UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Exploring opportunities in TinyML

A study into TinyML-ODL and its future

Author
Juan Diego Rubio Serrano

Director
Dr. Felix Freitag
Department of Computer Architecture

A thesis submitted in fullfillment of the requirements for the degree of Informatics Engineering specializing in Computing

Barcelona, September 2022

Abstract

Internet of Things (IoT) has acquired useful and powerful advances thanks to the Machine Learning (ML) implementations. But the implementation of Machine Learning in IoT devices with data centers has some serious problems (data privacy, network bottleneck, etc). Tiny Machine Learning (TinyML) arose in order to have an independent edge device executing the ML program without the necessity of any data center. But there is still the need for high performance computers to train the ML model. But, can this situation improve?

This project goes through TinyML and two TinyML techniques capable to train the ML model on-device (what we call TinyML On-Device Learning or TinyODL): TinyML with Online-Learning (TinyOL) and Federated Learning (FL). We study both techniques in a theoretical analysis and try to develop one TinyODL app.

Resum: Internet of Things (IoT) ha obtingut uns forts avantatges molt usables gràcies a les implementacions del Machine Learning (ML). Però la implementació del Machine Learning en dispositius IoT utilitzant centres de dades porta una sèrie de problemes a tenir en compte (privacitat de les dades, el coll d'ampolla de la xarxa, etc.). Tiny Machine Learning (TinyML) va sorgir amb l'objectiu de tenir dispositious IoT independents executant el programa d'ML sense la necessitat d'un centre de dades. Però encara hi ha la necessitat de fer servir ordinadors d'alta potència per poder entrenar el model d'ML. Així i tot, es pot millorar aquesta situació?

Aquest projecte estudia el TinyML i dues de les seves tècniques, del que anomenem TinyML On-Device Learning o TinyODL, capaces d'entrenar el model d'ML en el mateix dispositiu (*on-device learning*): TinyML with Online-Learning (TinyOL) i Federated Learning (FL). S'estudien les dues tècniques des d'una anàlisi teòrica i provem de desenvolupar una aplicació TinyODL.

Resumen: Internet of Things (IoT) ha obtenido unas muy buenas y usables mejoras gracias a las implementaciones del Machine Learning (ML). Pero la implementación de Machine Learning en dispositivos IoT utilizando centros de datos conlleva una serie de problemas a tener en cuenta (privacidad de los datos, el cuello de botella de la red, etc.). Tiny Machine Learning (TinyML) surgió con el objetivo de tener dispotivios IoT independientes ejecutando el programa de ML sin la necesidad de un centro de datos. Pero aún existe la necesidad de usar ordenadores de alta potencia para poder entrenar el modelo de ML. Aún así, se puede mejorar esta situación?

Este proyecto estudia el TinyML y dos de sus técnicas, de lo que llamamos TinyML On-Device Learning o TinyODL, capaces de entrenar el model de ML en el mismo dispotivio (*on-device learning*): TinyML with Online-Learning (TinyOL) y Federated Learning (FL). Se estudian las dos técnicas desde un anáisis teórico y probamos de desarrollar una aplicación TinyODL.

Contents

1 Introduction and Context

1.1 Introduction

Internet of Things (IoT) has acquired useful and powerful advances thanks to the Machine Learning (ML) implementations. IoT devices (e.g. smartphones, house alarms) collects data through its sensors (e.g. cameras, microphones), which are sent to data centers. These data centers can offer high computing power, so they process all the data, execute the ML algorithm and sends back the result to its corresponding edge device. This approach allows high computing power applications to be used by low-powered devices, but it has some disadvantages and requirements that cannot be met in all scenarios. For instance, the edge devices and data centers must have internet connection to send data between them. Also sending data to separate locations can cause some latency in the execution and the possible exposure to data privacy (data leakage).

TinyML is a Machine Learning field where its goal is to execute the ML algorithms inside low-powered devices. TinyML consists of[1] training an ML model with pre-collected data and flash it into an MCU so it can execute the algorithm by itself and get an answer nearly as accurate as using a remote ML model. This allows IoT devices to respond to its tasks without sending any data outside the device and can reduce the response time without worsen the device's performance.

As TinyML evolves, new applications are found but obstacles may arise. For example, the ML model's flexibility or the dependency with high performance computers. New approaches with on device learning, like TinyOL or *Federated Learning*, are developed or researched to solve these obstacles.

1.1.1 Context

This Bachelor's thesis is submitted in fulfillment of the requirements of the degree in Informatics Engineering specialising in Computing, coursed in the Barcelona School of Informatics of the Polytechnic University of Catalonia. The thesis is authored by Juan Diego Rubio Serrano and supervised by Felix Freitag.

1.1.2 Concepts definition

This thesis works with a deep knowledge in Machine Learning and microcontrollers. Therefore, the reader should be familiar with the next concepts to understand the thesis correctly.

Microcontroller

A microcontroller (MCU) is an integrated circuit device designed to govern a specific operation in an embedded system. It is important not to confuse a microcontroller with a microprocessor, as the latter is used in general-purpose computers. Microcontrollers typically include a processor (CPU), memory and input/output peripherals. Microcontrollers are ubiquitous and can be found in a wide range of devices (e.g., washing machines, cars, stove, etc.).

[1] Later in this thesis one can find a methodical understanding of what TinyML is and how it works

Machine Learning

Being one of the most important and known fields inside Artificial Intelligence, Machine Learning (ML) [1] is devoted to understand and build methods that can *learn*. That is, methods that leverage data to improve performance on some set of tasks.

Machine Learning algorithms build models based on sample data, or *training data*. This data serves as experience and the model's goal is to generalize from it. In this context generalize means to perform accurately on new, unseen examples/tasks after having experienced a training dataset.

ML approaches have different ways to be divided, but traditionally they are split into these three broad categories, which correspond to learning paradigms dependable on the nature of the data available to the learning system:

- *Supervised learning*: The model receives example inputs and their desired outputs. Its goal is to learn a general rule in order to map inputs to outputs.

- *Unsupervised learning*: Unlike the supervised learning, no outputs are given, leaving it on its own to find structure in its input. This can be a goal itself (discover hidden patterns in data) or a mean towards an end.

- *Reinforcement learning*: The program must to perform a certain goal in a dynamic environment. As it navigates its problem space, the program is provided with feedback analogous to rewards, with the goal to maximize.

We can find a lot of ML algorithms like *Clustering*, *Linear regression*, *Random forests*, *Support Vector Machines (SVM)*, *K-Nearest Neighbour (KNN)*, etc. In this project, though, we focus our attention in the supervised learning algorithms, concretely in *Artificial Neural Networks*.

Artificial Neural Network

Artificial Neural Networks (ANN) [2] or simply Neural Networks (NN) are computing systems inspired by the biological neural networks that constitute animal brains. They are a type of Machine Learning algorithm that work in all learning paradigms, but mainly used in supervised learning.

An ANN consists of connected units of nodes called artificial neurons, which loosely model the neurons in a biological brain. Each artificial neuron receives signals, then processes them and can signal neurons connected to it, like the synapses. The connections are called edges, which typically have a weight that adjusts as learning proceeds. This weight increases or decreases the strength of the signal.

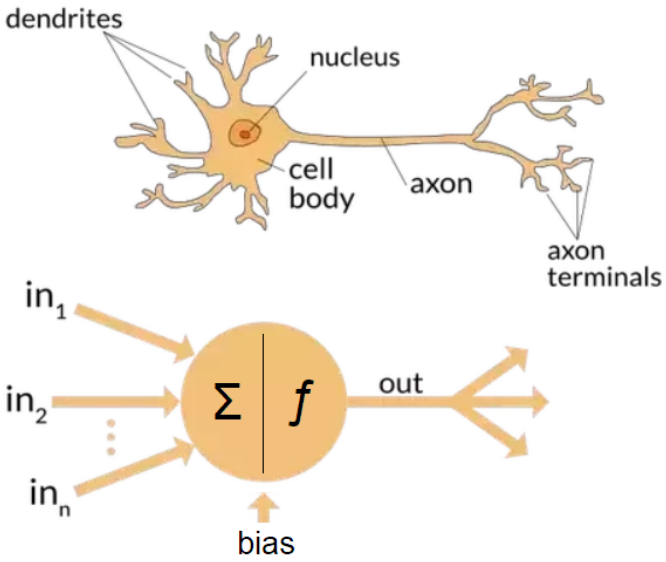

Figure 1: Representation of a neuron and an artificial neuron.

An ANN is formed by a set of layers, which are a set of non-connected neurons, that are connected between them sequentially. We can differentiate 3 types of layers: the input layer, which receives the signals from the program instead from another layer, the output layer, which sends the signals to the program instead to another layer, and the hidden layers, which receives the signals from the previous layer and sends the signal to the next layer.

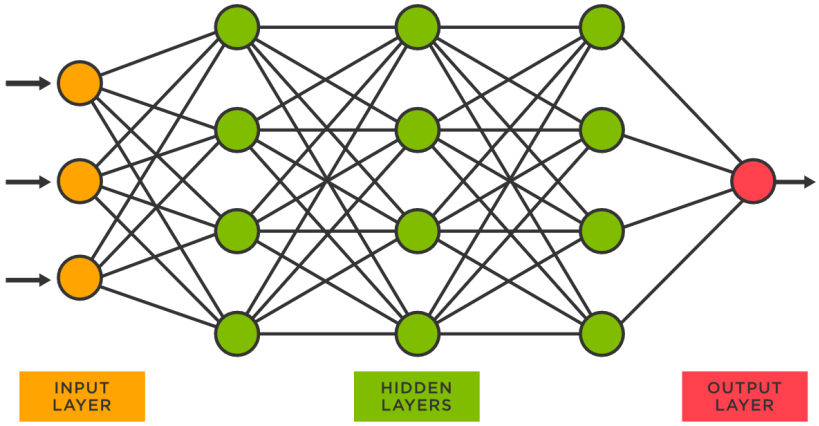

Figure 2: Representation of an artificial neural network.

On one hand, input and output layers are essential to the network, since both read the outside data and shows the results. On the other hand, hidden layers are not essentially required in a neural network, although ANN with no hidden layers are developed to do easy tasks. Even networks with one hidden layer are for easy tasks as AND and OR functions, but the XOR function can not be generalized with one hidden layer.

The workflow of an artificial neuron is simple: takes all the values from the incoming edges, multiplies them with its respective weight (associated to the edge) and sums them all into a unique value. Then a function called *activation function* takes the resulting value as input and decides whether the neuron outputs or not a signal.

The network works as follows: Each input value is stored in each neuron from the input layer. The layer sends all its values to the next layer, concretely to the next layer's neurons the outgoing edges are connected to. After processing the values inside the neurons, this processed is repeated until reaching the output layer, where after processing the values it returns the resulting outputs to the program.

1.2 The Problem

As stated before, TinyML tries to implement ML applications on MCUs. Therefore, the main challenge is to develop ML algorithms for low-powered devices. This is no easy task since we need to handle with some MCU constraints against the Machine Learning trend:

Hardware constraints

Low power Low power consumption is one of the defining features of TinyML systems. But TinyML devices can consume different amounts of power, which makes maintaining accuracy across the range of devices difficult. Hence makes things even difficult for benchmarking. Not only that, but it is difficult to determine when data paths and pre-processing steps can vary significantly between devices. Then there are other factors like chip peripherals and underlying firmware can impact the measurements. [3]

Limited memory While traditional ML systems like smartphones cope with resource constraints in the order of a few GBs, TinyML systems are typically coping with resources that are two orders of magnitude smaller. This also complicates the deployment of a benchmarking suite as any overhead can significantly impact power consumption or even make the benchmark too big to fit. A variety of benchmarks should be chosen such that the diversity of the field is supported. [4]

Processor capacity Majority of tiny edge devices have 10–1000 MHz clock frequency. Although it is acceptable for other applications, it can restrict the complex learning models from running efficiently at the edge. [5]

Software constraints

One of the greatest obstacles TinyML faces is the lack of flexibility of the ML model. We train the ML model outside the device because of its constraints (low power, limited memory). This means the resulting program needs to be updated every time the execution environment has new data the device can collect. For example, when the monitored environment (or machine) reaches a new unknown state and needs to be classified as a dangerous situation.

Machine Learning trends

In a recent paper called 'Compute Trends Across Three Eras of Machine Learning' [6], the authors have studied the compute, data, and algorithmic advances that eventually guide the progress of Machine Learning. It can be seen in the next figure, the training compute of models increases exponentially. This is because of the improvement in performance of newer computers and the complexity the ML models are getting into. For example, GPT-3 has more than 175 billion parameters (nearly 800GB of memory) [7] and DALL·E 2 trained with 250 million images. [8]

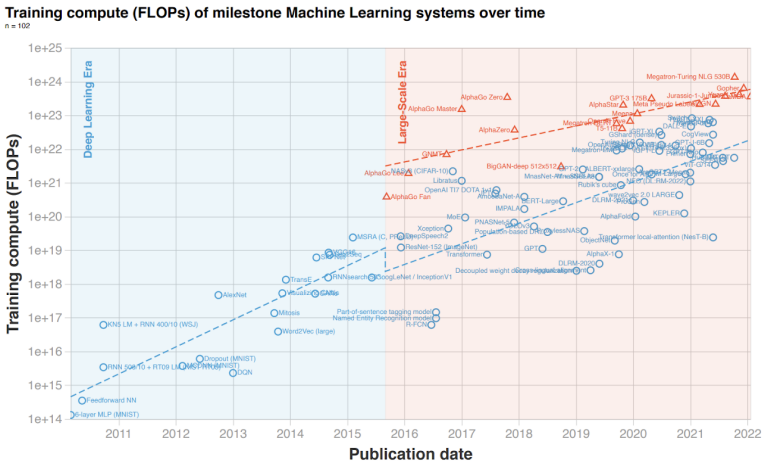

Figure 3: Trends in training compute of $n102$ milestone ML systems between 2010 and 2022. Notice the emergence of a possible new trend of large-scale models around 2016. The trend in the remaining models stays the same before and after 2016.

Figure 3: Training compute (FLOPs) of milestone Machine Learning systems over time

All the previous constraints makes TinyML hard to evolve. But is it possible to relax at least one of the constraints? Trying to answer this question a new TinyML subfield arise. Although it has no specific name, its main purpose is to prove ML models can be trained inside the TinyML devices obtaining (nearly) the same results relaxing the flexibility constraint. In this thesis this subfield is going to be referred as *TinyML-ODL* (TinyML On Device Learning) or *TinyODL* directly.

1.2.1 Stakeholders

- **Research team**: By the end of this thesis the research team will have become an expert of TinyML and its new subfield TinyML-ODL, which it is expected to be decisive for the future of Machine Learning.
- **Companies**: TinyML undoubtedly has interesting and useful applications in our daily tasks. Being able to not only prepare programs to resolve some tasks but to solve new unknown situations is what these applications lack. TinyODL models capable to solve this obstacle will impact positively to companies that invest and research about it.
- **Scientific community**: With this thesis we hope it inspires new TinyODL approaches, even new general purpose ML approaches.

1.2.2 Justification

TinyML became popular with its deployment of ML applications in edge devices with no transmission data. Training the model previously in a high performance computer and deploy it in the edge device makes possible to solve all the transmission data obstacles we found so far. But this new approach has some obstacles as well. We are still dependable of high performance computers to create and update the model. Also updating the ML model could be a hard task or even impossible being subject to the edge device's accessibility. This results in a static model, hard to adapt to new data and impossible to adjust for different scenarios, which impedes the flexibility of the Internet of Things.

TinyML is a recent field in the industry of Machine Learning, so solutions to that field are more expected to find that already found. Nevertheless, some researchers started to investigate for new approaches in TinyML with on device training. These new approaches seem promising, but they are very recent and still pending to evolve.

1.3 Objectives

1.3.1 Project objectives

This thesis aims to study two of the known techniques in TinyODL we mentioned previously: TinyOL[2] and Federated Learning. Two main objectives emerge from it:

O1: The off-device training of TinyML

To be capable to understand how TinyODL techniques work and what needs serve we must know what TinyML is and how it works first. This objective can be broken down into the next sub-objectives:

- *Understand the TinyML approach*
- *Understand the steps to develop a TinyML app*
- *Develop a TinyML app*

O2: The on-device training of TinyML

This objective aims to get to know both TinyOL and Federated Learning techniques to its core theoretically. Then choose one of them to develop and see practical results. It encompasses the next goals:

- *Learn TinyOL technique*
- *Learn Federated Learning technique*
- *Compare both techniques*
- *Develop a TinyODL app*

[2]Clarification: TinyODL and TinyOL are not the same. *TinyODL* stands for the set of techniques that trains the TinyML model inside the edge device, while TinyOL is an example of a *TinyODL* technique.

1.3.2 Potential obstacles and risks

TinyODL techniques are a very recent working field in TinyML. That is why we can encounter some software errors. This could happen due to software that is not updated to its final version or because of software not ready to work in a TinyML environment or with TinyODL techniques. All these possible problems can come from the fact that ML software is not prepared to work with low powered devices or vice versa.

Similar to the previous possible obstacle, we can find ourselves in a situation where no software was developed to program our desired application. The risk to develop an app with new libraries to develop for the field can deeply slow us down.

To end with these 'young field' related problems, all the information we can find about the field is very limited. Moreover, some data can be biased or not firm enough. This fact can lead us into some dead ends or open questions we will not be able to solve in this project.

One more obstacle we find is the debugging of code in MCUs. Since we are working with low powered devices and high performance programs it is possible to have some debugging restrictions, for example with memory usage or print logs to console.

To enclose this section, the main problem we face in this project is the time available. It is required be software and embedded-hardware expertise just to develop TinyML applications. Furthermore, a machine learning expertise is demanded with TinyODL techniques. Thereby, the time invested in the previous study to master them can affect the quality of the work, and hence the objectives mentioned before.

2 Project Planning

We introduced the thesis, defined the problem and point out the objectives we work. In this section we write down how we plan to perform this thesis. We explain a methodology and the resources we are provided to work with, we define all the tasks we should complete to achieve our objectives and how are they distributed in our limited time. Finally we explain how to handle the potential obstacles and risks we mentioned in the previous section.

2.1 Methodology

For this thesis we considered the Agile methodology would be a good base method to develop the project. The Agile methodology, as stated in its manifesto [9], work with short iterations (normally a fortnight long) consisting of the phases of analysis, planning, development, testing, review and maintaining.

But for this project we take the idea of a two-week review. Instead, we will define the project tasks to work on each fortnight and review how it went, check the doubts and problems and entrust the next tasks to work on. All the tasks are going to be stored in a Kanban board where we can easily see its status and check its progress and obstacles (in the next section we define which tools we use to accomplish this).

2.2 Resources

The next resources were used to develop this thesis.

2.2.1 Staff

Here there are all the jobs titles needed for the project:

- **MAN - *Project Manager***: The project manager looks out for the project and makes the decisions for it. The project manager is in charge of organizing the project to meet the content and deadlines.
- **RE - *Researcher***: This is the assigned person (or people) to investigate the field and its latest news.
- **PR - *Programmer***: This member is needed to implement and execute the models correctly.
- **DA - *Data analyst***: This member is responsible for all the data analysis of the project, from machine learning data analysis for a good model to the analysis of the results given by the experimentation.

2.2.2 Material resources

Here are exposed all the materials needed for the project:

- **PC - *Computer***: A computer was needed to train the model outside the microcontroller at first and if needed create the model. My laptop model is Huawei Matebook 14 AMD RyzenTM 5 4600H 16GB + 512GB [10], but any computer capable to support the software and processing needed, and with a USB port, fit the necessities.
- **PH7 - *Arduino Portenta H7***: Obviously without a microcontroller where to execute the model it is impossible to perform the project. In our case we used an Arduino Portenta H7, which has a 2MB flash memory, 8MB SDRAM, a STM32H747 Dual-Core processor (cores M7 and M4) with a 480MHz clock, 5V of input voltage and 3.3V operating voltage, etc. You can find all the specifications here [11].
- **VS - *Arduino Portenta Vision Shield - LoRa***: Added to the Portenta H7, this module consists of a Himax camera, a microphone and LoRa connection [12].
- **USB - *USB type C - USB type A cable***: To flash code from the laptop to the Arduino a cable is needed. In our case we operated with a cable with both a USB type A male for the laptop and a USB type C male for the Portenta H7.

