

---

**UNIVERSITAT POLITÈCNICA DE CATALUNYA**  
**BARCELONATECH**

**Facultat d'Informàtica de Barcelona**



**UNIVERSIDAD POLITÉCNICA DE CATALUÑA**  
**FACULTAD DE INFORMÁTICA DE BARCELONA**

Grado en Ingeniería Informática  
Especialidad de Computación

---

# **Desarrollo de una aplicación TinyML**

---

Ivan Ramírez Mijarra

Director: Felix Freitag (Departamento de Arquitectura de Computadores)  
Memoria Final TFG

17 de abril de 2022

# Resumen

En los últimos años el mundo de TinyML se ha expandido a un ritmo constante y rápido. Gracias a la posibilidad de implementar aplicaciones de aprendizaje automático en dispositivos de bajo consumo y menor tamaño, reduciendo así en gran medida los costes energéticos, se ha vuelto el punto de atención de muchos desarrolladores e investigadores de aplicaciones de aprendizaje automático. Este proyecto tiene la intención de investigar las herramientas y técnicas usadas para lograr aplicaciones de TinyML, concretamente en aplicaciones capaces de utilizar algoritmos de aprendizaje federado. Estos algoritmos se centran en el uso de varios dispositivos para conseguir juntar la información conseguida por todos ellos, con el objetivo de crear un modelo global compartido entre todos ellos, consiguiendo de esta manera compartir el aprendizaje entre todos.

Para ello, el proyecto siguió el hilo empezado por Marc Monfort en su trabajo de fin de grado, donde se conseguía crear una aplicación de aprendizaje federado capaz de distinguir entre diferentes clases entrenadas con señales de audio.

A partir de este punto el proyecto investiga primero la posibilidad de crear una aplicación de aprendizaje federado capaz de distinguir entre diferentes clases, pero esta vez entrenadas con imágenes en vez de audio. Y segundo la posibilidad de implementar la tecnología de Bluetooth en la aplicación de aprendizaje federado para transmitir la información entre los clientes y el servidor.

Los resultados del proyecto demuestran la viabilidad de implementar un aplicación de aprendizaje federado entrenada con un conjunto de imágenes y un camino a seguir para lograr la correcta implementación de la tecnología Bluetooth en los microcontroladores para conseguir intercambiar información por este medio en vez de los puertos seriales conectados por cables.

**Palabras clave:** TinyML, aprendizaje automático, aprendizaje federado

# Índice

---

<b>Índice</b>	<b>3</b>
<b>1 Contextualización y extensión del proyecto</b>	<b>6</b>
1.1 Contextualización	6
1.2 Definición de conceptos	6
1.2.1 Microcontrolador	6
1.2.2 TinyML	6
1.2.3 Aprendizaje automático	7
1.2.4 Redes neuronales artificiales	7
1.2.5 Aprendizaje federado	8
1.3 Identificación del problema	8
1.3.1 Objetivo de la aplicación	8
1.3.2 Uso de nuevas tecnologías	8
1.3.3 Limitaciones de los microcontroladores	9
1.4 Actores implicados	9
1.5 Justificación	9
1.6 Alcance	10
1.6.1 Objetivos genéricos	10
1.6.2 Subobjetivos	10
1.6.3 Obstáculos y riesgos	11
1.7 Metodología y rigor	12
1.7.1 Herramientas de seguimiento	12
<b>2. Planificación temporal</b>	<b>13</b>
2.1 Extensión temporal del proyecto	13
2.2 Definición de tareas	13
T1 Gestión del proyecto	13
T2 Formación básica inicial	14
T3 Extensión Bluetooth	15
T4 Modificación objetivo de la aplicación	16
T5 Microcontrolador como servidor	17
T6 Documentación del proyecto	18
2.3 Recursos	18
2.3.1 Recursos humanos	18
2.4 Gestión del riesgo: Planes alternativos	22
2.4.1 Tiempo	22
2.4.2 Desarrollo de las aplicaciones	22
2.5 Cambios en la planificación inicial	23

<b>3. Presupuesto</b>	<b>25</b>
3.1 Costes de personal	25
3.2 Costes genéricos	27
3.3 Costes de contingencia	29
3.4 Costes de imprevistos	29
3.5 Coste final del proyecto	30
<b>4. Control de gestión</b>	<b>30</b>
<b>5. Contexto</b>	<b>31</b>
5.1 Herramientas de desarrollo	31
5.1.1 PlatformIO	31
5.1.2 Edge Impulse	33
5.2 Hardware	35
5.2.1 Arduino Nano 33 BLE Sense	35
5.2.2 Cámara OV7675	36
5.2.3 Arduino Tiny Machine Learning Shield	36
5.3 TinyML	38
5.4 Aprendizaje Federado (Federated Learning)	39
<b>6. Desarrollo de la aplicación</b>	<b>42</b>
6.1 Objetivos	42
6.2 Preparación hardware	42
6.3 Análisis código inicial	43
6.3.1 Microcontrolador	43
6.3.2 Servidor	46
6.4 Red neuronal	51
6.5 Modificaciones audio a imagen	52
<b>7. Pruebas</b>	<b>58</b>
7.1 Primera prueba	58
7.2 Segunda prueba	59
7.3 Tercera prueba	61
7.4 Cuarta prueba	62
7.5 Quinta prueba	63
7.6 Primera prueba Aprendizaje Federado	64
7.7 Segunda prueba Aprendizaje Federado	65
7.8 Tercera prueba Aprendizaje Federado	67
<b>8. Análisis Sostenibilidad</b>	<b>69</b>
8.1 Proyecto Puesto en Producción (PPP)	69
8.1.1 Ambiental	69
8.1.2 Económico	70

8.1.3 Social	70
8.2 Vida útil	71
8.2.1 Ambiental	71
8.2.2 Económico	71
8.2.3 Social	71
8.3 Riesgos	71
8.3.1 Ambiental	71
8.3.2 Económico	72
8.3.3 Social	72
8.4 Resultados análisis de sostenibilidad	72
<b>9. Trabajos futuros</b>	<b>73</b>
9.1 Bluetooth	73
9.2 Red convolucional	74
9.3 Arduino servidor de Federated Learning	77
<b>10. Conclusiones</b>	<b>79</b>
<b>11. Referencias</b>	<b>82</b>
<b>Anexo Código Bluetooth</b>	<b>84</b>

# 1 Contextualización y extensión del proyecto

## 1.1 Contextualización

Este proyecto es un trabajo de fin de grado del Grado de Ingeniería Informática realizado en la Facultad de Informática de Barcelona centrado en el campo del aprendizaje automático, rama del campo de la inteligencia artificial.

Este proyecto es una propuesta del director de proyecto, Félix Freitag, del Departamento de Arquitectura de Computadores de la Facultad de Informática de Barcelona.

El objetivo del proyecto es estudiar el nuevo campo de TinyML y las herramientas que se usan para poder llevar a cabo sus funciones, con el fin de crear una aplicación capaz de resolver problemas mediante el uso de estos dispositivos y técnicas.

La aplicación realizada estará basada en el trabajo de fin de grado de Marc Monfort[1], que realizó una aplicación de aprendizaje federado, la cual será nuestro punto de partida y se investigarán posibles mejoras usando otras tecnologías para algunos procesos.

## 1.2 Definición de conceptos

En esta sección se definirán conceptos importantes necesarios para poder comprender adecuadamente el tema que se está estudiando, con el fin de poder facilitar al lector la comprensión de la temática del proyecto y su evolución.

### 1.2.1 Microcontrolador

Un microcontrolador es un circuito integrado programable, el cual dispone de una memoria programable y ejecuta las órdenes desde esta. Los microcontroladores están compuestos por tres unidades principales, una o varias unidades de procesamiento (CPUs), periféricos de entrada y salida y la memoria. Debido a que no utiliza memoria RAM convencional sino memoria programable y no volátil provoca que los tiempos de acceso a datos e instrucciones sean mucho mayores y por tanto que tenga que funcionar a velocidades de reloj muy bajas, hecho que les permite consumir mucho menos a nivel energético.

### 1.2.2 TinyML

TinyML es una nueva rama del aprendizaje automático creada recientemente que tiene como objetivo implementar técnicas de aprendizaje automático en microcontroladores

capaces de procesar estos datos con un consumo energético bastante reducido. Hecho, que permite a estos dispositivos trabajar por períodos de tiempo largos e ininterrumpidos sin necesidad de estar conectados a una fuente de alimentación considerable y por tanto realizar tareas nuevas, tales como, el seguimiento de animales salvajes y poder reaccionar ante peligros que puedan aparecer ante ellos (cazadores), seguimiento de pacientes con enfermedades graves que necesitan de tratamientos espontáneos y muchos más.

### 1.2.3 Aprendizaje automático

Dentro de la inteligencia artificial existe una rama llamada aprendizaje automático, centrada en desarrollar técnicas para que las computadoras aprendan a resolver problemas a base de datos recibidos del medio, sin la necesidad de tener una solución previamente programada para cada caso.

Para calcular nuevas soluciones estos dispositivos utilizan los datos que reciben para detectar patrones en ellos y ajustar las acciones del programa en consecuencia.

### 1.2.4 Redes neuronales artificiales

Las redes neuronales artificiales están diseñadas con el fin de emular los cerebros de los animales. Una red neuronal artificial contiene varias capas con nodos dentro de estas, cada nodo simboliza una neurona de un cerebro real y está conectado con otros nodos mediante aristas, como si de un grafo se tratase como se puede apreciar en la Figura 1. Cada nodo está diseñado para realizar una función matemática con los datos que recibe en la entrada y propagar el resultado a los nodos conectados a él.

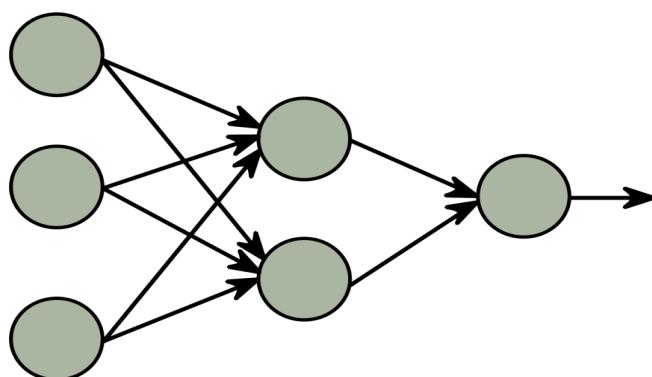


Fig.1 Ejemplo simple de una red neuronal artificial con 6 nodos

### 1.2.5 Aprendizaje federado

El aprendizaje federado es una técnica de aprendizaje automático que consiste en entrenar un algoritmo a través de una arquitectura descentralizada. Para lograr esto se dispone de varios dispositivos con sus propios datos locales y privados, cada dispositivo recibe su propio entrenamiento y luego se envía el modelo entrenado al servidor que conecta todos los dispositivos. Este servidor tiene la tarea de fusionar todos los modelos que recibe de cada dispositivo conectado a su red y luego el nuevo modelo resultado de esta fusión distribuirlo a todos los dispositivos para que estos lo utilicen.

## 1.3 Identificación del problema

Como ya se ha mencionado previamente nuestro punto de partida es una aplicación de aprendizaje federado, más concretamente una con la tarea de “keyword spotting”, la cual se enfoca en poder reconocer unas palabras claves luego de que la aplicación haya sido entrenada mediante el aprendizaje federado.

Este aprendizaje federado ha sido realizado pocas veces en microcontroladores y por tanto es un campo con poca información por el momento. El problema existente consistirá en investigar a fondo este campo de aprendizaje federado, hallar diferentes posibilidades respecto a nuestro punto de partida con el fin de poder mejorar procesos o tener más variedad en cuanto a opciones de aplicaciones de aprendizaje federado.

### 1.3.1 Objetivo de la aplicación

Nuestro punto de partida es un aplicación que tiene como objetivo reconocer una serie de palabras claves previamente entrenadas mediante el aprendizaje federado. Existen muchas más aplicaciones diferentes en el campo del aprendizaje automático y por tanto un objetivo será poder crear una aplicación con otro tipo de utilidad mediante el uso del aprendizaje federado en microcontroladores. Para lograr esto habrá que tratar con cuidado la cantidad de memoria que necesita una posible aplicación diferente, ya que los microcontroladores tienen su capacidad limitada y no tenerla en mente provocará el mal funcionamiento de la aplicación.

### 1.3.2 Uso de nuevas tecnologías

Para poder crear la red de aprendizaje federado se usaron unas tecnologías concretas para comunicar los dispositivos con el respectivo servidor, realizar la tarea de servidor, la ejecución del código y otras muchas.

Uno de los desafíos del proyecto será investigar el uso de tecnologías que permitan nuevos usos a este tipo de aplicaciones, tanto por el hecho de mejorar procesos como la comunicación o velocidad de cálculos, como poder dar alternativas viables para procesos que se llevan a cabo en el transcurso de la aplicación.

### 1.3.3 Limitaciones de los microcontroladores

Hay que tener en cuenta que los microcontroladores son dispositivos mucho más limitados que otros tipos de dispositivos a los que estamos acostumbrados a usar. Tanto en cómputo, memoria y almacenamiento los microcontroladores tienen cientos de veces menos capacidad que un ordenador. Por tanto es un problema que habrá que tener en mente para poder lograr aplicaciones eficientes en estos tipos de dispositivos.

## 1.4 Actores implicados

Los principales interesados y beneficiados de los resultados de este proyecto será la comunidad de desarrolladores de TinyML, los cuales podrán acceder a la información y código producido con el fin de poder desarrollar sus aplicaciones en TinyML enfocadas al aprendizaje federado.

Los resultados obtenidos también podrán motivar a futuros emprendedores a definir una idea existente que se pueda lograr eficientemente con este tipo de aplicaciones y dispositivos de bajo consumo que, anteriormente, debido a la tecnologías consideradas no era una opción viable o que generara beneficios.

## 1.5 Justificación

Como ya se ha mencionado para el desarrollo de la aplicación disponemos de un punto de partida. A dicha aplicación se le harán estudios de posibles cambios que mejorarián su capacidad de realizar diferentes tareas. Con el fin de poder entender el por qué de cada mejora habrá que determinar su estado actual y que beneficios implicaría una mejora en ese campo.

La aplicación de aprendizaje federado dispone de un servidor ejecutado en un ordenador que se conecta a los diferentes dispositivos mediante el uso de cables. Una de las funciones de los microcontroladores usados en este proyecto es el hecho de poder utilizar bluetooth como medio de transmisión de datos. El hecho de poder comunicarse con el servidor mediante bluetooth abrirá muchas posibilidades debido a que la distancia entre servidor y dispositivo habrá aumentado sin necesidad de cables entre ellos.

Una opción a contemplar sería el uso de un microcontrolador como servidor principal y poder rechazar el uso de cualquier dispositivo avanzado en la aplicación. Para poder lograr esto es bastante seguro que habría que mejorar la capacidad de procesamiento de los microcontroladores para poder soportar las tareas que implica el servidor.

Para ello, estos microcontroladores disponen de una tecnología llamada RTOS (Real-time operating system o sistema operativo de tiempo real), la cual permitiría la organización de las tareas de cada microcontrolador y el hecho de poder ejecutarlas en paralelo para poder mejorar la eficiencia de estas.

El mundo del aprendizaje automático es muy extenso y sumado al mundo de los microcontroladores que tienen tantas limitaciones puedes encontrar casos en que una aplicación funcione correctamente usando un método y otra aplicación sea incapaz de siquiera ejecutarse por el tipo de datos que utilice. Por tanto, experimentar con los límites de las aplicaciones y con diferentes aplicaciones ayudará a comprender mejor con qué limitaciones estamos trabajando y hasta qué punto se podría llegar.

## 1.6 Alcance

En esta sección se definirán los objetivos genéricos y subobjetivos del proyecto, los requisitos funcionales y potenciales riesgos e inconvenientes que se pueden presentar durante la ejecución del proyecto.

### 1.6.1 Objetivos genéricos

Este proyecto consta de dos objetivos genéricos que dividen el proyecto en dos etapas. Una primera de iniciación en el mundo del aprendizaje federado en TinyML y la segunda en la experimentación e investigación profunda de este campo.

El primer objetivo consiste en investigar, aprender y familiarizarse con las herramientas y programas necesarios para realizar aprendizaje federado y conseguir replicar la aplicación del punto de partida.

El segundo objetivo consiste en investigar en profundidad todas las tecnologías que contienen los microcontroladores que pueden servir de ayuda para poder mejorar las aplicaciones de aprendizaje federado.

### 1.6.2 Subobjetivos

Dentro del primer objetivo se pueden incluir los siguientes subobjetivos para poder facilitar la realización de este:

- Comprender y ser capaz de generar cualquier tipo de aplicación de aprendizaje federado.
- Comprender y ser capaz de generar cualquier tipo de aplicación TinyML en los microcontroladores.
- Comprender y ser capaz de generar un servidor capaz de enviar y recibir información a los dispositivos conectados a él mediante procesos de comunicación.
- Comprender las limitaciones que tienen los microcontroladores con los que se trabajará.

Dentro del segundo objetivo se pueden incluir los siguientes subobjetivos para poder facilitar la realización de este:

- Comprender el funcionamiento de la tecnología de Bluetooth y aplicarla a la aplicación
- Experimentar con diferentes redes neuronales artificiales para poder evaluar cuál tiene mejores resultados
- Experimentar con diferentes aplicaciones de aprendizaje federado y comprobar su correcto funcionamiento en los microcontroladores
- Probar la utilización de un microcontrolador como servidor del aprendizaje federado, esto llevará a la necesidad de estudiar y aplicar la tecnología de RTOS para los microcontroladores

### 1.6.3 Obstáculos y riesgos

El desarrollo de este proyecto se lleva a cabo entre dos mundos muy diferentes, los microcontroladores y el aprendizaje automático. Entonces claramente un obstáculo grande que se encontrará en el proyecto será la interacción entre estos dos. El aprendizaje automático requiere de un gran número de cálculos y por otro lado tenemos los microcontroladores que son muy limitados en cuanto a la capacidad que pueden proporcionar. Una mala organización de los recursos disponibles puede acabar fácilmente en que la aplicación no sea capaz de ejecutarse por la necesidad de alguno de estos.

Otro gran obstáculo que podemos encontrar es el tiempo, estos dos mundos son muy diferentes entre ellos y para poder tratar los dos de manera correcta para que funcionen sin problemas requerirá de la investigación y estudio dedicada a cada caso, pero el tiempo del proyecto es limitado y puede llegar a no ser suficiente para poder llegar a cumplir todos los objetivos y subobjetivos propuestos.

En cuanto al aprendizaje de la aplicaciones se suelen usar grandes datasets de miles de ejemplos para entrenar los modelos, pero para este proyecto no disponemos de estos y eso puede producir que nuestros resultados no sean del todo satisfactorios.

Las limitaciones de los microcontroladores de solo poder tener un código programado en ellos para ejecutar también ralentiza el entrenamiento y testeo de los modelos, debido a que solo se podrá ir de uno en uno cada vez. No será hasta el final del entrenamiento con todos los dispositivos con el mismo modelo que se podrá determinar si el modelo es correcto y funcional o el caso contrario.

## 1.7 Metodología y rigor

Este proyecto se dividirá en dos claras fases que corresponden cada una con cada objetivo genérico mencionado anteriormente. La primera parte consistirá en realizar una formación sobre TinyML mediante el uso de 3 cursos publicados por la Universidad de Harvard en la plataforma Edx [2]. Con estos cursos realizados y el trabajo de Marc Monfort podré recrear la aplicación de aprendizaje federado que hará la función de punto de partida.

La segunda parte del proyecto consistirá en la investigación de las tecnologías mencionadas en los subobjetivos del segundo objetivo genérico y su implementación en una aplicación de aprendizaje federado.

La metodología escogida está basada en la metodología Agile, la cual trata en ciclos cortos de 1 a 2 semanas en los cuales se planean al principio unas metas que conseguir una vez acabe el ciclo, las cuales serán tratadas si se han cumplido o no junto al director del proyecto y se planificará el siguiente ciclo.

La ventaja de esta metodología es poder avanzar con unas metas a corto plazo el proyecto y de esta manera tener una mejor idea del estado del mismo. En caso de que un inconveniente apareciera sería tratado lo antes posible y de esta manera no ralentizará el proyecto en gran medida.

### 1.7.1 Herramientas de seguimiento

Para poder llevar un seguimiento adecuado de las metas de cada ciclo, planificar las futuras metas y debatir sobre el progreso del proyecto se llevarán a cabo reuniones cada 2 semanas con el director del proyecto Félix Freitag, estas reuniones se realizarán en la plataforma de Jitsi[3], la cual permite realizar llamadas por internet en los que se

puede compartir la pantalla, activar la cámara entre otras funciones para poder mostrar adecuadamente los avances del proyecto.

Para almacenar el código de las aplicaciones se usará un repositorio público de la plataforma GitHub[4] y para la documentación del trabajo una carpeta de google drive compartida entre el autor y el director del trabajo.

## 2. Planificación temporal

### 2.1 Extensión temporal del proyecto

La planificación para el proyecto contempla su comienzo el 11 de agosto de 2021, momento en el que participé en los cursos de desarrollo de aplicaciones con TinyML de edX para empezar a familiarizarme con las herramientas usadas para crear aplicaciones para esta plataforma con el fin de tener un punto de partida más sólido y poder afrontar mejor la gestión inicial del proyecto. La fecha estimada para la finalización del proyecto y todas las tareas que lo engloba es el 10 de enero de 2022. Por tanto la extensión del trabajo consta de un total de 21 semanas, dedicando 23 horas a cada semana de media y un total de 483 horas al proyecto entero.

### 2.2 Definición de tareas

Para realizar una planificación adecuada del proyecto es necesario definir previamente las tareas a realizar en este. A continuación se definirán todas las tareas y subtareas de cada una de ellas, con sus respectivas dependencias. La tabla 1 muestra un resumen de las tareas con sus respectivas dependencias y recursos.

#### T1 Gestión del proyecto

En esta tarea se lleva a cabo todo lo relacionado con la presentación de proyecto antes de ser realizado. Esto engloba la contextualización, planificación, costes y el seguimiento que se llevará a cabo en el proyecto.

**T1.1 Contextualización y alcance del proyecto:** Definición del contexto en el que se encuentra el proyecto, el problema que se está tratando, el alcance del proyecto marcando los objetivos que se pretenden alcanzar y la metodología que se seguirá para su realización.

**T1.2 Planificación temporal del proyecto:** Planificación y organización de las tareas que se seguirán durante la realización del proyecto para poder alcanzar los objetivos marcados previamente. Para realizar esta tarea se necesita que la tarea T1.1 se haya finalizado previamente.

