hands-on learning approach. Through detailed examples and exercises, readers can explore basic to advanced array operations, learn efficient data manipulation techniques, and apply NumPy in practical scenarios such as image processing and financial modeling [**NumPy:2024**].
Each of these resources provides unique insights and practical knowledge that can help both novices and seasoned users of NumPy expand their skills and understanding of numerical computing with Python.

22. Pandas

22.1. Introduction

Pandas is a powerful and popular open-source data analysis and manipulation library for Python. It provides easy-to-use data structures and data analysis tools, making it highly efficient for working with structured data, such as spreadsheets, SQL tables, and time series data.
This report aims to provide a comprehensive overview of the Pandas package, including its description, installation process, and several examples demonstrating its usage in different scenarios. Additionally, further reading resources will be suggested for those interested in diving deeper into the Pandas library [**McKinney:2011**].
The version of the Pandas package used here is 1.4.4. This package is used in the project to load the data from CSV files, transform the data (e.g. handle missing values), add columns, and etc. All these functions of the Pandas package are explained in the upcoming sections[**McKinney:2015**].

22.2. Description

Pandas is a powerful and popular open-source data analysis and manipulation library for Python. It provides easy-to-use data structures and data analysis tools, making it highly efficient for working with structured data, such as spreadsheets, SQL tables, and time series data.
At its core, Pandas introduces two primary data structures: Series and DataFrame.

- Series is a one-dimensional labeled array that can hold any data type. It is similar to a column in a spreadsheet or a database table and can be accessed using labels or positions. With Series, data can be efficiently manipulated and analyzed, enabling tasks such as filtering, aggregation, and transformation [**McKinney:2011**].

- DataFrame, on the other hand, is a two-dimensional labeled data structure with columns of potentially different types. It can be thought of as a tabular data structure, similar to a spreadsheet or SQL table. DataFrames provide a rich set of operations, such as indexing, merging, reshaping, and grouping, allowing for extensive data manipulation and analysis capabilities.

Pandas offers a wide range of functionalities, including data cleaning, transformation, filtering, merging, reshaping, and visualization. It provides tools for handling missing data, working with time series data, and performing statistical operations. With its intuitive and expressive syntax, Pandas simplifies data manipulation tasks and accelerates the data analysis process [**Heydt:2017**].
Furthermore, Pandas seamlessly integrates with other popular Python libraries, such as NumPy, Matplotlib, and scikit-learn, making it an essential tool in the data science and analytics ecosystem. Its versatility, efficiency, and extensive documentation make it a preferred choice for data professionals, researchers, and analysts working with structured data in Python.

22.2.1. Data Suitability

Pandas is highly compatible with various types of data.

- **Data Structure:** Pandas primarily works with two main data structures: Series and DataFrame. Series is a one-dimensional labeled array capable of holding any data type, while DataFrame is a two-dimensional labeled data structure resembling a table or spreadsheet. Ensure that your data is organized in a structure that can be represented by these data structures for effective usage with Pandas.
- **Data Format:** Pandas supports various data formats, including CSV, Excel, SQL databases, JSON, HTML,HDF5 (Hierarchical Data Format),Parquet, Feather, and more. Ensure that your data is in a compatible format that can be easily read and loaded into Pandas. Additionally, Pandas provides functions to read and write data in different formats, allowing you to seamlessly work with different types of data sources.
- **Data Size:** Consider the size of your data when working with Pandas. While Pandas can handle large datasets, excessively large datasets may result in performance issues or memory constraints. If you have a large dataset, you may need to optimize your code, use appropriate data types, or consider using alternative tools like **Dask** or **Apache Spark** for distributed computing.
- **Performance Considerations:** Depending on the size and complexity of your data, certain operations in Pandas may be more computationally expensive. It is essential to optimize your code and leverage Pandas' built-in vectorized operations and efficient indexing techniques to achieve better performance.

22.2.2. Key Features of Pandas

Pandas excels in the following areas [**Betancourt:2019**]:

- Easy management of missing data (represented as NaN) in both floating point and non-floating point data.
- Size mutability, allowing the insertion and deletion of columns from DataFrame and higher-dimensional objects.
- Automatic and explicit data alignment, enabling explicit alignment of objects to a set of labels or automatic alignment by Series, DataFrame, etc. during computations.
- Powerful and flexible group by functionality, facilitating split-apply-combine operations on data sets for both aggregation and transformation.
- Seamless conversion of ragged and differently-indexed data from other Python and NumPy data structures into DataFrame objects.
- Intelligent label-based slicing, fancy indexing, and subsetting of large data sets.
- Intuitive merging and joining of data sets.
- Flexible reshaping and pivoting of data sets.
- Hierarchical labeling of axes, allowing the presence of multiple labels per tick.
- Robust I/O tools for loading data from flat files (CSV and delimited formats), Excel files, databases, and efficient saving/loading of data in the HDF5 format.
- Time series-specific functionality, including date range generation, frequency conversion, moving window statistics, date shifting, and lagging.

22.2.3. Why Multiple Data Structures in Pandas?

Pandas employs multiple data structures for a specific purpose. It is best to view pandas data structures as flexible containers for lower-dimensional data. For instance, DataFrame acts as a container for Series, and Series acts as a container for scalars. This allows us to insert and remove objects from these containers in a manner similar to dictionaries [**McKinney:2015**].
Additionally, pandas provides sensible default behaviors for common API functions, considering the typical orientation of time series and cross-sectional data sets. When using N-dimensional arrays (ndarrays) to store 2- and 3-dimensional data, it burdens the user to consider the data set's orientation while writing functions. In pandas, the axes are designed to impart more semantic meaning to the data. For a given data set, there is usually a preferred orientation. The objective is to reduce the mental effort required to code data transformations in downstream functions [McK12].
For example, when dealing with tabular data (DataFrame), it is more semantically helpful to think in terms of the index (representing rows) and the columns, rather than axis 0 and axis 1. Iterating through the columns of a DataFrame results in more readable code:

```
for col in df.columns:
    series = df[col]
    # do something with series
```

Listing 22.1.: Columns of DataFrame

22.3. Installation

Installing pandas as a component of the cross-platform Anaconda distribution, which is used for data analysis and scientific computing, is the simplest way to do so. For the majority of users, this installation technique is advised.

22.3.1. Python version support

Pandas officially supports a range of Python versions, including Python 3.8, 3.9, 3.10, and 3.11.

22.3.2. Installing with Anaconda

For less experienced users, installing pandas and the rest of the NumPy and SciPy stack can be challenging. With Anaconda, a cross-platform (Linux, macOS, Windows) Python distribution for data analytics and scientific computing, it is easy to install not only pandas, but Python and the most well-known tools that make up the SciPy stack (IPython, NumPy, Matplotlib,...) as well. After launching the installer, the user won't need to install anything else or wait for any software to compile in order to utilize pandas and the rest of the SciPy stack.

22.3.3. Installing using terminal or command prompt

To install Pandas, you can use the Python package manager, pip, by running the following command in your terminal or command prompt:
Ensure that you have a compatible version of Python installed on your system before running the installation command.

```
pip install pandas
```

Listing 22.2.: Installing Pandas using terminal

22.3.4. Required dependencies

Pandas requires the following dependencies:

Package	Minimum supported version
NumPy	1.20.3
python-dateutil	2.8.2
pytz	2020.1

22.4. Example - Manual

22.4.1. User Manual for the Example Python File in PyCharm

Installation and Setup

1. **Download and install Python:**

   Visit the official Python website `https://www.python.org` and download the latest version of Python for your operating system. Follow the installation instructions provided.

2. **Install PyCharm:**

   Visit the JetBrains website `https://www.jetbrains.com/pycharm` and download PyCharm Community Edition, which is the free version. Install PyCharm by following the installation instructions specific to your operating system.

Opening the Python File in PyCharm

1. Launch PyCharm: Open the PyCharm application from your desktop or applications menu.

2. Create a new project: Click on `Create New Project` or go to `File` → `New Project`. Choose a suitable location for your project and provide a name.

3. Open the Python file: Once the project is created, navigate to the project directory in the PyCharm project view. Right-click on the desired folder and select `New` → `Python File`. Provide a name for the file and click `OK`.

4. Copy your Python code: Open your Python file (with .py extension) in a text editor and copy the contents.

5. Paste the code: Paste the copied code into the newly created Python file in PyCharm.

Working with the Python File

1. Running the script: To run the Python script, right-click anywhere within the Python file and select `Run` → `Run ExampleManual.py`. Alternatively, you can use the keyboard shortcut `Shift + F10`. The script will execute, and the output will be displayed in the PyCharm console.

2. Debugging the script: To debug the Python script and set breakpoints for analysis, click on the left gutter of the Python file, next to the line where you want to set the breakpoint. A red dot will appear, indicating the breakpoint. Click on the `Debug` button or use the keyboard shortcut `Shift + F9` to start debugging the script.

3. Interacting with the script: If your script expects user input or provides interactive prompts, you can provide the input in the PyCharm console. The console allows you to interact with the script while it is running.

Modifying the Python File

1. Editing the code: To make changes to the Python code, simply locate the section you want to modify and edit the code accordingly.

2. Saving the changes: PyCharm automatically saves your changes as you work. However, you can manually save the file by going to `File` → `Save` or using the keyboard shortcut `Ctrl + S`.

Further Assistance and Resources

- PyCharm Documentation: PyCharm offers comprehensive documentation to help you understand its features and functionality. You can access it online at `https://www.jetbrains.com/help/pycharm`.