2.2.3 Software resources

The next software resources were used to develop the practical parts of this project. It may not be necessary to obtain all software frameworks to replicate some results in this project.

- **GCO - *Google Colaboratory*** Colaboratory or Colab [13] is a product from Google Research where one can develop, test and debug Python code through the browser. Based on Jupyter [14] it allows to use and share Jupyter notebooks without downloading, installing or running anything in one's computer. We use this website to develop and train NN models in Python. Evidently Jupyter is a good alternative.
- **TFL - *Tensorflow Lite*** Google developed an end-to-end open source platform called Tensorflow [15] to develop ML models. Later, they added to it what is called Tensorflow Lite [16], a mobile library for deploying models on mobile, microcontrollers and other edge devices. In this project we employ this platform to develop and deploy TinyML models.
- **EI - *Edge Impulse*** One way to develop TinyML models easily is by using a framework, for example Edge Impulse. Edge Impulse [17] is a development platform for machine learning on edge devices. One of the TinyML apps in this project was done using this platform.
- **PIO - *Visual Studio Code + PlatformIO*** As we work with an Arduino, a framework to develop and flash code to is required. We decided to employ PlatformIO [18], a VS Code [19] extension for embedded development.
- **IDE - *Arduino IDE*** Arduino IDE [20] is the main IDE to develop and flash Arduino code. Although not being our IDE for development, we made use of it.

- **GIT - *Git & GitHub*** For the code implementation and usage it is essential to work with Git [21], a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Moreover, with GitHub [22] we upload the code to the Internet, being capable to download the code from wherever computer we want to and share our progress with other people if we want to.

2.2.4 Software tools for project management

The next software resources were used to manage the project.

- **NO - *Notion*** To manage all the tasks inside the project we used Notion [23], a project management and note-taking software.
- **GCA - *Google Calendar*** To manage the available time and meeting appointments we used Google Calendar [24], a time-management and scheduling calendar service developed by Google.
- **GM - *Google Meet*** We make use of the Google Meet service to do the meetings.
- **GAN - *Gantter*** To do the Gantt chart and all the tasks management it is useful to use a tool like Gantter [25].
- **OV - *Overleaf*** Finally, to write this document we used Overleaf [26] web service, a collaborative cloud-based LaTeX editor.

2.3 Time management

The period to develop the project started February 14th of 2022, but due to some personal situations the project actually started March 17th of 2022. Its final delivery was planned to be June 27th of 2022 and the presentation between the final delivery and June 31st, but after starting June we decided to extend the period to next semester until September 17th of 2022, and make the presentation between October 17th and 21st. Nevertheless, we first explain our time distribution based on the first deadline, and afterwards we add the extra period.

First period (March 17th - April 24th)

There is a total of 102 days between March 17th and June 27th. From it, we need to substract 14 days due to a competition I [3] had to assist between May 1st and 7th and its preparation one week before it. That leaves us with 84 days. We expected to dedicate 4 hours per day from March 17th until April 24th. This is no arbitrary number, since is the time we thought we would had available to work on the project while working on the competition's project and the Data Mining (DM) subject I enrolled. That is 39 days working 4 hours/day, a total of 156 hours. The figure below represents the weekly distribution of time during this first period.

[3] This document is written by Juan Diego, who 'I' refers to.

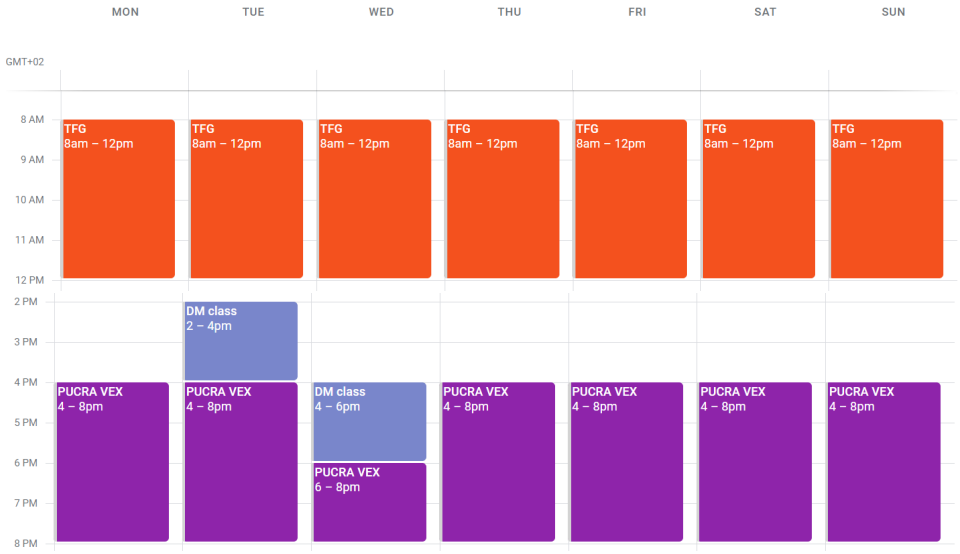

Figure 4: Timetable from March 17th until April 24th in Google Calendar. *TFG* stands for '*Treball Final de Grau*' which is Final Degree Thesis in Catalan.

Second period (May 8th - June 27th)

After the competition, starting May 8th, the expected time to invest in the project is 8 hours per day on average. Since our first idea was to end June 27th, we should be working 50 days and 8 hours/day, which is 400 hours. The figure below represents the weekly distribution of time of this second period. The missing hours (for example Wednesday evenings) would be distributed during the week.

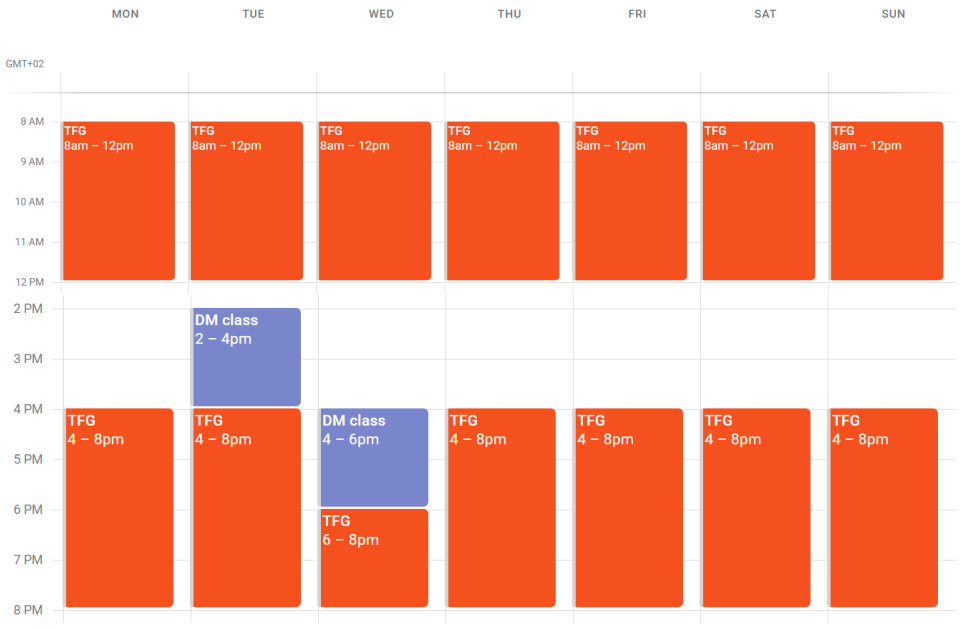

Figure 5: Timetable from March 17th until April 24th in Google Calendar. *TFG* stands for '*Treball Final de Grau*' which is Final Degree Thesis in Catalan.

Summing up, we expected to invest 156+400 = 556 hours in the project. That is a quite optimistic value and not realistic at all, that is why we narrowed it down to 500 hours approximately. Albeit being a fairly high number, it served us to see which tasks we were and were not able to accomplish.

Third period (June 28th - September 17th)

Starting June, we began to realize we were running out of time for developing the final steps of the project. Since we were in that situation due to some personal issues, we thought it would be interesting to extend the working period to all summer. That is why we added a third period to the previous time distribution. From June 28th until September 17th we decided to work 5 hours a day, 5 days a week. Also we would meet in July to see the advancements and leave August to end the project and write this document.

The next figure represents an approximate distribution during this period. Since this period is located in summer vacations we could not define which exact days we were working on it, but at least there was 5 days per week.

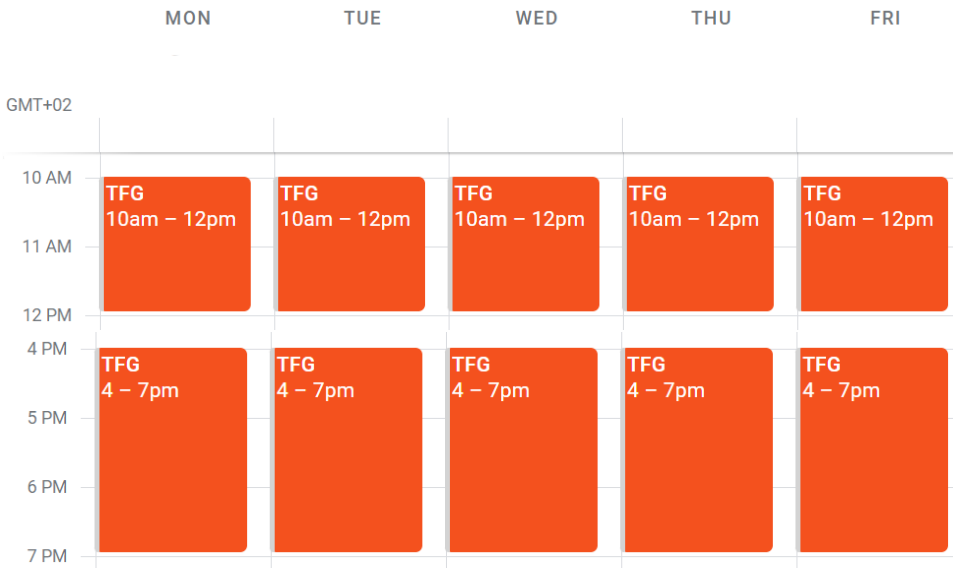

Figure 6: Timetable from March 17th until April 24th in Google Calendar. *TFG* stands for '*Treball Final de Grau*' which is Final Degree Thesis in Catalan.

In this period we expected to work 55 days and 5 hours/day, which is 275 hours. Similar to the previous periods, this could be an optimistic value, so we narrowed it down to 230 hours.

To conclude, this project was expected to took 500+230 = 730 hours to develop, manage and report.

2.4 Task definition

In this section we break down all the project goals and management into tasks. To make easier to understand the tasks we decided to group them.

Project Management (PM)

This group contains all the tasks related to project management and report. That is, all tasks that are not study or development of the objectives presented above. By accomplishing all these tasks it is expected to organize and present correctly the project.

PM-1 ***Scope*** We must define the extension of the project. That is what we want to do, what we do not want to do and what we set aside.

PM-2 ***Planning*** To achieve our goals it is mandatory to make a good planning about the resources we dispose, the tasks we need to complete and the time available and its distribution.

PM-3 ***Budget management*** We plan the budget we need to develop the project and the budget we can spend.

PM-4 ***Sustainability*** We perform a study about how sustainable is this project so the reader has an idea.

PM-5 ***Meetings*** As stated in our methodology, we meet each two weeks approximately to see the project advancements and make decisions.

PM-6 ***Final report*** Written document explaining the project definition and its development, with conclusions and references used.

PM-7 ***Final presentation*** Visual document used as an auxiliar tool for the final presentation of the project.

Previous Study (PS)

Before go deep into the project subject we first need to learn the basic tools and concepts.

PS-1 ***Research Arduino Portenta H7 specifications*** This task consists of research about Arduino Portenta H7 specifications so we know what it is capable of.

PS-2 ***Arduino environment adaptation*** Since it is our first time working with Arduino we need to have some previous knowledge about its tools. We break down this task into:

- **PS-2.1** *Arduino IDE environment adaptation*: Get basic knowledge and practice to work with Arduino IDE.
- **PS-2.2** *PlatformIO environment adaptation*: Get basic knowledge and practice to work with PlatformIO.

PS-3 ***Develop LED program*** We develop a program where we practice the Portenta's LED usage.

PS-4 ***Develop camera program*** Program where we practice the Portenta's camera usage. We can divide this task into two:

- **PS-4.1** *Research Portenta's camera usage*
- **PS-4.2** *Develop camera program*

TinyML approach (TML)

Once we completed the previous study we can move on with our first assignment: To study TinyML and develop a TinyML app. We divided it as follows:

TML-1 ***Define TinyML*** First of all we have to fully understand what the TinyML approach is. This task does an exhaustive research about TinyML and its workflow.

TML-2 ***Define steps to develop a TinyML application*** This task's purpose is to research and understand the steps to develop a TinyML application.

TML-3 ***Define TinyML app to develop*** Definition and design of the TinyML app we want to develop.

TML-4 ***TinyML environment adaptation*** We research the TinyML tools needed to develop the app and practice with them.

TML-5 ***Develop a TinyML model*** We create the ML model for the app and convert it into a TinyML model. We break the task down:

- **TML-5.1** *Collect and preprocess data*
- **TML-5.2** *Design ML model*
- **TML-5.3** *Train & test ML model*
- **TML-5.4** *Research how to convert model into TinyML model*: We study how to convert a ML model into a TinyML model.
- **TML-5.5** *Convert model into TinyML model*: We convert our model into a TinyML model.

TML-6 ***Develop & Test TinyML app*** Final steps to complete our TinyML app where we develop a program that uses the TinyML model inside the Portenta H7 and test it.

Study of TinyOL (TOL)

Another assignment consists of a theoretical study about one of the on-device learning approaches: TinyOL. It contains the next tasks:

TOL-1 ***Define and understand TinyOL technique*** Introduction to what is the TinyOL technique and its theoretical justification to work correctly.

TOL-2 ***Steps to develop a TinyOL application*** Step by step about how one can develop a TinyML application with the TinyOL technique.

TOL-3 ***Research tools to develop a TinyOL application*** We research the tools we can or need to use to develop a TinyOL application.

TOL-4 ***Benefits & Disadvantages*** We list all the benefits and disadvantages about this technique, its usage and its development.

Study of Federated Learning (FL)

One more assignment is the theoretical study of the Federated Learning technique. Similar to the TinyOL study we define the next tasks:

FL-1 ***Define and understand Federated Learning technique*** Introduction to what is the Federated Learning technique and its theoretical justification to work correctly.

FL-2 ***Steps to develop a Federated Learning application*** Step by step about how one can develop a TinyML application with the Federated Learning technique.

FL-3 ***Research tools to develop a Federated Learning application*** We research the tools we can or need to use to develop a Federated Learning application.

FL-4 ***Benefits & Disadvantages*** We list all the benefits and disadvantages about this technique, its usage and its development.

Development of a TinyODL app (ODL)

Last but not least, we develop and analyse one of the techniques we studied. We can break it down into the following tasks:

ODL-1 ***Choose a TinyODL technique*** Compare the TinyODL techniques and choose one to develop explaining the choice.

ODL-2 ***TinyODL environment adaptation*** We practice with the tools to develop a TinyML app using the TinyODL technique.

ODL-3 ***Define application*** Description about the application we want to develop.

ODL-4 ***Develop & Test TinyML application*** We develop the desired TinyML application without the TinyODL technique. The steps to develop a TinyML application can be seen above in the TinyML approach section.

ODL-5 ***Develop TinyODL application*** We develop the TinyML application with the TinyODL technique. Depending on the selected technique this task contains different sub-tasks.

ODL-6 ***Deploy & Test TinyODL application*** Flash the application to Portenta H7 and train the model.

ODL-7 ***Compare results*** Analyse the TinyML and TinyODL application ouptuts. Compare the model's accuracy and loss, the outputs obtained when running inference, their performance inside the Portenta H7 and any interesting data.

2.5 Task planning

Once we defined all the tasks to develop the project, we made a table with the duration, dependencies [4], the people assigned and the resources from each task. You can find the table in the Annexes section. Below the table we can find the Gantt chart. where we exemplify the task planning in a calendar with their dependencies and time we expected to spend. It is important to point out that the Gantt chart was developed before requesting the project extension.

2.6 Risk management

Previously we stated the obstacles we can find during the project. In this section we are going to explain how we prevent or manage them.

Lack of TinyODL software tools

The developed software related to TinyODL techniques is an obstacle out of our range of prevention. To manage it, we shall try to squeeze the TinyML and ML tools to its limits to develop our TinyODL app, and try to develop an app not too complex.

Lack of information

Same goes to the information we can find about TinyODL techniques. Moreover, we can take advantage of the situation to contribute new knowledge to the field.

Debug inside the MCU

To debug inside the microcontroller a good option is to test each function independently of the rest of the code, with a test control for each function. When debugging the TinyML model and the TinyODL technique we first make sure they work correctly in our computer, trying to simulate as best as we can the microcontroller environment. Then we flash the app and run it in our microcontroller, and we compare the results obtained with the ones from the computer.

Time availability

The time available is one of the biggest problems. What we can do to prevent it is try to work as parallel as possible all the non-dependent tasks, try to analyse other solutions when facing a development problem and make periodic reviews of the project to adapt the tasks to the time left.

[4] Dependencies are recursive. For example, if task C depends on B and B depends on A this means that C depends also on A.

3 Budget and Sustainability

3.1 Budget management

In this section we propose the expected budget to do the project. This can be broken down into 5 parts.

3.1.1 Personnel costs

In this project 4 roles are required in order to fulfill all the tasks correctly, which are the *Project Manager*, the *Researcher*, the *Programmer* and the *Data analyst*. Since we are performing this project in the FIB in Barcelona, we can make an approximation of their salaries by analysing the mean salaries in Barcelona.

Project Manager salary: We find the mean annual salary for this job in Barcelona is up to 37.000€, which is nearly 19€ per hour.

Researcher salary: Similarly the researcher job has an approximate 20.400€ annual salary, translated as a 10.50€ per hour.

Programmer salary: '*Programmer*' is not the best word to describe a job, so we decided the best job that fits this position is the *Software Engineer*. We get a mean of 36.000€ yearly, but in this case we are talking about experienced software engineers like seniors, but with standard or junior software engineers we can find a mean of 30.000€ yearly, or 15.50€ hourly.

Data analyst salary: With the data analyst salary we can find some differing information, from 27.000€ to 34.000€. In that case we take an approximation to these values but trying to take the higher values in order to be prepared. In conclusion, we take a salary of 32.000€ yearly, 16.50€ hourly.

All the salaries, yearly and hourly, were round up. We collect all the salaries in the next table:

Personnel	**Annual salary** (€)	**Hourly salary** (€)
Project Manager	37.000	19
Researcher	20.400	10.50
Programmer	30.000	15.50
Data analyst	32.000	16.50

Table 1: Personnel salaries by year and hour. Own creation.

The next table breaks down by each task its personnel cost.

Task	Time (h)	People Assigned	Cost (€)
PM	**93**	**-**	**2.052**
PM-1	5	MAN	95
PM-2	12	MAN	228
PM-3	3	MAN	57
PM-4	3	MAN	57
PM-5	15	MANx2	570
PM-6	40	MAN	760
PM-7	15	MAN	285
PS	**38**	**-**	**949,5**
PS-1	6	RE	63
PS-2	10	PR	155
PS-2.1	3	PR	46,5
PS-2.2	7	PR	108,5
PS-3	5	PR	77,5
PS-4	17	PR	263,5
PS-4.1	5	RE	52,5
PS-4.2	12	PR	186
TML	**93**	**-**	**3.420**
TML-1	6	RE	63
TML-2	8	RE	84
TML-3	3	MAN, PR, DA	127,5
TML-4	13	PR	201,5
TML-5	48	-	1.232
TML-5.1	10	PR, DA	320
TML-5.2	8	PR	124
TML-5.3	12	PR, DA	384
TML-5.4	8	RE	84
TML-5.5	10	PR, DA	320
TML-6	15	PR, DA	480
TOL	**47**	**-**	**741**
TOL-1	10	RE	105
TOL-2	10	RE	105
TOL-3	12	RE	126
TOL-4	15	RE, DA	405
FL	**47**	**-**	**741**
FL-1	10	RE	105
FL-2	10	RE	105
FL-3	12	RE	126
FL-4	15	RE, DA	405
ODL	**176**	**-**	**4.991,5**
ODL-1	10	MAN	190
ODL-2	25	PR	387,5
ODL-3	3	MAN, PR, DA	153
ODL-4	48	PR, DA	1.536
ODL-5	55	PR, DA	1.760
ODL-6	25	PR, DA	800
ODL-7	10	DA	165
TOTAL	**494**	**-**	**12.895**

Table 2: Personnel cost breakdown.