**T1.3 Presupuesto:** Estudio del coste económico para la realización del proyecto. Se identificarán todos los gastos relacionados a cada una de las tareas descritas en la planificación y se elaborará un informe con todos los gastos detectados. Para realizar esta tarea se necesita que la tarea T1.2 se haya finalizado previamente.

**T1.4 Definición final del proyecto:** Se redactará un documento final sobre la definición del proyecto juntando las tres tareas realizadas anteriormente y se revisará que todos los datos sean correctos. Para realizar esta tarea se necesita que la tarea T1.3 se haya finalizado previamente.

**T1.5 Seguimiento del proyecto:** Esta tarea contempla todo el conjunto de reuniones organizadas para discutir el estado del proyecto y el progreso del mismo.

## T2 Formación básica inicial

En esta tarea se lleva a cabo el aprendizaje de todos los conocimientos necesarios previos a empezar a desarrollar extensiones sobre la aplicación de aprendizaje federado.

**T2.1 Curso edX 1 “Fundamentals of TinyML”:** Realización del primero de los tres cursos disponibles en la plataforma de edX sobre TinyML. En este primer curso se introduce el mundo de TinyML y se lleva a cabo la realización de simples modelos de aprendizaje automático.

**T2.2 Curso de edX 2 “Applications of TinyML”:** Realización del segundo curso de la plataforma edX, donde se explica el proceso de entrenar y optimizar los modelos de aprendizaje automático para que un microcontrolador, dispositivo más limitado, pueda ejecutar sin ninguna dificultad.

**T2.3 Curso de edX 3 “Deploying TinyML”:** Realización del último curso de la plataforma edX, en el cual se explica la implementación de aplicaciones en microcontroladores, como cargar los modelos, recoger de manera correcta la información de entrada esperada para el modelo del aprendizaje automático, etc...

**T2.4 Aprendizaje de herramientas básicas:** Estudio de las herramientas necesarias para poder realizar aplicaciones de aprendizaje federado en TinyML:

- Python y Jupiter Notebooks
- Google Colab
- TensorFlow

**T2.5 Recreación de aplicación de aprendizaje federado:** Esta tarea consiste en recrear en dos microcontroladores la aplicación final de aprendizaje federado que realizó Marc Monfort en su proyecto de fin de grado. Para realizar esta tarea se necesita que la tareas T2.1, T2.2, T2.3, T2.4 hayan finalizado previamente de manera satisfactoria o de lo contrario no se tendrá un completo conocimiento del desarrollo de esta aplicación.

### T3 Extensión Bluetooth

En esta tarea se desarrollará la extensión de aplicar la tecnología de Bluetooth en el microcontrolador con el fin de mantener la comunicación con el servidor del aprendizaje federado. Esta comunicación inicialmente se lleva a cabo mediante el uso de cables, por tanto no se dispone de ningún conocimiento previo en este campo, ni en su realización.

**T3.1 Estudio de la tecnología Bluetooth:** Debido a la falta de información la primera tarea que hay que realizar para lograr con éxito la extensión es el estudio de esta tecnología y qué procedimientos se aplican para mantener la comunicación con esta.

**T3.2 Estudio de la aplicación de Bluetooth en microcontroladores:** Debido a las limitaciones de los controladores es necesario estudiar la manera en la que se puede aplicar la tecnología del Bluetooth en estos dispositivos. Para realizar esta tarea se necesita que la tarea T3.1 se haya finalizado previamente.

**T3.3 Desarrollo de la aplicación Bluetooth:** En esta tarea se llevará a cabo la modificación del código tanto del servidor como de los microcontroladores con el fin de que puedan mantener una comunicación activa entre los dos y poder ejecutar la aplicación de aprendizaje federado sin inconvenientes. Para realizar esta tarea se necesita que la tarea T3.2 se haya finalizado previamente.

**T3.4 Testeo y recolección de datos:** Una vez realizada una aplicación de aprendizaje federado que utilice Bluetooth, se pondrá a prueba y se comprobará su correcto funcionamiento. También se recolectarán datos sobre las posibles mejoras o lo contrario sobre la eficiencia de la aplicación. Para realizar esta tarea se necesita que la tarea T3.3 se haya finalizado previamente.

## T4 Modificación objetivo de la aplicación

En esta tarea se realizará la modificación del objetivo de la aplicación de aprendizaje federado. El objetivo inicial de esta consiste en la detección de palabras clave (“keyword spotting”), la aplicación graba el audio tras apretar un botón y es capaz de reconocer si es una palabra clave de las que tiene registradas o una aleatoriedad. Este objetivo se modificará para que la aplicación consista en la detección de imágenes (“image detection”), la aplicación capturará imágenes con la cámara de la que dispone y será capaz de detectar si un objeto se encuentra en ellas o no.

**T4.1 Estudio de la detección de imágenes (“image detection”):** Debido al conocimiento previo sobre técnicas de detección de imágenes estudiadas en la asignatura de Visión de Computadores, se realizará un repaso sobre las técnicas implementadas anteriormente para la realización de este tipo de aplicaciones.

**T4.2 Estudio del módulo de cámara:** Los microcontroladores dispondrán de un módulo añadido con la función de capturar imágenes en ellos. Se necesitará la investigación de en qué tipo de datos se guardan las imágenes y cómo trabajar con ellos.

**T4.3 Desarrollo de la aplicación de detección de imágenes:** En esta sección se modificará el código de la aplicación para que esta trabaje con la detección de imágenes y no la detección de palabras claves. Para lograr esto se deberá cambiar tanto el método de recolección de datos del microcontrolador como el modelo de aprendizaje federado usado previamente y requerirá un breve estudio sobre el diseño de la red neuronal necesaria para lograr este objetivo. Para realizar esta tarea se necesita que las tareas T4.1 y T4.2 hayan finalizado previamente.

**T4.4 Entrenamiento del modelo:** Una vez la aplicación de detección de imágenes funcione correctamente se procederá al entrenamiento del modelo de aprendizaje automático. Necesario para que la aplicación sea capaz de detectar los objetos. Durante este proceso será necesario arreglar todos los posibles

fallos en el código nuevos debidos al entrenamiento del modelo. Para realizar esta tarea se necesita que la tarea T4.3 se haya finalizado previamente.

**T4.5 Testeo y recolección de datos:** Cuando el modelo para la detección de imágenes haya sido entrenado se pondrá a prueba y se comprobará su correcto funcionamiento. También se recolectarán datos sobre la tasa de aciertos en las predicciones de la aplicación y su eficiencia en cuanto a lograr el objetivo de esta. Para realizar esta tarea se necesita que la tarea T4.4 se haya finalizado previamente.

## T5 Microcontrolador como servidor

Esta tarea tiene como objetivo poder realizar las operaciones de las que se encarga el servidor de una aplicación de aprendizaje federado en un microcontrolador. Para poder lograr este objetivo será necesario optimizar la ejecución del programa en los microcontroladores mediante técnicas de paralelismo. Con el fin de lograr esto será necesario el uso de la tecnología RTOS implementada en los microcontroladores.

**T5.1 Estudio RTOS en microcontroladores:** la tecnología RTOS es usada en muchos dispositivos diferentes, pero el uso de esta tecnología en microcontroladores es muy reducido y se dispone de poca información al respecto. Por tanto se necesitará un estudio sobre su implementación en estos dispositivos previamente a aplicarla.

**T5.2 Conversión a C:** los microcontroladores usan código escrito en C, pero el código dedicado al servidor de la aplicación inicial está escrito en Python, por tanto será necesario reescribir todo el código al lenguaje C si se quiere que un microcontrolador lo ejecute.

**T5.3 Aplicación de RTOS:** Una vez ya dispongamos del código en C del servidor tendremos que aplicar la tecnología de RTOS en él para que los microcontroladores puedan ejecutar diferentes funciones en paralelo y por tanto soportar el trabajo del servidor. Para realizar esta tarea se necesita que las tareas T5.1 y T5.2 hayan finalizado previamente.

**T5.4 Testeo y recolección de datos:** Finalmente habrá que cargar el código nuevo de servidor en un microcontrolador y crear un canal de comunicación entre los demás microcontroladores y el que está haciendo de servidor. Se recolectarán datos del funcionamiento de la aplicación para poder determinar si se ha logrado el objetivo de usar un microcontrolador como servidor. Para realizar esta tarea se necesita que la tarea T5.3 se haya finalizado previamente.

## T6 Documentación del proyecto

En esta tarea se elaborarán el conjunto de documentos para la información del proyecto.

**T6.1 Memoria final:** Redacción del documento final del proyecto. Este documento juntará toda la recolección de datos de cada tarea relacionada con la extensión de la aplicación inicial y la documentación sobre la gestión del proyecto.

**T6.2 Presentación del proyecto:** En esta última tarea se realizará todos los procesos necesarios para llevar a cabo la presentación y defensa del proyecto, tales como la preparación de las diapositivas usadas y el ensayo de la presentación.

## 2.3 Recursos

### 2.3.1 Recursos humanos

**Au - Autor:** Autor del proyecto. Encargado de la realización de todas las tareas del proyecto.

**Di - Director:** Director del proyecto. Responsable del seguimiento y desarrollo del proyecto. Interviene en las reuniones de seguimiento y se le puede consultar en caso de dudas.

### 2.3.2 Recursos materiales

**PC - Ordenador:** Ordenador común de escritorio. Se usará para los diferentes estudios que se llevan a cabo en las tareas, para la edición del código y como servidor del aprendizaje federado en las primeras tareas del proyecto.

**KA - Kit Arduino para TinyML:** Kit que contiene el microcontrolador *Arduino Nano 33 BLE Sense*, una placa y un módulo cámara compatible con el microcontrolador.

**CH - Componentes de Hardware:** componentes que se usarán para crear pequeños circuitos para agregar botones disponibles al microcontrolador.

**JM - Jitsi Meet:** plataforma para realizar las reuniones.

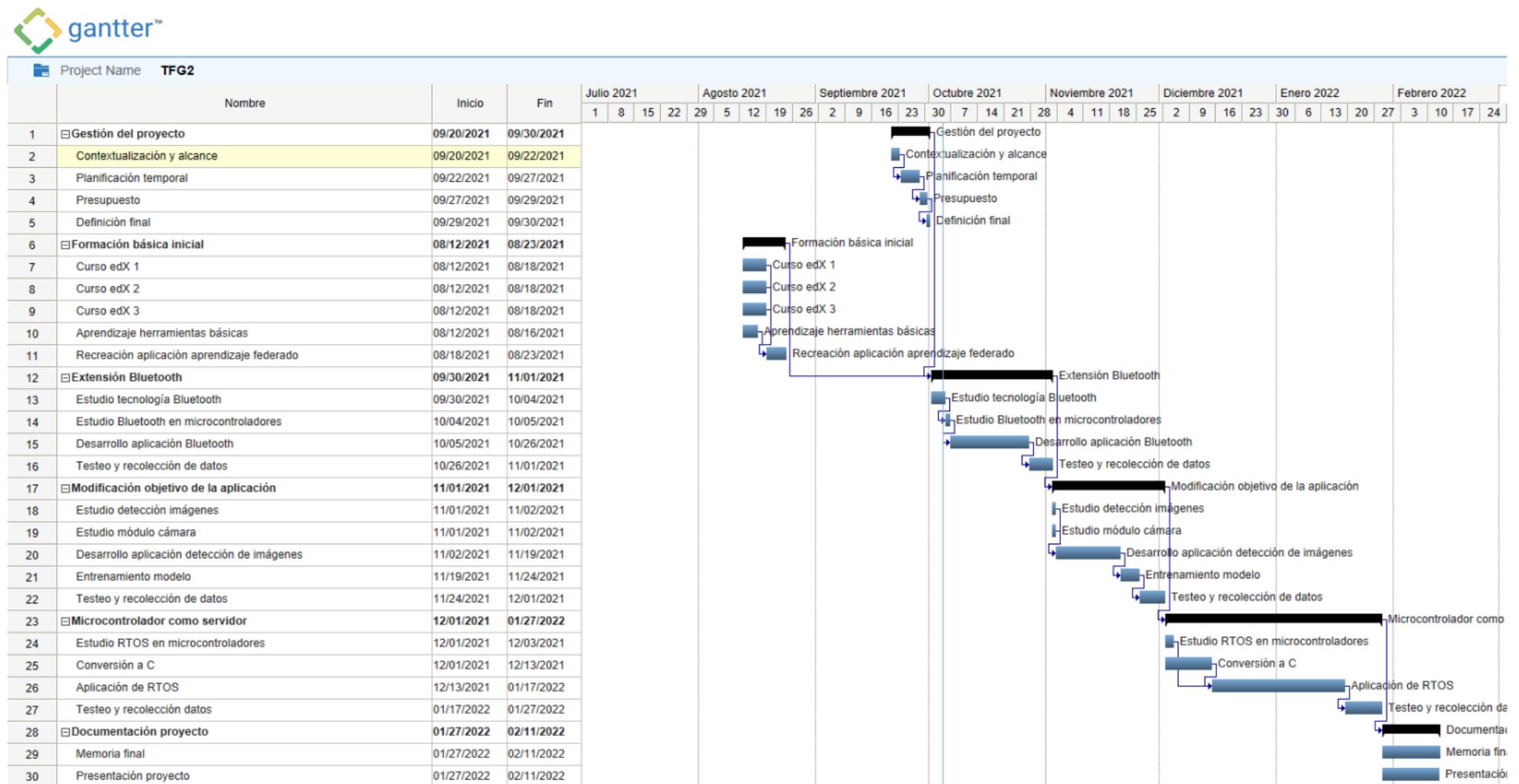
**GD - Google Drive:** plataforma para almacenar la documentación del proyecto

ID	Tarea	Tiempo (h)	Dependencias	Recursos
<b>T1</b>	<b>Gestión del proyecto</b>	<b>50</b>		<b>Au</b>
T1.1	Contextualización y alcance	10		PC, GD
T1.2	Planificación temporal	10	T1.1	PC, GD
T1.3	Presupuesto	6	T1.2	PC, GD
T1.4	Definición final	4	T1.3	PC, GD
T1.5	Seguimiento	20		DI, JM
<b>T2</b>	<b>Formación básica inicial</b>	<b>65</b>		<b>Au, PC</b>
T2.1	Curso edX 1	15		KA
T2.2	Curso edX 2	15		KA
T2.3	Curso edX 3	15		KA
T2.4	Aprendizaje herramientas básicas	8		
T2.5	Recreación aplicación aprendizaje federado	12	T2.1, T2.2, T2.3, T2.4	KA, CH
<b>T3</b>	<b>Extensión Bluetooth</b>	<b>75</b>	<b>T2</b>	<b>Au, PC</b>
T3.1	Estudio tecnología Bluetooth	5		
T3.2	Estudio Bluetooth en microcontroladores	5	T3.1	
T3.3	Desarrollo aplicación Bluetooth	50	T3.2	KA, CH
T3.4	Testeo y recolección datos	15	T3.3	KA, CH
<b>T4</b>	<b>Modificación objetivo de la aplicación</b>	<b>75</b>	<b>T3</b>	<b>Au, PC</b>

---

<b>T4.1</b>	<b>Estudio detección imágenes</b>	<b>3</b>		
<b>T4.2</b>	<b>Estudio módulo cámara</b>	<b>3</b>		
<b>T4.3</b>	<b>Desarrollo aplicación detección de imágenes</b>	<b>44</b>	<b>T4.1, T4.2</b>	<b>KA, CH</b>
<b>T4.4</b>	<b>Entrenamiento modelo</b>	<b>10</b>	<b>T4.3</b>	<b>KA, CH</b>
<b>T4.5</b>	<b>Testeo y recolección datos</b>	<b>15</b>	<b>T4.4</b>	<b>KA, CH</b>
<b>T5</b>	<b>Microcontrolador como servidor</b>	<b>150</b>	<b>T4</b>	<b>Au, PC</b>
<b>T5.1</b>	<b>Estudio RTOS en microcontroladores</b>	<b>10</b>		
<b>T5.2</b>	<b>Conversión a C</b>	<b>30</b>		
<b>T5.3</b>	<b>Aplicación de RTOS</b>	<b>85</b>	<b>T5.1, T5.2</b>	<b>KA, CH</b>
<b>T5.4</b>	<b>Testeo y recolección datos</b>	<b>25</b>	<b>T5.3</b>	<b>KA, CH</b>
<b>T6</b>	<b>Documentación proyecto</b>	<b>68</b>	<b>T5</b>	<b>Au,PC,GD</b>
<b>T6.1</b>	<b>Memoria final</b>	<b>40</b>		
<b>T6.2</b>	<b>Presentación proyecto</b>	<b>38</b>		
<b>Total</b>		<b>483</b>		

*Tabla 1: Tareas del proyecto y recursos (elaboración propia)*



## *Diagrama Gantt del proyecto*

## 2.4 Gestión del riesgo: Planes alternativos

Todo proyecto conlleva riesgos que hay que tener en cuenta ya que la realización de un proyecto perfecto y sin inconvenientes es muy poco probable. Por ello en esta sección se describirán los posibles riesgos detectados que pueden surgir durante la realización del proyecto y respectivos planes alternativos a llevar a cabo para solucionar estos inconvenientes.

### 2.4.1 Tiempo

Uno de los principales problemas que pueden aparecer en proyectos de duración avanzada es el tiempo. En la planificación temporal se han estimado unas horas aproximadas para cada tarea, pero en el mundo de la programación los errores de código son muy comunes y a veces difíciles de encontrar la solución. Es por eso que es difícil poder predecir la cantidad de horas que se necesitaran para realizar cada tarea. Existe el caso de que debido a que el mundo de TinyML es reciente y la información sobre él es limitada, sobre todo en campos como el aprendizaje federado, que el tiempo previsto para dedicar al proyecto sea insuficiente para lograr los objetivos marcados.

En caso de que este problema sea detectado mediante las reuniones de seguimiento como está previsto, la primera solución que se tomará para ello es aumentar las horas de trabajo que se le dedican al proyecto semanalmente. Es una solución simple pero puede ser suficiente para arreglarlo. En el caso de que no sea así se dispone de dos semanas extras antes de la presentación del trabajo que no se han contado para la realización de este.

### 2.4.2 Desarrollo de las aplicaciones

Las tareas dedicadas a las extensiones de la aplicación tienen todas en común que son progresivas. Por tanto si una subtarea se hace imposible completarla o no se encuentra una posible solución hará que la tarea sea imposible de realizar y las siguientes subtareas a esta. Debido a la poca aplicación de la metodología de aprendizaje federado en microcontroladores no es imposible encontrarse con un punto incapaz de resolver. Es por eso que es necesario un plan alternativo para cada tarea.

**Bluetooth:** en el caso de no ser capaz de poder mantener un canal de comunicación entre servidor y microcontroladores mediante el uso de la tecnología de Bluetooth se redactará un informe explicando detalladamente el

problema que no se ha sabido resolver y se explicaran las diferentes soluciones probadas para solucionarlo.

**Detección de imágenes:** en el caso de no ser capaz de que la aplicación realice esta tarea correctamente se volverá al uso de la detección de palabras claves y se hará un estudio sobre los diferentes tipos de redes neuronales que se pueden usar para esa aplicación y poder testear las limitaciones del microcontrolador con respecto a las redes neuronales.

**Microcontrolador como servidor:** en el caso de no ser capaz de tener un microcontrolador como servidor de la aplicación de aprendizaje federado se creará un informe explicando el problema encontrado y se substituirá esta extensión por la extensión de aplicar RTOS para mejorar el rendimiento de la aplicación.

## 2.5 Cambios en la planificación inicial

Durante la realización del proyecto se encaró un problema con la tarea T3, la aplicación Bluetooth. Primeramente se contemplaron diferentes alternativas para encarar el objetivo de realizar la aplicación de aprendizaje federado mediante el sistema de Bluetooth. La plataforma de Python dispone de varias librerías para poder crear sockets de Bluetooth, estos sockets eran una opción a tener en cuenta para realizar la comunicación entre los dispositivos, y fue la primera opción en la que se empezó a trabajar. Esta primera aproximación dió muchos problemas y no se conseguía establecer una conexión entre ambos dispositivos, lo que provocó un estancamiento en esta tarea. Después de probar otras opciones como PyBluez e investigar sobre el posible origen del problema, se detectó que el hecho que generaba problemas en las opciones tratadas previamente era el tipo de Bluetooth usado por los microcontroladores, BLE (Bluetooth Low Energy). Este Bluetooth dispone de diferentes funcionalidades y son más reducidas con el objetivo de consumir menos energía en el proceso de comunicación.

Para poder usar esta tecnología correctamente investigué sobre cómo crear una aplicación simple para poder conectar un microcontrolador al ordenador y poder transmitir información. Eventualmente conseguí realizar una aplicación capaz de ello mediante el uso de una librería llamada Bleak.

Debido a este estancamiento no calculado previamente y una cantidad de horas semanales dedicadas (10-12h) menor a la esperada en la planificación del proyecto

(23h). Entre director del proyecto y autor se llegó a la conclusión de que la mejor solución sería cambiar el turno del proyecto de Enero a Abril, obteniendo así tres meses adicionales para la realización del proyecto.

Posteriormente a la realización del cambio se siguió trabajando en la aplicación de Bluetooth con la nueva tecnología encontrada. Pero el estancamiento no cambió debido a que el conocimiento sobre esta tecnología no era muy avanzado y nos encontrábamos errores desconocidos a la hora de aplicarla para la realización del aprendizaje federado. Es por eso que se decidió dejar la aplicación de Bluetooth para futuros planes del proyecto y se empezó a trabajar en la T4 del proyecto, desarrollo de una aplicación de aprendizaje federado usando una cámara para recolectar la información.

Por último se ha decidido eliminar como tarea del proyecto la Tarea 5, donde se esperaba usar un microcontrolador como servidor en vez de un ordenador. Esta decisión ha sido tomada debido a que para poder realizar este objetivo se necesitaba previamente haber completado la Tarea 3 (extensión de bluetooth) sin problemas, que no ha sido el caso.

### 3. Presupuesto

Para calcular el presupuesto del proyecto se tienen que tener en cuenta los costes de cada una de las tareas identificadas en la planificación del proyecto. Estos gastos se dividen en 4 grupos diferentes, costes de personal, costes genéricos, costes de contingencia y costes imprevistos. A continuación, se detallan en concreto cada uno de estos costes.

#### 3.1 Costes de personal

Los costes de personal son los que se asocian a los recursos humanos necesarios para realizar cada una de las tareas del proyecto. Para realizar una estimación adecuada de los costes personales del proyecto será necesario en primer lugar diferenciar los perfiles de los trabajadores necesarios para la realización de las tareas, para posteriormente atribuir una estimación del sueldo de dicho perfiles basándose en los datos obtenidos de la plataforma Payscale [5][6].