- Python Documentation: The official Python documentation provides detailed information about the Python language, libraries, and best practices. It is available at `https://docs.python.org`.

- Online Python Communities: Joining online communities like Stack Overflow `https://stackoverflow.com` or the Python subreddit `https://www.reddit.com/r/Python` can provide valuable insights and assistance from experienced Python developers.

22.4.2. How to import

The code begins by importing the required libraries, numpy (`np`) and pandas (`pd`).

```
import numpy as np
import pandas as pd
```

Listing 22.3.: How to Import the package

22.4.3. Object creation

- A pandas Series `s` is created with some values, including a NaN (not a number) value.

- A sequence of dates is generated using the `pd.date_range()` function and stored in the `dates` variable.

- A DataFrame `df` is created with random values using the `np.random.randn()` function. The DataFrame has 6 rows and 4 columns, with the dates as the index and column labels as 'A', 'B', 'C', and 'D'.

Listing 22.1: Object Creation Example

```
s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s)

dates = pd.date_range("20130101", periods=6)
print(dates)

df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
print(df)
```

```
 0   1.0
 1   3.0
 2   5.0
 3   NaN
 4   6.0
 5   8.0
 dtype: float64
  DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-
04',
 '2013-01-05', '2013-01-06'],
 dtype='datetime64[ns]', freq='D')
 A      B      C      D
 2013-01-01 -1.473364  1.116274 -1.815663  0.198003
 2013-01-02 -0.094806 -0.501628 -1.722026  0.190776
 2013-01-03  0.163170  0.995486  0.854393 -2.657105
 2013-01-04  0.064040 -0.041879  0.581580 -0.896344
 2013-01-05 -0.209241 -0.910417  0.517225  0.097267
 2013-01-06  0.669742  0.226435 -0.609790 -0.741223
```

22.4.4. Viewing data

- The `head()` method is used to display the first few rows of the DataFrame `df`.
- The `tail()` method is used to display the last 3 rows of `df`.
- The `describe()` method provides a statistical summary of the DataFrame.
- The `sort_values()` method is used to sort the DataFrame `df` by the values in column 'B'.

```
print(df.head())
print(df.tail(3))
print(df.describe())
```

Listing 22.4.: Viewing Data

The results are not shown from now on due to the shortage of space. The manual for the user is provided, so by runing that, the results would appear in the console.

22.4.5. Selection

Getting

- Column 'A' of the DataFrame `df` is selected using `df["A"]`.
- The first three rows of the DataFrame `df` are selected using slicing with `df[0:3]`.

selection by label

- Using label-based indexing with `df.loc`, all rows and columns 'A' and 'B' are selected.

Selection by position

- The fourth row of the DataFrame `df` is selected using `df.iloc[3]`.
- Rows 3 to 4 and columns 0 to 1 are selected using `df.iloc[3:5, 0:2]`.

Boolean indexing

- The DataFrame `df` is filtered using a Boolean condition `df["A"] > 0`.

```
# getting
print(df["A"])
print(df[0:3])

# selection by label
df.loc[:, ["A", "B"]]

# Selection by position
print(df.iloc[3])
print(df.iloc[3:5, 0:2])

# Boolean indexing
print(df[df["A"] > 0])

df.iloc[0, 1] = 0
print(df.iloc[0, 1])
```

Listing 22.5.: Boolean Indexing Example

22.4.6. Setting

- The value at row 0, column 1 of the DataFrame `df` is modified to 0 using `df.iloc[0, 1] = 0`.

```
# Setting
df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ["E"])
df1.loc[dates[0] : dates[1], "E"] = 1
```

Listing 22.6.: Setting Example

22.4.7. Handling missing data

- A new DataFrame `df1` is created by reindexing `df` and adding a new column 'E'.
- Values in column 'E' for the first two rows of `df1` are set to 1.
- The `dropna()` method is used to drop rows with any NaN values from `df1`.
- The `pd.isna()` function is used to check for NaN values in `df1`.

```
# Handling Missing Data
df1.dropna(how="any")
pd.isna(df1)
```

Listing 22.7.: Handling Missing Data

22.4.8. Operations

- The `apply()` method is used to calculate the difference between the maximum and minimum values of each column in `df`.

```
#Operations
df.apply(lambda x: x.max() - x.min())
```

Listing 22.8.: Operations Example

22.4.9. Merge

- The `concat()` function is used to concatenate three chunks of the DataFrame `df`.
- Two DataFrames, `left` and `right`, are created with a common column 'key'.
- The `merge()` function is used to merge `left` and `right` DataFrames based on the 'key' column.

```
# Concat
df = pd.DataFrame(np.random.randn(10, 4))
df

pieces = [df[:3], df[3:7], df[7:]]
pd.concat(pieces)

#Merge
left = pd.DataFrame({"key": ["foo", "foo"], "lval": [1, 2]})
right = pd.DataFrame({"key": ["foo", "foo"], "rval": [4, 5]})
left
right

pd.merge(left, right, on="key")
```

Listing 22.9.: Merge Example

22.4.10. Grouping

- A DataFrame `df` is created with 'Animal' and 'Max Speed' columns.
- The `groupby()` method is used to group the DataFrame `df` by the 'Animal' column and calculate the mean of 'Max Speed' for each group.

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
    'Parrot', 'Parrot'],
    'Max Speed': [380., 370., 24., 26.]})

df.groupby(['Animal']).mean()
```

Listing 22.10.: Grouping Example

22.4.11. Reshaping

- A DataFrame `df_single_level_cols` is created with two rows, two columns, and single-level column labels.
- The `stack()` method is used to stack the DataFrame, returning a Series.
- The `unstack()` method is used to unstack the Series, reverting it.

```
df_single_level_cols = pd.DataFrame([[0, 1], [2, 3]],
        index=['cat', 'dog'],
        columns=['weight', 'height'])

# Stacking a dataframe with a single level column axis returns a Series:
stacked  = df_single_level_cols.stack()
stacked
# the inverse operation of stack() is unstack(), which by default unstacks the last
    ↪ level:
stacked.unstack()
```

Listing 22.11.: Reshaping Example

22.4.12. Importing and exporting data

1. `df.to_csv("foo.csv")`: This line takes a Pandas DataFrame, `df`, and exports it to a CSV file named "foo.csv". The `to_csv` method is used to convert the DataFrame into a CSV format and save it as a file. The resulting CSV file will contain the data from the DataFrame, with each row representing a data entry and each column representing a variable.
2. `pd.read_csv("foo.csv")`: This line reads the contents of the CSV file "foo.csv" and creates a new DataFrame using the data from the file. The `read_csv` function from the Pandas library is used to read the CSV file and convert it back into a DataFrame.

```
df.to_csv("foo.csv")
pd.read_csv("foo.csv")
```

Listing 22.12.: Reading CSV file

22.4.13. Pandas Error Handling

Pandas provides various error handling mechanisms to handle exceptions and errors that may occur during data manipulation and analysis. These error handling techniques help in diagnosing and addressing issues that arise when working with data in Pandas. Some common error handling techniques in Pandas include:

1. **Try-Except Blocks:** You can use standard Python try-except blocks to catch and handle specific exceptions that may occur during Pandas operations. For example, you can wrap a Pandas function call within a try block and use except blocks to handle specific exceptions, such as `ValueError` or `KeyError`, and perform appropriate error handling actions.

```
try:
    # Pandas operation
    df = pd.read_csv('data.csv')
except FileNotFoundError:
    # Error handling code
    print("File not found. Please check the file path.")
except ValueError:
    # Error handling code
    print("Error in data. Please ensure correct data format.")
```

Listing 22.13.: Try Except Blocks

2. **Error Reporting:** Pandas provides descriptive error messages that provide information about the nature of the error and the location where it occurred. These error messages can be useful for diagnosing and debugging issues in the data or the code. When an error occurs, Pandas displays an error message along with a traceback that helps identify the source of the error.

3. **Error Handling Functions:** Pandas provides specific functions to handle errors and exceptions. For example, the `pd.options.mode` function allows you to set error handling modes, such as `raise` to raise exceptions for errors, `warn` to display warning messages instead of raising exceptions, and `ignore` to ignore errors and proceed with the operation.

```
# Set error handling mode to 'raise'
pd.options.mode.chained_assignment = 'raise'

# Set error handling mode to 'warn'
pd.options.mode.chained_assignment = 'warn'

# Set error handling mode to 'ignore'
pd.options.mode.chained_assignment = 'ignore'
```

Listing 22.14.: Error Handling Functions

4. **Error Handling with DataFrame and Series:** Pandas provides methods and properties for error handling specific to DataFrame and Series objects. For example, the `.at` and `.iat` attributes allow for safe access and assignment of scalar values, raising exceptions if the provided index is not found. The `.get()` method allows retrieving values with a default value, avoiding exceptions when the key is not found.