3.1.2 Material costs

As related to the costs for the materials and software tools listed in the Resources subsection, we describe the next budget. The amortization cost is calculated as the cost per hour of the material times the hours we used the material

Material	Price (€)	Cost per hour (€/h)
Huawei MateBook 14 AMD 2020	699,00	0.016
Arduino Portenta H7	99,00	0.214
Vision Shield Lo-Ra	60,00	0.15
USB cable	7,21	0.04
Total	**865,21**	**-**

Table 3: Material costs: Prices.

Material	Used hours	Amortization (€)
Huawei MateBook 14 AMD 2020	469	7,48
Arduino Portenta H7	115	24,61
Vision Shield Lo-Ra	110	16,5
USB cable	115	4,6
Total	**-**	**53,19**

Table 4: Material costs: Amortization cost.

In our case we used a *Huawei MateBook 14 AMD 2020* laptop, but any computer capable of running the software tools and frameworks used and with a USB port shall work as well.

Related to the software tools, all the tools we used are open source software or free to use frameworks, so there is no need to spend any budget in order to replicate the project. Nevertheless, some of the tools have a subscription option to obtain more access to it.

3.1.3 Indirect costs

Here we make a budget about the some indirect costs the project has.

- **Work space**: The project has been developed remotely, with a monthly rent of approximately 400€. The project has a total of 494 hours, so its cost is:

$$WSCost = 400 * \frac{1}{30days} * \frac{1}{24hours} * 494 = 329,33$$

- **Electricity**: Since the average price of the electricity is very variable, we estimate an approximation of 0,226 €/kWh. The laptop we used for this project consumes 56 Wh, in a total of 494 hours this goes up to:

$$ECost = 494 * 0,226 * 56 = 6,252$$

- **Internet**: There is a monthl cost of 48€. Average of 6 working hours and 494 project hours.

$$ICost = 48 * \frac{1}{30days} * \frac{1}{8hours} * 494 = 98,8$$

Resource	Cost (€)
Work space	329,33
Electricity	6,252
Internet	98,8
Total	**434,382**

Table 5: Indirect costs

3.1.4 Contingency

Unforeseen events are common in any project. To prevent them we need to adapt our budget. We decided to set a 18% of the sum of the personnel, material amortization and indirect costs seen before. This results in:

$$(12.895 + 53,19 + 434,382) * 0.18 = 2.408,86$$

3.1.5 Incidental costs

We also take into account the obstacles we may encounter while working on the project. Similar to the contingency, we need to add a plus in the general budget to mitigate the consequences. The next table shows the possible incidents and the added budget for every one.

Incident	**Estimated cost** (€)	**Risk** (%)	**Cost** (€)
Debug time	20.687,35	20	4137,47
Material damage	865,21	10	86,521
Total	**-**	**-**	**4.223,97**

Table 6: Incidental costs

3.1.6 Total cost

Finally, we represent all the previous costs in a final table.

Activity	**Cost** (€)
Personnel costs	12.895
Material costs	865,21
Indirect costs	434,382
Contingency	2.408,86
Incidental costs	4.223,97
Total	**20.827,422**

Table 7: Summary of costs

3.2 Sustainability

A project is not just plan and develop. We must be aware how the project will affect our environment, economy and society. In this section we talk about this project influence, concluding with the final question: is this project sustainable?

3.2.1 Environment influence

The environmental footprint of this project is measured by how many resources we used to make it happen. On one hand we need to think about the energy we consumed by the light office, powering the computers, etc. We also have to take into account all the raw material and energy to create of the used products.

We can calculate the footprint of the energy consumed during the project. Using a computer that consume an average of 56W a total time of 469 hours, that gives us a total of 26,26KWh or 6,1Kg of CO2. The microcontroller power usage is about 1 µA in sleep mode, which will be most of the time.

3.2.2 Economic influence

The project has a total cost of 20.827€, which is a considerable amount to invest in a project. Moreover, this total cost can vary. At first, it does not seem a good invest.

But on the other hand we avoid to have a high computer to train the model and all the process to get access to the microcontroller in order to flash the model. That in the long term saves us a lot of money.

3.2.3 Social influence

First, I want to explain my personal experience, since this project has taught me new concepts and helped me to understand better the artificial neural networks. I also learned about microcontrollers and its capabilities with ML, which made me understand this field has very interesting projects.

The influence this project has in our society can seem null, but the possibility of edge devices being able to train ML models in order to accomplish complex tasks is a big step, bigger if we think about the accessibility of edge devices to anyone.

But all that glitters is not gold. Being able to work with ML models could also become one thing to fear. That is why with this technology shall be accompanied with awareness.

3.2.4 So, is this project sustainable?

Yes, although the economical aspect seems a bit much, but in general terms this project not only does not affect severe in the environment, but a lot of economical and social benefits come with it.

4 Study of TinyML context

In this section we explain all the knowledge we obtained before doing the true goal of this project.

4.1 Arduino Portenta H7

We go deep into what we learned about the Arduino Portenta H7 during the previous study. Also we explain the applications we developed to get in touch with the environment.

The Arduino Portenta H7[27][28] is a microcontroller that follows the Arduino MKR form factor, but enhanced with the Portenta family 80 pin high-density connector. Designed by Arduino [29], it can run high level code with some real time tasks thanks to its two processors working in parallel and capable of communicate with each other.

4.1.1 Specifications

Here we show some of the specifications this Arduino has and we thought were interesting to point out. Figures 17, 18, 19 shows the hardware of Arduino Portenta H7 and its Vision Shield.

ST STM32H747XI Processor

- **Dual-Core**:
  - *ARM Cortex-M7 core* up to 480MHz with double precision FPU
  - *ARM 32-bit Cortex-M4 core* up to 240MHz with FPU
- **Flash memory 2MB**: with read-while-write support
- **SRAM 1MB**
- **On-chip GPU** Chrom-ART Accelerator (DMA2D)
- **Up to 35 communication peripherals**
- **11 analog peripherals**

External memories

- **SDRAM 8MB** than can go up to **64MB**
- **Flash QSPI 16MB** than can go up to **128MB**

WiFi/BT module Murata 1DX

- *WiFi 802.11*b/g/n 65 Mbps
- *Bluetooth 5.1* BR/EDR/LE

Others

- *10/100 Ethernet PHY*
- *High Speed USB PHY*
- *Crypto Chip*
- *UFL Antenna option*
- *DisplayPort over USB-C*
- *5V Input - 3.3V Operating*

Portenta Vision Shield

Moreover, we had the opportunity to add to the Portenta H7 its Vision Shield. It has the next technical specifications:

- **Camera module Himax HM-01B0**
  - Ultra Low Power Image Sensor
  - High sensitivity 3.6µ BrightSenseTM pixel technology
  - Supports QQVGA (160x120) at 15, 30, 60 and 120 FPS (1.1mW at 30 FPS)
  - Supports QVGA (320x240) at 15, 30, 60 and 120 FPS (2mW at 30 FPS)
- **PDM Digital Microphone 2x MP34DT06JTR MEMS**
  - 64 dB signal-to-noise ratio
  - Omnidirectional sensitivity
  - –26 dBFS ± 1 dB sensitivity
- **LoRa connectivity**: LoRa Module with ARM Cortex-M0+ working at 868/915Mhz
- **Micro SD card slot**

4.1.2 Why choose Arduino Portenta H7 ?

Thanks to the dual core processor and low-power capabilities, Portenta supports a wide array of applications.

Machine Vision: In combination with the Portenta Vision Shield it can run machine vision applications. This allows to detect the presence or movement of objects in a video stream.

AI & Machine Learning: Thanks to the power of the two cores it can simultaneously read data from sensors or other devices on one core while the other core processes the data stream and uses machine learning to make sense of the data. When used with the Portenta Vision Shield its camera module or the two directional microphones can be used as data sources

Connectivity: The Portenta features on-board Bluetooth and WiFi capabilities which makes it the perfect candidate for reliable IoT applications. When used together with the Portenta Vision Shield it enables LoRa communication in places where it needs to communicate efficiently over a long distance. The Vision Shield also features an Ethernet port that allows for wired networking applications.

4.1.3 Arduino programs review

We can now start with the development of Portenta programs, but first we are going to review how an Arduino program works.
An Arduino program consists of three essential parts:

- ***#include "Arduino.h"***: This line of code makes available all the basic Arduino functions.
- ***setup()* function**: Function executed at first
- ***loop()* function**: Function executed indefinitely in an infinite loop.

Alternatively we can write an arduino program execution as follows:

```
#include "Arduino.h"
setup()
while do not stop do
  loop()
end
```

Nevertheless, the program can be stopped by keyboard (Ctrl+c), by an error or exception when executing the code or by unplugging the microcontroller.

Once we have written our code inside the *setup* and *loop* functions we flash the code with our preferred IDE. In this project we used both PlatformIO for development and Arduino IDE to support some development situations. Inside both it can be found a button called *Upload* to compile and flash the code to the plugged microcontroller.

4.1.4 LED Program

In order to get to know how to work with Portenta and its development tools we developed an easy program. This program consisted of making the LED blink as we wanted to.

Tools to develop

To develop this program we used the next tools:

- Portenta H7
- USB cable
- PlatformIO

Development

The program is pretty easy, since most of the Portenta management is done by the Arduino functions. Inside the *setup()* function we start the LED with a function called *pinMode*. Then inside the *loop()* function we change the state of the LED (light or not) with the *digitalWrite* function.

When flashing code to the Portenta we found out that when pressing the button twice it enters into the Bootload mode, where the RGB LED blinks green with fading. By observing this we tried to recreate this fading.

To make a LED fade in Arduino we thought we just needed to change the *digitalWrite* function to the *analogWrite* function and put a number between 0 and 255 as an argument along with the LED pin. But that did not work, and after some tries and a lot of research we could not find a solution or an alternative to it.

Here one can find all the source code for this program -¿ [link to github (pending)]

Problems & Errors

Facing the fade program, we found that there is not a lot of information about this microcontroller. The Arduino documentation is pretty complete, but outside from it we could not find much more information.

Albeit it is not a problem, it is worth pointing out that we could not make the LED fade. Seems like this option it is not available for developers since the Pinout document does not clarify if the LED pins are analog.

4.1.5 Camera Program

Next, we developed a program to control the Portenta's Vision Shield camera in order to adapt ourselves to the Vision Shield environment. The goal of the program was to show on screen the images captured by the camera.

Tools to develop

We started developing this program inside the PlatformIO environment but we got some library problems with the camera, so in order to not slow us down we decided to develop it in the Arduino IDE and afterwards solve that problem. Moreover, we used OpenMV IDE to ease us all the graphical tasks for showing the image. To sum up, we use the next tools:

- Portenta H7
- USB cable
- Arduino IDE
- OpenMV IDE

Development

The problem's complexity upgraded compared to the one before. This time we were using the Arduino IDE (or PlatformIO) to initialize the camera and pick up the images from it, while the OpenMV IDE collect the images and show them in *Java*! But, the good thing is that an OpenMV code had already been programmed to collect and show the images, so our worries should been focused in the camera usage.

Arduino code: First the *camera.h* and *himax.h* need to be included, where all the camera and image management functions are declared. Next the camera is initialized. Inside the *setup()* function the *Serial* is started and the camera is started with some specifications (resolution, image format, framerate). Finally, the *loop()* function reads from the camera and sends the image via the Serial connection.

OpenMV code: As a simple review, the OpenMV code initializes the Serial object, receives the images from it and shows them inside the IDE. You can find the code here [link to github] and here its explanation.

Problems & Errors

As stated before, we had problems with the library dependencies with PlatformIO and the camera and himax library. This was because PlatformIO could not find the libraries, they were not inside the Arduino libraries. After adding libraries from the PlatformIO option, we decided the best option was to add the libraries ourselves to the code and the problem was solved.

Another obstacle was the need to work with OpenMV if we wanted to see the images. This hinders the usage, not only by sending the data between IDEs, but because a previous Java knowledge is demanded if we want the program work with the camera specifications we declared.

Finally, we must mention the final code did not worked properly. There were some running attempts that the images were not shown (although the camera did get images and sent them via Serial) or the video froze. When debugging the codes it seemed the problem was coming from the OpenMV code, so we decided to set it aside.

4.2 TinyML study

Before starting to study the TinyODL techniques we first needed to be familiar with Tiny Machine Learning. That is what this section is for. We are going to explain TinyML, deepen into its workflow and try to develop an app to show what we learned.

Tiny Machine Learning is broadly defined as a fast growing field of machine learning technologies and applications including hardware, algorithms and software capable of performing on-device sensor data analytics at extremely low power, typically in the mW range and below, and hence enabling a variety of always-on use-cases and targeting battery operated devices.

The goal of TinyML is to bring ML inference to ultra-low-power devices, typically under a mW, and thereby break the traditional power barrier preventing widely distributed machine intelligence. This area is committed to democratizing deep learning for all pervasive MCUs. While TinyML will not replace current high performance AI-based services, it will complement them with machine learning capability within the IoT.

4.2.1 How it works & Why?

TinyML works on pushing the IoT devices to their limits by storing in the memory a trained Machine Learning model. The problem TinyML faces is the model's size in comparison with the device's memory size. By optimizing the model's size and performance, the model fits the memory size so the program can work with it. The model is trained, optimized and then flashed into the device with the program.

One way to accomplish this is by designing and training an ML model as usual. Then the model data is optimized and reviewed to check the model performs as we expect. A common technique to optimize is [quantization], where each model value is transform into an 8-bit value, reducing the size of the data and so the model's size. But as we modify the values we need to make sure the model still works as we want to, which the review part is for.

Another option is reversing the steps: quantize the model data and then train it. For example, instead of designing a model with weights as float values (16 or 32 bit long) we restrict the weights as 8-bit integer values. Then, the model will train inside the size range we want it to be.

4.2.2 Problems and obstacles

TinyML has a lot of advantages in IoT applications. But it is not perfect, and has some obstacles to face when developing. Starting with the memory constraint, it is one problem from the IoT field that still drags. The main efforts developing a TinyML program are focused on fitting the model into the memory size, which can be a really hard task, even impossible in some situations. Furthermore, the extra space the model leaves in memory can be very small, even to force us to change the application workflow if it is not enough space left.

Related to the memory constraint, the Machine Learning models designed and trained for TinyML applications shall be reduced in space, which usually makes the model pretty simple. This means that hard tasks or complex situations will not be faced correctly by the model. Or even it cannot face the task it was design to by its simplicity, a more complex model and bigger in size is needed.

Once the model is working inside the low-powered device it runs inference as we expect to in the program, but it does not train. This is not an obstacle to deal with any TinyML program, but it can be decisive in some circumstances. Being the model able to adapt to new scenarios can lead to a better performance, and its non-flexibility can lead to untrue answers and inaccurate actions.

Another obstacle found during this research is that most of the TinyML programs are oriented to Neural Network models, concretely *Multi-Layer Perceptron* and deep learning models, which is good if we want to develop one, but we can find more useful to develop other ML models (for example, Random Forests, SVM, etc) for some problems.

4.2.3 Steps to develop a TinyML app

Developing a TinyML app is pretty much developing an ML model and create a program that works with the model. We now explain step by step how that works once we defined what the app is for. In this steps we explain the development of a supervised model, which is what work on this project.

Data collection Our first step is always collect all the data we can for the model. This is an essential step to know in what environment the model is going to work. Sometimes instead of collecting the data by ourselves it is interesting to search for a useful dataset.

Data preprocessing Not all the data we have is valid, and maybe the data is represented by unreadable values or ranges that will make the training invalid. To convert our dataset into a readable and useful dataset for the model we process it by what it is called *preprocessing*. Although it is not stritcly defined how all datasets must be preprocessed, there are some guides to check in our dataset:

- *Missing values*: We shall inspect the dataset for missing or null values and decide what to do with those examples. We can toss them, or fill them with the probabilistic distribution of other valid values (for example, the *normal distributon*).

- *Handle outliers*: The dataset can have some values that extremely differs with the vast majority of the data. This values can affect our model performance by biasing the output by learning them. To prevent that, a search and process of these values is done. Normally we treat these values similar as missing values, changing them with the normal distribution of valid values for example.

- *Mixed data types*: Inside a dataset two main types of data can be found: numerical values and categorical values. Some models can handle both, but there are others that just can handle one type. To solve this obstacle we can transform all data to the type we want or the model needs.

- *Feature extration*: Sometimes our dataset can have some implicit information it is interesting to extract as new values to our dataset.

- *Standarization*: Usually our data is stored in a range that can bias the training. To prevent that we standarize the values. One common standarization is *normalization*.

Model design The next step is to design our ML model. In this step we decide the structure of the model and its *hyperparameters*. This step can (and probably will) be revisited to improve the model's training or its structure depending on the training and testing results.

Train & Test Once our model is designed we can start training it with a subset of the previous dataset. Once the training ended successfully we shall test the model with a subset of the previous dataset that was not used for training so we can simulate the model performance in the work environment. It is important to check the accuracy and loss of the model so they are in our expected range to work correctly. Also it is interesting to store those values to compare them with the TinyML model.

What we explained so far is the same as developing a Machine Learning model. Next, we explain the TinyML steps.

Convert to TinyML By now the model shall be prepared to be deployed in a ML app. Our goal now is to fit the model inside the microcontroller we want to use. To do so we can use the quantization technique, where the bits are reduced. Normally this technique maps decimal values into integer values, which can result into a bad performance of the model. That is why we shall test again the model and compare its accuracy and loss with the non-quantized model.

Create the app The TinyML model is completely developed. Now we just need to create the program we want and add the model to it. This step can be accomplished by different options: developing a code that uses the model weights or using a website that handles all that.

Flash the app Lastly, we flash the app into the microcontroller and check that everything works as we want. We can also check the energy consumption.

4.2.4 TinyML Wake Word app

To familiarize us with TinyML we developed a Wake Word app. The goal of the app is to recognise whether a *yes* or *no* is said and print it on screen. To develop the app we used the framework Edge Impulse, which eases the development of a TinyML app.

As stated in the TinyML steps to develop, first we need to collect data. This can be done with the Edge Impulse option *Data acquisition*, where by plugging our Portenta H7 with the Vision Shield module we can record our own voice, trim it and label it as training or testing data. As one can suppose, all data preprocessing is managed by Edge Impulse, but we can also decide if some records are not valid for the dataset.

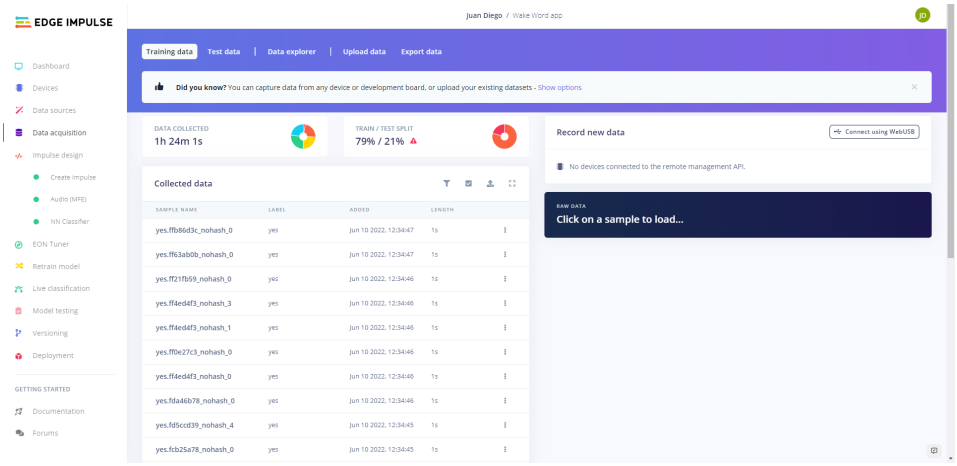

Figure 7: Edge Impulse framework: Data acquisition

Next, we design our ML model. Inside the *Impulse design* option we can create the structure by just selecting and placing processing blocks. In our case we add an *Audio (MFE)* block, to process the audio data, and a *Classification (Keras)* block, a pre-established model to classify data into groups. In our example we have 4 groups: yes, no, unknown and noise.