A partir de las tareas descritas anteriormente se identifican dos perfiles necesarios para la realización del proyecto: el director de proyecto y un ingeniero de informática (especializado en inteligencia artificial).

Para el cálculo del salario de cada perfil se asume que trabajan 40 horas a la semana durante 52 semanas al año, a lo que hay que restar 14 días festivos anuales, registrados en el calendario laboral de 2020 [7], y 22 días laborables de vacaciones. Para calcular la cotización de la seguridad social se ha aplicado un 28,30 basado en los datos del SEPE sobre bases y tipos de cotización para contingencias comunes del 2021 [8]. La Tabla 1 contiene los perfiles con sus correspondientes salarios estimados y los costes derivados de cada uno.

Perfil	Salario anual	Cotización	Coste anual	Coste por hora	Horas	Coste total
Ingeniero de software	23.921,99 €	9.442,01 €	33.364 €	18,61€	483	8.988,63 €
Director de proyecto	53.853,15 €	21.255,85 €	75.109 €	41,91€	20	838,2 €

Tabla 2: Perfiles del proyecto y sus costes (elaboración propia)

La Tabla 2 muestra el coste de cada tarea teniendo en cuenta los costes personales anteriores.

ID	Tarea	Tiempo (h)	Roles	Cost (€)
T1	Gestión del proyecto	50	Ingeniero de software	1.768,64
T1.1	Contextualización y alcance	10		186,1
T1.2	Planificación temporal	10		186,1
T1.3	Presupuesto	6		111,6
T1.4	Definición final	4		74,44
T1.5	Seguimiento	20	Director de proyecto	1.210,4
T2	Formación básica inicial	65	Ingeniero de software	1.209,65
T2.1	Curso edX 1	15		279,15
T2.2	Curso edX 2	15		279,15
T2.3	Curso edX 3	15		279,15
T2.4	Aprendizaje herramientas básicas	8		148,88
T2.5	Recreación aplicación aprendizaje federado	12		223,32
T3	Extensión Bluetooth	75	Ingeniero de software	1.395,75
T3.1	Estudio tecnología Bluetooth	5		93,05
T3.2	Estudio Bluetooth en microcontroladores	5		93,05
T3.3	Desarrollo aplicación Bluetooth	50		930,5
T3.4	Testeo y recolección datos	15		279,15
T4	Modificación objetivo de la aplicación	75	Ingeniero de software	1.395,75
T4.1	Estudio detección imágenes	3		55,83
T4.2	Estudio módulo cámara	3		55,83
T4.3	Desarrollo aplicación detección de imágenes	44		818,84

<b>T4.4</b>	<b>Entrenamiento modelo</b>	<b>10</b>		<b>186,1</b>
<b>T4.5</b>	<b>Testeo y recolección datos</b>	<b>15</b>		<b>279,15</b>
<b>T5</b>	<b>Microcontrolador como servidor</b>	<b>150</b>	<b>Ingeniero de software</b>	<b>2.791,5</b>
<b>T5.1</b>	<b>Estudio RTOS en microcontroladores</b>	<b>10</b>		<b>186,1</b>
<b>T5.2</b>	<b>Conversión a C</b>	<b>30</b>		<b>558,3</b>
<b>T5.3</b>	<b>Aplicación de RTOS</b>	<b>85</b>		<b>1.581,85</b>
<b>T5.4</b>	<b>Testeo y recolección datos</b>	<b>25</b>		<b>465,25</b>
<b>T6</b>	<b>Documentación proyecto</b>	<b>68</b>	<b>Ingeniero de software</b>	<b>1.451,58</b>
<b>T6.1</b>	<b>Memoria final</b>	<b>40</b>		<b>744,4</b>
<b>T6.2</b>	<b>Presentación proyecto</b>	<b>38</b>		<b>707,18</b>
<b>Total</b>				<b>10.012,87</b>

Tabla 3: Tareas del proyecto y costes (elaboración propia)

### 3.2 Costes genéricos

Para calcular correctamente los costes genéricos del proyecto hay que tener en cuenta todo tipo de material necesario para la realización de este.

Uno de los componentes esenciales del proyecto es la disponibilidad de un equipo informático adecuado para las tareas que se le tienen asignado. Asumiré que el precio de un ordenador de sobremesa suficiente para todo el trabajo será de 800€ como mucho y unos periféricos estándar para acompañar este ordenador de 150€ como mucho. Para calcular el coste que repercute sobre el proyecto de estos componentes obtendré la amortización correspondiente a estos usando las siguientes fórmulas (3)(4). Los resultados se pueden observar en la Tabla 4 a continuación.

$$\text{Coste por hora} = \text{Precio recurso} / (\text{esperanza de vida} * \text{días de trabajo} * \text{horas de trabajo}) \quad (3)$$

$$\text{Amortización} = \text{coste por hora} * \text{horas usado}$$

Se asume que la esperanza de vida de los dispositivos es de 5 años, los días laborales en Cataluña son 252 y 8 horas de trabajo diarias para el cálculo del coste por hora de estos dispositivos.

Dispositivo	Precio (€)	Horas usado	Amortización (€)
Ordenador de sobremesa	800	483	38,33
Periféricos	150	483	7,19

Tabla 4: Costes de amortización (elaboración propia)

Como costes genéricos también tenemos 3 Kits de desarrollo TinyML de Arduino [9], necesitaremos 3 kits por el momento para poder desarrollar aplicaciones de aprendizaje federado y poder testearlas, ya que al servidor tiene que haber como mínimo dos dispositivos diferentes conectados y en el caso que queramos tener un microcontrolador como servidor necesitaremos 3 kits. Y necesitaremos también 2 kits de componentes hardware para crear circuitos sencillos [10] para poder entrenar cómodamente los microcontroladores.

Para el cálculo de electricidad, internet y espacio de trabajo usaré como referencia un espacio fijo unipersonal de coworking que incluye todo lo mencionado anteriormente y está disponible las 24 horas del día. Este espacio tiene un coste de 349 € al mes [11], teniendo en cuenta que el proyecto abarca un total de 5 meses se tendrá que multiplicar el precio por 5.

A continuación la Tabla 5 resume todos los costes genéricos del proyecto y el total de todos estos costes sumados.

Recurso	Coste (€)
Ordenador de sobremesa	38,33
Periféricos	7,19
3 x Kit de desarrollo TinyML de Arduino	126
2 x componentes hardware	44,97
Espacio de trabajo + internet + electricidad	1.745
<b>Total</b>	<b>1.961,49</b>

Tabla 5: Costes genéricos del proyecto (elaboración propia)

### 3.3 Costes de contingencia

Los costes de contingencia corresponden a cualquier tipo de imprevisto que pueda aparecer y no haya sido contemplado previamente. Este coste se calcula como un porcentaje de la suma de los dos costes anteriores, personales y genéricos. Se suele asociar un porcentaje entre 10% y 20% a los proyectos de desarrollo, debido a la previa planificación del proyecto se le asignará un porcentaje de 13% ya que la aparición de imprevistos ya se tiene en cuenta en la asignación de horas a cada tarea.

Por tanto, el coste de contingencia asciende a un total de **1.556,66 €**.

### 3.4 Costes de imprevistos

Los costes de imprevistos corresponden a situaciones que no son seguras que vayan a suceder y por tanto se calculan a parte. Para calcular estos costes se multiplicarán los costes asociados a los imprevistos con la probabilidad de que ocurra cada uno de ellos.

En este proyecto se pueden detectar dos imprevistos que generarán costes adicionales, la necesidad de adquirir un nuevo kit de desarrollo TinyML de Arduino, tanto por si se decide ampliar la aplicación como por el mal funcionamiento de alguno de ellos, y la ampliación de tiempo en caso de que se necesite más tiempo para poder finalizar todas las tareas planificadas. En el caso de la ampliación de tiempo se asumirá un incremento del 10% de los costes totales de personal, lo que corresponde a 1.001,29 €.

La Tabla 6 resume los costes de los imprevistos, la probabilidad de que estos acontecimientos se den y el coste final calculado.

<b>Imprevisto</b>	<b>Coste (€)</b>	<b>Probabilidad</b>	<b>Coste final (€)</b>
Compra de kit de desarrollo TinyML de Arduino adicional	42,00	20,00%	8,4
Ampliación de tiempo	1.001,29	30,00%	300,39
<b>Total</b>			<b>308,79</b>

Tabla 6: Costes de imprevistos del proyecto (elaboración propia)

### 3.5 Coste final del proyecto

Para finalizar en la Tabla 7 se puede observar un resumen del coste total del proyecto donde intervienen todos los costes previamente calculados.

Tipo de coste	Coste (€)
Costes de personal	<b>10.012,87</b>
Costes genéricos	<b>1.961,49</b>
Costes de contingencia	<b>1.556,66</b>
Costes de imprevistos	<b>308,79</b>
<b>Total</b>	<b>13.839,81</b>

Tabla 7: Coste final del proyecto (elaboración propia)

## 4. Control de gestión

El control de gestión del proyecto se realizará en las sesiones de seguimiento junto al director del proyecto donde además de supervisar la planificación de las tareas se revisará que se hayan adquirido los recursos necesarios para cada tarea y que las horas destinadas correspondan con las planificadas.

Para poder evaluar correctamente la evolución de los costes se realizará un cálculo de sobre la desviación del coste y la desviación del consumo. Las siguientes dos fórmulas (8) (9) expresan el procedimiento para realizar los cálculos.

$$\text{Desviación de coste} = (\text{Coste estimado} - \text{Coste real}) * \text{Horas consumidas} \quad (8)$$

$$\text{Desviación de consumo} = (\text{Horas estimadas} - \text{Horas consumidas}) * \text{Consumo estimado} \quad (9)$$

## 5. Contexto

### 5.1 Herramientas de desarrollo

Para la realización del código del proyecto se ha usado el editor de código fuente desarrollado por Microsoft llamado Visual Studio Code, el cual dispone de un IDE (Integrated Development Environment) como extensión llamada PlatformIO, este IDE hace la tarea de creación y distribución de código para los Sistemas Embebidos[12] mucho más fácil y rápida. Estos Sistemas Embebidos se basan en usar microprocesadores o microcontroladores para realizar una o más tareas simples, de manera frecuente en un sistema de computación en tiempo real.

Dado que el proyecto parte de una aplicación donde se usan microcontroladores para obtener información, procesarla y enviarla al servidor principal, es correcto asumir que se trata de un Sistema Embebido. Por tanto PlatformIO resulta ser una herramienta muy conveniente a la hora de programar en el proyecto.

#### 5.1.1 PlatformIO

Como ya se ha comentado previamente, PlatformIO[13] consiste en un framework especializado en Sistemas Embebidos, pudiendo trabajar en múltiples plataformas y con un soporte en diferentes *software development kit (SDK)*[14] y frameworks. Incluye herramientas para debugar, análisis de código automático, Unit Testing (para poder testear fragmentos del código por separado) y desarrollo remoto. En la figura 2 se puede apreciar la pantalla de inicio de PlatformIO.

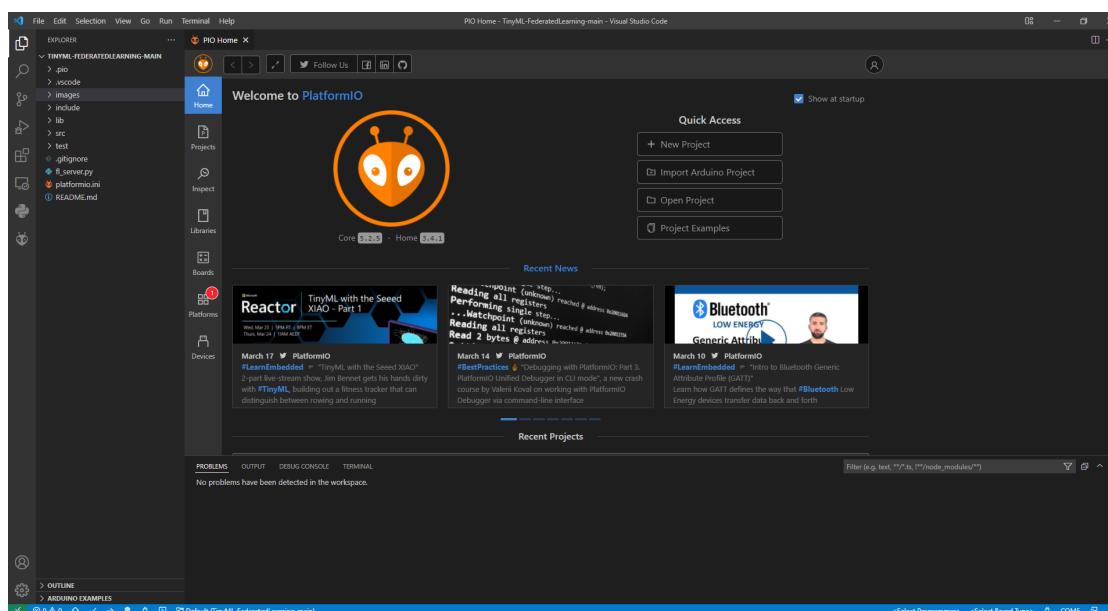
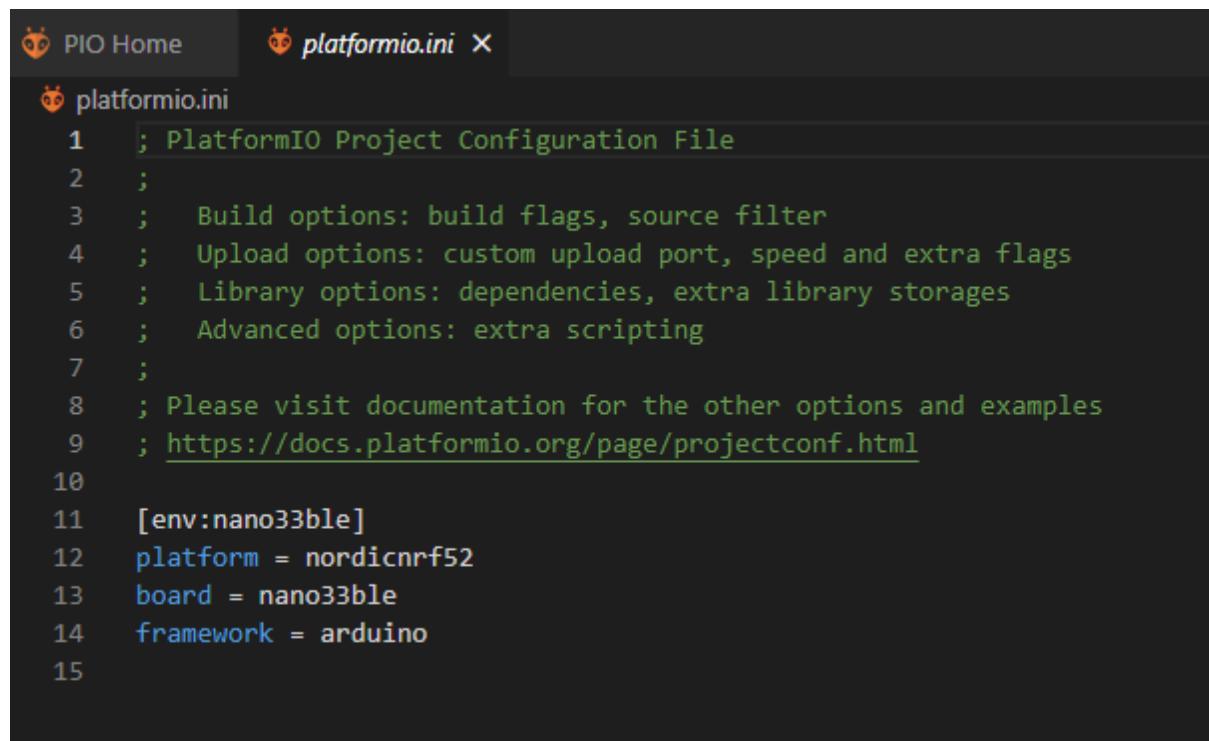


Fig 2: Página de inicio del IDE PlatformIO

Todo proyecto hecho en PlatformIO dispone de un fichero llamado platform.ini. Este fichero se encarga de configurar los parámetros que usará el IDE para tratar el código. En este fichero el usuario tiene que comunicar el entorno en el que trabajará en el proyecto, para ello tiene que definir:

- Con que placa está trabajando (Arduino UNO, Arduino 33 Ble Sense...)
- El framework que usa la placa (Arduino, raspberry...)
- La plataforma que se usará (nordicnrf52...)

Luego de definir estos parámetros es PlatformIO el que se encarga de descargar e instalar todos los controladores necesarios para usar esa placa en el código. En la figura 3 se puede observar un ejemplo de platform.ini.



```
PIO Home    platformio.ini ×

platformio.ini
1 ; PlatformIO Project Configuration File
2 ;
3 ; Build options: build flags, source filter
4 ; Upload options: custom upload port, speed and extra flags
5 ; Library options: dependencies, extra library storages
6 ; Advanced options: extra scripting
7 ;
8 ; Please visit documentation for the other options and examples
9 ; https://docs.platformio.org/page/projectconf.html
10
11 [env:nano33ble]
12 platform = nordicnrf52
13 board = nano33ble
14 framework = arduino
15
```

Fig 3: Código de ejemplo de platform.ini

Una vez platform.ini y el código fuente de la aplicación están acabados, PlatformIO facilita el proceso de compilar y cargar el código en el microcontrolador. Todo este proceso se puede realizar desde la barra horizontal azul que se encuentra en la parte inferior de la pantalla. En esta barra el usuario puede seleccionar entre los diferentes entornos declarados previamente en el fichero platform.ini, para luego o bien definiendo el puerto en el que se encuentra conectado el microcontrolador o dejando que lo detecte automáticamente, cargar el código fuente en el microcontrolador

simplemente presionando el botón de la flecha. La figura 4 muestra la barra horizontal previamente nombrada.

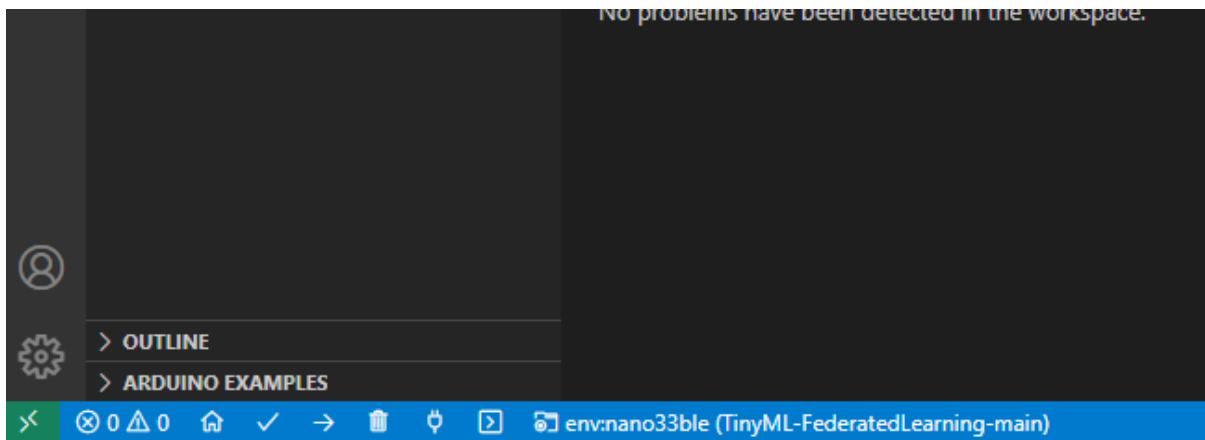


Fig 4: Barra horizontal de PlatformIO para realizar las operaciones

### 5.1.2 Edge Impulse

Edge Impulse consiste en una plataforma Online la cual se fundó en 2019, que permite crear un modelo de aprendizaje automático y entrenarlo en línea para posteriormente poder cargarlo en el dispositivo que se desee y poder usar la red neuronal, entrenada previamente en línea, en el microcontrolador en cualquier lugar sin necesidad de entrenarla de nuevo.

Para ello la plataforma requiere que el usuario rellene 3 bloques necesarios para poder realizar la aplicación. En la figura 5 se puede observar el menú inicial de la plataforma con los tres bloques a llenar.

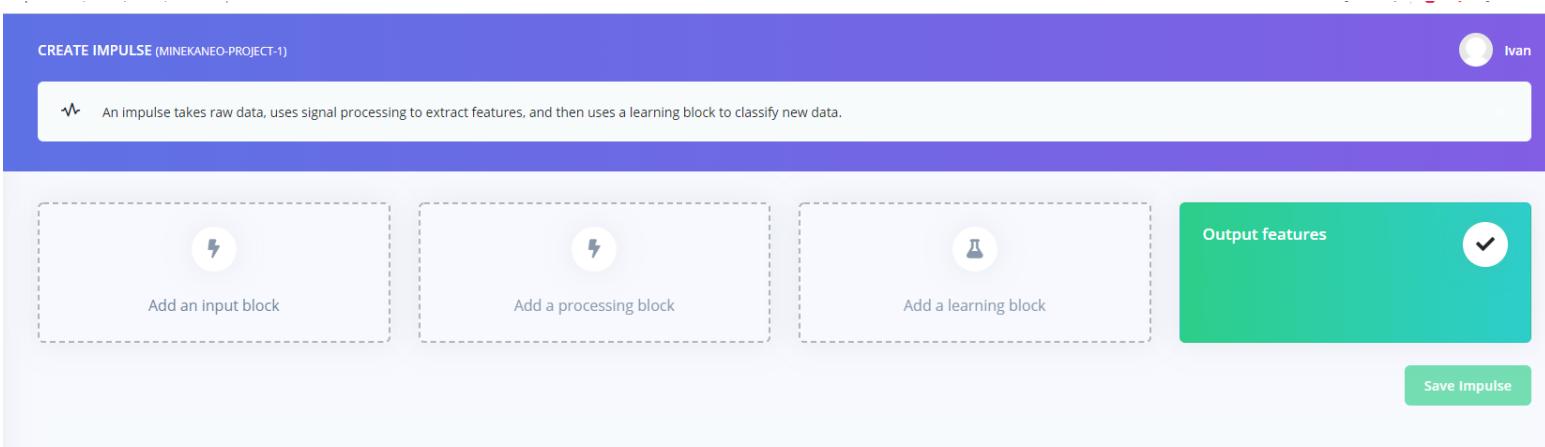


Fig 5: Esquema básico de la plataforma Edge Impulse

**Input Block.** El primer bloque corresponde a qué tipo de datos le vamos a dar como entrada a nuestra red neuronal para poder entrenarse. Para ello disponemos de dos opciones, imágenes como data o valores obtenidos por sensores tales como audio, vibraciones, temperaturas, etc...