```
# Safe access using .at attribute
value = df.at[index, column]

# Safe assignment using .iat attribute
df.iat[index, column] = value

# Safe retrieval with default value using .get() method
value = df.get(key, default_value)
```

Listing 22.15.: Error Handling with DataFrame and Series

By leveraging these error handling techniques, you can effectively handle exceptions and errors that may arise during data operations in Pandas, ensuring smooth data processing and analysis.

22.5. Further Reading

22.5.1. A Comprehensive Overview of Pandas

Explore pandas, a Python library tailored for structured data analysis in statistics, finance, and social sciences. This paper covers its design, features, and its role as a foundational layer in Python's statistical computing landscape. Discover how pandas complements the scientific Python stack and improves upon data manipulation tools from languages like R. Additionally, insights into future development and growth opportunities for statistical applications in Python are discussed [**McKinney:2011**]. Find it here : `https://www.dlr.de/sc/portaldata/15/resources/dokumente/pyhpc2011/submissions/pyhpc2011_submission_9.pdf`

22.5.2. pandas: a python data analysis library

This book, written by Wes McKinney (the creator of Pandas) [**McKinney:2015**], is a valuable resource for learning Pandas. It covers various aspects of data manipulation, analysis, and visualization using Pandas. The book also explores practical examples and real-world use cases. Find it here: https://www.oreilly.com/library/view/python-for-data/9781491957653/

22.5.3. Pandas Library

This chapter presents pandas, a powerful Python library that introduces the DataFrame data structure. Derived from NumPy arrays, pandas revolutionizes data analysis by offering a structured framework. The core elements, Series and DataFrame, enable the organization of varied data types into a unified structure, facilitating easy application of methods or functions to the entire dataset or specific segments. This capability expands the horizons of efficient and comprehensive data analysis [**Betancourt:2019**]. Find it here: `https://link.springer.com/chapter/10.1007/978-1-4842-5001-3_3`

22.5.4. Python for Data Analysis

The provided Books page with the book titled "Python for Data Analysis" by Wes McKinney. This book extensively covers the pandas library, a key tool for data analysis in Python. The link allows access to the book for further exploration of pandas and its applications in the context of data analysis using Python. [McK12]. Access it here: `https://books.google.de/books?hl=en&lr=&id=v3n4_AK8vu0C&oi=fnd&pg=PR3&dq=pandas+library&ots=riDL2lxAsD&sig=652ON3D9JVUQ4BBtgvBuo-4tdgc&redir_esc=y#v=onepage&q=pandas%20library&f=false`

22.5.5. Pandas Learning

The provided link directs to a Google Books page for the book titled "Python for Data Science For Dummies" by John Paul Mueller and Luca Massaron. This book likely covers various aspects of Python for data science, including the pandas library. The link allows access to the book for further exploration of pandas and its applications within the broader context of data science using Python [**Heydt:2017**]. You can find the official Pandas learning book here: `https://books.google.de/books?hl=en&lr=&id=Sng5DwAAQBAJ&oi=fnd&pg=PP1&dq=pandas+library&ots=J-L7JhMo75&sig=9RaIsH-obmG0vNc9_qjL6AnNUHA&redir_esc=y#v=onepage&q=pandas%20library&f=false`

Remember to explore the official documentation first as it covers the most up-to-date information about Pandas. The books, online resources, and video tutorials mentioned above will complement your learning and provide additional insights into working with Pandas.

23. Keras

Introduction to Keras

Keras and TensorFlow are among the most widely adopted frameworks in deep learning, each offering distinct strengths and capabilities.[Nam24] Keras, initially introduced as a high-level API for neural network development, is renowned for its simplicity and user-friendly design, enabling researchers and developers to prototype and build sophisticated models with minimal code. Supporting multiple backends, including TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK), Keras provides flexibility and ease of integration.[Nam24] On the other hand, TensorFlow, developed by the Google Brain team, is a robust and versatile framework that offers a comprehensive suite of tools for designing, training, optimizing, and deploying machine learning models.[Nam24] With both high-level and low-level APIs, TensorFlow caters to a wide range of applications, from exploratory research to large-scale production systems, and includes an extensive ecosystem with tools like TensorFlow Extended (TFX), TensorFlow Lite, and TensorFlow Serving.[Nam24]
The integration of Keras as the official high-level API for TensorFlow combines the simplicity of Keras with the power and scalability of TensorFlow, providing a unified and seamless environment for deep learning development and deployment.[Nam24] This paper provides an in-depth review of Keras and TensorFlow, highlighting their features, capabilities, and applications. We begin by discussing the evolution of deep learning frameworks and the contributions of Keras and TensorFlow in advancing the field. Next, we examine related studies that compare these frameworks and their applications, followed by a detailed methodology for evaluating their performance and capabilities.[Nam24]

Description of Keras

Keras is an open-source, high-level neural networks API written in Python, designed to facilitate fast experimentation and efficient model development in deep learning. Initially developed as an interface for TensorFlow, it now supports multiple backends, including TensorFlow, JAX, and PyTorch. Keras emphasizes ease of use with a clean, human-centric API, enabling developers to quickly build and train models with minimal code. It also promotes rapid debugging, code elegance, and performance optimization, while being highly deployable across various platforms. Keras models are scalable and can be used in both research and industrial applications, with widespread adoption in organizations like CERN, NASA, and NIH.[Nam24]

Understanding the Structure of Keras

At the core of Keras are several essential components that together create the framework's structural integrity. Understanding these components is crucial for effective utilization of Keras, as they play pivotal roles in the model-building process.[Nam24]

Models

In Keras, models are abstractions that represent neural network architectures. There are two primary types of models in Keras:

- **Sequential Model:** A linear stack of layers, suitable for simple feedforward networks and problems with a uniform sequence of layers.
- **Functional API:** Allows for the creation of complex architectures with shared layers, multiple inputs, and multiple outputs, catering to advanced use cases like multi-task learning.[Ker24]

Layers

Layers are the building blocks of any neural network, and Keras provides a variety of pre-built layers to accommodate different functions:

- **Dense Layers:** Fully connected layers that apply linear transformations, crucial for fully connected network architectures.
- **Convolutional Layers:** Used for image processing tasks, enabling the network to learn spatial hierarchies from images.
- **Recurrent Layers:** Designed for sequence data, ideal for tasks such as natural language processing (NLP).[Ker24]

Activations

Activation functions define the output of each layer in a network, injecting non-linearity into the model. Common activation functions include:

- **ReLU:** Mitigates the vanishing gradient problem, widely used in hidden layers.
- **Sigmoid:** Maps predictions to probabilities between 0 and 1, ideal for binary classification tasks.
- **Softmax:** Used in the output layer for multi-class classification problems.[Ker24]

Training and Evaluation in Keras

The Compile Method

The `compile` method specifies the optimizer, loss function, and evaluation metrics. Popular optimizers include:

- **Adam:** Combines the benefits of both RMSprop and SGD.
- **RMSprop:** Suitable for recurrent neural networks.
- **SGD:** Effective in convergence with large datasets.

The Fit Method

The `fit` method trains the model using backpropagation and gradient descent. Users can specify:

- Batch size
- Number of epochs
- Validation data

Callbacks, such as early stopping, can be implemented for enhanced training strategies.

Model Evaluation and Prediction

Post-training, the `evaluate` method quantifies the model's performance on unseen data, while the `predict` method generates predictions for new inputs.[Ker24]

Manual

23.0.1. Installation of keras

- **TensorFlow:**
  - TensorFlow is a versatile and widely-used machine learning framework, maintained as an open-source project.
  - It includes tools for building, training, evaluating, and deploying models.
  - Keras, its high-level API, simplifies the creation and training of deep learning networks.
  - TensorFlow Lite enables deploying TensorFlow models on mobile and embedded devices.[Ker24]
- **Using Google Colab:**
  - Colab is recommended for running TensorFlow code interactively.
  - A button in the notebook allows you to load it directly into Colab.
  - If issues arise when accessing a GitHub-hosted notebook, manually construct the URL for Colab.[Ker24]
- **Notebook Preparation:**
  - Before running the code, clear all existing outputs for a fresh start.
- **Dependencies:**
  - Install TensorFlow (`tensorflow==2.0`) and import essential libraries like NumPy and Matplotlib for mathematical operations and visualization.[Ker24]

23.0.2. Code Example:

To set up your environment, use the following code snippet:

Listing 23.1: Example command to check Python version

```
# Install TensorFlow
!pip install tensorflow==2.0

# Import TensorFlow
import tensorflow as tf

# Import NumPy for numerical computations
import numpy as np

# Import Matplotlib for visualizations
import matplotlib.pyplot as plt
```

Installing Keras on Different Environments

1. **Installing Keras with TensorFlow (Recommended):**

Listing 23.2: Installation of keras in tensor flow

```
pip install tensorflow
python -c "import tensorflow as tf; print(tf.keras.version)"
```

This method ensures compatibility between TensorFlow and Keras

2. **Standalone Keras Installation (Legacy):**

Listing 23.3: installation of keras in Legacy version

```
pip install keras
python -c "import keras; print(keras.__version__)"
```

Note: Some features may not be supported without TensorFlow as the backend.

3. **Installing on Windows:**
   - Open Command Prompt with administrator privileges.
   - Install TensorFlow (and Keras with it):

Listing 23.4: Installation of keras on Windows

```
pip install tensorflow
```

   - For hardware-specific optimizations, install TensorFlow with GPU support:

Listing 23.5: Installation of Keras with GPU support

```
pip install tensorflow-gpu
```

4. **Installing on macOS:**

Listing 23.6: Installation of Keras on macOS

```
pip install tensorflow
brew install python3
python3 -c "import tensorflow as tf; print(tf.keras.__version__)"
```

5. **Installing on Linux:**

Listing 23.7: Installation of Keras on Linux

```
python3 -m pip install --upgrade pip
pip install tensorflow
pip install tensorflow-gpu
python3 -c "import tensorflow as tf; print(tf.keras.__version__)"
```

6. **Installing in Virtual Environments:**

Listing 23.8: Installation of Keras on Virtual Env

```
python3 -m venv keras_env
source keras_env/bin/activate  # On Windows: keras_env\Scripts\activate
pip install tensorflow
```

Verify the installation inside the virtual environment.

7. **Installing for TensorFlow Lite for Microcontrollers:**

Listing 23.9: Installation of Keras on Microcontroller

```
pip install flatbuffers
pip install tensorflow
```

Set up Arduino IDE or other microcontroller-specific tools to deploy models.

23.0.3. Verify TensorFlow Installation

To verify that TensorFlow is installed correctly, run the following command in your terminal or command prompt:

- **Check TensorFlow Version:**

Listing 23.10: Checking TensorFlow Version

```
python -c "import tensorflow as tf; print(tf.__version__)"
```

This command should display the TensorFlow version number. Ensure that the version is compatible with the Keras and TensorFlow Lite requirements for your project.

23.0.4. Create a Test Keras Model

Test the Keras installation by creating a simple model with the following Python script:

- **Create and Compile a Keras Model:**

Listing 23.11: Creating a Keras Model

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Create a simple model
model = Sequential([
Dense(10, activation='relu', input_shape=(5,)),
Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics
    ↪ =['accuracy'])

print("Model created successfully.")
```

If the script runs without errors, Keras is functioning correctly.