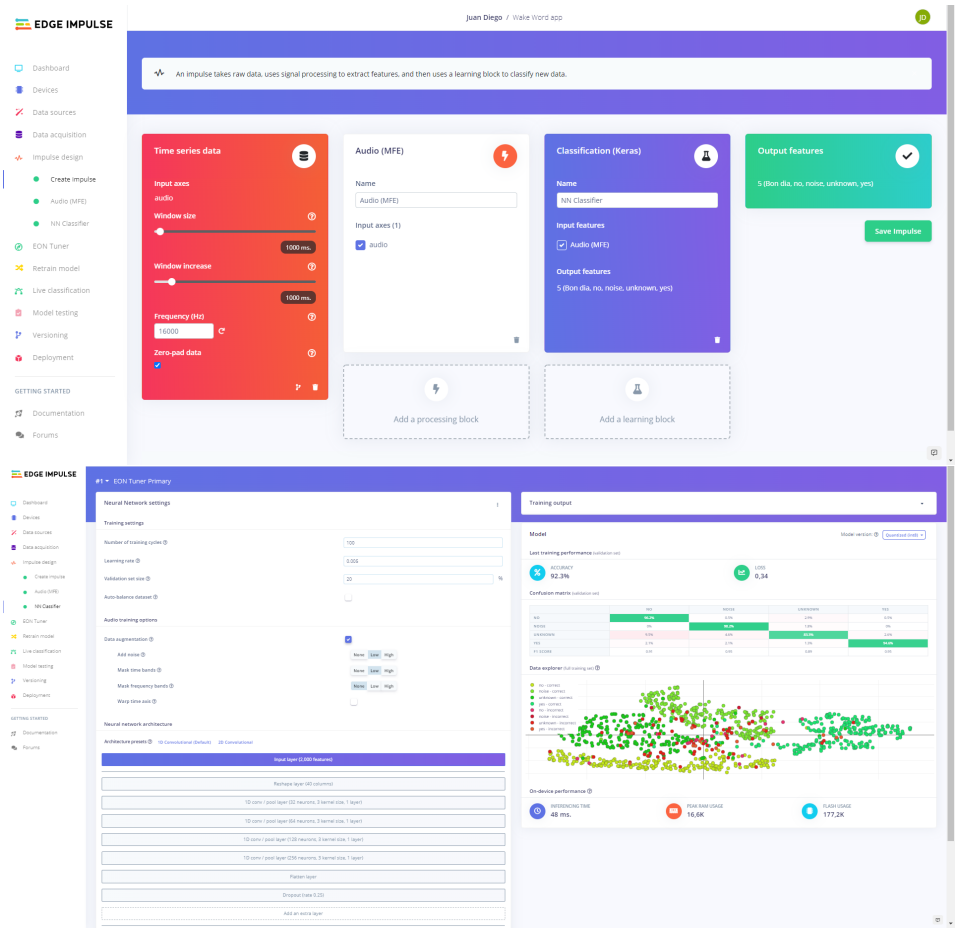

Figure 8: Edge Impulse framework: Impulse design

Once the model is ready to train, we can choose two train options inside Edge Impulse. The first one is *Retrain model*, which is simply training the model, and it shows some data about the training results. The other option is called *EOT Tuner*, which runs multiply training executions with different hyperparameters, and shows some interesting data from its performance (accuracy, latency, RAM and ROM used, etc). At first we used the first option, but as soon as we found the *EOT Tuner* we used it to train the model and know its best hyperparameters.

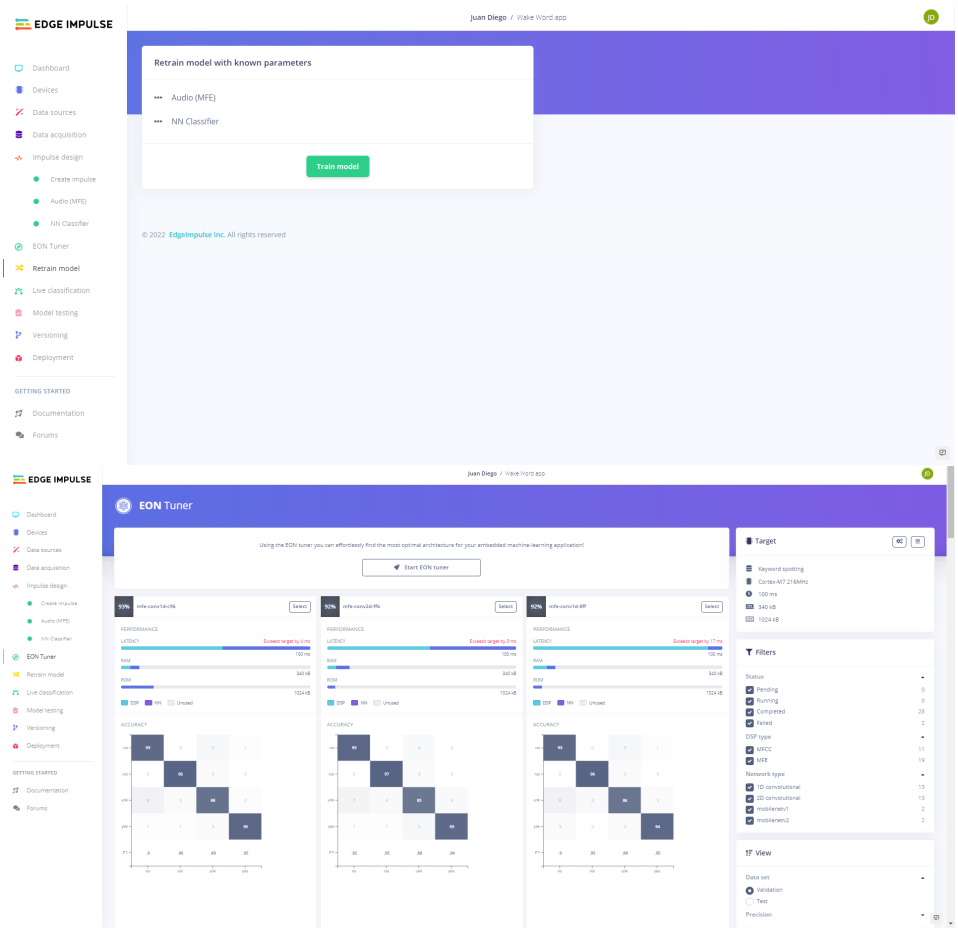

Figure 9: Edge Impulse framework: Retrain model & EOT Tuner

To end with the model development, we test it with the *Model testing* option, which is a *Retrain model* option but with the test dataset. Alternatively, there is the *Live classification* option, that can simulate how the model will work by inserting new data.

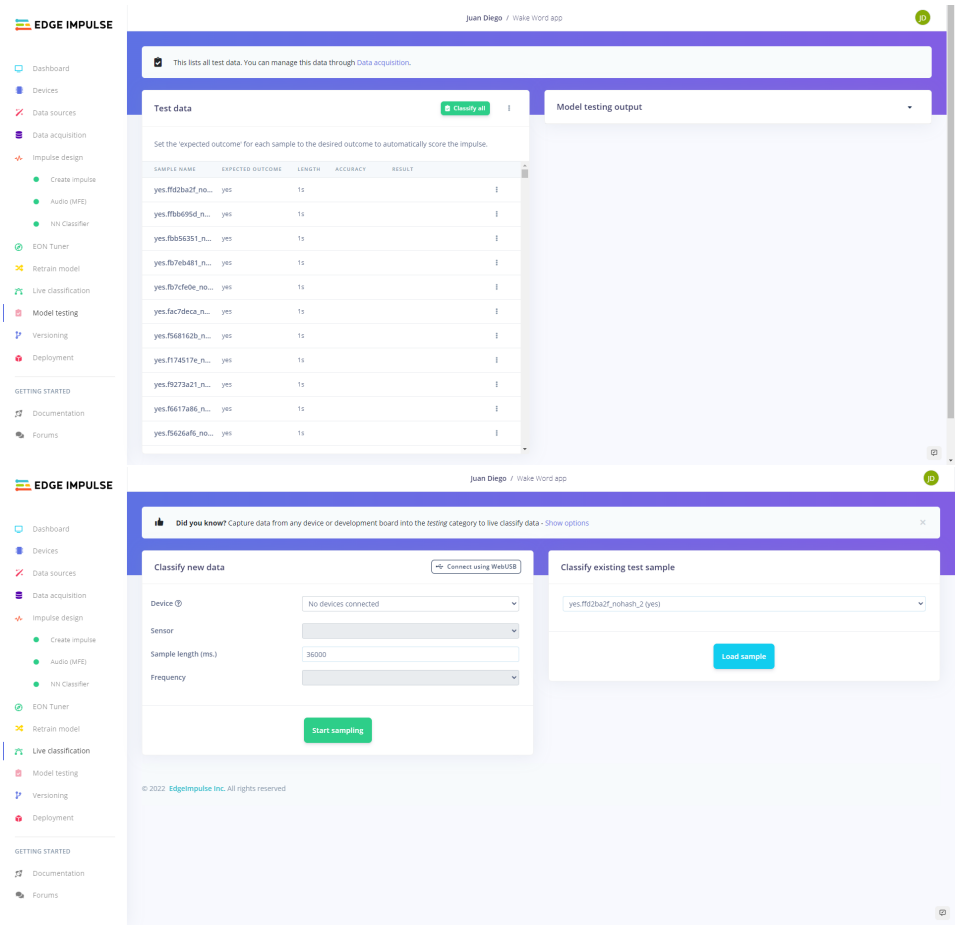

Figure 10: Edge Impulse framework: Model testing & Live classification

Finally, we deploy the model with the *Deployment* option, to choose the library and firmware desired, build and deploy the model. This step without Edge Impulse would have been more complex since we would have had to create an Arduino program, transform the model into an array of weights and more.

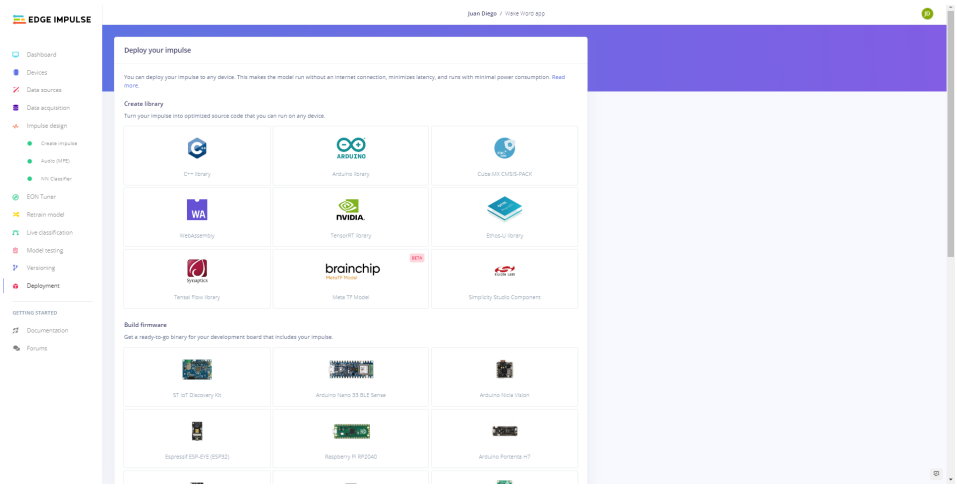

Figure 11: Edge Impulse framework: Deployment

The resulting app worked as planned. The Portenta started the app and printed *noise* when we were not speaking. Then when saying *yes* or *no* the model printed them respectively, and with other words it just printed *unknown*. In a not too noisy environment the app worked as expected.

Furthermore, to prove ourselves we understood how to develop a TinyML app and use the Edge Impulse framework we added a new label, *Bon dia* (which is Good morning in Catalan) to see if we can make the app recognise the word.

We restarted the development but this time adding the *Bon dia* dataset. The model hyperparameters were not very different from the ones we found previously. The final app worked fine as well, it detected the *Bon dia* words and had no error. The development of a TinyML app was a success. Now we can move on to the next part.

5 TinyODL techniques

This section dives into the two techniques of TinyODL we propose: TinyOL and Federated Learning. We explain for each one how it works, its steps and how to develop them.

5.1 TinyOL study

In TinyML we pre-train the model before deploying it into the microcontroller, which only performs inference. This strategy treats the model as a static object, making the deployment of TinyML in the industry environment a challenging task:

- *Costly updates*: Updating each one of the edge devices can be really costly considering its enormous amount, and some of them being in hard accessible places.
- *Inconsistency*: Every machine is different, and so the models trained in different machines, even though they use the same dataset.
- *Data transmission*: Transmitting the field data to the data center is expensive and can cause delay.
- *Concept drift*: The ML model's performance will drop if the input data distribution evolves.

To tackle these challenges, *Haoyu Ren*, *Darko Anicic* and *Thomas Runkler* in his paper [30] proposed a novel system called TinyOL, which means TinyML with Online-Learning.

TinyOL is a system where an additional layer is attached to existing networks. By updating the layer's weights or modifying its structure the application can accommodate to new data classes, and there is no need to store historical data for training since incremental learning is applied. The model is always up to date and can thus deal with the concept drift.

5.1.1 How it works

The TinyOL system proposed starts having a TinyML NN model. This model shall work as usual: it receives data stream, inference every input and returns the predicted results. Then the TinyOL system is attached to the existing model. This attachment can be accomplished by modifying the last layer of the model or adding an extra layer after the model's output. To explain the system easily we talk about the second one, albeit its structure can vary.

The core of the TinyOL system is the additional layer, consisting of several neurons that can be customized, initialized and updated on the fly. It works as the new last layer of the model. Since this layer runs in the RAM it can be trained, unlike the flashed NN model uploaded as a C array, similar to transfer learning.

The workflow of the system can be defined as in the algorithm below. We first initialize the TinyML model and the TinyOL system. Then, for each input received from the streaming data we first inference it, obtaining the output of the model. This output is passed as an input to the TinyOL system making the prediction. But first we can accumulate the mean and variance and standardize the input depending on the tasks.

If a corresponding label is available, the evaluation metrics and the weights in the additional layer will be adapted using online gradient descent algorithms, e.g., stochastic gradient descent (SGD). Thus the training and prediction steps are interleaved. Once the neurons are updated, the sample pairs can be discarded effectively. In other words, at a time, only one data pairs of the stream live in the memory, and there is no need to store the historical data.

Seems like the main goal of this technique is to convert static NN models already developed for MCUs into flexible ones. We also must point out that the robustness obtained against concept drift implies that the field data's statistical properties might vary over time. Moreover, since the model cannot foresee changes in the training face its performance will drop significantly without post-training.

5.1.2 Why it works

But is this technique giving a solution to the challenges TinyML faces? Does it work properly with just training a layer or is it just a theoretical proposition? In the paper we can see a practical example and the mathematical demonstration about how it works.

At first it seems like a non-valid solution since we are running a model that cannot adapt to the environment and we expect a single layer to do so. But the truth is that this approach works and this is why.

Suppose a NN classification model as our TinyML model. We are given a stream of inputs that each one of them is inferred by the model, returning us an array of probabilities for each class the model learned. These probabilities are clue to our training, because when facing unknown classes for the model they will tell some details about them. For example, when recognising animals in images, a *cheetah* class can be represented with high probability values in *tiger* and *zebra* classes, while a *platypus* might have high probability values in *duck* and *mole* classes.

The model output (and its implicit feature extraction) is passed to the TinyOL system. Training the layer with the expected output works as a normal NN model, so there is no need to prove the training algorithm is correct. Also, proving the system does not work correctly (or at least as expected) with one layer is no obstacle, since it only shows the need to add more than one layer to the system, which can be plausible depending on the RAM space left and the total performance.

We encourage the readers to take a look at the practical example proposed in the paper [30]. It is simple and eases the task to understand this technique. In a few words, the practical example shows an TinyML model that tells the state of a fan depending on its performance. The tilted and struck states are not known by the model, and using TinyOL they study its adjustment.

5.1.3 Steps to develop a TinyOL app

Previously we talked about how the TinyML with Online-Learning technique works and why, and the steps to develop a TinyOL have been nearly exposed. Nevertheless, we find interesting to enumerate the tasks needed to apply this technique serving as a guide to anyone interested in developing a TinyOL app.

1. *Create a NN model*: This technique was planned to work as an extension of Neural Network models in TinyOL. That is why we shall design an NN model.

2. *Train & Test*: We train and test the model with the representative dataset of the environment.

3. *Convert to TinyML model*: Once the model is ready to be deployed we convert it into a TinyML model so it can be executed inside our microcontroller.

4. *Design the TinyOL extension*: After having the TinyML model, we design the structure of the trainable part inside the app. This step is normally revisited at least once to improve the final model's training performance.

5. *Deploy the TinyOL app*: Then, we develop the app and deploy it inside the MCU so we can test it.

6. *Train & Test the TinyOL system*: Finally, we check the app works as expected and the TinyOL system learns new data and adapts correctly. If the system performance is not enough we may revisit the previous steps.

To sum up, developing a TinyOL app is in fact developing a TinyML app and add a TinyOL system. This idea reminds us about the idea of developing a TinyML app, which is developing an ML model and convert it into a TinyML one.

5.1.4 Tools to develop a TinyOL app

This technique is very recent, so there are no useful tools to develop these type of apps more than the TinyML tools. In fact, the tools used to develop TinyML apps can be useless in situations where we need to modify the model's last layer. Of course though, the best potential for this technique is to add a layer (or layers) as an extension to the existing model.

To develop the model's extension it may be useful to search for a C library that eases the development. In this case Python's packages for modelling ML models would be very useful.

5.1.5 Benefits & disadvantages

Now, we are going to process all the information we obtained from the study and conclude the advantages and obstacles or problems this technique entails.

Benefits

- *Flexibility*: This technique makes our program capable of adapting to changes in the environment. The model can learn new data and respond with a considerable accuracy.

- *Adaptability*: One of the main advantages, and probably the most iconic, is the possibility of converting normal TinyML static programs into dynamic ones with just adding a TinyOL system. This characteristic makes the TinyOL technique very powerful.

- *Simplicity*: Another improvement is that there is no need to know new algorithm or concepts to apply this technique. With just knowing how to develop a Neural Network model we can make our own TinyOL system.

Disadvantages

- *Limited to NN*: TinyOL is still young and it only has been proven with Neural Network models. Although a lot of TinyML models are artificial Neural Network, some other useful ML models cannot be implemented with this technique.

- *Complex to develop*: There are not any *keras* or *pandas* packages in C to design NN models, so here the technique takes a level of complexity. Moreover, we can face some situations where it is required to develop the structure from scratch.

- *High performance training*: It is true that TinyOL uses a reduced model to train, but we are still performing high computation (the model update) inside the microcontroller, which restricts the possible TinyOL systems to very simple NN models.

- *High computing dependency*: Countering the benefit of converting existing TinyML models, the TinyOL technique cannot model complex and flexible ML from scratch, and one of the reasons for it is the previous obstacle. In some situations we will still require a powerful computer to create our TinyML model.

5.2 Federated Learning study

Federated Learning [31] is a Machine Learning technique that trains an algorithm across multiple decentralized edge devices or servers holding local data samples, without exchanging them. This approach stands in contrast to traditional centralized machine learning techniques where all the local datasets are uploaded to one server, as well as to more classical decentralized approaches which often assume that local data samples are identically distributed.

Its general principle consists in training local models on local data samples and exchanging parameters between these local nodes at some frequency to generate a global model shared by all nodes. This technique allows for smarter models, lower latency and less power consumption, all while ensuring privacy.

The main difference between Federated Learning and distributed learning lies in the assumptions made on the properties of the local datasets, as distributed learning originally aims at parallelizing computing power where federated learning originally aims at training on heterogeneous datasets. The local datasets are not assumed to be independent and identically distributed or roughly the same size. Instead, they are typically heterogeneous and different size that can vary in several orders of magnitude.