**Processing Block.** Una vez hemos decidido que tipo de datos queremos como entrada para nuestra red neuronal, la plataforma nos pide que seleccionemos un bloque de procesado. Este bloque se encarga de recoger la data de entrada que nosotros le proporcionamos a la aplicación y procesarla para obtener las características con las que entrenará la red neuronal. Este bloque vendrá determinado también por el tipo de dato de entrada que queramos para nuestra aplicación, ya que hay diferentes algoritmos con mejores resultados para señales de audio como por ejemplo el MFCC(Mel Frequency Cepstral Coefficients) o el MFE(Mel-filterbank Energy) y otros para imágenes donde simplemente se obtiene el valor entre 0 y 1 de cada píxel de la imagen.

**Learning Block.** El último bloque que la plataforma nos pide seleccionar es el encargado del aprendizaje. Este bloque es el equivalente a la red neuronal de la aplicación y dependiendo del objetivo de nuestra aplicación decidiremos escoger uno u otro.

Una vez se han seleccionado todos los bloques solo resta cargar en la plataforma todos los datos de entrada con los que se desea entrenar la red neuronal indicando a qué clase corresponde cada uno. Posteriormente, la plataforma se encarga de entrenar la red neuronal con toda la data facilitada por el usuario y de esta manera generar la aplicación resultante entrenada por esta data.

Esta plataforma ha sido de gran ayuda para el proyecto a la hora de entender el funcionamiento de las redes neuronales aplicadas a microcontroladores gracias a que gran parte del código usado por esta plataforma se encuentra abierto al público [15]. Por tanto gracias a esto, los dos primeros bloques que la plataforma pide al usuario seleccionar en su página web, el bloque de la selección del tipo de data de entrada y el bloque encargado de procesar esa data, han sido posible de investigarse a fondo el funcionamiento de las diferentes alternativas ofrecidas por la plataforma y poder así aplicarlas al código de la aplicación desarrollada en el proyecto.

## 5.2 Hardware

En el proyecto trabajaremos con los componentes del Arduino Tiny Machine Learning Kit. En este paquete podemos encontrar el Arduino Nano 33 BLE Sense, que será el microcontrolador con el que trataremos en el proyecto, la cámara OV7675 y un Shield que nos permite conectar ambos dispositivos y ofrece diferentes pines digitales y analógicos para utilizar.

### 5.2.1 Arduino Nano 33 BLE Sense

La placa Arduino Nano 33 BLE Sense es la placa de Arduino con un voltaje de 3.3V más pequeña hasta la fecha, con unas dimensiones de 45x18mm.

Esta placa ofrece una gran variedad de sensores a su disposición, con un total de 9 sensores diferentes. Dispone de sensores capaces de capturar las temperaturas, humedad, presión, audio, luz entre otros. La figura 7 muestra el microcontrolador y todos los sensores situados en él.

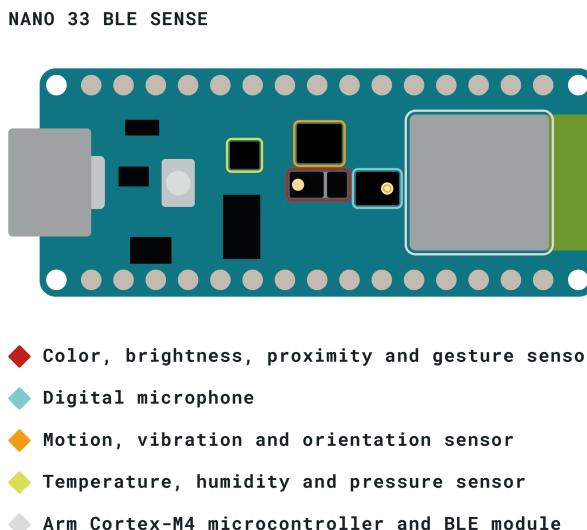


Fig 7: Arduino 33 Ble Sense y sus sensores

Se podría considerar que la Arduino Nano 33 BLE Sense es la continuación de la Arduino UNO, una placa muy extendida en la comunidad Arduino, pero teniendo a su disposición un procesador mucho más potente, el nRF52840, con un CPU 32-bit ARM® Cortex®-M4 trabajando a una frecuencia de 64MHz. Con una memoria de programa de 1MB, 32 veces más grande que la del Arduino UNO y una RAM también 128 veces más grandes, permite ejecutar códigos mucho más elaborados y con una cantidad de variables mayor.

Una de las características más importantes del Arduino Nano 33 BLE Sense es la posibilidad de poder ejecutar programas basados en Inteligencia Artificial usando TinyML. Es por ello que ha sido valorado como la mejor opción para el proyecto.

Por último, el Arduino Nano 33 BLE Sense dispone de la tecnología Bluetooth® y Bluetooth® Low Energy. Característica bastante única en el mundo de los microcontroladores, permitiéndo así poderse conectar a diferentes dispositivos sin la necesidad del uso de cables.

### 5.2.2 Cámara OV7675

Junto al Arduino Nano 33 BLE Sense disponemos de la cámara OV7675[16] para acompañarlo. Esta cámara dispone de una resolución de 640x480px, con YUV422, Raw RGB, ITU656, RGB565 como formatos de salida de las imágenes y una tasa máxima de transferencia de imagen de 30 fps para VGA, 60fps para QVGA y 240fps para QQVGA.

Con estas características la cámara OV7675 es más que capaz para las necesidades del proyecto y dado la compatibilidad entre el Arduino Nano 33 BLE Sense y la cámara, hace que sea la mejor opción para capturar imágenes para nuestro programa. En la figura 8 se puede apreciar la apariencia de la cámara OV7675.



Fig 8: Imagen de la cámara  
OV7675

### 5.2.3 Arduino Tiny Machine Learning Shield

El Arduino Tiny Machine Learning Shield se nos presenta junto al Arduino Nano 33 BLE Sense y la cámara OV7675 en el kit presentado a la venta por Arduino llamado Tiny Machine Learning Kit. Este shield nos permite conectar la cámara al microcontrolador de una manera muy sencilla y sin tener que preocuparnos por las conexiones de cada pin.

Con el Arduino y la cámara conectados al shield, queda a nuestra disposición 6 conectores de 4 pines, entre los que podemos encontrar dos pines digitales y dos pines analógicos. Pines que irán muy bien para poder construir un pequeño circuito de botones ya que la cámara ocupa la gran mayoría de estos pines. Por último el shield también dispone de un botón adicional que corresponde al pin digital 13 del microcontrolador. La figura 9 muestra los componentes del TinyML Kit de arduino sin

conectar al shield. La figura 10 muestra el shield con los componentes conectados en él y en la figura 11 se puede observar el diagrama interno de las conexiones del shield de arduino.

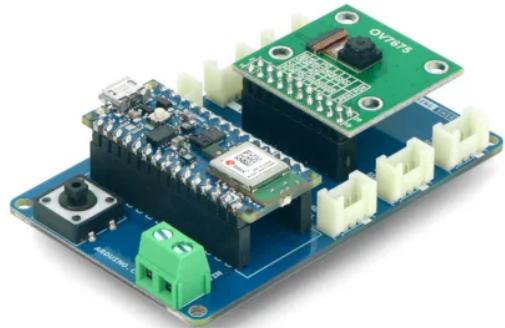
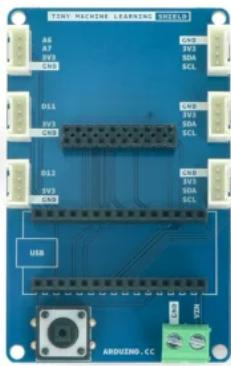


Fig 9: Componentes del TinyML Kit de arduino

Fig 10: Componentes del kit conectados correctamente

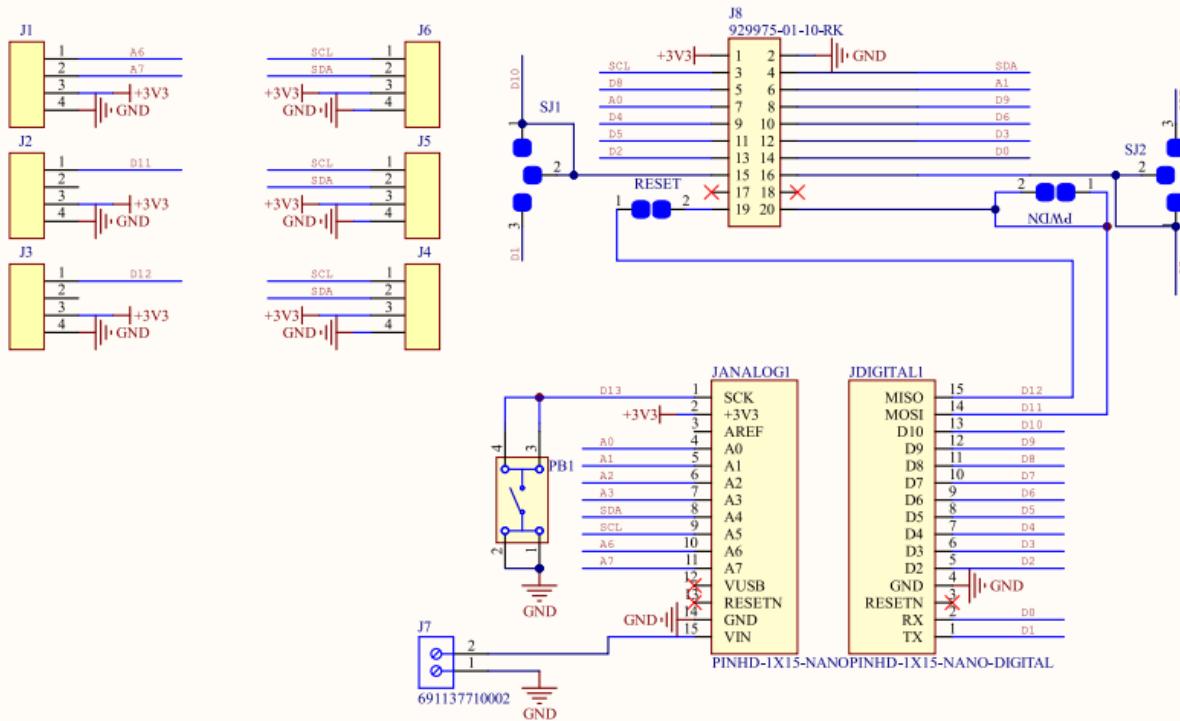


Fig 11: Diagrama interno del Arduino Machine Learning Shield

### 5.3 TinyML

Como ya se ha definido previamente, TinyML es un campo de la inteligencia artificial que se enfoca en la implementación de aprendizaje automático en microcontroladores, dispositivos de menores capacidades que los típicos ordenadores en los que se realizan estas técnicas.

Desde hace ya unos años, el aprendizaje automático se ha ido expandiendo a un ritmo acelerado entre las disciplinas de la inteligencia artificial. Varios modelos de redes neuronales han ido apareciendo, con versiones cada vez más potentes y grandes a las anteriores, siendo la GPT-3 la red neuronal más grande creada hasta la fecha, con más de 175 billones de neuronas artificiales.

Estas redes neuronales a cambio de la gran potencia obtenida por el aumento de tamaño, tienen unos costes de electricidad absurdos (aproximadamente la producción de tres plantas nucleares para poder entrenar la red durante una hora). Obviamente esto ha llevado a una gran parte de la industria de la informática a criticar estos consumos eléctricos desorbitados, pero también ha hecho que el interés por el desarrollo de ideas que permitan una redes neuronales más eficientes en cuanto al coste haya aumentado últimamente. Ideas como algoritmos, representaciones de datos y computación más eficientes han sido el objetivo que han perseguido varios investigadores en los últimos años.

La idea tradicional de IoT (Internet of Things) consistía en enviar la data desde los dispositivos a una nube donde son procesados. Este proceso tenía varios problemas, entre los cuales destacaban más:

- Eficiencia en el coste energético: transmitir datos cuesta mucha más energía que mantenerlos en el dispositivo. Crear dispositivos capaces de procesar su propia data es la mejor solución en cuanto a coste energético.
- Privacidad de los datos: el hecho de enviar los datos de un dispositivo crea la posibilidad de acceder maliciosamente a esos datos desde fuera del sistema, mantener los datos en el mismo dispositivo evita esa posibilidad haciéndolo así más seguro.
- La latencia: los dispositivos que tienen la necesidad de enviar los datos a la nube para recibir posteriormente una respuesta de esta, se encuentran con la dependencia de la velocidad de la red para el tiempo de respuesta. Un dispositivo capaz de generar su propia respuesta no depende de la latencia de la red.

- Almacenamiento: gran parte de la data obtenida por los dispositivos no es de utilidad para el comportamiento de la aplicación, un ejemplo claro serían las cámaras de seguridad las cuales están grabando las 24h pero gran parte de la grabación no se usa. Un dispositivo inteligente capaz de saber cuándo retener información y cuándo no reduce en gran medida la cantidad de almacenamiento necesario.

Todos estos problemas acabaron potenciando la creación de una nueva rama de investigación en la inteligencia artificial que acabó cogiendo el nombre de Tiny Machine Learning (TinyML). Una metodología que tiene como objetivo la obtención de data y tratamiento de esta en el mismo dispositivo, dispositivos normalmente de capacidades reducidas y con costes energéticos menores.

Actualmente en TinyML existen dos grandes áreas de interés:

- Keyword Spotting: La mayoría de la gente ya conoce los típicos ejemplos como “OK Google” o “Hey Siri”. Estos dispositivos están permanentemente escuchando el audio que reciben por el micrófono y reaccionan a ciertas secuencias de palabras que detectan, las cuales han aprendido previamente.
- Visual Wake Words: Estas aplicaciones consisten en la detección de un elemento concreto en las imágenes y son capaces de decir si aparece en ellas o no. Ejemplos de esto serían luces automáticas que se encienden al detectar personas, cámaras que hacen fotos o grabaciones cuando detectan animales o personas, entre otros.

## 5.4 Aprendizaje Federado (Federated Learning)

El aprendizaje federado es una técnica de aprendizaje automático, creada por Google en el año 2017, que consiste en entrenar un algoritmo o modelo global a través de una arquitectura formada por múltiples dispositivos los cuales contienen sus propios datos locales y privados. Este hecho hace que esta técnica haya llamado la atención en los últimos años ya que al no tener la necesidad de enviar los datos locales y privados a una nube donde se procesan, estos evitan posibles problemas relacionados con la privacidad de los datos. Sectores como el médico, defensa, telecomunicaciones, farmacia son ampliamente beneficiados por el uso de esta técnica al poder evitar filtraciones de los datos confidenciales que manejan como podrían ser informes militares, médicos entre otros.

El aprendizaje federado dispone a día de hoy de tres algoritmos diferentes para realizar el proceso de aprendizaje: el algoritmo centralizado, descentralizado y heterogéneo.

- El **algoritmo centralizado** se basa en el uso de un servidor central para coordinar los pasos del algoritmo y los dispositivos. Este servidor es el responsable de la elección de los dispositivos participantes, de recolectar la data necesaria y las actualizaciones pertinentes de cada dispositivo.
- El **algoritmo descentralizado** a diferencia del anterior no usa ningún servidor para recolectar y tratar la data de los dispositivos participantes en el aprendizaje federado. En este entorno se sigue una topología de red, donde serán los propios dispositivos quienes se coordinen entre sí para transmitir la data necesaria para crear un modelo global sin la necesidad de un servidor central. Este algoritmo evita el posible problema de cuello de botella que ocurría en la versión centralizada, donde el servidor tenía que obtener la data de todos los dispositivos para poder proceder con la actualización del modelo global.
- El **algoritmo heterogéneo** ha sido desarrollado recientemente para ser capaz de abordar clientes heterogéneos equipados con capacidades de computación y comunicación muy diferentes. Esta técnica permite el entrenamiento de modelos locales heterogéneos con complejidades de cálculo que varían dinámicamente, sin dejar de generar un único modelo global.

Otro factor importante a tener en cuenta a la hora de realizar aprendizaje federado es el algoritmo con el que crearemos el modelo global a partir de los modelos locales. Actualmente existen dos muy populares para este trabajo: el “Federated Stochastic Gradient Descent” (FedSGD) y el “Federated Averaging” (FedAVG).

- En el **FedSGD**, los dispositivos conectados a la red de aprendizaje federado envían al servidor el error generado por la data de entrada que han recibido en cada entrenamiento. El servidor tras recibir el error de los dispositivos que han realizado un entrenamiento realizará la media y entrenará el modelo global con el nuevo valor obtenido.
- Por lo contrario, en el **FedAVG**, los dispositivos entran sus modelos locales con el error generado por la data de entrada, actualizando los parámetros de sus modelos. Luego, los dispositivos envían al servidor los parámetros de sus

modelos entrenados y este se encarga de realizar la media entre todos ellos para crear un nuevo modelo global a partir de todos los locales.

La mayor diferencia entre los dos métodos se refleja en la opción que ofrece FedAVG de poder entrenar los modelos locales varias veces antes de tener que ser enviados al servidor para crear el modelo global, cosa que FedSGD no permite ya que solo puede enviar el error generado por un entrenamiento.

El aprendizaje federado es una herramienta con muchas ventajas a la hora de mantener la privacidad de la data e incrementar la velocidad a la que se puede entrenar un modelo pero a cambio de estas ventajas aparece una desventaja que tiene que ser controlada para el correcto funcionamiento del algoritmo. Esta desventaja cae en la comunicación entre el servidor y los clientes. Es importante controlar que los paquetes de data enviados desde los dispositivos lleguen en el correcto orden al servidor y sin pérdidas para poder realizar correctamente las actualizaciones del modelo global, para ello, protocolos de comunicación basados en el ACK(acknowledgment)[17] son necesarios para asegurar estos criterios. A la vez que, se tiene que reducir al máximo el tiempo de transmisión de los datos ya que se acabará convirtiendo en el cuello de botella de la red de aprendizaje federado.

## 6. Desarrollo de la aplicación

### 6.1 Objetivos

Como se ha comentado anteriormente en el campo de TinyML las dos áreas en las que se está enfocando gran parte de las aplicaciones actuales son el Keyword Spoting y el Visual Wake Words. El comportamiento de la aplicación consiste en un mezcla de estas dos variantes, a lo que se le podría llamar Image Spoting.

La aplicación será entrenada en el dispositivo con diferentes imágenes correspondientes a una de las clases, **el objetivo es que tras el entrenamiento del modelo la aplicación sea capaz de diferenciar las imágenes entre las clases entrenadas.**

Para comprobar que se cumpla el objetivo se observará el gráfico generado por el error producido al entrenar cada imagen para determinar si una prueba es satisfactoria o no, satisfactoria en caso de que el error converge hacia el 0 o cerca de él y no satisfactoria en caso contrario.

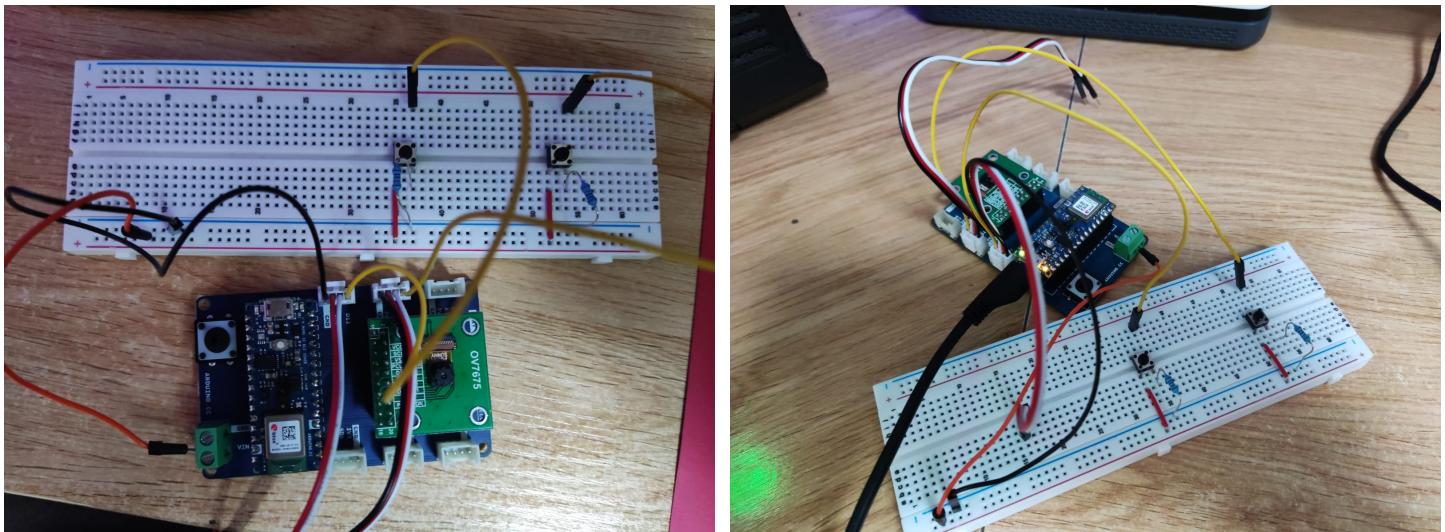
### 6.2 Preparación hardware

El hardware necesario para la aplicación es sencillo de preparar. Necesitaremos los componentes integrados en el Tiny Machine Learning Kit de Arduino, el cual contiene un Arduino Nano 33 BLE Sense, un shield y la cámara OV7675. Tanto el microcontrolador como la cámara usarán sus respectivas posiciones en el shield.

En un protoboard aparte tendremos que preparar dos botones necesarios para el funcionamiento de la aplicación. Para ello conectaremos dos cables de 4 pines en los encajes de 4 pines correspondientes al pin digital 11 y 12 en el shield, que corresponden a los dos encajes de abajo a la izquierda del shield sosteniendo el microcontrolador con la cámara en la zona superior.

Estos cables nos permitirán acceder individualmente a los pines digitales que necesitaremos para el uso de dos botones. Para ello conectaremos un botón a cada salida de los cables, una resistencia en una de las salidas del botón que se irá a tierra y la otra salida al VIN del shield.

En las figuras 12 y 13 se pueden apreciar cómo se conectan los diferentes elementos necesarios para que la aplicación funcione correctamente.



*Fig 12 y 13: ejemplos de las conexiones hardware para la aplicación*

## 6.3 Análisis código inicial

El proyecto tiene como punto de partida la aplicación de aprendizaje federado realizada por Marc Monfort, por tanto, para poder continuar adecuadamente desde este punto era necesario comprender el funcionamiento de dicha aplicación. En esta sección se analizará la estructura de la aplicación inicial y de su funcionamiento.