23.0.5. Test TensorFlow Lite Conversion

To test TensorFlow Lite conversion, use the following script:

- **Convert the Keras Model to TensorFlow Lite:**

Listing 23.12: TensorFlow Lite Conversion

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model as a .tflite file
with open("model.tflite", "wb") as f:
f.write(tflite_model)

print("Model converted to TensorFlow Lite successfully.")
```

Make sure that a file named `model.tflite` is created in your working directory. This ensures that the conversion process works.

23.0.6. Arduino Compatibility Test

If you plan to deploy the model on an Arduino, follow these steps:

- Install the Arduino IDE.
- Install the TensorFlow Lite Arduino library.
- Convert the `model.tflite` file to a C array using the `xxd` tool:

Listing 23.13: Converting Model to C Array

```
xxd -i model.tflite > model.cc
```

This generates a C array from the TensorFlow Lite model that can be included in your Arduino project.

23.0.7. How to import

Installation

```
pip install keras
```

Getting Started

Once Keras is installed, you can import it in your Python scripts or interactive sessions:

Listing 23.14: Example of gesture data with sensor readings

```
import keras
```

Creating a Sequential Model

Keras provides a Sequential model API for building neural networks layer-by-layer. Here's an example of how to create a simple Sequential model with Keras:

Listing 23.15: Example of a neural network model with Keras

```
from keras.models import Sequential
from keras.layers import Dense

# Initialize the model
model = Sequential()

# Add layers to the model
model.add(Dense(units=64, activation='relu', input_shape=(100,)))
model.add(Dense(units=10, activation='softmax'))

# Compile the model
model.compile(loss='categorical_crossentropy',
optimizer='sgd',
metrics=['accuracy'])
```

Training the Model

Once the model is defined, you can train it on your data. Here's an example of how to train the model:

Listing 23.16: Training the Keras model

```
# Assuming X_train and y_train are your training data
model.fit(X_train, y_train, epochs=10, batch_size=32)
```

23.0.8. Run a Pre-trained Example

To further verify your setup, use a pre-trained TensorFlow Lite model from the TensorFlow repository. Deploy it on your target device and confirm that it produces the expected output.

Evaluating the Model

After training, you can evaluate the performance of the model on your test data:

Listing 23.17: Evaluating the Keras model

```
# Assuming X_test and y_test are your test data
loss, accuracy = model.evaluate(X_test, y_test)
print("Test Loss:", loss)
print("Test Accuracy:", accuracy)
```

For a detailed guide covering in-depth usage of different parts of the Keras API, you can refer to the Keras developer guides. These guides are deep-dives into specific topics such as layer subclassing, fine-tuning, or model saving.[**Keinstall:2024**]

23.0.9. Important attributes of keras

1. **Simplicity**: Keras is designed to be simple and easy to use. It provides a high-level interface that abstracts away many complexities of building neural networks, making it accessible to beginners and experts alike.

2. **Modularity**: Keras allows for easy construction of complex neural network architectures through a modular approach. Neural network models are built by stacking layers on top of each other, and each layer can be easily added, removed, or modified.

3. **Flexibility**: Keras supports both sequential and functional API for building models. The sequential API is used for simple linear stacks of layers, while the functional API allows for more complex architectures with shared layers, multiple inputs, and multiple outputs.[**Attrik:2020**]

4. **Compatibility**: Keras is compatible with multiple backend engines, including TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK). This allows users to choose the backend that best suits their needs.

5. **Customization**: Keras provides a wide range of built-in layers, loss functions, optimizers, and metrics. Additionally, users can easily define custom layers, loss functions, and metrics to meet specific requirements.

6. **Integration**: Keras seamlessly integrates with other popular Python libraries and frameworks, such as TensorFlow and scikit-learn. This allows for easy interoperability and integration into existing workflows and projects.

7. **Community and Documentation**: Keras has a large and active community of developers and users who contribute to its development and provide support. The official documentation is comprehensive and includes tutorials, guides, and API references to help users get started and learn more about Keras.

These attributes make Keras a powerful and versatile tool for building and training neural networks for a wide range of applications, including the Magic Wand project with the Arduino Nano 33 BLE Sense.

23.1. Example - Version

The version information for Keras can be retrieved programmatically within Python environment.

Listing 23.18: Getting the version of Keras

```
import keras

# Get the version of Keras
keras_version = keras.__version__

# Print the version
print("Keras Version:", keras_version)
```

You can run this code in your Python environment to print out the version of Keras that you have installed. This information can be useful for ensuring compatibility and understanding which features are available in your Keras installation for your Magic Wand project with the Arduino Nano 33 BLE Sense.[**Vke:2024**]

23.1.1. updating the version of Keras in Python

To update the version of Keras in Python, you can use the pip package manager. Here are the steps:

1. Open your command line or terminal.
2. Run the following command:

   Listing 23.19: Upgrading Keras using pip

   ```
   pip install --upgrade keras
   ```

   This command will upgrade the Keras package to the latest version available on the Python Package Index (PyPI).
3. Once the upgrade is complete, you can verify the updated version by running:

   Listing 23.20: Check the version of Keras using Python

   ```
   python -c "import keras; print(keras.__version__)"
   ```

   This command will print out the version of Keras installed in your Python environment, confirming that the update was successful.

By following these steps, you can ensure that you have the latest version of Keras installed.[**Lak:2023**]

23.2. Example

This document demonstrates how to use TensorFlow Lite with the Arduino Nano 33 BLE Sense. Specifically, it uses data from the `LSM9DS1` accelerometer to classify gestures or motion using a simple TensorFlow model.

23.2.1. Step 1: Create and Compile a Keras Model

- **Create and Compile a Keras Model:**

  Listing 23.21: Creating a Keras Model

  ```
  import tensorflow as tf
  from tensorflow.keras import Sequential
  from tensorflow.keras.layers import Dense
  ```

```
import numpy as np

# Example training data: Accelerometer data (X, Y, Z)
train_data = np.array([[0.1, 0.2, 0.3], [0.4, 0.5, 0.6], [0.7, 0.8,
    ↪  0.9]])  # Example accelerometer data
labels = np.array([0, 1, 0])  # Example labels for classification (
    ↪ gesture class)

# Create a simple model for classification
model = Sequential([
Dense(16, activation='relu', input_shape=(3,)),  # 3 input features
    ↪  (X, Y, Z)
Dense(1, activation='sigmoid')  # Binary output (gesture class 0 or
    ↪  1)
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics
    ↪ =['accuracy'])

# Train the model
model.fit(train_data, labels, epochs=10)

# Convert the model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model as a .tflite file
with open("model.tflite", "wb") as f:
f.write(tflite_model)

print("Model converted to TensorFlow Lite successfully.")
```

23.2.2. Step 2: Convert the Model to C Array

- **Convert the Model to C Array:**

Listing 23.22: Converting the Model to C Array

```
xxd -i model.tflite > model.cc
```

23.2.3. Step 3: Arduino Code to Run the TensorFlow Lite Model

- **Arduino Code to Run the TensorFlow Lite Model:**

Listing 23.23: Arduino Code to Run the TensorFlow Lite Model

```
#include <Arduino.h>
#include <TensorFlowLite.h>
#include <Wire.h>
#include <LSM9DS1.h>

// Include the generated C array from the TensorFlow Lite model
extern "C" {
    #include "model.cc"
}

// Instantiate the LSM9DS1 sensor object
LSM9DS1 imu;

// Define the TensorFlow Lite interpreter and model
tflite::MicroInterpreter* interpreter;
tflite::Model* model;
tflite::MicroAllocator* micro_allocator;
tflite::MicroInterpreter* micro_interpreter;

void setup() {
    Serial.begin(9600);
    Wire.begin();

    // Initialize the IMU sensor (LSM9DS1)
    if (!imu.begin()) {
```

```
            Serial.println("Failed to initialize IMU sensor!");
            while (1);
        }

        // Load the TensorFlow Lite model
        model = tflite::GetModel(model_tflite);
        interpreter = tflite::MicroInterpreter(model, micro_allocator,
            ↪ tflite::kTensorArenaSize);

        Serial.println("Setup complete!");
    }

    void loop() {
        // Read accelerometer data from LSM9DS1 sensor
        imu.readAccel();
        float x = imu.accelX();
        float y = imu.accelY();
        float z = imu.accelZ();

        // Prepare the input tensor (accelerometer data)
        float input_data[] = {x, y, z};
        float* input = interpreter->input(0)->data.f;
        for (int i = 0; i < 3; i++) {
            input[i] = input_data[i];
        }