To sum up, the idea is to train a model as a combination of models pre-trained with local datasets. We can find variations in the Federated Learning settings:

Centralized Federated Learning A central server is used to orchestrate the different steps of the algorithms and coordinate all the participating nodes during the learning process.

Decentralized Federated Learning the nodes are able to coordinate themselves to obtain the global model, preventing single point failures as the model updates are exchanged only between interconnected nodes. This is the best setting when performing in a collection of edge devices.

Heterogeneous Federated Learning It can enable the training of heterogeneous local models with dynamically varying computation and non-iid data complexities while still producing a single accurate global inference model. This technique was recently proposed to address heterogeneous clients equipped with very different computation and communication capabilities.

5.2.1 How it works

The Federated Learning technique relies on an iterative process broken up into what is called as a *federated learning rounds*. Each round works as follows:

1. **Initialization**: A Machine Learning model is chosen to be trained on local nodes and initialized. The local nodes are activated and wait for the central server.

2. **Selection**: A subset of the local nodes is selected to train local models. The current global model state is transmitted to the participating local nodes.

3. **Local training**: Each node trains a local model with its local dataset and produces a set of potential model updates.

4. **Reporting**: The local updates are sent to the central server to be aggregated and processed into a single global update. The central server also handles failures for disconnected nodes or lost model updates. This global update is sent back to the local nodes.

We repeat the process until a condition is met, where the central server aggregates the global update into the global model and ends the Federated Learning technique.

The steps above are explained with a central server orchestrating all the participating nodes and global states and updates. But with a *peer-to-peer* approach (using *gossip* or *consensus* methodologies) we can get the same results. For example, this paper converts a centralized FL algorithm to work without a central server with peer-to-peer settings.

In a decentralized FL the main idea is to train the local models inside the local nodes. A concept known as *knowledge learning* steps in. The knowledge learning is simply input random data from the domain (known as a transfer vector) to, lets say, two local models, and use the output of one model as th expected output to train the other model. To do so, at least one model shall be pre-trained to ensure

We can find variations of the Federated Learning technique like *Federated Stochastic Gradient Descent*, *Federated averaging*, *Federated Learning with Dynamic Regularization*, etc.

5.2.2 Steps to develop a FL app

To develop a TinyML app with Federated Learning we first need to define the model design and its specifications. It is also important to define the Federated Learning settings, since some model specifications may vary for a better adaptation. Next we design the model based on the previous decisions.

Unlike TinyOL, we do not need to focus in the model composition, but the algorithm used to train it. We start the program development before the model training because the program itself is the training.

One thing to define before developing the program is the communication method, which depends on the available resources and the Federated Learning settings.

Centralized FL

With a centralized setting we need to develop:

1. Central server program: This is the main program since it is the one who will call the local nodes to train and infer. We shall aim to develop a fast model update aggregation and an optimized node selection function (if needed). In this paper we can find some robust aggregation algorithms.
2. Local node program: In this program we train a local model with a local dataset that can be either in memory or obtained by the device sensors.
3. Communication: A well defined and optimized communication is desired since this is our bottleneck.

Decentralized FL

A decentralized setting demands to develop the next functions for the node program:

- Model initialization: We shall initialize the local model. We can initialize the model from scratch or as a pre-trained model. This decision is up to the developer.
- Model training: The training algorithm for a local dataset.
- Inference: A function to infer the received data.

We also need to develop the main program with:

- Communication between nodes: A peer-to-peer communication shall be well defined and optimized in order to reduce the bottleneck.
- The knowledge learning function: First we need to obtain the random data, which can be accomplished with the device sensors. Then with the training and inference functions we communicate models outputs in order to train.

5.2.3 Benefits & disadvantages

Given the previous information, we can define the next advantages and conflicts:

Benefits

- *Distributed training*: In a well distributed node system the cost of training the model is divided by the training of local models and their aggregation.
- *Model learning*: Federated Learning is applied to the model that infers the data, whereas in TinyOL we create an extended system to learn.
- *Heterogeneity*: Since the local models are trained with different datasets with no previous assumption we develop different models that learn to respond better in some situations than others, but in general there is at least one model with a good response.
- *Adaptability*: Unlike TinyOL, this technique is available to any supervised ML model. Furthermore, it can be adapted to reinforcement learning if needed.

Disadvantages

- *Complex development*: Although the training of local models have no difficulty, its aggregation to a general model and the communication between nodes makes the development even more complex.
- *Communication bottleneck*: Communication between devices can be a severe bottleneck in the performance of the algorithm.
- *Lack of libraries*: There are not developed libraries for applying this technique, which makes harder its programming.

6 Develop TinyOL app

Once we studied both TinyOL and FL techniques, understood its steps to develop and listed their benefits and problems, we decide to make some practical study. The main purpose is to develop a TinyML application and apply the TinyODL technique to compare results. Albeit developing both technique would be ideal, the time constraint of this project force us to decide which technique develop.

6.1 Why we chose TinyOL?

Our goal is to see how a TinyODL technique improves the program adaptability. We want to develop the program as quick as possible and TinyOL is, by far, faster to develop than a TinyML with FL program. We do not need to worry about communication between nodes or a central program, and more.

We want to compare the outputs between a TinyML and a TinyODL app. Since TinyOL needs first a TinyML model to be developed, it seems the best option since we do both apps simultaneously.

6.2 Image classification app

Image classification in edge devices is really demanded these days. There are already models capable of classify images or detect people [32][33], but they are static models incapable of learn new data. That is why developing a TinyOL image classification app was an interesting opportunity we do not want to waste.

6.2.1 App definition

We want to develop a program that when executed inside our Portenta H7 it can classify objects with the Vision Shield camera. First we want the program to classify easy objects, fruits at first, and try to raise the difficulty to more complex objects when the fruits classifier works.

The workflow of the app will be as follows: First, an image is captured from the camera. Then it is preprocessed and inferred into the TinyML model and the TinyOL system. The output is shown to the user and it can decide whether tell the program the expected output or just restart the process with a new image. In case of taking the first action, the expected output is read by the TinyOL system with the output inferred previously, and trains the TinyOL system.

To create the TinyOL app we first need a TinyML app that works. Here two options arise: create our own TinyML app or take one already created. At first it seemed obvious to not waste time in creating the app ourselves, but thought twice we decided it is best to know how the model is designed and being able to tune it to our interest. The final decision is to create the TinyML app ourselves. Image classification models are currently modeled as a Convolutional Neural Network, and so is our model.

6.2.2 Tools

Previously in this document, we created a TinyML model with Edge Impulse. In this case, though, we want to deeply know the inner structure of the model, so we design and train the model with the next tools:

- *Tensorflow Lite*: Tensorflow helps us to design a CNN and tune parameters and hyperparameters easily. With Tensorflow Lite we convert our model into a TinyML model. Inside Tensorflow Lite we can find Tensorflow Lite for Microcontrollers, which helps the development of TinyML apps with Tensorflow models.
- *Google Colab*: Our preferred editor to create the TinyML model.
- *xxd*[34]: To transform the TinyML model into a C array for the microcontroller.
- *PlatformIO*: Editor to develop the TinyML and TinyOL apps.

6.3 Development

We are going to split this part in two. First we explain the development of the TinyML app and finally we explain how we develop and add the TinyOL system to the TinyML app so it can train.

All the source code to develop the TinyOL app can be found at my GitHub repository: *juandiegorubio/TinyODL-TFG*[5], inside the *TinyOL/image-recognition* folder.

6.3.1 TinyML app

We learned in the TinyML study that TinyML models are just ML models optimized for edge devices. Our first task is to develop a Convolutional Neural Network capable of classify fruits in images.

The fruits dataset we used for the training is from Kaggle, called Fruits 360[6]. This dataset has up to 133 different fruits, so our model is going to classify from 133 different classes.

We first try some CNN models with the Keras package in Tensorflow. With the Convolutional layers we tried to lower down the parameters to train. By testing models we ended up with the below model structure. Albeit it has quite few convolutional layers it had the best accuracy and the lowest loss compared to all the other models, so we move on with this one. We can always turn back to this step if the model will not fit the microcontroller limitations.

[5] https://github.com/juandiegorubio/TinyODL-TFG
[6] https://www.kaggle.com/datasets/moltean/fruits

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 16)        448

max_pooling2d (MaxPooling2D  (None, 33, 33, 16)        0
)

dropout (Dropout)            (None, 33, 33, 16)        0

conv2d_1 (Conv2D)            (None, 31, 31, 32)        4640

max_pooling2d_1 (MaxPooling  (None, 15, 15, 32)        0
2D)

dropout_1 (Dropout)          (None, 15, 15, 32)        0

conv2d_2 (Conv2D)            (None, 13, 13, 64)        18496

max_pooling2d_2 (MaxPooling  (None, 6, 6, 64)          0
2D)

dropout_2 (Dropout)          (None, 6, 6, 64)          0

conv2d_3 (Conv2D)            (None, 4, 4, 128)         73856

max_pooling2d_3 (MaxPooling  (None, 2, 2, 128)         0
2D)

dropout_3 (Dropout)          (None, 2, 2, 128)         0

flatten (Flatten)            (None, 512)               0

dense (Dense)                (None, 100)               51300

dropout_4 (Dropout)          (None, 100)               0

dense_1 (Dense)              (None, 70)                7070

=================================================================
Total params: 155,810
Trainable params: 155,810
Non-trainable params: 0
_________________________________________________________________
```

Figure 12: Image classification model structure

Next, the model is trained and converted into a TinyML model. To convert a Tensorflow model into a TinyML model we use the Tensorflow Lite object called *TFLiteConverter*[7]. In our case we just want the model to be quantized, so the default optimizer is a good option. With that defined, we can convert the model with the *convert* function of the object, which returns us our CNN model as a TinyML model.

[7] https://www.tensorflow.org/api_docs/python/tf/lite/TFLiteConverter

```
converter = tf.lite.TFLiteConverter.from_saved_model("CNN_image_classification")
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()
```

Figure 13: Convert CNN model into a TinyML model

Up until now we have a TFLite model, which is what we want. But there is one problem, we have the model readable for Python and the microcontroller works in C. To make the model readable for C programming we want to use the xxd command, which transforms our TFLite model into a C array of weights, our TinyML model representation in C.