La aplicación se divide en dos códigos principales, un código es el comportamiento de los microcontroladores, estos microcontroladores tienen como objetivo entrenar en el propio dispositivo capturando señales de audio, asociándolas a cada clase, para luego enviar los modelos entrenados al servidor y recibir el modelo fusionado de los demás. El otro código corresponde a las funciones del servidor, el servidor se encarga de conectar con cada microcontrolador para obtener sus modelos entrenados, una vez tiene todos los modelos locales, este se encarga de juntarlos en un mismo modelo global y distribuir este nuevo modelo global a todos los microcontroladores que lo reemplazarán como su modelo local.

### 6.3.1 Microcontrolador

Los microcontroladores son los encargados de recolectar los datos que se usarán en el entrenamiento y de entrenar sus propios modelos locales para luego enviarlos al servidor y recibir el modelo global unificado de todos los microcontroladores conectados al servidor.

Para ello los microcontroladores siguen un patrón sencillo. Primeramente inicializan todos los elementos necesarios que necesitará el código para funcionar, como LEDs y botones como se puede observar en la figura 14. Luego de la fase de inicializar los componentes llama la función `ini_network_model()` de la figura 15, la cual se queda a la espera de recibir y guardar el modelo inicial desde el servidor. Una vez recibe este modelo, el microcontrolador entra el main loop que ejecutará hasta la finalización de la aplicación.

```

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(9600);

    // Start button_configuration
    pinMode(button_1, INPUT);
    pinMode(button_2, INPUT);
    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(LED_R, OUTPUT);
    pinMode(LED_G, OUTPUT);
    pinMode(LED_B, OUTPUT);
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_G, HIGH);
    digitalWrite(LED_B, HIGH);

    digitalWrite(LED_BUILTIN, HIGH);      // ON

    //init Network
    // myNetwork.initWeights();
    init_network_model();
    digitalWrite(LED_BUILTIN, LOW);      // OFF

    num_epochs = 0;
}

```

Fig 14: función `setup()` del código

```

void init_network_model() {
    char signal;
    do {
        signal = Serial.read();
        Serial.println("Waiting for new model...");
    } while(signal != 's'); // s -> START

    Serial.println("start");

    char* myHiddenWeights = (char*) myNetwork.get_HiddenWeights();
    for (uint16_t i = 0; i < (InputNodes+1) * HiddenNodes; ++i) {
        Serial.write('\n');
        while(Serial.available() < 4) {}
        for (int n = 0; n < 4; n++) {
            myHiddenWeights[i*4+n] = Serial.read();
            // delay(1);
        }
    }

    char* myOutputWeights = (char*) myNetwork.get_OutputWeights();
    for (uint16_t i = 0; i < (HiddenNodes+1) * OutputNodes; ++i) {
        Serial.write('\n');
        while(Serial.available() < 4) {}
        for (int n = 0; n < 4; n++) {
            myOutputWeights[i*4+n] = Serial.read();
            // delay(1);
        }
    }
}

```

Fig 15: función `init_network_model()` del código

El main loop de los microcontroladores comprueba activamente si alguno de los botones ha sido pulsado, en ese caso hace el tratamiento especificado para dicho botón. En el caso en que ningún botón haya sido pulsado comprueba si el servidor ha intentado comunicarse con él para realizar el Federated Learning. Si en el puerto Serial lee el byte ‘>’ empezará a realizar el ‘Handshake’ con el servidor y después de asegurar la comunicación entre los dos dispositivos procederá a enviar el modelo actual en ese microcontrolador como se observa en la figura 16. Luego se quedará a la espera del modelo que tiene que recibir desde el servidor, indicando que está listo con el byte ‘n’ y en cuanto detecte bytes en el puerto Serial los empezará a leer y guardar como su nuevo modelo, proceso reflejado en la figura 17.

```

else {
    if (Serial.read() == '>') { // s -> FEDERATED LEARNING
        //*****
        * Federate Learning
        *****/
        Serial.write('<');
        digitalWrite(LED_BUILTIN, HIGH);      // ON
        delay(1000);
        if (Serial.read() == 's') {
            Serial.println("start");
            Serial.println(num_epochs);
            num_epochs = 0;
            // Serial.println(HiddenNodes);

            //*****
            * Sending model
            *****/
            // Sending hidden layer
            char* myHiddenWeights = (char*) myNetwork.get_HiddenWeights();
            for (uint16_t i = 0; i < (InputNodes+1) * HiddenNodes; ++i) {
                Serial.write(myHiddenWeights+i*sizeof(float), sizeof(float));
            }

            // Sending output layer
            char* myOutputWeights = (char*) myNetwork.get_OutputWeights();
            for (uint16_t i = 0; i < (HiddenNodes+1) * OutputNodes; ++i) {
                Serial.write(myOutputWeights+i*sizeof(float), sizeof(float));
            }
        }
    }
}

//*****
* Receiving model
*****/
// Receiving hidden layer
for (uint16_t i = 0; i < (InputNodes+1) * HiddenNodes; ++i) {
    Serial.write('\n');
    while(Serial.available() < 4) {}
    for (int n = 0; n < 4; n++) {
        myHiddenWeights[i*4+n] = Serial.read();
        // delay(1);
    }
}

// Receiving output layer
for (uint16_t i = 0; i < (HiddenNodes+1) * OutputNodes; ++i) {
    Serial.write('\n');
    while(Serial.available() < 4) {}
    for (int n = 0; n < 4; n++) {
        myOutputWeights[i*4+n] = Serial.read();
        // delay(1);
    }
}

//*****
* END - Federate Learning
*****/

```

Fig 16 y 17: fase encargada del aprendizaje federado del código del microcontrolador

En el caso que se haya pulsado un botón la aplicación graba con el micrófono una fracción de un segundo. El audio grabado durante este segundo será procesado por la clase signal con el fin de extraer las características que se usarán en el entrenamiento de la red neuronal.

Para extraer las características se usan los Coeficientes Cepstrales en las Frecuencias de Mel (o **Mel Frequency Cepstral Coefficients** MFCC en inglés). Estos coeficientes son los encargados de extraer de la señal de audio los fragmentos con información valiosa y obviar los que no aportan información útil como podría ser el ruido de fondo, tono, volumen, emociones.

Para obtener estos coeficientes se siguen 5 pasos:

- Primeramente dividimos la señal en tramos pequeños
- A cada tramo le aplicamos la Transformada de Fourier (transformada que mediante la transformación de una función matemática a otra, obtiene en el dominio una representación de la frecuencia) para obtener la potencia espectral de la señal.
- Se aplican los filtros asociados a la escala Mel a los espectros obtenidos anteriormente y se suman todos las energía obtenidas.
- Se obtiene el logaritmo de todas las energías obtenidas de cada frecuencia Mel
- Aplicar la transformada de coseno discreta a los logaritmos

Una vez ya se han obtenido las características necesarias de las señales de audio la aplicación procede al entrenamiento de la red o directamente al reconocimiento en caso de que se haya seleccionado la opción sin entrenamiento.

Para el entrenamiento se realiza la técnica de backward sobre los nodos de la red neuronal con las características obtenidas. Esta técnica propaga primeramente la señal por la red neuronal para obtener una salida, la salida que se obtiene de propagar la señal con la red actual es comparada con la señal deseada y se obtiene un error en la diferencia entre las dos. Este error es entonces propagado hacia atrás pasando por todas las neuronas, de esta manera cada neurona recibe la contribución relativa que ha aportado al error.

Finalmente se realiza la técnica de forward, donde la señal de audio se propaga desde el inicio de la red hasta el final y se obtiene una señal de salida donde se puede observar a qué clase de las creadas esta señal tiene más similitud.

### 6.3.2 Servidor

La aplicación inicial estaba basada en el algoritmo centralizado para ejecutar el aprendizaje federado. Como se ha comentando anteriormente este algoritmo consiste en tener un dispositivo actuando como servidor encargado de conectarse a todos los clientes de los cuales obtiene los parámetros necesarios de sus modelos locales para poder realizar el modelo global a partir de estos.

Para la creación del modelo global de la red de aprendizaje federado en la aplicación inicial se usaba el algoritmo FedAVG, donde cada cliente tiene como objetivo enviar los parámetros de sus modelos entrenados localmente y el servidor realiza una media entre los valores que obtiene para crear el modelo global. De esta manera se permitía a la aplicación entrenar más de una vez los modelos locales antes de juntarlos en el global.

Lo primero que hace la aplicación del servidor es preguntar al usuario por el número de dispositivos con el que se trata y por cada dispositivo pide especificar el puerto en el que se encuentra conectado, ya que será necesario para poder iniciar una conexión serial con los microcontroladores y poder transmitir los datos de servidor a microcontrolador y viceversa. La figura 18 muestra el código de este proceso.

```

# def main():

    num_devices = read_number("Number of devices: ")

    devices = [read_port(f"Port device_{i+1}: ") for i in range(num_devices)]

    size_hidden_layer = (650+1)*16
    size_output_layer = (16+1)*3

    np.random.seed(12345)
    hidden_layer = np.random.uniform(-0.5, 0.5, (650+1)*16).astype('float32')
    output_layer = np.random.uniform(-0.5, 0.5, (16+1)*3).astype('float32')

    for d in devices:
        init_network(hidden_layer, output_layer, d)

    devices_connected = devices

```

Fig 18: fase inicial del código del servidor

Una vez estos parámetros han sido especificados por el usuario el servidor envía a cada microcontrolador una red neuronal inicial aleatoria mediante la función `init_network()`. Esta función recibe como parámetros la capa hidden y la capa output y se las transmite al dispositivo *arduino*. Para ello anuncia al microcontrolador con un byte ‘s’ que va a empezar a transmitir datos y espera antes de cada mensaje la confirmación del microcontrolador de que está listo para recibir. En la figura 19 se puede observar la función `init_network()`.

```

def init_network(hidden_layer, output_layer, arduino):
    arduino.reset_input_buffer()
    arduino.write(b's')
    print_until_keyword('start', arduino)
    for i in range(len(hidden_layer)):
        arduino.read() # wait until confirmation of float received
        float_num = hidden_layer[i]
        data = struct.pack('f', float_num)
        arduino.write(data)

    for i in range(len(output_layer)):
        arduino.read() # wait until confirmation of float received
        float_num = output_layer[i]
        data = struct.pack('f', float_num)
        arduino.write(data)

```

Fig 19: función `init_network()` del código del servidor

Una vez esta primera fase de setup ha acabado, la aplicación entra en el main loop observable en la figura 20. Este loop se encarga cada 10 segundos de empezar el aprendizaje federado, para ello, intenta establecer una conexión con cada microcontrolador especificado al principio para recibir su modelo entrenado.

```
#####
# Infinite loop
#####
while True:

    max_time = 10
    # max_time = 100000000 # Modified to graph

    for d in devices:
        d.timeout = 0
        # d.timeout = None # Modified to graph

    countdown_print = [20,15,10, 5, 4, 3, 2, 1]
    ini_time = time.time()

    while True:
        for d in devices:
            msg = d.readline().decode()
            if (len(msg) > 0):
                print(f'{d.port}:', msg, end=' ')
                # Modified to graph
                # if msg[:-2] == 'graph':
                #     n_epoch = int(d.readline()[:-2])
                #     n_error = d.read(4)
                #     [n_error] = struct.unpack('f', n_error)
                #     n_button = int(d.readline()[:-2])
                #     graph.append([n_epoch,n_error, n_button])

        countdown = max_time - int(time.time() - ini_time)
        if countdown <= 0:
            break
        elif countdown in countdown_print:
            print(f'Starting FL in {countdown} seconds')
            countdown_print.remove(countdown)

    print('Starting Federated Learning')
    old_devices_connected = devices_connected
    devices_connected = []
    devices_hidden_layer = np.empty((0,size_hidden_layer), dtype='float32')
    devices_output_layer = np.empty((0,size_output_layer), dtype='float32')
    devices_num_epochs = []
```

Fig 20: Bucle main del código del servidor

El Federated Learning consta de tres fases, una primera fase donde recibe los modelos de cada microcontrolador, una segunda donde los procesa todos y los junta en un mismo modelo y una última fase donde envía este nuevo modelo a todos los microcontroladores conectados.

En la primera fase la aplicación empieza un tratamiento automático de negociación llamado ‘handshake’ con los dispositivos conectados. Este tratamiento consta de tres mensajes, primeramente el servidor envía un mensaje llamado SYN al microcontrolador, en el caso de la aplicación este SYN corresponde al byte ‘>’, si el microcontrolador recibe este byte es entonces responsable de comunicar al servidor que lo ha recibido para ello tiene que enviar un mensaje llamado SYN ACK al servidor, en el caso de la aplicación este SYN ACK corresponde al byte ‘<’, y por último cuando el servidor recibe el SYN ACK es este el encargado de comunicar al microcontrolador que ha recibido su mensaje y por tanto la negociación ha sido un éxito.

Una vez este proceso ha finalizado es correcto asumir que no habrá interferencia en los datos desde otros dispositivos y por tanto se puede recibir los datos del microcontrolador con el que nos hemos conectado sin problema. Es entonces cuando la aplicación empieza a recibir el modelo entrenado por el dispositivo y los va acumulando en las variables `devices_hidden_layer` (la capa hidden de la red) y `device_output_layer` (la capa output de la red) como se puede observar en la figura 21.

```
#####
# Receiving models
#####
for d in devices:
    d.reset_input_buffer()
    d.reset_output_buffer()
    d.timeout = 5

    print(f'Starting connection to {d.port} ...') # Handshake
    d.write(b'>') # Python --> SYN --> Arduino
    if d.read() == b'<': # Python <-- SYN ACK <-- Arduino
        d.write(b's') # Python --> ACK --> Arduino

    print('Connection accepted.')
    devices_connected.append(d)
    devices_hidden_layer = np.vstack((devices_hidden_layer, np.empty(size_hidden_layer)))
    devices_output_layer = np.vstack((devices_output_layer, np.empty(size_output_layer)))
    d.timeout = None

    print_until_keyword('start', d)
    devices_num_epochs.append(int(d.readline()[:-2]))

    print(f'Receiving model from {d.port} ...')
    ini_time = time.time()

    for i in range((650+1)*16): # hidden layer
        data = d.read(4)
        [float_num] = struct.unpack('f', data)
        devices_hidden_layer[-1][i] = float_num

    for i in range((16+1)*3): # output layer
        data = d.read(4)
        [float_num] = struct.unpack('f', data)
        devices_output_layer[-1][i] = float_num

    print(f'Model received from {d.port} ({time.time()-ini_time} seconds)')
```

```

# if it was not connected before, we dont use the devices' model
if not d in old_devices_connected:
    devices_num_epochs[-1] = 0
    print(f'Model not used. The device {d.port} has an outdated model')

else:
    print(f'Connection timed out. Skipping {d.port}.')

```

*Fig 21: Fase de recepción de los modelos locales del código del servidor*

Una vez la aplicación ha obtenido todos los posibles nuevos modelos de los dispositivos, esta se encarga de procesarlos. Esta acción es muy simple y se resume en hacer una media de los modelos obtenidos (FedAVG) y guardarlos en las dos variables previamente nombradas. En el cálculo de esta media es posible otorgar diferentes rangos de importancia a los dispositivos pero inicialmente se espera que todos los dispositivos tengan el mismo rango de importancia y por eso hace una media estándar entre ellos. La figura 22 muestra el código de tal proceso.

```

#####
# Processing models
#####

# if sum == 0, any device made any epoch
if sum(devices_num_epochs) > 0:
    hidden_layer = np.average(devices_hidden_layer, axis=0, weights=devices_num_epochs)
    output_layer = np.average(devices_output_layer, axis=0, weights=devices_num_epochs)
    # We can use , weights to change the importance of each device
    # example weights = [1, 0.5] -> giving more importance to the first device...
    # is like percentage of importance : sum(a * weights) / sum(weights)

```

*Fig 22: Fase de procesamiento de los modelos recibidos del código del servidor*

Por último, una vez ya se han recibido los modelos y procesado, es momento de enviar el nuevo modelo global resultado de la unión de los otros a todos los dispositivos conectados al servidor. Para el envío de estos modelos sigue el mismo patrón que la función de inicialización, el servidor espera a que el microcontrolador le confirme que está listo para recibir los datos y le va enviando en paquetes pequeños tanto la capa hidden como la output.

## 6.4 Red neuronal

Como se ha podido observar en el análisis de la aplicación de “keyword Spotting” la red neuronal que se usó consistía en una red neuronal prealimentada (feed-forward en inglés). Estas redes son las más simples y por tanto las más pequeñas en cuanto a tamaño. Su trabajo consiste en mover la información recibida en una única dirección: hacia adelante. De esta información se puede obtener un error al obtener la salida y entrenar la red con este error, tal y como se ha podido comprobar en el código anterior.

La red neuronal prealimentada no es óptima para el reconocimiento de imágenes pero dado a su tamaño, cantidad de memoria disponible en los microcontroladores y los conocimientos obtenidos, se ha decidido reutilizar esta red neuronal para la aplicación de reconocimiento de imágenes.

La red neuronal viene originalmente de una plantilla realizada por Ralph Heymsfeld en la placa Arduino UNO [18] y fue modificada para la aplicación de “keyword Spotting” por Marc Monfort.

La red está compuesta por una capa de entrada de 650 nodos, una capa oculta de 16 nodos y una capa de salida de 3 nodos (uno por cada clase), diseño apreciable en la figura 23. Como función de activación de la red neuronal se usa un función sigmoide[19]. Esta configuración genera un total de 10467 parámetros, convirtiéndolos en 41868 bytes al usar un float 4 byte para representarlos. Cantidad de memoria capaz de almacenarse sin problemas en la SRAM de la placa Arduino Nano 33 la cual dispone de un total de 256 kB.

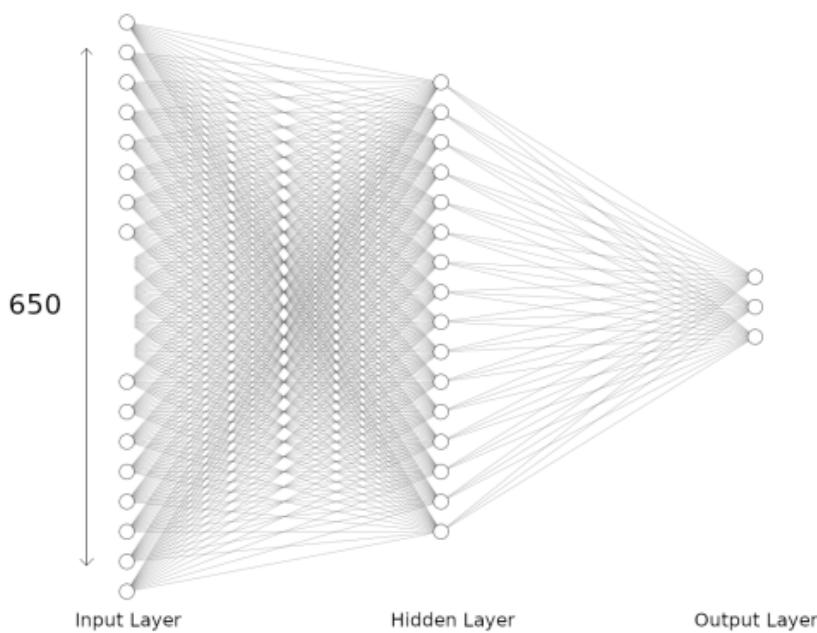


Fig 23: diagrama de la red neuronal

Como se ha explicado previamente en el análisis, este modelo se entrena con las características (features en inglés) obtenidas, en nuestro caso serán características obtenidas de las imágenes tomadas como entrada para la red neuronal. Estas características fluctúan a través de la red generando el porcentaje de coincidencia con cada una de las tres clases, luego se usará ese porcentaje con el conocido al seleccionar una clase y el error se enviará hacia atrás para que cada neurona aprenda cuán errónea ha sido su contribución y así entrenarse.

En orden de optimizar el modelo durante las pruebas se tocarán dos parámetros importantes para definir el comportamiento de la red. La velocidad de aprendizaje (learning rate en inglés) y el momentum.

- La velocidad de aprendizaje es el encargado de definir en cuanto se entrena el modelo por cada error generado por las imágenes. Esta velocidad por defecto es de 0.3, con las pruebas se investigará si una velocidad menor de aprendizaje puede ser mejor, teniendo en cuenta la cantidad de data necesitada para entrenar (cuanto menor la velocidad más se necesitará) y la estabilidad de los resultados (cuanto mayor la velocidad menos estables serán).
- El momentum es el responsable de definir cuánto porcentaje de la data anterior se usa para la dirección de la data que se está entrenando en ese momento. Por defecto el valor del momentum es de 0.9, significando que se añade un 90% de la dirección anterior a la actual.

## 6.5 Modificaciones audio a imagen

Las principales modificaciones realizadas han sido en el código de los microcontroladores, ya que el servidor lo que recibe es un modelo ya entrenado y por tanto el proceso que se haya seguido para entrenarlo no afecta a la estructura de datos que recibirá. No obstante se le ha añadido una funcionalidad al servidor para disponer de la posibilidad de visualizar la imagen realizada. Para ello, cuando se establece la conexión con un microcontrolador, además de recibir el modelo, también recibe los bytes que definen la última imagen realizada por el microcontrolador con el fin de poder visualizar la imagen que hemos hecho con la cámara. La figura 24 muestra la sección del código encargada de esta función.

```

if d.read() == b'i':
    print('Receiving photo.')
    image = np.zeros((160*120,3), dtype=int)

    # Loop through all of the pixels and form the image
    for i in range(176*144):
        #Read 16-bit pixel

        data = d.read(2)
        pixel = struct.unpack('>h', data)[0]

        #Convert RGB565 to RGB 24-bit
        r = ((pixel >> 11) & 0x1f) << 3
        g = ((pixel >> 5) & 0x3f) << 2
        b = ((pixel >> 0) & 0x1f) << 3
        image[i] = [r,g,b]

    image = np.reshape(image,(120, 160,3)) #QQVGA resolution

    # Show the image
    plt.imshow(image)
    plt.show()

```

Fig 24: código para recibir la imagen en el servidor

Esta sección del código se encarga de leer los bytes que conforman la imagen de dos en dos, ya que un píxel de la cámara está definido por dos bytes, y luego convierte estos píxeles desde el formato RGB565 al RGB888 de 24-bit aplicando unas conversiones. Una vez tenemos la imagen en RGB888 24 bit, organizamos los píxeles en la misma resolución que usa la cámara de 160x120 y ya podemos visualizar correctamente la imagen con la función show() de la librería matplotlib de python.

No obstante, esta funcionalidad permanece desactivada durante la fase de entrenamiento de la aplicación, ya que al mostrar la imagen con el plot había que cerrar la ventana para que el programa siguiese funcionando, esto hacía que la fase de entrenamiento de la aplicación durará mucho más tiempo del necesario. Por tanto, esta posibilidad de visualizar la imagen sirve para identificar la distancia correcta y el enfoque de la cámara antes de realizar las imágenes para el entrenamiento del modelo.