        // Invoke the model to make predictions
        interpreter->Invoke();

        // Get the output tensor
        float* output = interpreter->output(0)->data.f;

        // Display the predicted output
        if (output[0] > 0.5) {
            Serial.println("Gesture Class: 1");
        } else {
            Serial.println("Gesture Class: 0");
        }

        delay(500);  // Delay before the next prediction
    }
```

23.2.4. Reshaping and Flattening

Reshaping and flattening operations are essential when preparing data for neural network models.

Reshaping with `tf.reshape()`: The `tf.reshape()` function allows you to change the shape of a tensor without changing its data. This is particularly useful when the input data needs to be reshaped to fit the input requirements of the model.

Listing 23.24: Reshaping a Tensor

```
import tensorflow as tf

# Create a tensor with shape (4, 3)
tensor = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

# Reshape the tensor to shape (2, 6)
reshaped_tensor = tf.reshape(tensor, (2, 6))
print(reshaped_tensor)
```

Flattening with the `Flatten()` Layer: In convolutional neural networks, the output of convolutional layers is often a multi-dimensional tensor. The `Flatten` layer is used to convert this tensor into a one-dimensional vector suitable for dense layers.

Listing 23.25: Flattening a Tensor

```
from tensorflow.keras.layers import Flatten

# Example input tensor of shape (batch_size, height, width, channels)
input_tensor = tf.random.normal([32, 64, 64, 3])  # Batch of 32 images, 64x64
    ↪  pixels, 3 color channels
```

```
flattened_tensor = Flatten()(input_tensor)
print(flattened_tensor.shape)  # Output shape will be (32, 64*64*3)
```

23.2.5. Stacking and Splitting Arrays

Handling multiple input or output arrays can be managed with `tf.stack()` and `tf.split()`.
Stacking Arrays with `tf.stack()`: The `tf.stack()` function stacks a list of tensors along a new axis.

Listing 23.26: Stacking Arrays

```
import tensorflow as tf

# Two 1D tensors
tensor_a = tf.constant([1, 2, 3])
tensor_b = tf.constant([4, 5, 6])

# Stack the tensors along the first axis
stacked_tensor = tf.stack([tensor_a, tensor_b])
print(stacked_tensor)
```

Splitting Arrays with `tf.split()`: The `tf.split()` function splits a tensor into smaller tensors along a specified axis.

Listing 23.27: Splitting Arrays

```
import tensorflow as tf

# A 1D tensor
tensor = tf.constant([1, 2, 3, 4, 5, 6])

# Split the tensor into 3 parts along the first axis
split_tensors = tf.split(tensor, 3)
print(split_tensors)
```

23.3. Managing Keras Versions

23.3.1. Check Installed Version

To check the version of Keras installed in your environment, use the following command:

Listing 23.28: Check Installed Keras Version

```
pip show keras
```

This command will display the installed version of Keras, along with other package information.

23.3.2. Upgrade to the Latest Version

To ensure you have the latest version of Keras installed, use the following command:

Listing 23.29: Upgrade Keras

```
pip install --upgrade keras
```

This will upgrade your installation to the latest version.

23.4. File Interaction: Saving and Loading Models

23.4.1. Saving Models

To save a trained Keras model, use the `model.save()` method. This stores the model's architecture, weights, and training configuration in a file.

Listing 23.30: Saving a Keras Model

```
# Save the trained model to a file
model.save('model.h5')
```

23.4.2. Loading Models

To reload a saved model, use `tf.keras.models.load_model()`:

Listing 23.31: Loading a Keras Model

```
from tensorflow.keras.models import load_model

# Load the model from the .h5 file
model = load_model('model.h5')
```

23.5. Error Handling

23.5.1. Common Issue: "Model size exceeds memory on microcontroller."

This issue occurs when the size of the TensorFlow model exceeds the available memory of a microcontroller. The solution to this problem is to quantize the model using TensorFlow Lite's post-training quantization.

Listing 23.32: Post-Training Quantization

```
import tensorflow as tf

# Convert the model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)

# Apply post-training quantization
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

# Save the quantized model
with open('quantized_model.tflite', 'wb') as f:
f.write(tflite_model)
```

This reduces the model size and makes it more suitable for microcontroller deployment.

23.5.2. Common Issue: "Incompatible Keras version."

Incompatibility between Keras and TensorFlow versions can lead to errors. To resolve this, ensure that the versions of Keras and TensorFlow you are using are compatible.
1. Uninstall the current versions of Keras and TensorFlow:

Listing 23.33: Uninstall Keras and TensorFlow

```
pip uninstall keras tensorflow
```

2. Install compatible versions:

Listing 23.34: Install Compatible Versions

```
pip install tensorflow
```

Alternatively, for specific versions:

Listing 23.35: Install Specific Version of Keras

```
pip install keras==2.9.0
```

23.6. Example- Files

23.6.1. Saving a Model

Keras provides a simple way to save both the architecture and weights of a trained model, allowing you to load it later for inference or further training.
To save a model in Keras, use the `model.save()` method:

Listing 23.36: Saving a Keras Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Define a simple model
model = Sequential([
Dense(10, activation='relu', input_shape=(5,)),
Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
    ↪ accuracy'])

# Save the model to a file (HDF5 format)
model.save('my_model.h5')
```

23.6.2. Loading a Saved Model

To load a previously saved model, use the `load_model()` function from `tensorflow.keras.models`:

Listing 23.37: Loading a Keras Model

```
from tensorflow.keras.models import load_model

# Load the model from the file
model = load_model('my_model.h5')

# Use the model for predictions or further training
predictions = model.predict(input_data)
```

23.6.3. Saving Only Model Weights

If you only want to save the weights (not the entire model), you can use the `save_weights()` method:

Listing 23.38: Saving Only Model Weights

```
# Save the model weights
model.save_weights('my_model_weights.h5')
```

To load the weights back into a model:

Listing 23.39: Loading Model Weights

```
# Load the model structure
model = load_model('my_model_structure.h5')

# Load the weights into the model
model.load_weights('my_model_weights.h5')
```

23.6.4. Saving and Loading Training Data

23.6.5. Saving Data to a CSV File

Suppose you have training data in a NumPy array and want to save it to a CSV file. You can use `numpy.savetxt()`:

Listing 23.40: Saving Data to a CSV File

```
import numpy as np

# Sample data: 100 samples with 5 features each
data = np.random.random((100, 5))

# Save the data to a CSV file
np.savetxt('training_data.csv', data, delimiter=',')
```

23.6.6. Loading Data from a CSV File

To load data back from a CSV file, you can use `numpy.loadtxt()` or `pandas.read_csv()` (if you prefer using pandas for convenience):

Listing 23.41: Loading Data from a CSV File

```
import numpy as np

# Load the data from the CSV file
data = np.loadtxt('training_data.csv', delimiter=',')
```

Alternatively, using pandas for loading the CSV:

Listing 23.42: Loading Data with Pandas

```
import pandas as pd

# Load the data into a DataFrame
data = pd.read_csv('training_data.csv', header=None)
```

23.6.7. Saving Data to HDF5

If you want to store large datasets efficiently, consider using the HDF5 format. You can use the `h5py` library to save and load data in HDF5 format:

Listing 23.43: Saving Data to HDF5

```
import h5py

# Create a sample dataset
data = np.random.random((100, 5))

# Save the data to an HDF5 file
with h5py.File('data.h5', 'w') as f:
f.create_dataset('dataset', data=data)
```

23.6.8. Loading Data from HDF5

To load data from an HDF5 file, use `h5py.File`:

Listing 23.44: Loading Data from HDF5

```
import h5py

# Load the data from the HDF5 file
with h5py.File('data.h5', 'r') as f:
data = f['dataset'][:]
```

23.6.9. Processing Image Files for Keras

Keras offers utility functions for loading and processing image data, especially for tasks like classification. The `ImageDataGenerator` class provides real-time data augmentation and loading from directories.

23.6.10. Loading and Processing Images with `ImageDataGenerator`

Listing 23.45: Loading and Processing Images

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Define an ImageDataGenerator for real-time augmentation
datagen = ImageDataGenerator(
rescale=1./255,  # Normalize the images to [0, 1]
rotation_range=40,  # Randomly rotate images
width_shift_range=0.2,  # Randomly shift images horizontally
height_shift_range=0.2,  # Randomly shift images vertically
shear_range=0.2,  # Random shear transformations
zoom_range=0.2,  # Random zoom
horizontal_flip=True,  # Randomly flip images horizontally
fill_mode='nearest'  # Fill missing pixels after transformations
)

# Load images from a directory
train_generator = datagen.flow_from_directory(
'path_to_train_data',  # Path to the training images
target_size=(150, 150),  # Resize images to 150x150
batch_size=32,
class_mode='binary'  # Binary labels (for binary classification)
)
```

23.6.11. Saving Processed Image Data

While you typically don't save processed image data in the same way as model weights, you can save transformed images if needed. You can save them using `ImageDataGenerator` or `PIL`:

Listing 23.46: Saving Processed Image Data

```
from tensorflow.keras.preprocessing.image import img_to_array, array_to_img
from PIL import Image

# Example: Save a transformed image
img = Image.open('image.jpg')
img_array = img_to_array(img)

# Perform some transformation (example: flipping)
img_array = np.fliplr(img_array)

# Convert back to image and save
img_processed = array_to_img(img_array)
img_processed.save('flipped_image.jpg')
```

23.6.12. Saving and Loading Text Data

If you're working with text data, such as in natural language processing (NLP), you can save and load data using standard text file handling methods.

23.6.13. Saving Text Data to a File

Listing 23.47: Saving Text Data to a File