```
# Install xxd if it is not available
!apt-get -qq install xxd
# Save the file as a C source file
!xxd -i cnn_image_classification.tflite > cnn_image_classification.cc
# Print the source file
!cat cnn_image_classification.cc
```

Figure 14: Convert TFLite model into a C array

Now we just need to develop our TinyML app in PlatformIO. This is done by using the TFLite for microcontrollers (in this document we also call it TFLite Micro) library developed by the TensorFlow team. Luckily, we have some examples of how a TFLite Micro app shall be. In this example we can find the next files:

output_handler.cc : File with the HandleOutput function. The function receives the resulting output of our model. In figure 20 the output values are shown with the *TF_LITE_REPORT_ERROR* function.

main.cpp : The principal code, where we initialize the model and other objects, take the input, get the model's output and call the output handler function. Below we explain in more detail this program. See figure 21.

constants.cc : Some constants for the main code.

We also add our C array model file into the *src* folder. The C array model will be called in the *main.cc* code. We now explain how the main code works.

How the TFLite Micro main program works:

We want the main program to initialize all the objects, read the input, infer the output and handle that output. To do all this we first need to prepare the memory space where the model will store the values and do all its work. First, some TFLite Micro global objects for the model management are declared. Then we define what is called the *TensorArena* and its size, which is the reserved memory space for the model to do its calculations.

Inside the *setup* function we initialize all the global objects, we create our model object with the function *tflite::GetModel(model_array_name)*, which takes as parameter the C array name of the model we converted previously with the *xxd* command. Finally, we initialize a *MicroInterpreter* object with all the model variables we initalized earlier (the model object, the tensor arena, etc). Last but not least, we store in *TFLiteTensor* objects the input and output tensor of our model.

Finally, the *loop* function (figure 23) reads an input, it stores the input inside the input tensor, takes resulting value from the output tensor and calls the *HandleOuptut* function to handle the model's output.

Seems pretty easy to develop, so we followed all the steps to do so. Here is when problems start to arise. After some difficulties to learn how to work with the environment, we encounter a problem of dependencies. When compiling the project the compiler tells us that the *utility.h* file is not found from a TFLite Micro file.

With some research we find this file comes from an essential library in C programming, called *asio*, inside the *boost* library. Stranged by the situation, we try to add it in any way possible by installing the library via PlatformIO, downloading ourselves the library and adding it to the project and more. But none of them worked.

6.3.2 TinyOL system

We did not have the chance to develop any TinyOL system due to the previous problem.

6.4 Results

The development of the TinyML model was impressively easy taking into account that we made the model from scratch. Also its adaptability to a C program is pretty useful, being able to transform any ML model into a C array TFLite model.

We also found pretty simple to work with TFLite Micro, but we got disappointed when a dependency problem from the TFLite Micro library arose. We tried our best to solve the problem but solving the dependency problem got us many more dependency problems, which became in an endless loop.

Since we arrived to this end point at the beginning of July, we thought it would be interesting to exploit the rest of the summer in developing our other TinyODL technique: Federated Learning. This can be found in the Develop Federated Learning app section.

7 Develop Federated Learning app

At this point of the project we encountered a dead end in the TinyOL app development and were a few weeks from the final delivery. So we decided it would be interesting to dig into the Federated Learning technique and try to develop an app with it.

As planned previously, we proposed an image classification app to run and learn inside the Portenta H7. Below we explain the idea and structure of the app. All the source code can be found at my GitHub repository: *juandiegorubio/TinyODL-TFG*[8], inside the *FL_with_CNN* folder.

7.1 Adjust image classification app

Our app goal is the same, to classify fruits based on real-time execution samples. The workflow will be quite similar, since an image will be captured, processed to be classified and either end here or used the input to train the model [9].

A change we find in this app development is that we do not need a TinyML model trained previously to work with it, since Federated Learning trains the model directly (unlike TinyOL that trains an add-on model), being able to develop directly the FL model without redoing steps previously studied.

7.1.1 Tools

As stated before in this document, there were not many software tools to develop TinyODL techniques, so most of our development was from scratch. Here are the tools we used to develop the app:

- *PlatformIO*: To develop, flash and debug the project.
- *RPC library*: Arduino library to connect both M7 and M4 cores and talk with each other.

We also support our implementation with Nil's Federated Learning implementation [35] inside the project *Federated Learning on embedded devices* [36].

7.2 Development

First things first, the app needs a CNN model. That is why our first task is to develop a CNN model in C. There were a few libraries in C to develop ML models [10] (for example OpenCV [37] or the Microsoft Cognitive Toolkit CNTK [38]), but none of them were libraries to work with edge devices. That is why we decided to code our own CNN model as simple as we can, fitting the model size to the microcontroller capacities.

[8] https://github.com/juandiegorubio/TinyODL-TFG

[9] It is important to point out that these similarities makes the possibility for both techniques to be adapted with each other and work together as idea and future proposals.

[10] In https://hackernoon.com/top-cc-machine-learning-libraries-for-data-science-nl183wo1 one can find more examples.

7.2.1 Similar NN implementations

A quick review in Nil's implementation, we see at first the class *Neural Network* consists of two layers, the *hidden* and the *output* layer (see figure 24). Both have 3 types of arrays to store information:

- *HiddenWeights / OutputWeights*: Stores the weights and biases of the layers.
- *Hidden / Output*: Stores the resulting value of each neuron in the layer.
- *ChangeHiddenWeights / ChangeOutputWeights*: Stores the changes added to the weights. This data is used only for training the network.

Additionally, the number of neurons of each layer, the input size and the maximum and minimum values of each weight are declared as global variables. About the functions, first some basic functions to initialize the network correctly with random weights inside the range of values (figure 25). Also some basic *getters*[11] are declared.

Next, the *forward* function, in figure 26, consist of simply the vectorial product of the input and the hidden layer weights, and the hidden layer's output and the output layer weights. The error is calculated while calculating the output values and returned. Each *for* loop stands for each of the layer's products we do, in this case two because we only have the hidden and output layers.

Finally, the *backward* function, in figures 27-28, first infers the input to the network, the same as the forward function. Once the output is obtained the gradients of the layer weights are calculated to change the weight values.

Once we confirmed that Nil's project worked correctly its time to develop our own neural network, in our case a Convolutional Neural Network. Since we do not know exactly the model structure we need to develop a code the number and type of layers can be easily exchangeable. Taking that into account, we decided to develop a code capable to simulate the workflow of developing a ML model in TensorFlow.

7.2.2 Layer implementation

A NN is simply a set of layers, which is why our first step is to declare a layer class (figure 29), which will be a common parent to all the layers we want to develop. This class essentially contains the *ReLu* and *Sigmoid* activation functions, the forward and backward function as *virtual*[12], and some common values as the input and output size, a function to get random integers and random float values, and more. The activation function derivative is also stored in order to calculate the output gradient.

The forward function starts taking the layer's input as argument and returns a pointer to the output obtained from inferring the input. Meanwhile, the backward function takes the layer's input and the gradient of the output, and returns the gradient of the previous layer's output.

[11] A *getter* is a common word to denote those functions which goal is to let the user obtain some private information about the object.

[12] A *virtual* function [39] is a member function which is declared within a base class and is re-defined (overridden) by a derived class.

Dense layer implementation

First we start with the dense layer, which we store as an array of weights as we see in the figures 30-31. We first initialize the weights as random values (either float or integer values, depending on the microcontroller capacity).

The forward function simply consists of calculating the product of the input and weights, resulting into the layer's output, which is returned. The backward function just takes the layer's input and gradient passed as arguments in order to calculate the changes in the weights to train the layer and get the output gradient of the previous layer. Finally the output gradient of the previous layer is returned. Both functions are in figure 32. To compute the change values we use the same calculations as Nil's implementation.

Convolutional layer implementation

As expected in a Convolutional Neural Network we need to develop a Convolutional layer. I never had the chance to study the convolutional layer implementation, so a previous study was needed. After understanding and contrasting information [40][41], we came up with the next implementation. For a deepen explanation of how a convolutional layer works we recommend to take a peek in these websites [42][43][44][45].

We store each filter and kernel as a unique array of weights (figure 33), similar to a dense layer. To access each filter and kernel we just need to multiply the desired filter or (either the first one *0*, the last one *n-1*, etc) with the filter size or kernel size, respectively. Differing from the dense layer implementation, the bias values are stored in another array as a pure decision to make the code lighter to understand. We initialize the kernels and bias values with random values as well (figure 34).

The forward function (figure 35) is the matrix product of the input and the kernel plus bias values. When it comes to the backward function (in figure 36, the computation of the change values is more complex. We strongly recommend to visit the previously cited web pages. In a summarized explanation, we compute the output gradient of the previous layer, the gradient of the kernels to change the weights, and the gradient of the bias values to change them.

The kernel and bias gradients are computed as the dense layer, but to compute the previous gradient we first need to convolve gradient of the actual layer and the kernel values (previous to be modified). In figure 37 one can see how this convolve function is implemented in order to obtain the previous gradient.

7.2.3 Loss function implementation

Similar to the layer class, we define in figure 38 the Loss class as parent of all the loss functions we want to develop. The class only contains two virtual functions: the loss function and the derivative of the loss function. We decided that for our initial necessities implementing the Mean Squared Error (MSE) function [46] was sufficient, and in future cases we could implement new loss functions just by creating new loss derived classes.

7.2.4 NN class implementation

With the loss function and layers implemented we can now move to the network implementation. The class essentially stores a set of layers systematically ordered, the model's error and the loss function used to compute it.

The *add* function is a public function to add a new layer as the last layer of the model. The forward function is a private function that given an input computes the model's output. Its implementation consists of a loop where we call all the model layer's forward function and take the output as the input for the next layer. When we get to the last layer, the final output is returned. The implementation is shown in figures 39-40.

The *backpropagate* function, also private, works similar to the forward function: we visit inversely the model layers and call the *backward* function, which returning gradient serves as the output gradient for the previous layer.

The *test* and *predict* functions (figure 42) are both public functions of the class. The *predict* is to infer inputs to the model and obtain an output, whereas the *test* infers a set of inputs and returns the resulting output loss for each inference.

Finally, the *train* function (in figure 41) is the public function in charge of training the model given a set of inputs and expected outputs, the epochs and batch size. The function does for each epoch *(number of inputs)/(batch size)* batches inferring all the inputs in each batch.

Once the batch is completed, the backpropagate function is called in order to train the model from all the gradients obtained from the predicted outputs and expected outputs in the batch. Finally, at the end of the epoch the error is weighted and shown if we want to.

7.2.5 Dual-Core communication

During the neural network implementation we also started to study how to communicate both M7 and M4 cores in the Portenta H7. To do so, we studied Nil's implementation of a Dual-Core program and the *RPC* library from Arduino. Thanks to it we were able to develop two simple codes: a simultaneous LED blinking and a program able to call functions from one core to another. Here are some interesting results we got when developing both programs.

Dual-Core LED blinking

The program had the M7 core blinking the green LED while the M4 core was blinking the red LED.

- We were able to made them blink both with different periods.
- The vatheriable storing the LED port can be global and modified inside each core functions with the *CORE_CM7* and *CORE_CM4* macros.
- We were not able to share data between cores, for example by sharing the same variable or memory address.

Calling Cores

The program consisted of two loops, one for each core. Each core had to compute a value calling the function to the other core, and show the resulting value via the serial port. This program used the RPC library and a thread object in order to run both loops.

- We are not able to print values from the M4 core direclty to the serial port, but with the RPC *read* and *println* function we can.
- We are still not able to share data between cores, but functions can called in both cores as separate executions.
- A thread is not required to call functions in both cores, but it is required when executing parallel.

7.3 Results

Once September came we were unable to continue developing those implementations. Here, we point out some interesting results we made so far.

First of all, the model network instance worked correctly, except for the training part. Either the train, backward or backpropagate function was not implemented correctly, so all the weights went to 0, infinite or non-representable numbers.

When testing the network class we confirmed the flexibility and commodity of creating and modifying the model structure.

The M7-M4 cores were able to work parallel and interact with each other, but an important restriction is that we could not find a way to share variables or memory between them. We would need more time to find a usable option.

8 Conclusion

8.1 Project conclusions

We have seen a TinyML application is easy to develop and deploy, and available to anyone interested since we just need an edge device and the free tools used in this project. When it comes to TinyODL techniques things got more complicated, since there is a previous knowledge required to develop the training technique.

With TinyOL we found out that developing the model is easier that expected thanks to TensorFlow Lite for Microcontrollers, but some ANN structure and working knowledge is demanded for the C development of the app. A challenge one can find, though, when developing a TinyOL app is facing dependency errors.

As for the Federated Learning development section we found out we can develop our own layers for ANN in order to find the most efficient and fast execution, adapting the model to our necessities. More time was needed in order to complete the model and develop the technique. We focused in analyse the technique.

8.2 Personal conclusions

I find both techniques interesting to develop and push beyond their limits. On one hand, I think TinyOL is the best option to develop easy TinyODL apps and it does not require a lot of knowledge as in FL. On the other hand, I think FL is the best option for professional TinyODL applications, by developing a model more exact to solve the task given the device and environment restrictions.

This project has been a hole new experience for me. I have been able to merge hardware and high level software into one objective. I also learned a lot of new concepts and techniques I expect to use in my new projects. One example is Edge Impulse, a framework to easily develop TinyML apps. Or the Federated Learning technique which can be useful for normal ML training in some projects, like the robot project I am working nowadays.

8.3 Future of TinyODL

Just by reading some of this project one can see TinyODL techniques have just started their way. The lasts August 31st and September 1st there was a forum [47] from the tinyML foundation [48] where they talked about new possible techniques for on-device learning. We can also find interesting papers like *On-Device Training of Deep Learning Models on Edge Microcontrollers* [49], where a novel Echo State Network model enabled on-device training on STM32 MCUs with good results.

Moreover, all TinyODL techniques are oriented to ANN, but other ML approaches are not visited, which can be a new path to follow. Either way, we see TinyODL has a lot to research for and seems the new step edge devices need to take for this new technology era.

9 Annexes

9.1 Task planning table

Task	Time (h)	Task Dependencies	Resources	People Assigned
PM	**93**	**-**	**PC, NO, GAN, GM, GCA, OV**	**MAN, RE, PR, DA**
PM-1	5	-	PC	MAN
PM-2	12	PM-1	PC, NO, GAN	MAN
PM-3	3	PM-1, PM-2	PC, NO	MAN
PM-4	3	PM-1, PM-2, PM-3	PC	MAN
PM-5	15	-	PC, GM, GCA	MANx2
PM-6	40	PM-1, PM-2, PM-3, PM-4	PC, OV	MAN
PM-7	15	PM-6	PC	MAN
PS	**38**	**-**	**PC, IDE, PIO, PH7, VS, US, GIT**	**RE, PR**
PS-1	6	-	PC	RE
PS-2	10	-	PC, IDE, PIO	PR
PS-2.1	3	-	PC, IDE	PR
PS-2.2	7	-	PC, PIO	PR
PS-3	5	PS-1, PS-2	PC, IDE/PIO, PH7, USB, GIT	PR
PS-4	17	PS-1, PS-2	PC, IDE/PIO, PH7, VS, USB, GIT	PR
PS-4.1	5	PS-1	PC	RE
PS-4.2	12	PS-1, PS-2, PS-4.1	PC, IDE/PIO, PH7, VS, USB	PR
TML	**93**	**-**	**PC, EI, PH7, VS, USB, GIT**	**PM, RE, PR, DA**
TML-1	6	-	PC	RE
TML-2	8	TML-1	PC	RE
TML-3	3	TML-1, TML-2	PC	MAN, PR, DA
TML-4	13	-	PC	PR
TML-5	48	TML-2, TML-3, TML-4	PC, EI, PH7, VS, USB, GIT	PR, DA
TML-5.1	10	-	PC, EI	PR, DA
TML-5.2	8	TML-2, TML-3	PC, EI	PR
TML-5.3	12	TML-5.1, TML-5.2	PC, EI	PR, DA
TML-5.4	8	TML-2	PC	RE
TML-5.5	10	TML-2, TML-5.(2,3,4)	PC, EI, PH7, VS, USB	PR, DA
TML-6	15	PS, TML-4, TML-5	PC, EI, PH7, VS, USB, GIT	PR, DA
TOL	**47**	**TML-1, TML-2**	**PC**	**RE, DA**
TOL-1	10	TML-1, TML-2	PC	RE
TOL-2	10	TML-2, TOL-1	PC	RE
TOL-3	12	TOL-2	PC	RE
TOL-4	15	TOL-1, TOL-2, TOL-3	PC	RE, DA
FL	**47**	**TML-1, TML-2**	**PC**	**RE, DA**
FL-1	10	TML-1, TML-2	PC	RE
FL-2	10	TML-2, FL-1	PC	RE
FL-3	12	FL-2	PC	RE
FL-4	15	FL-1, FL-2, FL-3	PC	RE, DA
ODL	**176**	**TOL, FL, TML-5, TML-6**	**PC, GCO, TFL, PIO, PH7, VS, USB**	**PM, PR, DA**
ODL-1	10	TOL-4, FL-4	PC	MAN
ODL-2	25	(TOL/FL), ODL-1	(GCO,TFL / -)	PR
ODL-3	3	ODL-1, ODL-2	PC	MAN, PR, DA
ODL-4	48	TML-5, TML-6, ODL-3	PC, PIO, PH7, VS, USB	PR, DA
ODL-5	55	(TOL-2/FL-2), ODL-2, ODL-3	PC, (GCO,TFL / -), PIO	PR, DA
ODL-6	25	ODL-5	PC, PIO, PH7, VS, USB	PR, DA
ODL-7	10	ODL-4, ODL-6	PC	DA
TOTAL	**494**	-	-	-

Table 8: Project tasks breakdown.

9.2 Gantt chart

	Nombre	Inicio	Fin
1	**Project Management (PM)**	**03/17/2022**	**06/27/2022**
2	PM-1	03/17/2022	03/18/2022
3	PM-2	03/18/2022	03/23/2022
4	PM-3	03/24/2022	03/24/2022
5	PM-4	03/25/2022	03/25/2022
6	PM-5	03/17/2022	06/17/2022
7	PM-6	03/28/2022	06/17/2022
8	PM-7	06/20/2022	06/27/2022
9	**Previous Study (PS)**	**03/17/2022**	**04/04/2022**
10	PS-1	03/17/2022	03/18/2022
11	**PS-2**	**03/18/2022**	**03/22/2022**
12	PS-2.1	03/18/2022	03/22/2022
13	PS-2.2	03/18/2022	03/22/2022
14	PS-3	03/23/2022	03/25/2022
15	**PS-4**	**03/23/2022**	**04/04/2022**
16	PS-4.1	03/23/2022	03/28/2022
17	PS-4.2	03/29/2022	04/04/2022
18	**TinyML approach (TML)**	**03/21/2022**	**04/12/2022**
19	TML-1	03/21/2022	03/22/2022
20	TML-2	03/22/2022	03/24/2022
21	TML-3	03/24/2022	03/25/2022
22	TML-4	03/23/2022	03/28/2022
23	**TML-5**	**03/29/2022**	**04/08/2022**
24	TML-5.1	03/29/2022	04/01/2022
25	TML-5.2	03/29/2022	04/01/2022
26	TML-5.3	04/04/2022	04/06/2022
27	TML-5.4	03/31/2022	04/06/2022
28	TML-5.5	04/06/2022	04/08/2022
29	TML-6	04/08/2022	04/12/2022
30	**Study of TinyOL (TOL)**	**04/11/2022**	**05/10/2022**
31	TOL-1	04/11/2022	04/13/2022
32	TOL-2	04/14/2022	04/18/2022
33	TOL-3	04/19/2022	04/22/2022
34	TOL-4	05/09/2022	05/10/2022
35	**Study of Federated Learning (FL)**	**04/11/2022**	**05/10/2022**
36	FL-1	04/11/2022	04/13/2022
37	FL-2	04/14/2022	04/18/2022
38	FL-3	04/19/2022	04/22/2022
39	FL-4	05/09/2022	05/10/2022
40	**Development of a TinyODL app (ODL)**	**05/11/2022**	**06/17/2022**
41	ODL-1	05/11/2022	05/12/2022
42	ODL-2	05/12/2022	05/16/2022
43	ODL-3	05/17/2022	05/17/2022

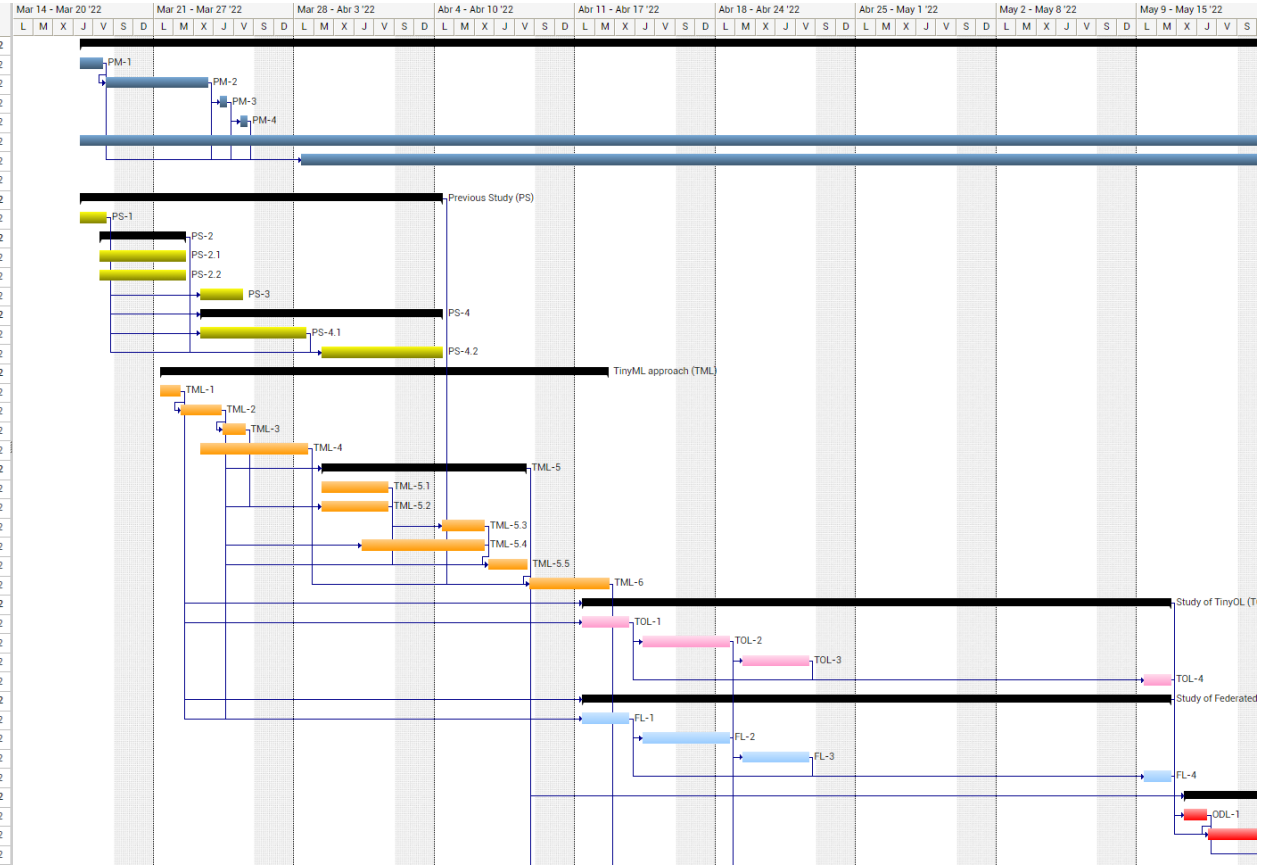

Figure 15: Gantt chart: First period.

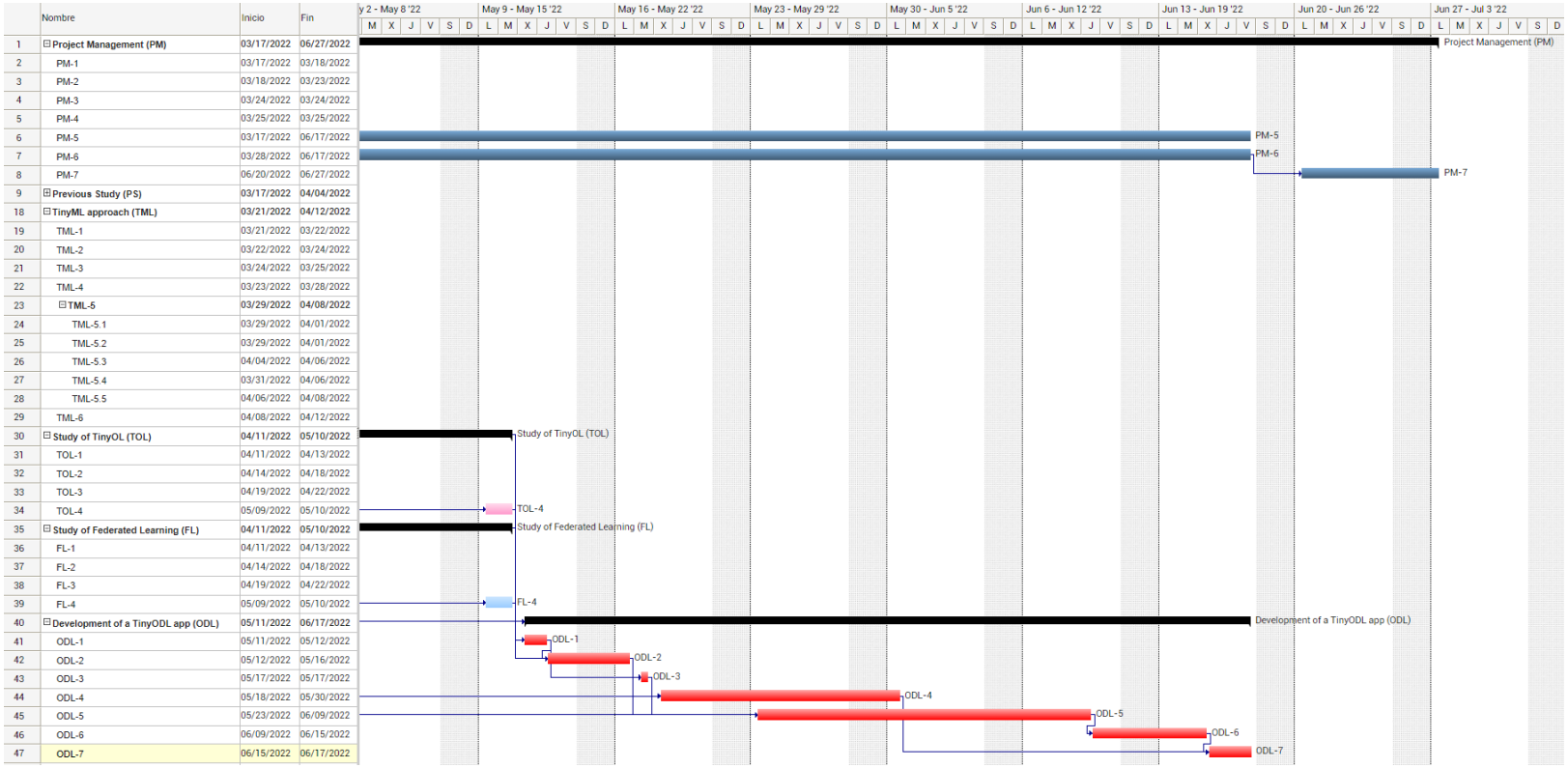

Figure 16: Gantt chart: Second period.

9.3 Figures

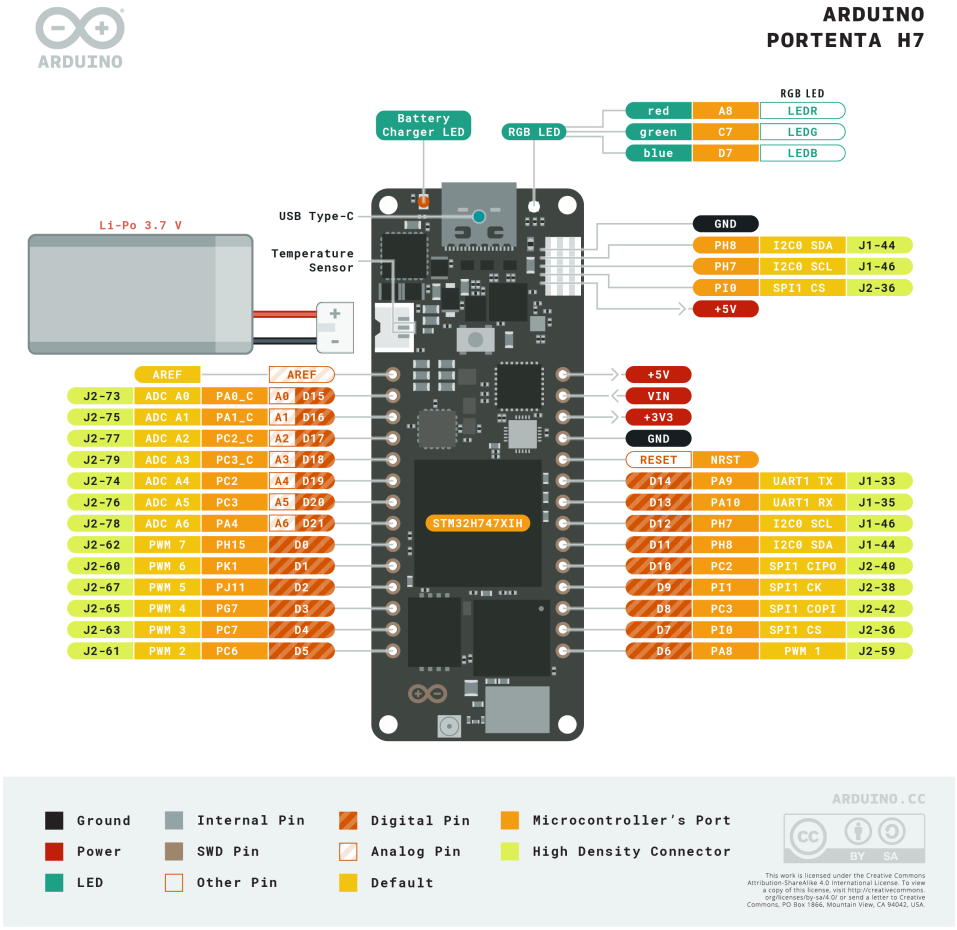

Figure 17: Arduino Portenta H7: Pinout figure

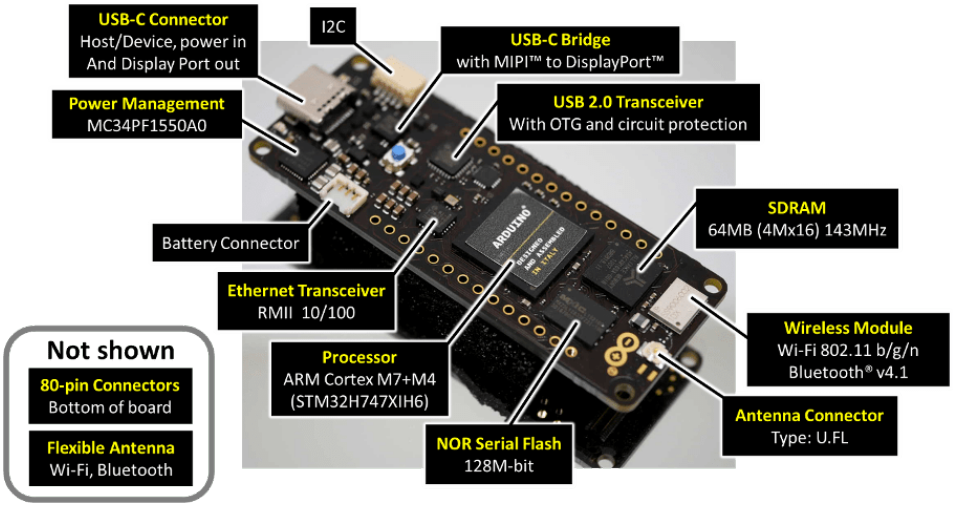

Figure 18: Arduino Portenta H7: Hardware

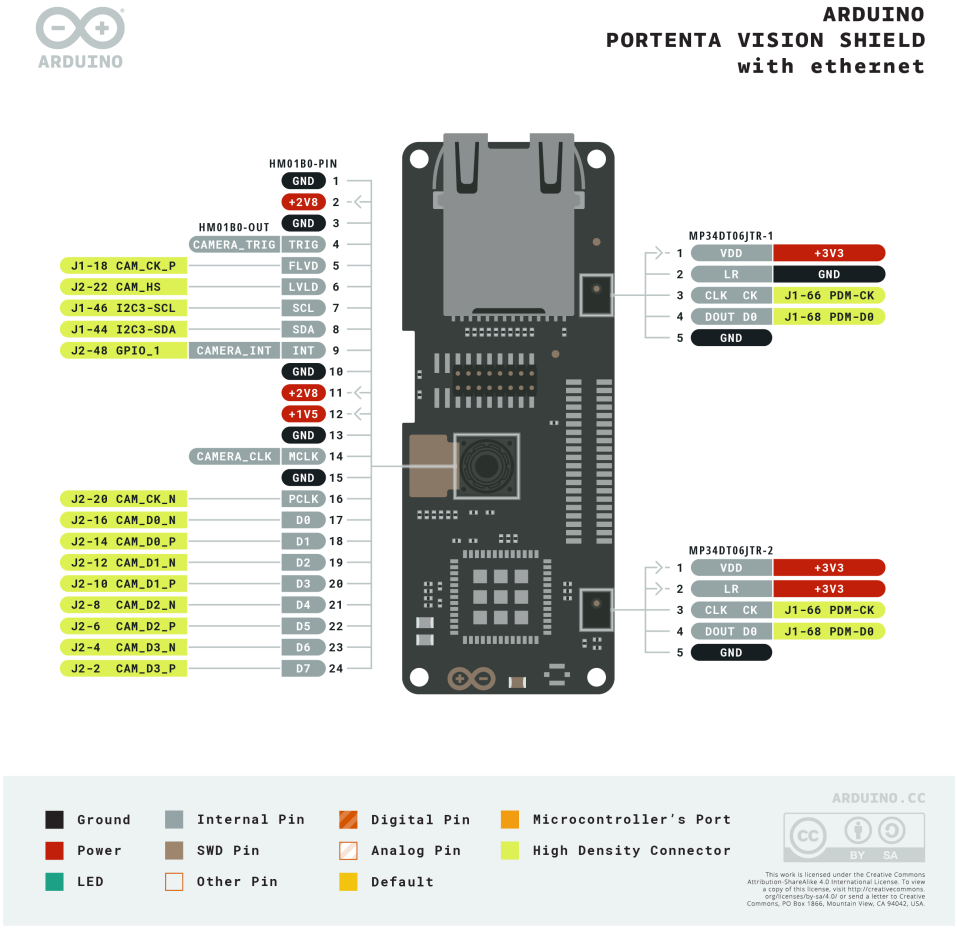

Figure 19: Arduino Portenta H7 Vision Shield: Pinout figure

output_handler.cc M

TinyODL-TFG > TinyOL > image-recognition > src > output_handler.cc > ...