En cuanto al microcontrolador, se han realizado varios cambios con el fin de cambiar el comportamiento de la aplicación. Para empezar, en la función de setup donde se inicializan los botones y los LEDs, se ha añadido la inicialización de la cámara. Esta se inicializa en la resolución QQVGA(160\*120 píxeles), la más pequeña posible entre

las opciones proporcionadas por la librería encargada de gestionar la cámara, y RGB565 como formato para representar los píxeles tomados por la cámara, observable en la figura 25.

```
bool ei_camera_init(void) {
    if (is_initialised) return true;

    if (!Cam.begin(QQVGA, RGB565, 1)) { // VGA downsampled to QQVGA (OV7675)
        ei_printf("ERR: Failed to initialize camera\r\n");
        return false;
    }
    is_initialised = true;

    return true;
}
```

Fig 25: función encargada de inicializar la cámara del código del microcontrolador

Respecto al main loop del microcontrolador se han tenido que realizar cambios más notorios. En la aplicación inicial, se usaban un total de 4 botones diferentes para el control del comportamiento del microcontrolador. Tres de estos botones corresponden a cada una de las tres clases que entrenaban los microcontroladores, cuando el usuario pulsaba uno de esos botones, el microcontrolador grababa los próximos segundos de audio y usaba los datos obtenidos para entrenar la clase seleccionada. El último botón era el encargado de decidir a qué clase correspondía la muestra que obtenía, indicando con los LEDs la clase con más porcentaje de coincidencia con la muestra.

Dado que la cámara usa gran parte de los pines digitales de los que dispone el microcontrolador, hizo que el uso de botones fuera bastante más limitado. Dejando solo dos pines digitales libres y un botón que venía integrado en el shield. Por tanto, con el fin de poder mantener una variedad de clases se cambió el comportamiento de los botones ligeramente. En esta nueva versión el microcontrolador consta solo de dos botones. Uno de los botones se encarga de cambiar el modo en el que nos encontramos y el otro se encarga de capturar la imagen, obtener las características necesarias de la imagen y el entrenamiento de la red neuronal correspondiente. La aplicación empieza con 4 modos, por tal de mantener el mismo número de clases que la aplicación inicial. Del modo 1 al modo 3, corresponden a las tres clases que se pueden entrenar en el modelo, el último modo sirve para que el microcontrolador decida a qué clase corresponde la muestra sin entrenar el modelo. En la figura 26 se puede apreciar el nuevo comportamiento de los botones en el código.

```

void loop()
{
    //Conditional to avoid multiple lectures of buttons
    if (digitalRead(button_1) == LOW && digitalRead(button_2) == LOW && button_pressed == true) {
        button_pressed = false;
        Serial.println("Buttons ready");
    }
    //Change mode
    if (digitalRead(button_1) == HIGH && button_pressed == false ) {
        digitalWrite(LED_R, LOW); // ON
        modo = modo + 1;
        if (modo >= 5) modo = 1;
        Serial.print("Modo:");
        Serial.println(modo);
        button_pressed = true;
    }
    //Foto button
    else if (digitalRead(button_2) == HIGH && button_pressed == false) {
        digitalWrite(LED_G, LOW); // ON
        button_pressed = true;
        foto = true;
        envfoto = true;
    }
}

```

Fig 26: comportamiento de los botones de la aplicación del microcontrolador

En cuanto al código del entrenamiento del modelo, al obtener una data de entrada distinta siendo esta una imagen en vez de una señal de audio, los Coeficientes Cepstrales en las Frecuencias de Mel no nos sirven para obtener características relevantes de las imágenes. Estos coeficientes son exclusivos para señales de audio y si se aplicaran a imágenes las características que estaría obteniendo serían impredecibles.

Para ello, se ha usado una función diferente para obtener las características. Esta función transforma todos los píxeles de la imagen en floats con un valor entre 0 y 1. La entrada de esta función consiste en una imagen con una resolución de 96x96 y un formato de RGB888, pero la imagen más pequeña que puede capturar la cámara del arduino tiene una resolución de 160x120 y el formato en el que se representa es el RGB565. Por tanto, es necesario redimensionar la imagen capturada y cambiar el formato en el que se representa para que pueda ser usada en la función encargada de obtener las características que se usarán para el entrenamiento del modelo.

Para poder redimensionar la imagen correctamente, primeramente se calcula el factor en el que habrá que redimensionar la imagen con la función calculate\_resize\_dimensions(), esta función mediante unos cálculos simples obtiene el factor de redimensión necesario a aplicar a la imagen. A continuación, se procede a capturar la imagen por la cámara en ese momento y mediante la función cropImage() se recorta y adapta a la resolución de 96x96, con la ayuda del factor calculado previamente, necesaria para obtener las características con las que se entrenará la red neuronal. Por último, usando las mismas conversiones usadas en el servidor para

mostrar la imagen, se convertirá la imagen en formato RGB565 representado como 96x96x2 bytes (ya que cada píxel se representa por dos bytes) a los tres valores rgb equivalentes que componen cada píxel de la imagen en el formato RGB888. Estos valores rgb son los que luego usaremos como parámetros en la función para obtener las características que se usarán para entrenar la red neuronal. Las figuras 27 y 28 contienen el código encargado de realizar todo este proceso.

```
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf)
{
    if (!is_initialised) {
        ei_printf("ERR: Camera is not initialized\r\n");
        return false;
    }

    if (!out_buf) {
        ei_printf("ERR: invalid parameters\r\n");
        return false;
    }

    // choose resize dimensions
    int res = calculate_resize_dimensions(img_width, img_height, &resize_col_sz, &resize_row_sz, &do_resize);
    if (res) {
        ei_printf("ERR: Failed to calculate resize dimensions (%d)\r\n", res);
        return false;
    }

    if ((img_width != resize_col_sz)
        || (img_height != resize_row_sz)) {
        do_crop = true;
    }

    Cam.readFrame(out_buf); // captures image and resizes

    if (do_crop) {
        uint32_t crop_col_sz;
        uint32_t crop_row_sz;
        uint32_t crop_col_start;
        uint32_t crop_row_start;
        crop_row_start = (resize_row_sz - img_height) / 2;
        crop_col_start = (resize_col_sz - img_width) / 2;
        crop_col_sz = img_width;
        crop_row_sz = img_height;

        //ei_printf("crop cols: %d, rows: %d\r\n", crop_col_sz,crop_row_sz);
        cropImage(resize_col_sz, resize_row_sz,
                  out_buf,
                  crop_col_start, crop_row_start,
                  crop_col_sz, crop_row_sz,
                  out_buf,
                  16);
    }
}
```

Fig 27: función encargada de realizar la captura de la imagen y adaptar la resolución

```
int ei_camera_cutout_get_data(size_t offset, size_t length, float *out_ptr) {
    size_t pixel_ix = offset * 2;
    size_t bytes_left = length;
    size_t out_ptr_ix = 0;

    // read byte for byte
    while (bytes_left != 0) {
        // grab the value and convert to r/g/b
        uint16_t pixel = (ei_camera_capture_out[pixel_ix] << 8) | ei_camera_capture_out[pixel_ix+1];
        uint8_t r, g, b;
        r = ((pixel >> 11) & 0x1f) << 3;
        g = ((pixel >> 5) & 0x3f) << 2;
        b = (pixel & 0x1f) << 3;

        // then convert to out_ptr format
        float pixel_f = (r << 16) + (g << 8) + b;
        out_ptr[out_ptr_ix] = pixel_f;

        // and go to the next pixel
        out_ptr_ix++;
        pixel_ix+=2;
        bytes_left--;
    }

    // and done!
    return 0;
}
```

Fig 28: función encarga de cambiar el formato de la imagen de RGB565 a RGB888

Se puede consultar el código final de la aplicación encargada de detectar imágenes y las librerías necesarias para ejecutarla en el siguiente GitHub:  
<https://github.com/Minekaneo/TFG>

## 7. Pruebas

Para poder comprobar el correcto funcionamiento de la aplicación capaz de diferenciar imágenes se realizarán pruebas con ejemplos simples. Para poder evaluar este comportamiento nos fijamos en la pérdida (loss) generada por cada imagen que queremos entrenar en el modelo (epoch). Esta pérdida se genera a la hora del entrenamiento, haciendo la diferencia entre el output generado al fluctuar la data por la red neuronal y el output esperado al seleccionar a qué clase corresponde esta data. Por tanto, la pérdida representa exactamente en cuanto ha confundido el programa la entrada por otra clase. En consecuencia el objetivo de las pruebas es obtener la pérdida más cercana a cero con tal de poder afirmar que el programa diferencia correctamente las clases entrenadas.

### 7.1 Primera prueba

La primera prueba consistirá en el entrenamiento de un microcontrolador único. A este microcontrolador se le entrenarán dos clases, una corresponderá a una imagen negra realizada sobre una hoja totalmente negra y la otra corresponderá a una imagen blanca realizada sobre una hoja totalmente blanca, hojas mostradas en la figura 29. Los parámetros usados en la primera prueba para la red neuronal son 0.3 como velocidad de aprendizaje y 0.9 de momentum, los cuales son los valores por defecto de la red neuronal y se espera que den un mejor resultado. El entrenamiento consistirá en realizar una fotografía a cada clase de manera intercalada hasta ver una tendencia en la gráfica hacia 0 o una tendencia irregular que no convergerá nunca.



Fig 29: las dos hojas usadas como ejemplos para los entrenamientos

Como se puede observar la figura 30, se visualiza una progresión de la pérdida hacia el 0 que consigue converger totalmente alrededor de las 35 imágenes entrenadas y por tanto, este hecho significa que la aplicación ha aprendido a diferenciar las dos clases y ha conseguido reducir el error en sus predicciones al punto en que prácticamente no confunde en lo más mínimo las imágenes entre las dos clases entrenadas.

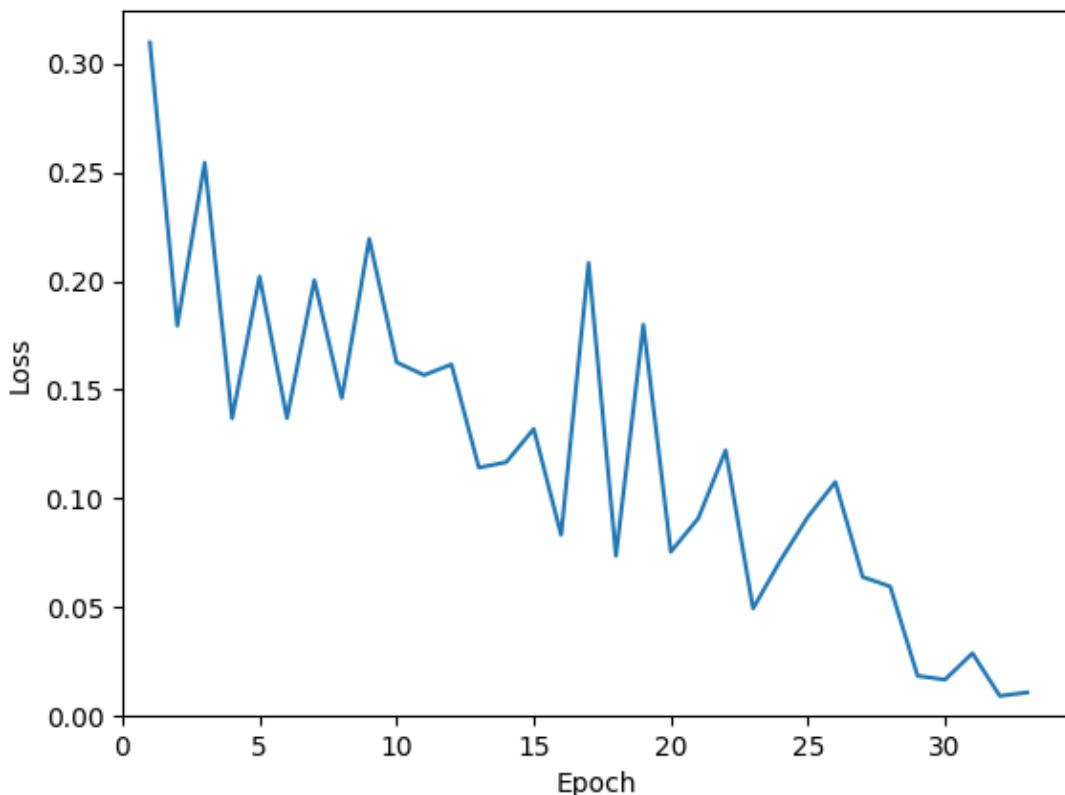


Fig 30: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la primera prueba, con una velocidad de aprendizaje de 0.3 y un momentum de 0.9

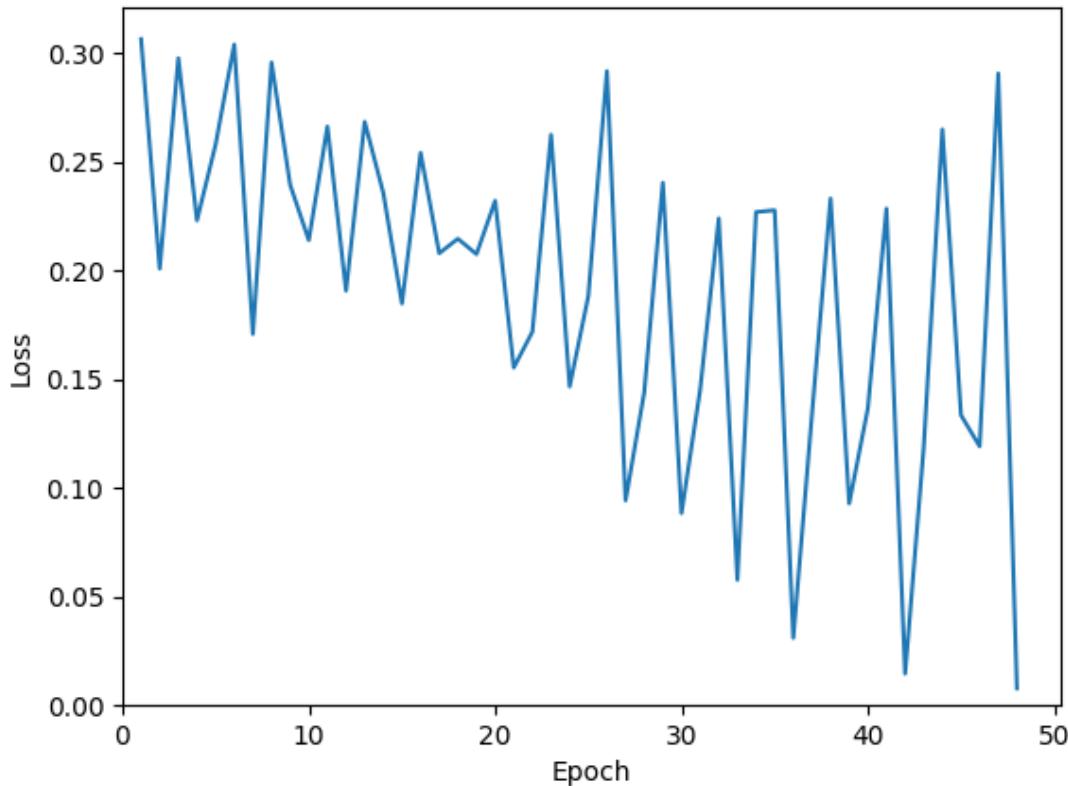
## 7.2 Segunda prueba

Para la segunda prueba se cambiarán el número y clases que se entranan. Ahora se entrenarán 3 clases diferentes. La primera corresponde al color azul, la segunda al color verde y la tercera al color rojo. Para entrenar estas tres clases igual que en la primera prueba se usarán tres hojas de colores, una totalmente azul, otra totalmente verde y por último una totalmente roja, hojas mostradas en la figura 31. El intervalo de aprendizaje y momentum se mantendrán idénticos a la anterior prueba siendo estos de 0.3 y 0.9 respectivamente. Al igual que en la primera prueba se tomará 1 foto de cada clase de manera intercalada hasta poder observar una tendencia en la gráfica generada.



*Fig 31: las tres hojas usadas como ejemplo para los entrenamientos*

Como se puede observar en la figura 32, se visualiza una progresión hacia el cero pero con picos constantes en medio. Estos picos han sido debidos a la confusión del color azul y el color verde, haciendo que cuando se le entrenaba la clase verde, tanto el verde como el azul tuvieran un gran porcentaje de coincidencia. Aunque el loss en esos casos era mucho mayor que las otras clases cabe recalcar que la clase con más porcentaje seguía siendo la correcta pero al tener un gran porcentaje con otra clase la pérdida se disparaba. Tanto azul como rojo por separado los diferencia de las otras dos sin problemas, por ello la gráfica iba en descenso llegando a situarse muy cerca del 0 alrededor de las 50 imágenes.



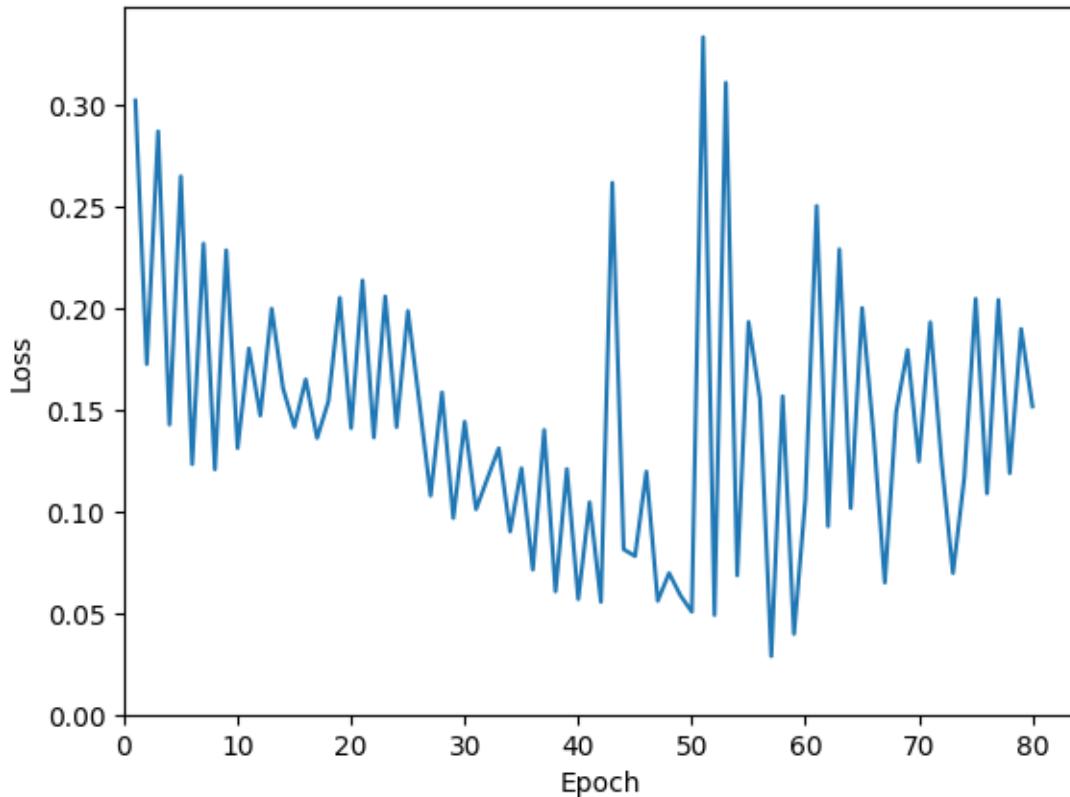
*Fig 32: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la segunda prueba, con una velocidad de aprendizaje de 0.3 y un momentum de 0.9*

### 7.3 Tercera prueba

Para la tercera prueba se investigará las consecuencias de reducir la velocidad de aprendizaje de 0.3 a 0.1, un tercio de la velocidad por defecto. Para esta prueba se usarán las mismas clases que en la primera, ya que son las que han dado un mejor resultado y en la que se pueden apreciar mejor los cambios. Por tanto, se tomará una foto a la clase negra y una a la clase blanca de manera intercalada hasta que se pueda observar una tendencia en el gráfico hacia el 0 o una tendencia irregular que nunca llegará a converger.

Como se puede observar en la figura 33, se visualiza una tendencia hacia el 0 desde el comienzo pero alrededor de las 40 imágenes esta tendencia desaparece y pasa a ser irregular con un incremento del error constante. Estos resultados son un poco desconcertantes, lo esperado al reducir la velocidad de entrenamiento era una tendencia hacia el 0 igual que con 0.3 pero más lenta como se podía apreciar en las primeras 40 imágenes. Aunque la tendencia del error hacia el final del gráfico es irregular, al no ser errores excesivamente grandes permitía a la aplicación diferenciar

las dos clases entrenadas, pero como el error no llega a converger a 0 no se puede afirmar que la diferenciación de las clases vaya a ser constante con cualquier entrada que se le de a la red neuronal.



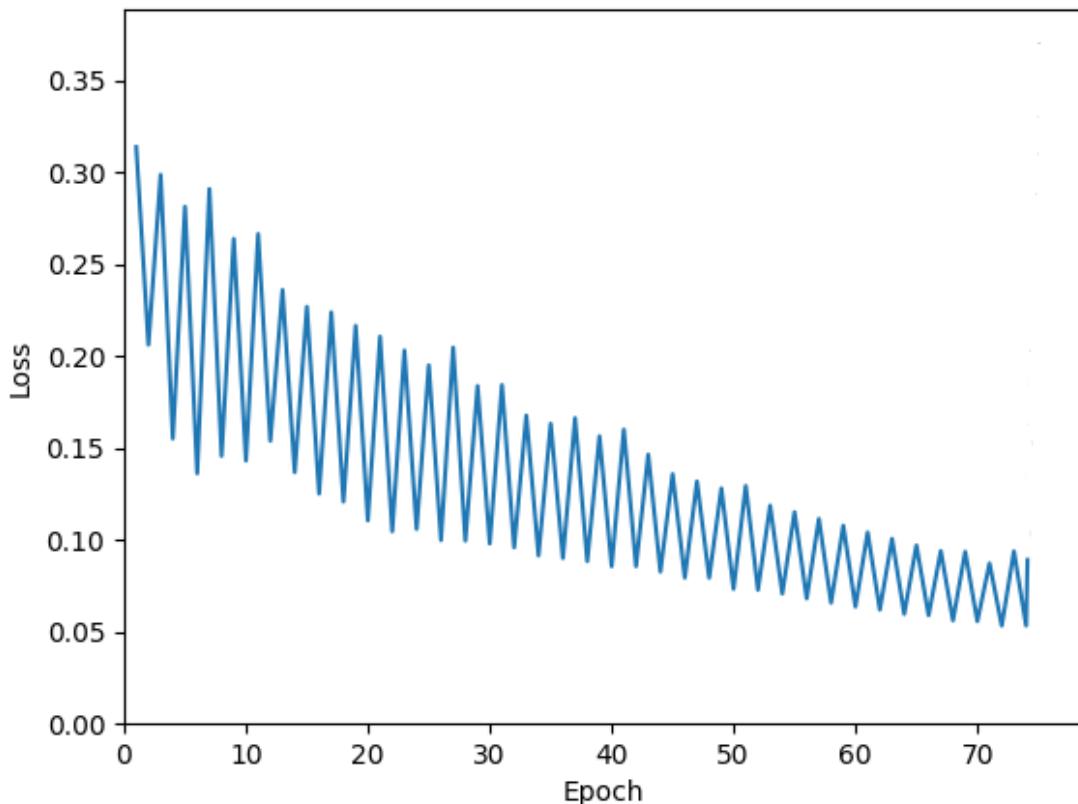
*Fig 33: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la tercera prueba, con una velocidad de aprendizaje de 0.1 y un momentum de 0.9*

## 7.4 Cuarta prueba

En la cuarta prueba se pretende seguir investigando las consecuencias de cambiar los valores por defecto de la red neuronal de la velocidad de aprendizaje y el momentum. En esta prueba a diferencia de la anterior se cambiará el momentum a 0 de la red neuronal y se mantendrá el valor por defecto de la velocidad de aprendizaje. Se entrenará las dos clases negro y blanco, realizando una fotografía a cada clase de manera intercalada hasta poder observar una tendencia en el gráfico generado por el error hacia el 0 o una tendencia irregular.