```
text_data = ["This is an example sentence.", "This is another example."]
```

```
# Save the text data to a text file
with open('text_data.txt', 'w') as f:
for line in text_data:
f.write(line + "\n")
```

23.6.14. Loading Text Data from a File

Listing 23.48: Loading Text Data from a File

```
# Load the text data from a file
with open('text_data.txt', 'r') as f:
text_data = f.readlines()

# Remove newline characters
text_data = [line.strip() for line in text_data]
```

23.7. Further Reading

Diving deeper into Keras and enhancing your skills in deep learning can be significantly aided by engaging with diverse educational materials. The following resources have been carefully selected to provide comprehensive learning opportunities:

23.7.1. TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers

Authored by Pete Warden, Daniel Situnayake, and others, this book is a comprehensive guide to deploying machine learning models on resource-constrained devices. It provides a practical, hands-on approach to using TensorFlow Lite to create and optimize models that run efficiently on microcontrollers such as Arduino boards. Ideal for those exploring the intersection of machine learning and embedded systems.[WS20a]

23.7.2. Deep Learning with Python (Second Edition)

Authored by François Chollet, the creator of Keras, this book provides an in-depth exploration of deep learning concepts, illustrated with practical examples using Keras. Covering key topics like neural network architecture, optimization, and advanced techniques, it is an excellent resource for building a solid foundation in deep learning while leveraging Keras's intuitive API. [Cho18]

23.7.3. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow

Written by Aurélien Géron, this comprehensive guide focuses on practical, hands-on applications of machine learning and deep learning using modern Python libraries. Combining theory and real-world examples, it covers Keras and TensorFlow for neural networks, reinforcement learning, and deep generative models.[Gér19]

23.7.4. Keras Documentation

The official Keras documentation is the most authoritative resource for learning about Keras features, including API guides, tutorials, and examples. Whether you're a beginner or an advanced user, this resource offers detailed explanations of Keras functionalities, model building, and deployment. [Ker24]

23.7.5. TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning

Authored by Bharath Ramsundar and Reza Bosagh Zadeh, this book focuses on TensorFlow and Keras as tools for solving a wide range of machine learning problems. It covers practical examples, from basic neural networks to advanced topics like reinforcement learning and transfer learning, with an emphasis on model building and deployment. [RZ18]

23.7.6. Advanced Deep Learning with Keras

Written by Rowel Atienza, this book dives into advanced deep learning techniques using Keras, including working with sequential data, generative adversarial networks (GANs), and implementing deep reinforcement learning. It is ideal for experienced users looking to enhance their knowledge of advanced Keras features and applications. [AS20]

23.7.7. TensorFlow Lite for Microcontrollers Guide

For those interested in deploying Keras models on edge devices and microcontrollers, the TensorFlow Lite for Microcontrollers guide offers in-depth resources and examples for optimizing and converting models for lightweight environments. [Ten24]
Each of these resources provides unique insights and practical knowledge to help both novices and seasoned Keras users expand their expertise in deep learning.

24. Matplotlib

24.1. Introduction

Matplotlib is a powerful and widely used Python library for creating visualizations and plots. It provides a comprehensive set of tools for generating a wide range of static, animated, and interactive visual representations of data. Matplotlib is highly flexible and customizable, allowing users to create visually appealing and informative plots for various purposes, such as data analysis, scientific research, presentations, and more. It is built on top of NumPy, another popular Python library for numerical computing, which makes it compatible with other scientific computing libraries in the Python ecosystem.
The version of the Matplotlib package used here is 3.5.2. All these functions of the Matplotlib package are explained in the upcoming sections.

24.2. Description

Matplotlib is a fundamental tool in the data science and visualization toolkit of Python, along with libraries like Pandas and NumPy. With Matplotlib, users can create a wide range of plots, including line plots, scatter plots, bar plots, histograms, pie charts, 3D plots, and more. It provides fine-grained control over plot elements such as axes, labels, colors, markers, and legends. Additionally, Matplotlib integrates seamlessly with Jupyter notebooks, making it a popular choice for interactive data exploration and visualization.
Key features and functions of Matplotlib function:

1. **Plotting Functions:** Matplotlib provides various plotting functions to create different types of plots, such as line plots, scatter plots, bar plots, histograms, pie charts, box plots, and more. These functions allow users to visualize data in different formats and representations.

2. **Customization Options:** Matplotlib offers extensive customization options to control every aspect of a plot. Users can modify plot elements like axes, labels, titles, colors, markers, line styles, legends, and annotations. This flexibility allows users to tailor the appearance of the plots to their specific requirements.

3. **Multiple Subplots:** Matplotlib enables the creation of multiple subplots within a single figure, allowing users to display and compare different plots side by side. This feature is particularly useful when visualizing multiple datasets or different aspects of a dataset simultaneously.

4. **3D Plotting:** Matplotlib includes capabilities for creating 3D plots and visualizations. Users can generate 3D scatter plots, surface plots, wireframe plots, and other three-dimensional representations of data. These plots are helpful when working with spatial or volumetric data.

5. **Colormaps:** Matplotlib provides a wide range of built-in colormaps for mapping data values to colors. Colormaps allow users to add depth and meaning to their plots, emphasizing patterns or variations in the data. Matplotlib also supports custom colormaps, giving users flexibility in choosing the color scheme that best suits their needs.

24.2.1. Data Suitability

When it comes to data suitability, Matplotlib is a versatile library that can handle a wide range of data types and formats.

- **Numerical Data:** Matplotlib is particularly well-suited for visualizing numerical data. Whether it's a simple line plot, scatter plot, histogram, or more complex plots like contour plots or heatmaps, Matplotlib provides the necessary tools to effectively represent and analyze numerical data. It offers customization options to highlight trends, patterns, and relationships within the data.

- **Categorical Data:** While Matplotlib is primarily designed for numerical data, it also supports visualizations for categorical data. Bar plots, pie charts, and stacked bar plots are useful for representing distributions, proportions, or comparisons among different categories. Matplotlib allows for customization of colors, labels, and other visual elements to enhance the understanding of categorical data.

- **Time Series Data:** Matplotlib is commonly used to visualize time series data. It provides various plot types suitable for displaying temporal trends, such as line plots, area plots, or candlestick plots. With Matplotlib, users can easily add date and time axes, format tick labels, and annotate events or significant time points. This makes it a valuable tool for analyzing and presenting data that changes over time.

- **Spatial Data:** Matplotlib also supports the visualization of spatial data. By leveraging its 3D plotting capabilities or using specialized modules like Basemap or Cartopy, Matplotlib can create maps, contour plots, and geospatial visualizations. It allows users to plot points, lines, polygons, and other geometric shapes on maps, making it suitable for analyzing and displaying spatial relationships.

- **Data Size:** Matplotlib can handle data of varying sizes, from small to large datasets. While it performs well with smaller datasets, it may face limitations in terms of performance and interactivity with extremely large datasets. In such cases, users may need to optimize the code, use data sampling techniques, or explore other specialized libraries for big data visualization.

- **Data Preprocessing:** Before visualizing data with Matplotlib, it is often necessary to preprocess and format the data appropriately. Matplotlib expects data to be in a suitable format, such as NumPy arrays or Pandas data structures. Therefore, users may need to apply data manipulation techniques using libraries like NumPy and Pandas to transform the data into the desired format before plotting.

24.3. Installation

24.3.1. Installing an official release

Matplotlib and its dependencies are available as wheel packages for macOS, Windows and Linux distributions:

Listing 24.1: Installing/upgrading pip and matplotlib

```
python -m pip install -U pip
python -m pip install -U matplotlib
```

24.3.2. Installing with Anaconda

Matplotlib is available both via the anaconda main channel

Listing 24.2: Installing matplotlib using conda

```
conda install matplotlib
```

as well as via the conda-forge community channel

Listing 24.3: Installing matplotlib from conda-forge

```
conda install -c conda-forge matplotlib
```

24.3.3. Required dependencies

Matplotlib requires the following dependencies:

Table 24.1.: Python Library Dependencies

Library	Minimum Version Required
Python	$\geq$ 3.6
FreeType	$\geq$ 2.3
libpng	$\geq$ 1.2
NumPy	$\geq$ 1.11
setuptools	–
cycler	$\geq$ 0.10.0
dateutil	$\geq$ 2.1
kiwisolver	$\geq$ 1.0.0
pyparsing	–

24.4. Example - Manual

24.4.1. General Concepts

"matplotlib" has an extensive codebase that can be daunting to many new users. However, most of matplotlib can be understood with a fairly simple conceptual framework and knowledge of a few important points.
Plotting requires action on a range of levels, from the most general (e.g., 'contour this 2-D array') to the most specific (e.g., 'color this screen pixel red'). The purpose of a plotting package is to assist you in visualizing your data as easily as possible, with all the necessary control – that is, by using relatively high-level commands most of the time, and still have the ability to use the low-level commands when needed.
Therefore, everything in matplotlib is organized in a hierarchy. At the top of the hierarchy is the matplotlib "state-machine environment" which is provided by the matplotlib.pyplot module. At this level, simple functions are used to add plot elements (lines, images, text, etc.) to the current axes in the current figure.
The next level down in the hierarchy is the first level of the object-oriented interface, in which pyplot is used only for a few functions such as figure creation, and the user explicitly creates and keeps track of the figure and axes objects. At this level, the user uses pyplot to create figures, and through those figures, one or more axes objects can be created. These axes objects are then used for most plotting actions.
For even more control – which is essential for things like embedding matplotlib plots in GUI applications – the pyplot level may be dropped completely, leaving a purely object-oriented approach.

How to import

To import functions from the Matplotlib library, you can use the import statement in Python. Here's how you can *import* Matplotlib functions:

Listing 24.4: Importing matplotlib for plotting