```cpp
#include "tensorflow/lite/micro/examples/hello_world/output_handler.h"

void HandleOutput(tflite::ErrorReporter* error_reporter, float x_value,
                  float y_value) {
    // Log the current X and Y values
    TF_LITE_REPORT_ERROR(error_reporter, "x_value: %f, y_value: %f\n",
                         static_cast<double>(x_value),
                         static_cast<double>(y_value));

}
```

Figure 20: *output_handler.cc* file

main.cpp 9+

TinyODL-TFG > TinyOL > image-recognition > src > main.cpp > ...

```cpp
#include <Arduino.h>

#include "constants.h"
#include "output_handler.h"
#include "mobilenet_v1_1.0_224_quant.h"
#include "tensorflow/lite/experimental/micro/all_ops_resolver.h"
#include "tensorflow/lite/experimental/micro/micro_error_reporter.h"
#include "tensorflow/lite/experimental/micro/micro_interpreter.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "tensorflow/lite/version.h"

// Globals, used for compatibility with Arduino-style sketches.
namespace {
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
tflite::MicroInterpreter* interpreter = nullptr;
TfLiteTensor* input = nullptr;
TfLiteTensor* output = nullptr;
int inference_count = 0;

// Create an area of memory to use for input, output, and intermediate arrays.
// Finding the minimum value for your model may require some trial and error.
constexpr int kTensorArenaSize = 2 * 1024;
uint8_t tensor_arena[kTensorArenaSize];
}  // namespace
```

Figure 21: *main.cc* file: global variables

main.cpp 9+

TinyODL-TFG > TinyOL > image-recognition > src > main.cpp > loop

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

  // Set up logging. Google style is to avoid globals or statics because of
  // lifetime uncertainty, but since this has a trivial destructor it's okay.
  // NOLINTNEXTLINE(runtime-global-variables)
  static tflite::MicroErrorReporter micro_error_reporter;
  error_reporter = &micro_error_reporter;

  // Map the model into a usable data structure. This doesn't involve any
  // copying or parsing, it's a very lightweight operation.
  model = tflite::GetModel(g_sine_model_data);
  if (model->version() != TFLITE_SCHEMA_VERSION) {
    error_reporter->Report(
        "Model provided is schema version %d not equal "
        "to supported version %d.",
        model->version(), TFLITE_SCHEMA_VERSION);
    return;
  }

  // This pulls in all the operation implementations we need.
  // NOLINTNEXTLINE(runtime-global-variables)
  static tflite::ops::micro::AllOpsResolver resolver;

  // Build an interpreter to run the model with.
  static tflite::MicroInterpreter static_interpreter(
      model, resolver, tensor_arena, kTensorArenaSize, error_reporter);
  interpreter = &static_interpreter;

  // Allocate memory from the tensor_arena for the model's tensors.
  TfLiteStatus allocate_status = interpreter->AllocateTensors();
  if (allocate_status != kTfLiteOk) {
    error_reporter->Report(format: "AllocateTensors() failed");
    return;
  }

  // Obtain pointers to the model's input and output tensors.
  input = interpreter->input(0);
  output = interpreter->output(0);

  // Keep track of how many inferences we have performed.
  inference_count = 0;
}
```

Figure 22: *main.cc* file: *setup* function

main.cpp 9+

TinyODL-TFG > TinyOL > image-recognition > src > main.cpp > ...

```
72  // The name of this function is important for Arduino compatibility.
73  void loop() {
74    // Calculate an x value to feed into the model. We compare the current
75    // inference_count to the number of inferences per cycle to determine
76    // our position within the range of possible x values the model was
77    // trained on, and use this to calculate a value.
78    float position = static_cast<float>(inference_count) /
79                     static_cast<float>(kInferencesPerCycle);
80    float x_val = position * kXrange;
81
82    // Place our calculated x value in the model's input tensor
83    input->data.f[0] = x_val;
84
85    // Run inference, and report any error
86    TfLiteStatus invoke_status = interpreter->Invoke();
87    if (invoke_status != kTfLiteOk) {
88      error_reporter->Report(format: "Invoke failed on x_val: %f\n",
89                             static_cast<double>(x_val));
90      return;
91    }
92
93    // Read the predicted y value from the model's output tensor
94    float y_val = output->data.f[0];
95
96    // Output the results. A custom HandleOutput function can be implemented
97    // for each supported hardware target.
98    HandleOutput(error_reporter, x_value: x_val, y_value: y_val);
99
100   // Increment the inference_counter, and reset it if we have reached
101   // the total number per cycle
102   inference_count += 1;
103   if (inference_count >= kInferencesPerCycle) inference_count = 0;
104 }
```

Figure 23: *main.cc* file: *loop* function

neural_network.h 1, M

Portenta-Dual-Core > neural_network.h > ...

```
 9   static const int PatternCount = 3;
10   static const int InputNodes = 650;
11   static const int HiddenNodes = 25;
12   static const int OutputNodes = 3;
13   static const float InitialWeightMax = 0.05;
14   static const float InitialWeightMin = -0.05;
15
16   class NeuralNetwork {
17     public:
18       NeuralNetwork();
19       void initWeights();
20       void initialize(float LearningRate, float Momentum, int DropoutRate);
21       // ~NeuralNetwork();
22       // void initWeights();
23       float forward(volatile float Input[], const float Target[]);
24       float backward(volatile float Input[], const float Target[]); // Input will be changed!!
25
26       float* get_output();
27       float* get_HiddenWeights();
28       float* get_OutputWeights();
29       float get_error();
30
31     private:
32       float Hidden[HiddenNodes] = {};
33       float Output[OutputNodes] = {};
34       float HiddenWeights[(InputNodes + 1) * HiddenNodes] = {};
35       float OutputWeights[(HiddenNodes + 1) * OutputNodes] = {};
36       float HiddenDelta[HiddenNodes] = {};
37       float OutputDelta[OutputNodes] = {};
38       float ChangeHiddenWeights[(InputNodes + 1) * HiddenNodes] = {};
39       float ChangeOutputWeights[(HiddenNodes + 1) * OutputNodes] = {};
40
41       float Error;
42       float LearningRate = 0.6;
43       float Momentum = 0.9;
44       // From 0 to 100, the percentage of input features that will be set to 0
45       int DropoutRate = 0;
46   };
```

Figure 24: Nil's NN implementation: Header file

neural_network.cpp 7

Portenta-Dual-Core > neural_network.cpp > clangd > NeuralNetwork::forward

```cpp
#include <arduino.h>
#include "neural_network.h"
#include <math.h>

NeuralNetwork::NeuralNetwork() {
  this->initWeights();
}

/**
 * Intialize weights to random numebrs
 */
void NeuralNetwork::initWeights() {
    for(int i = 0 ; i < HiddenNodes ; i++ ) {
        for(int j = 0 ; j <= InputNodes ; j++ ) {
            ChangeHiddenWeights[j*HiddenNodes + i] = 0.0 ;
            HiddenWeights[j*HiddenNodes + i] = random(InitialWeightMin, InitialWeightMax);
        }
    }
    for(int i = 0 ; i < OutputNodes ; i ++ ) {
        for(int j = 0 ; j <= HiddenNodes ; j++ ) {
            ChangeOutputWeights[j*OutputNodes + i] = 0.0 ;
            OutputWeights[j*OutputNodes + i] = random(InitialWeightMin, InitialWeightMax);
        }
    }
}

void NeuralNetwork::initialize(float LearningRate, float Momentum, int DropoutRate) {
    this->LearningRate = LearningRate;
    this->Momentum = Momentum;
    this->DropoutRate = DropoutRate;
}
float* NeuralNetwork::get_output(){
    return Output;
}

float* NeuralNetwork::get_HiddenWeights(){
    return HiddenWeights;
}

float* NeuralNetwork::get_OutputWeights(){
    return OutputWeights;
}

float NeuralNetwork::get_error(){
    return Error;
}
```

Figure 25: Nil's NN implementation: Function implementations.

neural_network.cpp 7, M

Portenta-Dual-Core > neural_network.cpp > ...

```
35  float NeuralNetwork::forward(volatile float Input[], const float Target[]){
36      float error = 0;
37
38      /***********************************************************************
39      * Compute hidden layer activations
40      ***********************************************************************/
41      for (int i = 0; i < HiddenNodes; i++) {
42          float Accum = HiddenWeights[InputNodes*HiddenNodes + i];
43          for (int j = 0; j < InputNodes; j++) {
44              Accum += Input[j] * HiddenWeights[j*HiddenNodes + i];
45          }
46          Hidden[i] = 1.0 / (1.0 + exp(X: -Accum));
47      }
48
49      /***********************************************************************
50      * Compute output layer activations and calculate errors
51      ***********************************************************************/
52      for (int i = 0; i < OutputNodes; i++) {
53          float Accum = OutputWeights[HiddenNodes*OutputNodes + i];
54          for (int j = 0; j < HiddenNodes; j++) {
55              Accum += Hidden[j] * OutputWeights[j*OutputNodes + i];
56          }
57          Output[i] = 1.0 / (1.0 + exp(X: -Accum));
58          // OutputDelta[i] = (Target[i] - Output[i]) * Output[i] * (1.0 - Output[i]);
59          error += 0.33333 * (Target[i] - Output[i]) * (Target[i] - Output[i]);
60      }
61      return error;
62  }
```

Figure 26: Nil's NN implementation: *Forward* function.

neural_network.cpp 7, M

Portenta-Dual-Core > neural_network.cpp > clangd > NeuralNetwork::backward

```
64  // Input will be changed!!
65  float NeuralNetwork::backward(volatile float Input[], const float Target[]){
66      float error = 0;
67      for (int i = 0; i < InputNodes; i++) {
68          if (rand() % 100 < this->DropoutRate) {
69              Input[i] = 0;
70          }
71      }
72
73      // Forward
74      /*****************************************************************
75      * Compute hidden layer activations
76      *****************************************************************/
77      for (int i = 0; i < HiddenNodes; i++) {
78          float Accum = HiddenWeights[InputNodes*HiddenNodes + i];
79          for (int j = 0; j < InputNodes; j++) {
80              Accum += Input[j] * HiddenWeights[j*HiddenNodes + i];
81          }
82          Hidden[i] = 1.0 / (1.0 + exp(X: -Accum));
83      }
84
85      /*****************************************************************
86      * Compute output layer activations and calculate errors
87      *****************************************************************/
88      for (int i = 0; i < OutputNodes; i++) {
89          float Accum = OutputWeights[HiddenNodes*OutputNodes + i];
90          for (int j = 0; j < HiddenNodes; j++) {
91              Accum += Hidden[j] * OutputWeights[j*OutputNodes + i];
92          }
93          Output[i] = 1.0 / (1.0 + exp(X: -Accum)); // Sigmoid, from 0 to 1
94          OutputDelta[i] = (Target[i] - Output[i]) * Output[i] * (1.0 - Output[i]);
95          error += 1/OutputNodes * (Target[i] - Output[i]) * (Target[i] - Output[i]);
96      }
97      // End forward
```

Figure 27: Nil's NN implementation: *Backward* function, inference part.

Figure 28: Nil's NN implementation: *Backward* function, update weights.

layer.hpp 7, M

FL_with_CNN > layer.hpp > ...

```cpp
class Layer {

protected:
    int InputSize, OutputSize;
    float* Output;

    float InitialWeightMax = 0.5;
    float LearningRate;
    float Momentum;

    float sigmoid(float value) { return (1.0 / (1.0 + exp(-value))); }
    float ReLU(float value) { return max(0,value); }
    float ReLU_prime(float value) { return value > 0 ? 1 : 0; }

    virtual float activation_function(float value) {
        return ReLU(value);
    }
    virtual float activation_function_prime(float value) {
        return ReLU_prime(value);
    }

public:
    int randomInt(int range=20) {
        int rand = random(range);
        return rand - range/2;
    }
    float randomFloat(int range=100) {
        float rand = float(random(range))/float(range);
        return 2.0 * (rand - 0.5);
    }

    virtual float* forward(const float Input[], bool verbose=false);
    virtual float* backward(float gradient[], const float Input[], bool verbose=false);

    virtual void printWeights();
    int getInputSize() { return InputSize; }
    int getOutputSize() { return OutputSize; }
};
```

Figure 29: Layer class: attributes and activation functions.

layer.hpp 7, M

FL_with_CNN > layer.hpp > ...

```
67  class Dense : public Layer {
68
69  public:
70      Dense(int InputSize, int OutputSize,
71            float LearningRate=0.3, float Momentum=0.6);
72      void initWeights();
73      void printWeights();
74
75      float* forward(const float Input[],
76                     bool verbose=false);
77      float* backward(float gradient[], const float Input[],
78                      bool verbose=false);
79
80  private:
81      float* Weights;
82      float* ChangeWeights;
83      float* InputGradient;
84
85  };
```

Figure 30: Dense layer class: Header file

layer.cpp 9+, M

FL_with_CNN > layer.cpp > ...

```
14  Dense::Dense(int InputSize, int OutputSize, float LearningRate, float Momentum)
15  {
16      this->InputSize = InputSize;
17      this->OutputSize = OutputSize;
18      this->LearningRate = LearningRate;
19      this->Momentum = Momentum;
20
21      Weights       = new float[(InputSize+1) * OutputSize];
22      ChangeWeights = new float[(InputSize+1) * OutputSize];
23      Output        = new float[OutputSize];
24      InputGradient = new float[InputSize];
25
26      initWeights();
27  }
28  void Dense::initWeights()
29  {
30      for(int i=0; i < (InputSize+1)*OutputSize; ++i) {
31          ChangeWeights[i] = 0.0;
32          Weights[i] = randomFloat() * InitialWeightMax;
33      }
34  }
```

Figure 31: Dense layer class: Function implementations.

layer.cpp 9+, M

FL_with_CNN > layer.cpp > ...

```cpp
float* Dense::forward(const float Input[], bool verbose)
{
    for(int o=0; o<OutputSize; ++o) {
        // Bias value
        float Accum = Weights[InputSize*OutputSize + o];
        // Weights
        for(int i=0; i<InputSize; ++i) {
            Accum += Input[i] * Weights[i*OutputSize + o];
        }
        Output[o] = activation_function(value: Accum);
    }
    return Output;
}

float* Dense::backward(float gradient[] , const float Input[], bool verbose)
{
    for(int i=0; i<InputSize; ++i) {
        InputGradient[i] = 0.0;
        for(int o=0; o<OutputSize; ++o) {
            // Backward the activation function
            gradient[o] = activation_function_prime(value: gradient[o]);
            // Calc input gradient
            InputGradient[i] += Weights[i*OutputSize + o] * gradient[o];
            // Update weights
            ChangeWeights[i*OutputSize + o] =
                LearningRate * Input[i] * gradient[o] + Momentum * ChangeWeights[i*OutputSize + o];
            Weights[i*OutputSize + o] = ChangeWeights[i*OutputSize + o];
        }
    }
    // Update bias
    for(int o=0; o<OutputSize; ++o) {
        ChangeWeights[InputSize*OutputSize + o] =
            LearningRate * gradient[o] + Momentum * ChangeWeights[InputSize*OutputSize + o];
        Weights[InputSize*OutputSize + o] = ChangeWeights[InputSize*OutputSize + o];
    }
    return InputGradient;
}
```

Figure 32: Dense layer class: Forward and backward functions.