Como se puede observar en la figura 34, se visualiza una tendencia en la reducción del error que se estabiliza entre 0.05 y 0.10 de error alrededor de las 70 imágenes. El gráfico resultado de la prueba no consigue converger en el 0 pero si relativamente cerca de él, por tanto se puede asumir que la aplicación es capaz de diferenciar correctamente las clases aunque puede darse el caso de algunas confusiones puntuales.

El motivo por el cual es posible que no llegué al 0 puede ser debido a que al no disponer de ningún momentum la clase con mayor error no fuera capaz de reducirlo lo suficiente gracias a la ayuda de la otra y por tanto se estabilizarán las dos un poco por encima del 0.



*Fig 34: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la cuarta prueba, con una velocidad de aprendizaje de 0.3 y un momentum de 0.0*

## 7.5 Quinta prueba

La quinta prueba tiene el objetivo de observar el comportamiento de la aplicación cuando tanto la velocidad de aprendizaje como el momentum son reducidos, siendo estos de 0.1 de velocidad de aprendizaje y 0 de momentum. Se seguirán usando las clases negro y blanco y se entrenará de la misma manera que en pruebas anteriores, realizando una fotografía en cada clase de manera intercalada hasta poder observar una tendencia en el gráfico generado por el error hacia el 0 o una tendencia irregular.

Como se puede observar en la figura 35, se visualiza una tendencia constante alrededor de 0.2 de error durante todo el gráfico. Este comportamiento no permite asumir que la aplicación funcione correctamente con estos valores de velocidad de aprendizaje y momentum debido a que puede tener una gran cantidad de confusiones en las predicciones.

Con estas pruebas se ha podido observar que reducir la velocidad de aprendizaje y el momentum no ayudan a mejorar los resultados del entrenamiento de la red neuronal y por tanto es mejor mantenerlos en sus valores por defecto para obtener mejores resultados con la aplicación.

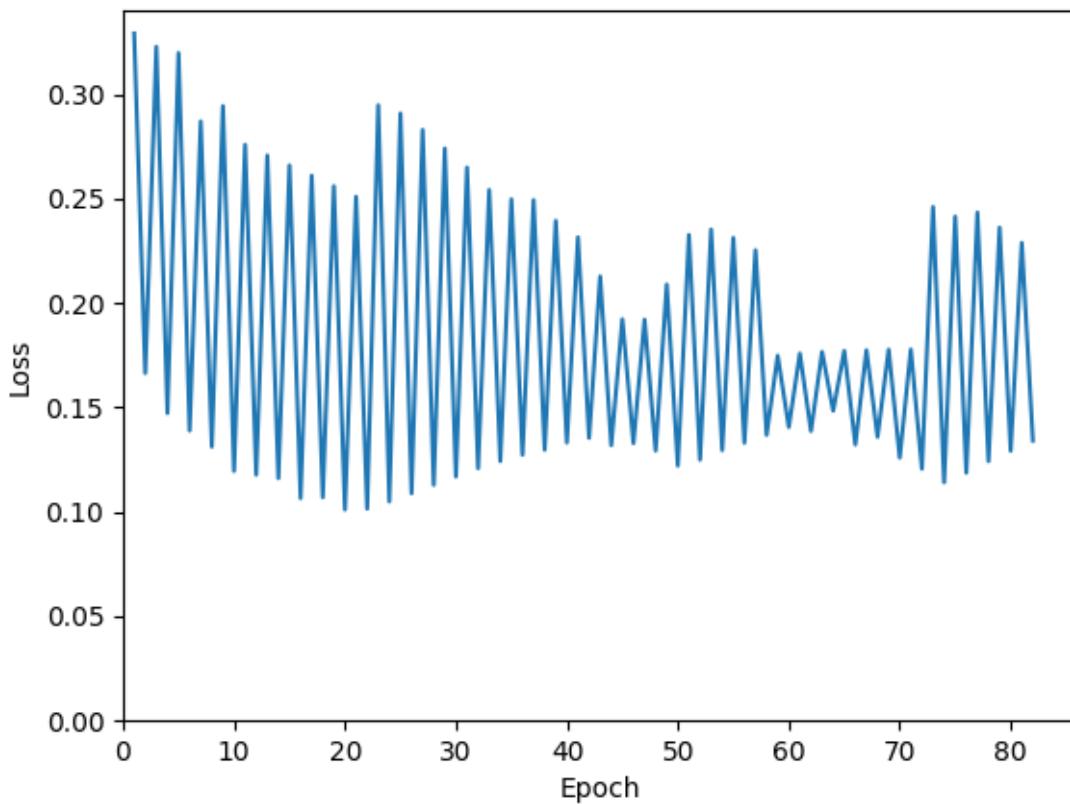


Fig 35: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la quinta prueba, con una velocidad de aprendizaje de 0.1 y un momentum de 0.0

## 7.6 Primera prueba Aprendizaje Federado

Las pruebas de Federated Learning han sido realizadas sobre dos microcontroladores idénticos, los dos con el mismo código en su memoria. El primer controlador corresponderá a la línea azul de los gráficos y el segundo a la línea naranja.

La primera prueba consiste en entrenar los dos microcontroladores de la misma manera. Para ello, se realizan 10 imágenes en total en cada microcontrolador antes de que el servidor junte los modelos. Estas 10 imágenes serán 5 en cada una de las dos clases que se entrenarán, la clase negra, la cual será una fotografía en una hoja negra y la clase blanca, la cual será una fotografía en una hoja blanca. Se repetirá este proceso hasta poder apreciar una tendencia hacia el 0 o una tendencia irregular que no sea capaz de converger hacia al 0.

Como se puede observar en la figura 36, se visualiza una tendencia hacia el 0 desde el principio llegando a converger en este entre las 20 y 25 imágenes. Esta prueba confirma que la aplicación de diferenciación de imágenes funciona correctamente aplicando el algoritmo de aprendizaje federado y como resultado de ello es capaz de converger el error, generado al entrenar las imágenes, al cero con una menor cantidad de imágenes por dispositivo en comparación a realizar la misma prueba pero sin aprendizaje federado. Este hecho es debido a que gracias a la creación del modelo global que se envía desde el servidor a los microcontroladores estos son capaces de recibir el entrenamiento realizado por los otros microcontroladores conectados al servidor, consiguiendo así un entrenamiento más rápido a costa de tener más dispositivos entrenando.

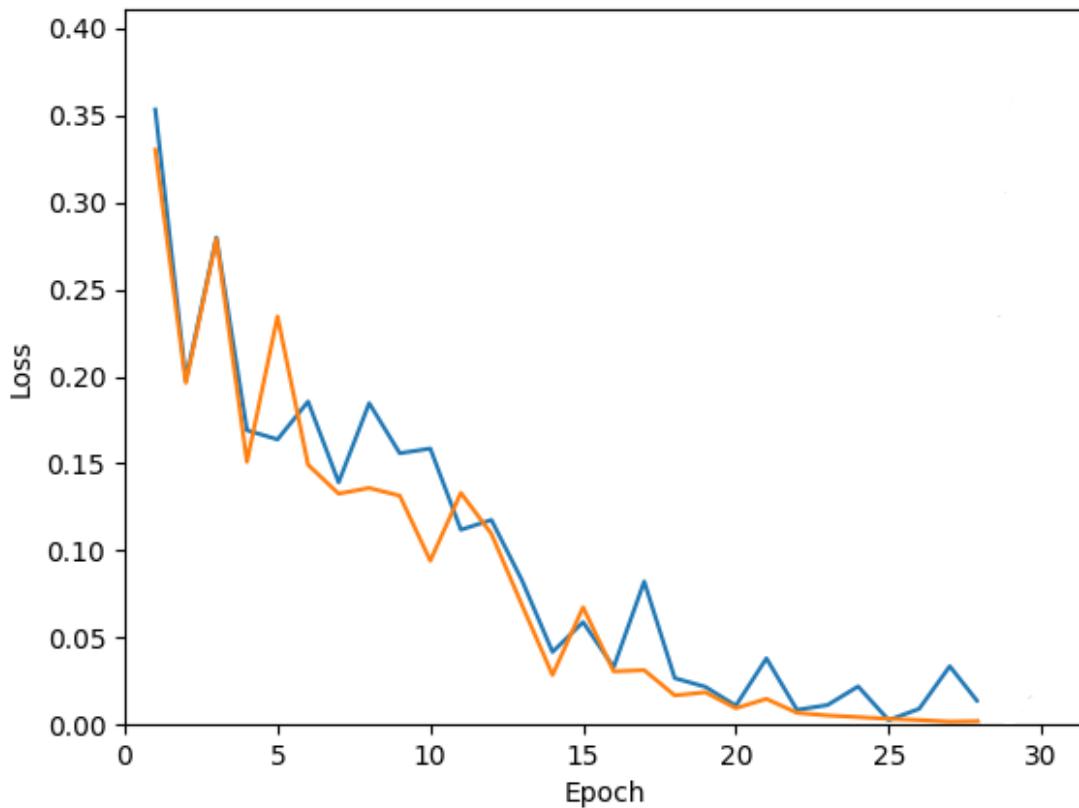


Fig 36: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la primera prueba de aprendizaje federado, con una velocidad de aprendizaje de 0.3 y un momentum de 0.9

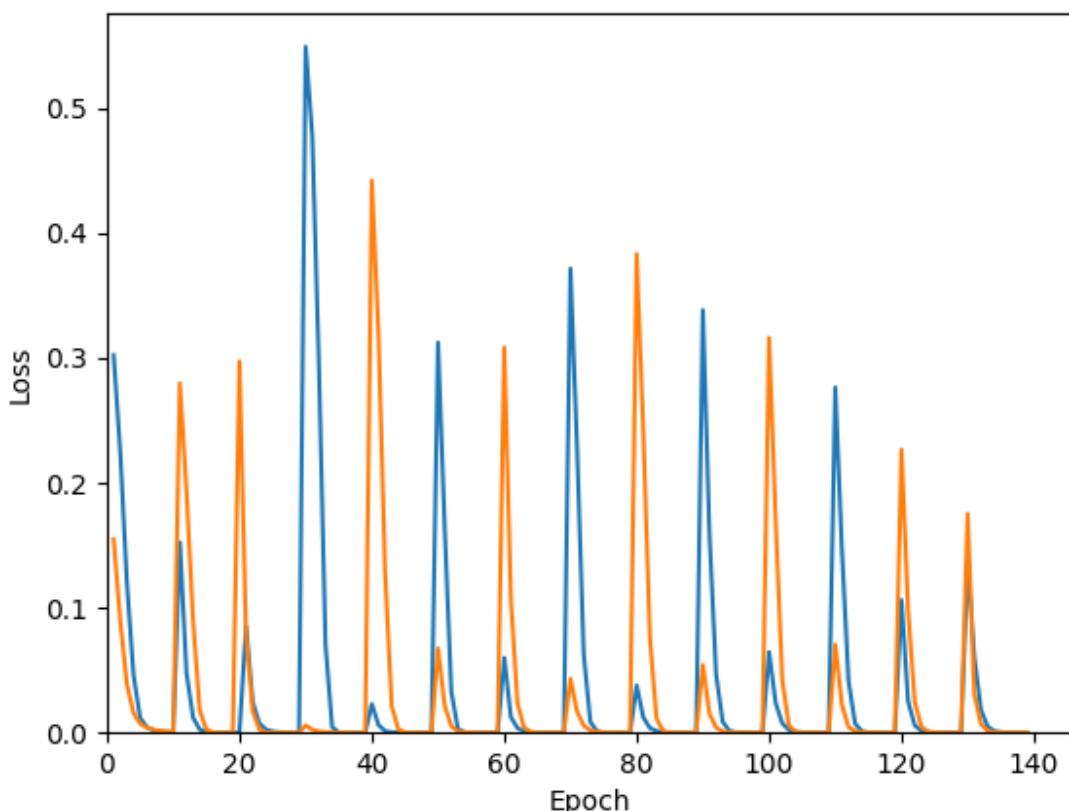
## 7.7 Segunda prueba Aprendizaje Federado

La segunda prueba de Federated Learning pretende observar el comportamiento cuando cada microcontrolador entrena una clase diferente del otro. En esta prueba el microcontrolador 1 (naranja) entrenará la clase del negro y el microcontrolador 2 (azul) entrenará la clase del blanco. Los entrenamientos consistirán en realizar 10

fotografías con cada microcontrolador para entrenar la clase que se les ha asignado y se esperará a que el servidor haga la fusión con los modelos antes de empezar la siguiente ronda de entrenamiento, y así hasta poder observar una tendencia en el gráfico.

Como se puede observar en la figura 37, se aprecia un pico cada 10 imágenes. Este pico es debido a la fusión que realiza el servidor con los dos modelos de los microcontroladores. Cada microcontrolador entrena una sola clase y por tanto reduce el error muy rápido pero cuando el servidor junta los modelos en uno nuevo los errores se disparan porque el microcontrolador opuesto no ha entrenado la clase del otro y por tanto, está atrasado respecto al otro en esa clase. Entonces, al hacer la media de los dos entrenamientos retrasa también al que entrenaba esa clase.

El error medio de los dos microcontroladores se empieza a estabilizar hacia el final del gráfico situándose alrededor de 0.1, un error medianamente bajo que puede generar algunas confusiones en las predicciones pero funcionar bien en general, pero nos empezamos a situar en las 140 imágenes entrenadas por cada microcontrolador, unas 280 imágenes en total, haciendo de este proceso uno muy costoso e ineficiente.



*Fig 37: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la segunda prueba de aprendizaje federado, con una velocidad de aprendizaje de 0.3 y un momentum de 0.9*

## 7.8 Tercera prueba Aprendizaje Federado

La tercera prueba de Federated Learning tiene como objetivo seguir estudiando el comportamiento de la aplicación cuando cada clase se entrena en cada microcontrolador. En esta prueba en vez de hacer diez imágenes en cada microcontrolador antes de que el servidor juntara los modelos, se realizará una única imagen en un microcontrolador y se esperará a que el servidor junte los modelos, luego se realizará una imagen en el otro microcontrolador y así sucesivamente hasta poder observar una tendencia en el gráfico. Las clases entrenadas seguirán siendo la negra y blanca.

Como se puede observar en la figura 38, se visualiza un comportamiento opuesto entre los dos microcontroladores. Cuando uno empieza a mejorar el error, el otro lo empeora al mismo ritmo, esto puede ser debido a que al juntar los modelos, el modelo resultante tenga preferencia por una de las dos clases en ciertos momentos creando así un error menor en la clase con preferencia y un error mayor en clase sin preferencia. Este error se ve reflejado en los microcontroladores ya que cada uno entrena una clase diferente.

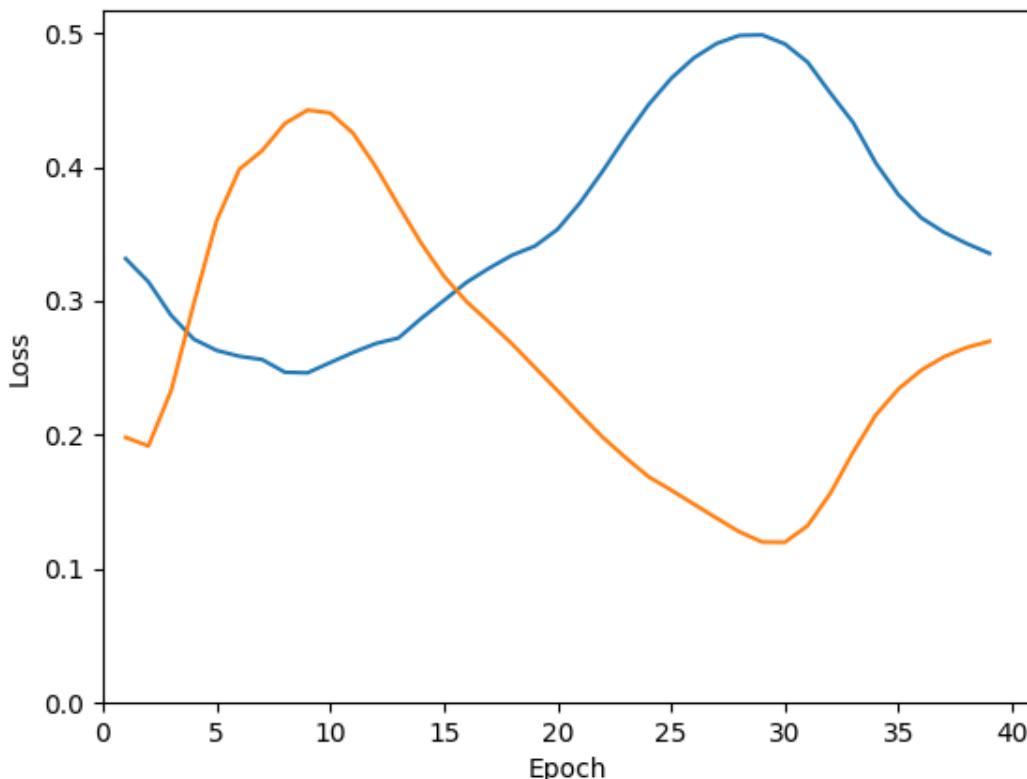


Fig 38: gráfico que relaciona el error obtenido con cada imagen entrenada resultante de la tercera prueba de aprendizaje federado, con una velocidad de aprendizaje de 0.3 y un momentum de 0.9

Se puede confirmar que este método de entrenamiento para la aplicación de federated learning es totalmente erróneo y los resultados que puede realizar la aplicación pueden no ser los esperados en la mayoría de los casos.

## 8. Análisis Sostenibilidad

En este apartado se analizará el impacto del proyecto en la sociedad desde los puntos de vista ambientales, económicos y sociales, mediante el uso de la matriz de sostenibilidad. La matriz divide el análisis en tres grandes bloques: Proyecto Puesto en Producción (PPP), vida útil y riesgos. Donde en cada bloque se evaluarán los tres puntos de vista nombrados anteriormente, ambiental, económico y social. Una vez realizado el análisis se asignará un valor a cada bloques representando el impacto generado por el proyecto, siendo 1 muy negativo y 10 muy positivo.

	<b>PPP</b>	<b>Vida Útil</b>	<b>Riesgos</b>
<b>Ambiental</b>	Impacto ambiental del proyecto	Huella Ecológica	Posibles aumentos huella ecológica
<b>Económico</b>	Costes del proyecto	Viabilidad del proyecto	Posibles perjuicios de la viabilidad del proyecto
<b>Social</b>	Impacto Personal	Impacto Social	Posibles perjuicios a la población

Tabla 8: matriz de sostenibilidad (elaboración propia)

### 8.1 Proyecto Puesto en Producción (PPP)

#### 8.1.1 Ambiental

Para cuantificar el impacto ambiental realizado por el proyecto se ha usado la métrica basada en el consumo de energía (Watts/h). El proyecto ha sido realizado por un ordenador y tres microcontroladores diferentes de Arduino 33 BLE Sense. Considerando que un ordenador consume unos 275 Watts de media por hora y que el proyecto ha requerido un total de 483 horas, la cantidad total de energía es de:

$$50\text{W} * 483\text{h} = 24,15\text{kWh.}$$

En cuanto a los microcontroladores, el consumo medio de energía es de 0.2W. Considerando que el uso de cada uno ha sido de alrededor de 5-6h, la cantidad total de energía es de:

$$0.2\text{W} * 6\text{h} * 3 \text{ microcontroladores} = 3,6\text{Wh}$$

Por tanto sumando los dos costes calculados previamente obtenemos un consumo energético total de:

$$24,15\text{kWh} + 0,0036\text{kWh} = 24,1536\text{kWh}$$

Con el objetivo de reducir el coste energético del proyecto dos de los tres microcontroladores y el ordenador han sido reutilizados de otros proyectos anteriores. Debido a que el coste energético de fabricar un único microcontrolador es bastante reducido y todos los demás componentes son reutilizados es la razón por la que se han omitido los costes de fabricación en el total. Se ha observado que en caso de querer realizar de nuevo el proyecto solo serían necesarios dos microcontroladores y no tres ya que con dos ya es posible observar el comportamiento del aprendizaje federado. Por tanto, si se quisiera reducir los costes para un futuro proyecto, usar un microcontrolador menos sería una opción.

### **8.1.2 Económico**

El coste del proyecto se analizó anteriormente en el capítulo 3. Presupuesto de la memoria. En esta sección se estima un coste final del proyecto de 13.839,81€. Pero, debido a cambios en el proyecto y la cancelación de la tarea 5: microcontrolador como servidor que tenía un coste de 2.791,5€ , el coste final del proyecto se ha reducido a 11.048,31€.

### **8.1.3 Social**

El proyecto ha sido una experiencia valiosa para mi carrera profesional, me ha ayudado a mejorar a la hora de documentar los trabajos e investigaciones realizados a nivel personal, aprendiendo a seguir una estructura correcta de planificación, ejecución y presentación de los proyectos. En cuanto al tiempo invertido el proyecto me ha ayudado a darme cuenta de errores que tenía a la hora de planificar las horas de trabajo a lo largo de los días, errores que desarrollaban en acumulaciones de tareas a realizar y estrés debido a ello. A lo largo del trabajo estos errores se han ido arreglando llegando a conseguir una distribución balanceada de las tareas a realizar a lo largo de los días. En resumen, creo que el proyecto me ha ayudado a mejorar tanto profesionalmente a la hora de expresar mis conocimientos, como personalmente a la hora de organizarme.