```
import matplotlib.pyplot as plt
```

Figure

The whole figure. The figure keeps track of all the child Axes, a smattering of 'special' artists (titles, figure legends, etc), and the canvas. (Don't worry too much about the canvas, it is crucial as it is the object that actually does the drawing to get you your plot, but as the user it is more-or-less invisible to you). A figure can have any number of Axes, but to be useful should have at least one.
The easiest way to create a new figure is with pyplot:

Listing 24.5: Creating a figure with matplotlib

```
fig = plt.figure()  # an empty figure with no axes
fig.suptitle('No axes on this figure')  # Add a title so we know which it is

fig, ax_lst = plt.subplots(2, 2)  # a figure with a 2x2 grid of Axes
```

Axes

This is what you think of as 'a plot', it is the region of the image with the data space. A given figure can contain many Axes, but a given Axes object can only be in one Figure. The Axes contains two (or three in the case of 3D) Axis objects (be aware of the difference between Axes and Axis) which take care of the data limits (the data limits can also be controlled via set via the set_xlim() and set_ylim() Axes methods). Each Axes has a title (set via set_title()), an x-label (set via set_xlabel()), and a y-label set via set_ylabel()).
The Axes class and its member functions are the primary entry point to working with the OO interface.

Axis

These are the number-line-like objects. They take care of setting the graph limits and generating the ticks (the marks on the axis) and ticklabels (strings labeling the ticks). The location of the ticks is determined by a Locator object and the ticklabel strings are formatted by a Formatter. The combination of the correct Locator and Formatter gives very fine control over the tick locations and labels.

Artist

Basically everything you can see on the figure is an artist (even the Figure, Axes, and Axis objects). This includes Text objects, Line2D objects, collection objects, Patch objects ... (you get the idea). When the figure is rendered, all of the artists are drawn to the canvas. Most Artists are tied to an Axes; such an Artist cannot be shared by multiple Axes, or moved from one to another.

24.4.2. Types of inputs to plotting functions

All of plotting functions expect `np.array` or `np.ma.masked_array` as input. Classes that are 'array-like' such as `pandas` data objects and `np.matrix` may or may not work as intended. It is best to convert these to `np.array` objects prior to plotting.
For example, to convert a **`pandas.DataFrame`**

Listing 24.6: Creating a DataFrame and converting it to a NumPy array

```
a = pandas.DataFrame(np.random.rand(4,5), columns = list('abcde'))
a_asarray = a.values
```

and to convert a `np.matrix`

Listing 24.7: Converting a NumPy matrix to an array

```
b = np.matrix([[1,2],[3,4]])
b_asarray = np.asarray(b)
```

24.4.3. Example

Listing 24.8: Example of a simple line plot using Matplotlib

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]

# Create a figure and axis
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y, marker='o', linestyle='-', color='b', label='Data')

# Set labels and title
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Line Plot')

# Add gridlines
ax.grid(True)

# Add legend
ax.legend()

# Display the plot
plt.show()
```

1. In this code example, we import the necessary modules from Matplotlib ('pyplot') and create sample data ('x' and 'y').

2. Create a figure and axis:

   Listing 24.9: Creating a subplot with Matplotlib

   ```
   fig, ax = plt.subplots()
   ```

   This creates a figure object ('fig') and an axis object ('ax'). The figure is the overall window or canvas, while the axis is the area within the figure where the plot is drawn.

3. Plot the data:

   Listing 24.10: Plotting data with a line plot

   ```
   ax.plot(x, y, marker='o', linestyle='-', color='b', label='Data')
   ```

   The 'plot()' function is used to create a line plot. Here, we specify the x and y data, marker style, line style, color, and label for the plot.

4. Set labels and title:

Listing 24.11: Setting labels and title for the plot

```
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')
ax.set_title('Line Plot')
```

The set_xlabel(), set_ylabel(), and set_title() functions are used to set the labels for the x-axis, y-axis, and the plot title, respectively.

5. Add legend:

Listing 24.12: Adding legend to the plot

```
ax.legend()
```

The 'legend()' function is used to add a legend to the plot, which shows labels for different elements of the plot.

6. Display the plot:

Listing 24.13: Displaying the plot

```
plt.show()
```

The 'show()' function is used to display the plot.

24.4.4. Matplotlib Error Handling

Error handling in Matplotlib, like in any other Python library, involves using try-except blocks to catch and handle potential errors that may occur during the execution of your code. Here's an example of how one can handle errors in Matplotlib:

Listing 24.14: Line Plot Example with Error Handling

```
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4]
y = [2, 4, 6]

try:
# Attempt to create a line plot
plt.plot(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot')
plt.show()

except ValueError as ve:
# Handle a specific type of error (ValueError in this case)
print(f"ValueError: {ve}")
# Take appropriate action to handle the error

except Exception as e:
# Handle any other type of exception
print(f"An error occurred: {e}")
# Take appropriate action to handle the error
```

In this example, a try-except block is used to handle potential errors that may occur during the creation and display of the line plot. Here's how the error handling works:

1. The code within the try block attempts to create a line plot using 'plt.plot()' and adds labels, title, and other plot elements.

2. If an error occurs during this process, the corresponding exception will be raised. In this example, we handle two types of exceptions:

a) **'ValueError':** This is a specific type of error that may occur if the input data for the plot is invalid or incompatible. We use the 'ValueError' as an example here, but you can handle other specific exceptions as needed.

b) **'Exception':** This is a generic exception that can handle any other type of exception that may occur during the execution of the code.

3. Inside each except block, you can define specific actions to be taken when the corresponding exception is encountered. In this example, we print out a customized error message and handle the error appropriately. You can replace the print statements with your desired error handling logic, such as logging the error, displaying a user-friendly message, or taking corrective actions.

24.5. Further Reading

24.5.1. Matplotlib Official Documentation

The official documentation is an extensive resource that provides comprehensive information about the Matplotlib library. It includes a user guide, API reference, tutorials, and examples. You can access it at:
`https://matplotlib.org/stable/users/index.html`

24.5.2. Mastering matplotlib

"Mastering Matplotlib" by [McG15] is a comprehensive guide that empowers readers to become proficient in creating professional-grade plots and visualizations using the Matplotlib library in Python. This book covers fundamental concepts, advanced techniques, and practical aspects to enhance data visualization skills. It can be accessed through:
`https://www.packtpub.com/product/mastering-matplotlib/9781783987542`

24.5.3. Scientific Visualization: Python + Matplotlib

"Scientific Visualization: Python + Matplotlib" by [Rou21] is a practical guide that demonstrates the use of Python and Matplotlib for scientific data visualization, catering to scientists, researchers, and data analysts. It offers insights into visualization techniques and customization to effectively present scientific data. It can be accessed through: `https://inria.hal.science/hal-03427242/`

24.5.4. Python for Data Analysis

This book, written by Wes McKinney (the creator of Pandas) [**Mckinney:python**], is a valuable resource for learning Pandas. It covers various aspects of data manipulation, analysis, and visualization using Pandas. The book also explores practical examples and real-world use cases. Find it here:
`https://www.oreilly.com/library/view/python-for-data/9781491957653/`

Literature

[AS20] R. Atienza and a. O. M. C. Safari. *Advanced Deep Learning with TensorFlow 2 and Keras - Second Edition*. Packt Publishing, 2020. URL: https://books.google.de/books?id=FJR5zQEACAAJ.

[Ada20] Adafruit. *Adafruit CircuitPython Documentation*. Accessed: 10 June 2024. 2020. URL: https://circuitpython.readthedocs.io.

[Ada21] Adafruit. *Adafruit LSM303DLHC Accelerometer and Magnetometer*. Accessed: 12 June 2024. 2021. URL: https://learn.adafruit.com/lsm303-accelerometer-and-compass.

[Ahm+13] N. Ahmad et al. "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications". In: *International Journal of Signal Processing Systems* 1.2 (2013). Emails: norhafizan, r_ariffin@um.edu.my, nazirahmkhairi@gmail.com, sebpeterkasi@gmail.com, pp. 256–262.

[Ala08] R. Alake. *Implementing AlexNet CNN Architecture Using TensorFlow 2.0+ and Keras*. 14.08.2020. URL: https://towardsdatascience.com/implementing-alexnet-cnn-architecture-using-tensorflow-2-0-and-keras-2113e090ad98.

[Ard21] Arduino, ed. *Arduino Nano 33 BLE Sense Rev2 with headers*. 2021. URL: https://store.arduino.cc/products/nano-33-ble-sense-rev2-with-headers.

[Ard22] Arduino, ed. *Arduino Documentation*. 2022. URL: https://docs.arduino.cc/.

[Ard23] Arduino, ed. *Arduino Nano 33 BLE Sense Rev2 - Product Reference Manual*. [pdf]. 2023. URL: https://docs.arduino.cc/resources/datasheets/ABX00069-datasheet.pdf.

[Ava15] Avago Technologies, ed. *APDS-9960 - Digital Proximity, Ambient Light, RGB and Gesture Sensor*. [pdf]. 2015. URL: https://docs.broadcom.com/doc/AV02-4191EN.

[Big21] S. J. Bigelow. *What is edge computing? Everything you need to know*. Accessed: 2023-02-26. 2021. URL: https://www.techtarget.com/searchdatacenter/definition/edge-computing.