layer.hpp 7, M

FL_with_CNN > layer.hpp > clangd > Layer > backward

```cpp
class Convolutional : public Layer {

  public:
    Convolutional(int InputDim[3], int KernelDim[2], int FilterSize,
                  float LearningRate=0.3, float Momentum=0.9);
    void initKernels();
    float* forward(const float Input[], bool verbose=false);
    float* backward(float gradient[], const float Input[], bool verbose=false);

  private:
    int InputHeight, InputWidth, InputDepth;
    int KernelHeight, KernelWidth, KernelSize, FilterSize;
    int OutputHeight, OutputWidth;

    float* Bias;
    float* Kernels;
    float* ChangeBias;
    float* ChangeKernels;

    void convolve(const float gradient[], const float Kernel[],
                  float* InputGradient);
    void correlate( float* Output, const float Input[], const float Kernel[],
                  const int InputHeight, const int KernelHeight,
                  const int InputWidth, const int KernelWidth);
};
```

Figure 33: Convolutional layer class: Header file

layer.cpp 9+, M

FL_with_CNN > layer.cpp > ...

```
86  Convolutional::Convolutional(int InputDim[3], int KernelDim[2], int FilterSize,
87                               float LearningRate, float Momentum)
88  {
89      this->InputHeight = InputDim[0];
90      this->InputWidth  = InputDim[1];
91      this->InputSize   = InputHeight*InputWidth;
92      this->InputDepth  = InputDim[2];
93
94      this->KernelHeight = KernelDim[0];
95      this->KernelWidth  = KernelDim[1];
96      this->KernelSize   = KernelHeight*KernelWidth;
97      this->FilterSize   = FilterSize;
98
99      this->OutputHeight = InputHeight - KernelHeight + 1;
100     this->OutputWidth  = InputWidth - KernelWidth + 1;
101     this->OutputSize = OutputHeight*OutputWidth;
102
103     this->LearningRate = LearningRate;
104     this->Momentum = Momentum;
105
106     Bias          = new float[FilterSize * OutputSize];
107     Kernels       = new float[FilterSize * InputDepth * KernelSize];
108     ChangeBias    = new float[FilterSize * OutputSize];
109     ChangeKernels = new float[FilterSize * InputDepth * KernelSize];
110     Output        = new float[FilterSize * OutputSize];
111
112     initKernels();
113 }
114 void Convolutional::initKernels()
115 {
116     for(int i=0; i < FilterSize * OutputSize; ++i) {
117         float rand = float(random(100))/100.0;
118         Bias[i] = 2.0 * (rand - 0.5) * InitialWeightMax;
119     }
120     for(int i=0; i < FilterSize * InputDepth * KernelSize; ++i) {
121         ChangeKernels[i] = 0.0;
122         float rand = float(random(100))/100.0;
123         Kernels[i] = 2.0 * (rand - 0.5) * InitialWeightMax;
124     }
125 }
```

Figure 34: Convolutional layer class: Function implementations.

layer.cpp 9+, M

FL_with_CNN > layer.cpp > ...

```cpp
float* Convolutional::forward(const float Input[], bool verbose)
{
    // For each output matrix
    for(int f=0; f<FilterSize; ++f) {
        // Fore each output value
        for(int o=0; o<OutputSize; ++o) {
            // First add the bias value
            float Accum = Bias[f*OutputSize + o];
            int rowO = o/OutputWidth, colO = o%OutputWidth;
            // For each input matrix
            for(int d=0; d<InputDepth; ++d) {
                // For each kernel value
                for(int k=0; k<KernelSize; ++k) {
                    int rowK = k/KernelWidth, colK = k%KernelWidth;
                    int inputPos = d*InputSize + (rowO+rowK)*InputWidth + (colO+colK);
                    // Add input value times kernel value
                    Accum += Input[inputPos] * Kernels[f*InputDepth*KernelSize + d*KernelSize + k];
                }
            }
            Output[f*OutputSize + o] = Accum;
        }
    }
    return Output;
}
```

Figure 35: Convolutional layer class: Forward function.

layer.cpp 9+, M

FL_with_CNN > layer.cpp > ...

```cpp
float* Convolutional::backward(float gradient[], const float Input[], bool verbose)
{
    float* InputGradient = new float[InputDepth * InputSize];
    // For each Output matrix
    for(int f=0; f<FilterSize; ++f) {
        // For each Input matrix
        for(int d=0; d<InputDepth; ++d) {
            // Calculate Input gradient
            convolve(gradient: &gradient[f*OutputSize],
                     Kernel: &Kernels[f*InputDepth*KernelSize + d*KernelSize],
                     InputGradient: &InputGradient[d*InputSize]);
            // Calculate Kernel gradient
            for(int k=0; k<KernelSize; ++k) {
                int rowK = k/KernelWidth, colK = k%KernelWidth;
                float Accum = 0.0;
                for(int o=0; o<OutputSize; ++o) {
                    int rowO = o/OutputWidth, colO = o%OutputWidth;
                    int whichInput = d*InputSize + (rowO+rowK)*InputWidth + (colO+colK);
                    Accum += Input[whichInput] * gradient[f*OutputSize + o];
                }
                ChangeKernels[f*InputDepth*KernelSize + d*KernelSize + k] = LearningRate * Accum +
                    Momentum * ChangeKernels[f*InputDepth*KernelSize + d*KernelSize + k];
                Kernels[f*InputDepth*KernelSize + d*KernelSize + k] =
                    ChangeKernels[f*InputDepth*KernelSize + d*KernelSize + k];
            }
        }
        // Calculate Bias gradient
        for(int o=0; o<OutputSize; ++o) {
            ChangeBias[f*OutputSize + o] = LearningRate * gradient[f*OutputSize + o] +
                Momentum * ChangeBias[f*OutputSize + o];
            Bias[f*OutputSize + o] = ChangeBias[f*OutputSize + o];
        }
    }
    return InputGradient;
}
```

Figure 36: Convolutional layer class: Backward function.

ç

layer.cpp 9+, M

FL_with_CNN > layer.cpp > ...

```cpp
void Convolutional::convolve(const float gradient[], const float Kernel[], float* InputGradient)
{
    // Calc new Output Gradient matrix
    int DimOffset = InputHeight - (KernelHeight-1);
    float newOG[(OutputHeight + 2*DimOffset) * (OutputWidth + 2*DimOffset)];
    // For every outer value before the inner matrix => 0
    for(int o=0; o < (DimOffset)*(OutputWidth + 2*DimOffset); ++o)
        newOG[o] = 0;
    // The inner matrix is the OutputGradient matrix, the values before and after inner values => 0
    for(int o=0; o < (OutputHeight)*(OutputWidth + 2*DimOffset); ++o) {
        if(o%(OutputWidth + 2*DimOffset) < DimOffset ||
           o%(OutputWidth + 2*DimOffset) > (OutputWidth + 2*DimOffset - 1)-DimOffset)
        {
            newOG[o] = 0;
        }
        else newOG[o] = gradient[o-DimOffset];
    }
    // For every outer value after the inner matrix => 0
    for(int o=0; o < (DimOffset)*(OutputWidth + 2*DimOffset); ++o)
        newOG[(OutputWidth + 2*DimOffset - 1) - o] = 0;

    // Correlate the Output Gradient matrix with the 180º rotated Kernel matrix
    for(int i=0; i<InputSize; ++i) {
        InputGradient[i] = 0;
        int rowI = i/InputWidth, colI = i%InputWidth;
        for(int k=0; k<KernelSize; ++k) {
            int rowK = k/InputWidth, colK = k%InputWidth;
            int outputPos = (rowI + rowK)*(OutputWidth + 2*DimOffset) + (colI + colK);
            InputGradient[i] += newOG[outputPos] * Kernel[(KernelSize-1)-k];
        }
    }
}
```

Figure 37: Convolutional layer class: Convolve function.

loss.hpp 1, M

FL_with_CNN > loss.hpp > ...

```cpp
#ifndef LOSS
#define LOSS

class Loss {
public:
    virtual float loss(const float output[], const float target[], int numItems) = 0;
    virtual float* loss_prime(const float output[], const float target[], int numItems) = 0;

};

class MSE : public Loss {
public:
    MSE() {};
    float loss(const float output[], const float target[], int numItems) {
        float loss = 0.0;
        for(int i=0; i<numItems; ++i)
            loss += (target[i]-output[i])*(target[i]-output[i]);
        loss /= numItems;
        return loss;
    }
    float* loss_prime(const float output[], const float target[], int numItems) {
        float* gradient = new float;
        for(int i=0; i<numItems; ++i)
            gradient[i] = 2 * (target[i]-output[i]) / numItems;
        return gradient;
    }
};
#endif
```

Figure 38: Loss class and MSE class.

C ModelNN.h 3, M

FL_with_CNN > C ModelNN.h > ...

```
4  #include "loss.hpp"
5  #include "layer.hpp"
6
7  static MSE loss_function = MSE();
8
9  class ModelNN {
10     public:
11         int nLayers=0;
12
13         ModelNN(int maxNumLayers);
14         void add(Layer* layer);
15
16         void train( int nItems, float X[], float Y[],
17                     int epochs=1, int batch_size=1, bool verbose=false );
18         float* test(int nItems, float X[], float Y[], bool verbose=false);
19         float* predict(float Input[]);
20
21     private:
22         float Error;
23
24         int InputSize;
25         int OutputSize;
26
27         Loss* loss = &(loss_function);
28         Layer** layers;
29
30         float* forward(float Input[], float* batchInputs[]);
31         void backpropagate(float* gradient, float* Input[]);
32 };
```

Figure 39: NN class: Header file

ModelNN.cpp 9+, M

FL_with_CNN > ModelNN.cpp > ...

```cpp
#include <Arduino.h>
#include "ModelNN.h"

ModelNN::ModelNN(int maxNumLayers) {
    layers = new Layer*[maxNumLayers];
}

void ModelNN::add(Layer* layer) {
    layers[nLayers] = layer;
    if(nLayers == 0) InputSize = layer -> getInputSize();
    OutputSize = layer -> getOutputSize();
    nLayers++;
}

float* ModelNN::forward(float Input[], float* batchInputs[]) {
    float* output = Input;
    bool store_inputs = batchInputs != nullptr;
    for(int l=0; l < nLayers; ++l) {
        if(store_inputs) batchInputs[l] = output;
        output = layers[l] -> forward(Input: output);
    }
}

void ModelNN::backpropagate(float* gradient, float* Inputs[]) {
    for(int l=nLayers-1; l >= 0; --l) {
        gradient = layers[l] -> backward(gradient,
                                         Input: Inputs[nLayers - (l+1)],
                                         verbose: false);
    }
}
```

Figure 40: NN class: Function implementations

Figure 41: NN class: Train function

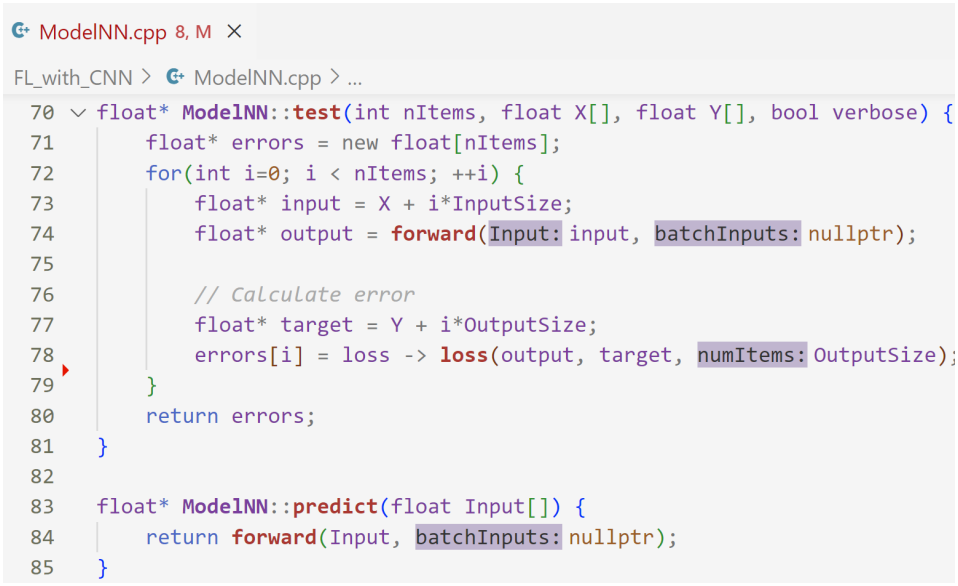

Figure 42: NN class: Test and predict functions.

Acronyms

AI Artificial Intelligence. 31

ANN Artificial Neural Network. 6, 7, 57

CNN Convolutional Neural Network. 47–50, 52

CPU Central Processing Unit. 5

DL deep learning. 31, 32

FL Federated Learning. 2, 39, 43, 44, 47, 52, 57

iid independent and identically distributed. 43

IoT Internet of Things. 2, 5, 10, 28, 31

MCU Microcontroller Unit. 5, 8, 11, 31, 40, 41, 57

MCU microcontroller. 33, 39, 41, 42

ML Machine Learning. 2, 5, 6, 8, 31–33, 43, 44

NN Neural Network. 6, 13, 32, 39–42, 53

TFLite TensorFlow Lite. 50, 51

TFLite Micro TensorFlow Lite for Microncontrollers. 50, 51

TinyML Tiny Machine Learning. 2, 5, 8, 31–34, 38–42

TinyODL TinyML On-Device Learning. 2, 9, 10, 31, 39

TinyOL TinyML with Online-Learning. 2, 5, 10, 39–42, 44

References

[1] Machine learning, Oct 2022. Available at https://en.wikipedia.org/wiki/Machine_learning.

[2] Artificial Neural Network, Oct 2022. Available at https://en.wikipedia.org/wiki/Artificial_neural_network.

[3] Ram Sagar. What are the challenges of establishing a tinyml ecosystem, Mar 2020. Available at https://analyticsindiamag.com/tinyml-ecosystem-challenges-machine-learning-iot/.

[4] Colby R. Banbury, Vijay Janapa Reddi, Max Lam, William Fu, Amin Fazel, Jeremy Holleman, Xinyuan Huang, Robert Hurtado, David Kanter, Anton Lokhmotov, David A. Patterson, Danilo Pau, Jae-sun Seo, Jeff Sieracki, Urmish Thakker, Marian Verhelst, and Poonam Yadav. Benchmarking TinyML Systems: Challenges and Direction. *CoRR*, abs/2003.04821, 2020. Available at https://arxiv.org/abs/2003.04821.

[5] Partha Pratim Ray. A review on tinyml: State-of-the-art and prospects, Nov 2021. Available at https://www.sciencedirect.com/science/article/pii/S1319157821003335.

[6] Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute Trends Across Three Eras of Machine Learning, 2022. Available at https://doi.org/10.48550/arxiv.2202.05924.

[7] Wikipedia contributors. GPT-3 – Wikipedia, The Free Encyclopedia, 2022. Available at https://en.wikipedia.org/w/index.php?title=GPT-3&oldid=1102671454.

[8] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical Text-Conditional Image Generation with CLIP Latents, 2022. Available at https://arxiv.org/abs/2204.06125.

[9] Manifesto for Agile software development. Available at https://agilemanifesto.org/.

[10] Huawei Matebook 14 2020 AMD, Mar 2018. Available at https://consumer.huawei.com/es/laptops/matebook-14-amd-2020/.

[11] The Arduino Team. Portenta H7: Arduino documentation. Available at https://docs.arduino.cc/hardware/portenta-h7.

[12] The Arduino Team. Portenta Vision Shield: Arduino documentation. Available at https://docs.arduino.cc/hardware/portenta-vision-shield.

[13] FAQs Google Colab. Available at https://research.google.com/colaboratory/faq.html#:~:text=Colaboratory%2C%20or%20%E2%80%9CColab%E2%80%9D%20for,learning%2C%20data%20analysis%20and%20education.

[14] Project jupyter. Available at https://jupyter.org/.

[15] Tensorflow. Available at https://www.tensorflow.org/.

[16] Tensorflow Lite: ML for Mobile and edge devices. Available at https://www.tensorflow.org/lite.

[17] Edge impulse. Available at https://www.edgeimpulse.com/.

[18] Platformio Ide - Visual Studio Marketplace. Available at https://marketplace.visualstudio.com/items?itemName=platformio.platformio-ide Platformio Ide - Visual Studio Marketplace.

[19] Microsoft. Visual studio code - code editing. redefined, Nov 2021. Available at https://code.visualstudio.com/.

[20] The Arduino Team. Software. Available at https://www.arduino.cc/en/software.

[21] Git webpage. Available at https://git-scm.com/.

[22] GitHub: Where the world builds software. Available at https://github.com/.

[23] Notion. One workspace, every team. Available at https://www.notion.so/product?fredir=1.

[24] About Google Calendar. Available at https://www.google.com/calendar/about/.

[25] Number 1 cloud-based project management software. Available at https://www.gantter.com/.

[26] Overleaf, online latex editor. Available at https://www.overleaf.com/.

[27] Arduino Pro: Portenta H7. Available at https://www.arduino.cc/pro/hardware/product/portenta-h7.

[28] The Arduino Team. Portenta H7: Arduino documentation. Available at https://docs.arduino.cc/hardware/portenta-h7.

[29] The Arduino Team. What is Arduino? Available at https://www.arduino.cc/en/Guide/Introduction.

[30] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. TinyOL: TinyML with Online-Learning on Microcontrollers. *CoRR*, abs/2103.08295, 2021. Available at https://arxiv.org/abs/2103.08295.

[31] Federated learning, Sep 2022. Available at https://en.wikipedia.org/wiki/Federated_learning.

[32] MJRoBot (Marcelo Rovai). ESP32-cam: Tinyml image classification - fruits vs veggies, Jan 2022. Available at https://www.hackster.io/mjrobot/esp32-cam-tinyml-image-classification-fruits-vs-veggies-4ab970.

[33] Tensorflow Lite examples: Machine Learning Mobile Apps. Available at https://www.tensorflow.org/lite/examples.

[34] Xxd - unix, linux command. Available at https://www.tutorialspoint.com/unix_commands/xxd.htm.

[35] Nil Llisterri. Nilllisterri/Portenta-dual-core. Available at https://github.com/NilLlisterri/Portenta-Dual-Core.

[36] Nil Llisterri Giménez. *Federated learning on embedded devices*. PhD thesis, UPC, Facultat d'Informàtica de Barcelona, Departament d'Arquitectura de Computadors, Jan 2022. Available at http://hdl.handle.net/2117/363727.

[37] OpenCV documentation. Available at https://docs.opencv.org/2.4/modules/ml/doc/ml.html.

[38] Microsoft. Microsoft/CNTK at hackernoon.com. Available at https://github.com/microsoft/CNTK?ref=hackernoon.com.

[39] Virtual function in C++, Jul 2022. Available at https://www.geeksforgeeks.org/virtual-function-cpp/.

[40] Difference between "kernel" and "Filter" in CNN, Sep 1962. Available at https://stats.stackexchange.com/questions/154798/difference-between-kernel-and-filter-in-cnn#:~:text=To%20be%20straightforward%3A%20A%20filter,x%20K%2D2%20x%20P%20.

[41] Renu Khandelwal. Convolutional Neural Network: Feature map and filter visualization, May 2020. Available at https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c.

[42] Mayank Mishra. Convolutional Neural Networks, explained, Sep 2020. Available at https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939.

[43] Introduction to convolutional neural networks. Available at https://aigents.co/data-science-blog/publication/introduction-to-convolutional-neural-networks-cnns.

[44] Convolutional Neural Network from scratch, May 2021. Available at https://www.youtube.com/watch?v=Lakz2MoHy6o&ab_channel=TheIndependentCode.

[45] Raimi Karim. 10 gradient descent optimisation algorithms, Jan 2022. Available at https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9.

[46] Mean squared error, Aug 2022. Available at https://en.wikipedia.org/wiki/Mean_squared_error.

[47] TinyML On Device Learning event. Available at https://www.tinyml.org/event/on-device-learning/.

[48] Home: Tinyml foundation. Available at https://www.tinyml.org/.

[49] Fabrizio De Vita, Giorgio Nocera, Dario Bruneo, Valeria Tomaselli, and Mirko Falchetto. On-Device Training of Deep Learning Models on Edge Microcontrollers. In *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 62–69, 2022. Available at https://ieeexplore.ieee.org/document/9903209.