[Cho18] F. Chollet. *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[Cho+21] D. H. K. Chow et al. "Comparison between accelerometer and gyroscope in predicting level-ground running kinematics by treadmill running kinematics using a single wearable sensor". In: *Sensors* 21.14 (2021), p. 4633.

[DMM20] K. Dokic, M. Martinovic, and D. Mandusic. "Inference speed and quantisation of neural networks with TensorFlow Lite for Microcontrollers framework". In: *2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM)*. 2020, pp. 1–6. DOI: 10.1109/SEEDA-CECNSM49515.2020.9221846.

[Dai] *AI Magic Wand with TensorFlow Lite for Microcontrollers and Arduino | Google Codelabs*. Accessed: 2024-06-17. 2021. URL: https://codelabs.developers.google.com/magicwand.

[Dat] *Sending Data between two Nano 33 BLE Senses to make a magic wand*. Accessed: 2024-06-17. 2021. URL: https://forum.arduino.cc/t/sending-data-between-two-nano-33-ble-sens2021.

[FAD18] M. Fezari and A. Al Dahoud. "Integrated Development Environment "IDE" For Arduino". In: (Oct. 2018).

[GOP12] C. Gomez, J. Oller, and J. Paradells. "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology". In: *Sensors* 12.9 (2012), pp. 11734–11753. ISSN: 1424-8220. DOI: 10.3390/s120911734. URL: https://www.mdpi.com/1424-8220/12/9/11734.

[Gér19] A. Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O'Reilly Media, Inc., 2019. ISBN: 9781492032649.

[Goo24] Google AI. *TensorFlow Lite: ML for Mobile and Edge Devices*. Accessed: 2024-06-17. 2024. URL: https://www.tensorflow.org/lite.

[Gu+18] J. Gu et al. "Recent Advances in Convolutional Neural Networks". In: *Pattern Recognition* 77 (2018), pp. 354–377. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2017.10.013. URL: http://arxiv.org/pdf/1512.07108v6.

[Gyr23] GyroPlace. *Gyroscope Sensor Calibration: A Comprehensive Guide*. Accessed: 18 August 2023. 2023. URL: https://gyroplacecl.com/gyroscope-sensor-calibration-a-comprehensive-guide/.

[Han+17] D. Hansen et al. "Real-Time Barcode Detection and Classification using Deep Learning". In: (Jan. 2017), pp. 321–327. DOI: 10.5220/0006508203210327.

[Har23] S. Harris. *Inertial guidance: a brief history and overview*. Advanced Navigation. [pdf]. Jan. 3, 2023. URL: https://www.advancednavigation.com/tech-articles/inertial-guidance-a-brief-history-and-overview/ (visited on 12/09/2023).

[Hol19] J. Hollingworth. *HMC5883L Python Library*. 2019. URL: https://github.com/hollingworth.

[Hon12] Honeywell. *HMC5883L 3-Axis Digital Compass IC Datasheet*. 2012.

[Jet24] JetBrains. *Learn PyCharm*. Accessed: 2024-06-17. 2024. URL: https://www.jetbrains.com/pycharm/learn/.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc., 2012, pp. 1097–1105. DOI: 0.1145/3065386. URL: proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[Ker24] Keras Team. *Keras Documentation*. Available at: https://keras.io. 2024.

[LP18] R. Lukac and K. N. Plataniotis. *Color Image Processing: Methods and Applications*. Image Processing Series. CRC Press, 2018. ISBN: 9781420009781. URL: https://books.google.de/books?id=oD8qBgAAQBAJ.

[Lai22] R. Laine. "Bluetooth Low Energy data transfer in energy harvester system: Comparison of platforms and boards". MA thesis. 2022.

[Li+21] A. Li et al. “A Review on Lithium-Ion Battery Separators towards Enhanced Safety Performances and Modelling Approaches”. In: *Molecules* 26.2 (2021). DOI: 10.3390/molecules26020478. URL: https://www.mdpi.com/1420-3049/26/2/478.

[MO19] M. Munoz-Organero. “Outlier detection in wearable sensor data for human activity recognition (HAR) based on DRNNs”. In: *IEEE Access* 7 (2019), pp. 74422–74436.

[Mac+23] S. R. P. Maciel et al. “Development of Low-Cost Bench Tribometer for Wear Tests by Sliding”. In: *IX Simpósio Internacional de Inovação e Tecnologia*. Vol. 10. 5. Blucher, 2023, pp. 220–227. DOI: 10.5151/siintec2023-305900. URL: https://doi.org/10.5151/siintec2023-305900.

[Mak24] MakerGuides. *How to Use an MPU6050 3-Axis Accelerometer and 3-Axis Gyro Sensor with Arduino*. Accessed: 17 December 2024. 2024. URL: https://www.makerguides.com/how-to-use-an-mpu6050-3-axis-accelerometer-and-3-axis-gyrosensor-with-arduino/.

[McG15] D. McGreggor. *Mastering matplotlib*. Packt Publishing, 2015, p. 19. ISBN: 9781783987559. URL: https://books.google.de/books?id=D_0KCgAAQBAJ.

[McK12] W. McKinney. *Python for Data Analysis*. "O'Reilly Media, Inc.", 2012. ISBN: 9781449319793.

[Mic14] A. K. Microdevices. *AK8963 Datasheet*. 2014. URL: https://akm.com.

[Nam24] A. Name. “Advancements in Deep Learning: A Review of Keras and TensorFlow Frameworks”. In: *Journal of Scientific and Engineering Research* 11.5 (2024), pp. 282–286. ISSN: 2394-2630. URL: http://www.jsaer.com.

[Pas+17] V. M. N. Passaro et al. “Gyroscope Technology and Applications: A Review in the Industrial Perspective”. In: *Sensors* 17.10 (2017), p. 2284. DOI: 10.3390/s17102284. URL: https://www.mdpi.com/1424-8220/17/10/2284.

[QG17] U. Qureshi and F. Golnaraghi. “An Algorithm for the In-Field Calibration of a MEMS IMU”. In: *IEEE Sensors Journal* 17.22 (2017), pp. 7479–7486.

[RZ18] B. Ramsundar and R. Zadeh. *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. O'Reilly Media, 2018. ISBN: 9781491980408. URL: https://books.google.de/books?id=GZ1ODwAAQBAJ.

[Raj19] A. Raj. *Arduino Nano 33 BLE Sense Review - What's New and How to Get Started?* 2019. URL: https://circuitdigest.com/microcontroller-projects/arduino-nano-33-ble-sense-board-review-and-getting-started-guide.

[Rou21] N. P. Rougier. *Scientific Visualization: Python + Matplotlib*. Ed. by N. P. Rougier. [pdf]. Nov. 2021. URL: https://inria.hal.science/hal-03427242.

[SJM18] N. Sharma, V. Jain, and A. Mishra. “An Analysis Of Convolutional Neural Networks For Image Classification”. In: *Procedia Computer Science* 132 (2018), pp. 377–384. ISSN: 18770509. DOI: 10.1016/j.procs.2018.05.198.

[STH24] M. Sharma, A. Tomar, and A. Hazra. “Edge Computing for Industry 5.0: Fundamental, Applications, and Research Challenges”. In: *IEEE Internet of Things Journal* 11.11 (2024), pp. 19070–19093. DOI: 10.1109/JIOT.2024.3359297.

[STM24] STMicroelectronics. *LSM9DS1: iNEMO inertial module—3D accelerometer, 3D gyroscope, 3D magnetometer*. Accessed: 17 December 2024. 2024. URL: https://www.st.com/en/mems-and-sensors/lsm9ds1.html#overview.

[Sch07] J. Schanda. *Colorimetry: Understanding the CIE System*. Wiley, 2007. ISBN: 9780470175620. URL: https://books.google.de/books?id=uZadszSGe9MC.

[Sem13] N. Semiconductors. *MAG3110 Datasheet*. 2013. URL: https://nxp.com.

[She13] T. Sherwood. *Magnetic Sensors in Navigation Systems*. Berlin: Springer, 2013.

[Soc20] I. R. Society. "Robotics with Sensors". In: *IEEE Journal of Robotics and Automation* 18.4 (2020), pp. 112–124.

[Ten24] TensorFlow Lite Team. *TensorFlow Lite for Microcontrollers Guide*. Available at: https://www.tensorflow.org/lite/microcontrollers. 2024.

[WS20a] P. Warden and D. Situnayake. *TinyML – Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers*. [pdf]. O'Reilly Media, Incorporated, 2020, p. 504. ISBN: 978-1-492-05204-3.

[WS20b] P. Warden and D. Situnayake. *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly Media, 2020.

[Wil15] D. Williams. "Metal Detection Techniques Using Magnetometers". In: *Security Technology Review* 15.6 (2015), pp. 45–52.

[Xu+22] L. Xu et al. "Gesture recognition using dual-stream CNN based on fusion of sEMG energy kernel phase portrait and IMU amplitude image". In: *Biomedical Signal Processing and Control* 73 (2022), p. 103364. ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2021.103364. URL: https://www.sciencedirect.com/science/article/pii/S1746809421009617.

[Zho+20] Q. Zhou et al. "A Novel MEMS Gyroscope In-Self Calibration Approach". In: *Sensors* 20.18 (2020). Received: 18 August 2020; Accepted: 16 September 2020; Published: 22 September 2020, pp. 1–18. DOI: 10.3390/s20185063. URL: https://www.mdpi.com/1424-8220/20/18/5063.

Index