## 8.2 Vida útil

### 8.2.1 Ambiental

Las aplicaciones de aprendizaje automático, una vez ya han sido entrenadas, no requieren de ningún dispositivo adicional para llevar a cabo sus tareas, por tanto, el consumo energético del proyecto durante su vida útil sería correcto asumir que equivale al consumo energético de los propios microcontroladores. Tal y como se ha comentado anteriormente este consumo es de 0.2W por hora, siendo un consumo relativamente bajo en comparación a aplicaciones de aprendizaje automático implementadas en otros dispositivos como ordenadores que consumen 275W por hora de media. Este hecho convierte a los microcontroladores en una opción energética mucho mejor en comparación a los ordenadores, mejorando así la huella ecológica de estas aplicaciones.

### 8.2.2 Económico

Como se ha comentado en el punto anterior, los microcontroladores una vez entrenados no tienen la necesidad de ningún dispositivo adicional y por tanto son capaces de reducir el consumo energético en gran medida. En consecuencia el coste energético de estos dispositivos es mucho menor y teniendo en cuenta que en los últimos meses el precio de la electricidad ha aumentado considerablemente esto comporta en un gran ahorro económico. Por tanto, es correcto asumir que el mundo de TinyML vaya a crecer rápidamente debido a su bajo consumo, haciendo así que la viabilidad del proyecto sea muy buena.

### 8.2.3 Social

Cualquier persona puede beneficiarse del contenido del proyecto, permitiendo la creación de nuevas aplicaciones de TinyML en diferentes sectores como defensa, medicina, etc... Pudiendo aumentar la calidad de vida de las personas ayudándoles en tareas simples y constantes que requieran poco esfuerzo para un microcontrolador pero mucho más esfuerzo para una persona.

## 8.3 Riesgos

### 8.3.1 Ambiental

Teniendo en cuenta que los microcontroladores son dispositivos muy pequeños y con un coste energético bajo para crearlos, se podrían considerar como la mejor opción actual a la hora de implementar aplicaciones para reducir la huella ecológica de estas.

Siendo de esta manera difícil visualizar un escenario donde aplicaciones TinyML, que usan estos dispositivos empeoren la huella ecológica.

### 8.3.2 Económico

Parecido al punto anterior, el coste económico de producir aplicaciones TinyML de aprendizaje federado como en el proyecto es muy reducido en comparación a sus otras opciones, haciendo de esta una de las mejores opciones en el mercado para producir. El único posible escenario en que no se cumpliera esta premisa, sería si se redujera la producción de microcontroladores, hecho bastante improbable debido al gran crecimiento del uso de estos dispositivos.

### 8.3.3 Social

Como en la gran mayoría de los proyectos tecnológicos que son capaces de mejorar las cualidades de vida de las personas, las poblaciones que no sean capaces de conseguir tal tecnología se verán perjudicadas respecto a las poblaciones que disponen de estas. Estos casos se pueden observar en el día a día comparando países de la unión europea y las tecnologías de las que disponen en estos y países del continente africano donde no disponen de esta gran variedad de tecnologías, haciendo que estos países se encuentren atrasados en cuanto a evolución tecnológica y en parte marginados por los otros países.

No se espera que el proyecto en sí sea capaz de realizar ninguna dependencia ya que la aplicación desarrollada en él es simple y tiene pocas funcionalidades. Pero sí es posible que en un futuro estas aplicaciones evolucionen hasta un punto en que la gente las vea necesarias en sus vidas y por tanto, en caso de que desaparecieran, sentir un vacío creado por la falta de la ayuda proporcionada por estas aplicaciones.

## 8.4 Resultados análisis de sostenibilidad

	<b>PPP</b>	<b>Vida Útil</b>	<b>Riesgos</b>
<b>Ambiental</b>	Impacto ambiental del proyecto <b>9</b>	Huella Ecológica <b>9</b>	Posibles aumentos huella ecológica <b>7</b>
<b>Económico</b>	Costes del proyecto <b>8</b>	Viabilidad del proyecto <b>10</b>	Posibles perjuicios de la viabilidad del proyecto <b>7</b>
<b>Social</b>	Impacto Personal <b>9</b>	Impacto Social <b>8</b>	Posibles perjuicios a la población <b>6</b>

Tabla 9: matriz de sostenibilidad con las respectivas notas

## 9. Trabajos futuros

### 9.1 Bluetooth

Uno de los subobjetivos iniciales del proyecto era desarrollar una aplicación de keyword spotting que enviara la información entre microcontroladores y servidor mediante el uso del Bluetooth. Esta tecnología permite a la aplicación el hecho de ignorar los cables para poder transmitir información y por tanto más libertad en cuanto a movilidad.

El microcontrolador Arduino 33 BLE Sense dispone tal y como en su nombre indica de la tecnología BLE (Bluetooth Low Energy). Este bluetooth a diferencia del normal utiliza menos capacidades, como la cantidad de información que se puede enviar por mensaje, a cambio de consumir muchísimos menos recursos.

Para poder utilizar esta tecnología es necesario que los dos dispositivos que desean enviar información dispongan de BLE, además de librerías adaptadas para esa tecnología. Este último dato fue el mayor problema que se enfrentó durante el proyecto respecto al Bluetooth. Librerías comunes para el Bluetooth de Python como PyBluez, la propia librería llamada Bluetooth, entre otras no funcionaban correctamente al estar usando el Bluetooth normal en vez de la versión Low Energy.

No fue hasta el uso de una librería llamada Bleak[20] que no se consiguió obtener un programa simple capaz de enviar un mensaje por bluetooth hacia el arduino y luego desde el arduino reenviar el mensaje recibido de vuelta al servidor y observar que el mensaje seguía intacto.

Bleak (Bluetooth Low Energy platform Agnostic Klient) es un software de cliente GATT(perfiles de atributos genéricos). Este término GATT es una especificación para enviar pequeños fragmentos de datos, denominados “atributos”, a través de un vínculo BLE. Todos los perfiles de aplicaciones de bajo consumo actuales se basan en GATT. Esta librería Bleak, es capaz de conectarse a los dispositivos actuando como un servidor GATT. Usado comúnmente para proveer a una API de Python asíncrona y multiplataforma las necesidades para conectarse y comunicarse con dispositivos de bajo consumo como por ejemplo sensores.

Fue entonces, a partir de esta librería Bleak, se creó un primer prototipo para poder enviar mensajes del servidor al microcontrolador y luego el microcontrolador leía el mensaje y lo devolvía sin cambiar ningún carácter.

Mediante testeos de este prototipo se descubrió que el tamaño de los mensajes que permitía enviar no podía exceder los 240-246 bytes, en el caso de que se intentara una función de la librería Bleak saltaba dando errores. El motivo más probable para este

hecho es que el tamaño máximo de los mensajes enviados usando la librería Bleak sean 256 bytes, que es múltiplo de 2 y suelen ser tamaños más normales, pero debido a una posible cabecera del mensaje para poder indicar a donde tiene que dirigirse y de quien es el mensaje, el tamaño efectivo al que se reduce el mensaje se encuentra alrededor de los 240-246 bytes. Este motivo no está confirmado pero es la explicación más razonable.

Entonces, la cantidad de bytes que se pueden enviar en los mensajes se reduce a un total de 240-246 bytes. Para solucionar el problema de poder enviar los mensajes grandes se hizo una pequeña modificación donde el mensaje que se quería transmitir se envía fragmentado en paquetes más pequeños y con un sistema de confirmación de recibo de los mensajes se puede garantizar que no se perderán los contenidos o mezclarán.

A partir de este punto, en el proyecto se empezó a juntar los dos códigos, el de enviar cualquier mensaje por bluetooth del servidor al microcontrolador y el de federated learning. Esta mezcla llevó a algunos errores de incompatibilidad inesperados e incluso en algunos puntos fue capaz de inhabilitar algunos microcontroladores por un tiempo hasta que fueron reiniciados y limpiados. Debido a estos hechos se decidió abandonar este camino y por tanto dejarlo como un trabajo futuro del proyecto.

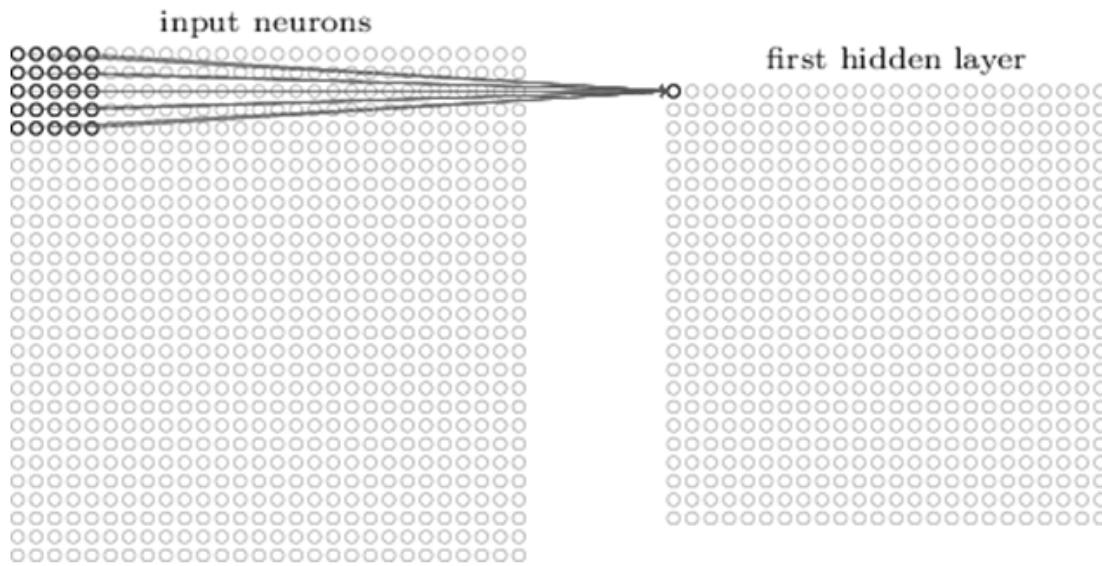
Ver [Anexo Código Bluetooth](#) para visualizar las funciones principales especificadas para la aplicación Bluetooth.

## 9.2 Red convolucional

Las redes convolucionales son un tipo de red neuronal artificial donde las neuronas artificiales, son muy similares a las neuronas en la corteza visual (parte del cerebro humano que se encarga del reconocimiento de patrones). Estas redes convolucionales por tanto son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes.

La primera capa en una red convolucional es siempre la Capa Convolucional. Esta capa se encarga de filtrar una zona de los datos de entrada (que suelen ser imágenes) para convertirlo en un solo número. Si nos imaginamos el filtro como una linterna, el área que quedaría iluminada sería la “receptive field”, dentro de esa área el filtro se encarga de multiplicar todos los valores originales de la imagen con los valores del filtro. Estas multiplicaciones luego se suman todas ellas y obtenemos un solo número que será el encargado de representar la zona que hemos tratado. Luego de calcular una zona el filtro se mueve 1 unidad a la derecha y repite el proceso hasta haber hecho cada una de las posibles zonas, por ejemplo, con un filtro de 5x5x3 en una imagen de 32x32x3 existen 784 diferentes zonas en las que colocar el filtro. Estas 784 zonas se

convertirán en números calculados por el filtro que compondrán una nueva imagen de 28x28.



*Fig 39: Ejemplo del trabajo del filtro en una red convolucional*

Los mayores responsables del resultado de los números calculados son los filtros. Dependiendo de que nos encontremos en el filtro, zonas de la imagen quedan totalmente descartadas por no contener la característica que andamos buscando con el filtro o al revés, debido al filtro obtendrán un gran número en caso de que se parezca mucho a lo que buscamos. Pongamos que tenemos el siguiente filtro de la figura 40:

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0



*Fig 40: ejemplo de un filtro posible para una red convolucional*

Este filtro como se puede observar corresponde a una línea curva, por tanto, en la imagen que usemos como entrada, las zonas con curvas obtendrán números mucho más elevados que las zonas sin curvas.

En el caso de la figura 41, la entrada es otra curva bastante parecida a la del filtro. Si hacemos los cálculos y multiplicamos cada pixel con su correspondiente píxel nos quedará la siguiente suma:  $(50*30)+(50*30)+(50*30)+(20*30)+(50*30)$ , esto después de sumarlo da un total de 6600, indicando que al ser un número bastante elevado el parecido de la zona input con el filtro es bastante alto.

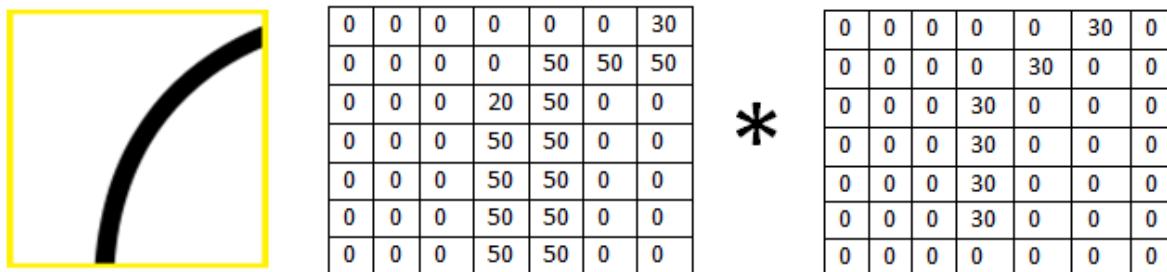


Fig 41: ejemplo de aplicación del filtro de la fig en un caso positivo

Vayamos ahora a otro caso totalmente distinto. En la figura 42 no se puede observar ninguna línea curva como la del filtro, por tanto, si tomamos un vistazo a las dos matrices veremos que el resultado de las multiplicaciones y la suma será cero, ya que cada número distinto a cero se multiplica por cero en la otra matriz.

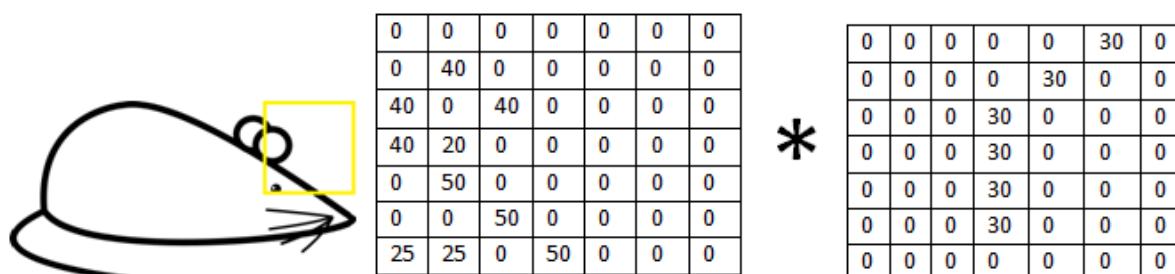


Fig 42: ejemplo de aplicación del filtro de la fig en un caso negativo

Entonces dependiendo de qué filtro usemos podemos decidir qué patrones en la imagen queremos encontrar, como círculos, rectas, curvas, etc.. De hecho lo más común es que después de aplicarle un filtro a la imagen, al resultado de esa operación se le aplique otro filtro y así varias veces hasta obtener el resultado deseado. Dependiendo del orden y filtros que se usen, se pueden acabar teniendo filtros mucho más complejos capaces de diferenciar entre garabatos o palabras.

Una vez hemos obtenido las características deseadas gracias a los filtros solo nos falta procesarlos. Para ello, se añade una capa totalmente conectada al final de la red. Esta capa será la encargada de transformar la entrada que le llega a un vector de las N clases que se quieran diferenciar, con una probabilidad de coincidencia para cada una. En la figura 43 se puede apreciar un ejemplo sencillo de una red convolucional.

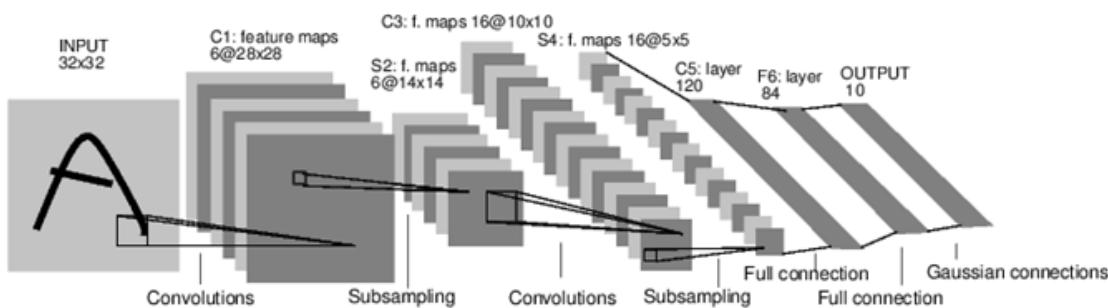


Fig 43: esquema general de una red convolucional

Las redes neuronales convolucionales como se puede observar son las mejores en cuanto a tratar imágenes, objetivo que perseguía la aplicación con la cámara del microcontrolador. Por ello, este tipo de red quedaría como trabajo futuro del proyecto, ya que muy probablemente haría mejorar significativamente los resultados de la identificación de las clases en la aplicación y su eficiencia.

Eso sí, habría que estudiar el tamaño que puede llegar a ocupar una red neuronal de este estilo y el espacio necesario para entrenarla en el dispositivo. Ya que, como ya sabemos los microcontroladores en cuanto a memoria són muy limitados y dependiendo de cuán grande se quisiera hacer esta red podría no haber suficiente espacio en la memoria del microcontrolador para aguantarla.

### 9.3 Arduino servidor de Federated Learning

Como último trabajo futuro, quedaría pendiente el tercer objetivo inicial. Este objetivo trataba un campo muy poco tratado dentro de los microcontroladores y por tanto esto lo hacía bastante complicado, pero unos de los requisitos para poderlo hacer era tener el bluetooth funcionando.

Esta tarea consiste en disponer como servidor un microcontrolador en vez de un ordenador como se ha estado usando hasta ahora. Para poder lograr esto, obviamente se necesitaba poder conectar por bluetooth los microcontroladores, ya que de lo contrario, no se podrían llegar a enviar información entre ellos. Pero en cuanto a complejidad el verdadero objetivo es conseguir que el microcontrolador pueda hacer el trabajo del ordenador a una velocidad suficiente para poder tratar con más de un microcontrolador diferente.

Los microcontroladores como se sabe están muy limitados en sus capacidades y obviamente respecto a un ordenador son muy inferiores. Esto hace que si intentamos usar el código del servidor en el arduino, el microcontrolador esté demasiado tiempo a la hora de recibir, tratar y enviar de vuelta los modelos. Esto haría que la aplicación fuera muy lenta y por tanto no muy eficiente.

Ahí es donde entran las optimizaciones de código y los threads. La idea de este trabajo futuro sería investigar si es posible optimizar lo suficiente el código, con la ayuda de la tecnología RTOS, para que un arduino pudiera ejecutarlo y así llegar a prescindir de la necesidad de un ordenador para aplicaciones de este tipo.

## 10. Conclusiones

Este proyecto estaba compuesto por dos objetivos genéricos los cuales conformaban una primera parte del proyecto y una segunda parte. El primer objetivo buscaba conseguir familiarizarse con las herramientas necesarias para poder reproducir aplicaciones TinyML de cualquier tipo para posteriormente ser capaz de reproducir aplicaciones de aprendizaje federado.

Para lograr este primer objetivo se realizaron los cursos de TinyML creados por Edx nombrados en la Tarea 2 del proyecto, gracias a ellos se obtuvo un rumbo inicial en el cual ya sería capaz de crear y cargar aplicaciones de TinyML en el microcontrolador integrado en el Kit de TinyML de arduino, el Arduino 33 BLE Sense. Posteriormente, una vez se probaron diferentes herramientas capaces de poder realizar aplicaciones TinyML en los microcontroladores, se llegó a la conclusión que la mejor entre todas ellas para el proyecto de aprendizaje federado era Edge Impulse junto al IDE PlatformIO. Fue entonces mediante estas aplicaciones que se reprodujo y se comprendió el funcionamiento de la aplicación inicial del proyecto creada por Marc Monfort completando así el primer objetivo del proyecto y la primera parte de este.

El segundo objetivo genérico del proyecto buscaba la investigación y experimentación con las aplicaciones avanzadas de aprendizaje federado. Esta segunda parte del proyecto estaba formada por tres bloques: adaptar la aplicación inicial para que use la tecnología de Bluetooth para transmitir la información entre servidor y cliente, creación de una aplicación de aprendizaje federado que usara imágenes como data para entrenar los modelos en vez de grabaciones de audio y observar su rendimiento, y por último, conseguir utilizar un microcontrolador como servidor de la aplicación de aprendizaje federado.

El primer bloque en ser tratado fue la adaptación de la aplicación para que usara la tecnología Bluetooth. Como se ha comentado anteriormente en los trabajos futuros del proyecto, este bloque generó muchos problemas al principio debido a las librerías de funciones escogidas para la realización del trabajo, siendo estas librerías incompatibles con la tecnología BLE (Bluetooth Low Energy) implementada en el microcontrolador. Estos problemas se solucionaron con el uso de la librería Bleak, la cual ya permitió poder establecer una conexión Bluetooth con el microcontrolador. A partir de este punto se empezó a investigar los límites que ofrece esta librería y en qué medida afectan estos a la aplicación. Fue entonces que se observó la gran diferencia en el tamaño de los mensajes que se podían enviar y los problemas que esto comportaba. Por motivos de tiempo disponible se decidió abandonar este primer bloque del segundo objetivo para poder investigar otros campos diferentes de la aplicaciones de aprendizaje federado en microcontroladores, pero en consecuencia a la investigación realizada en este campo se decidió dejar como trabajo futuro del proyecto, ya que, aunque no se consiguió reproducir la aplicación de Bluetooth se

observó que era totalmente posible la realización de esta. Para ello, se tendrían que adaptar la cantidad de mensajes y sus tamaños para poder enviar toda la información necesaria y usar protocolos de comunicación para poder asegurar que no hay pérdidas de paquetes en el envío de los datos.

El segundo bloque correspondiente al segundo objetivo consistía en la realización de una aplicación de aprendizaje federado en microcontroladores que utilizara imágenes como data para entrenar los modelos locales de los clientes. Para ello se usó la cámara integrada en el Kit TinyML de Arduino, la cual ya venía preparada para poder ser usada sin problema con el Arduino 33 BLE Sense gracias a un shield que permitía la conexión entre los dos. Con cambios en la obtención de la data y las características de esta para entrenar el modelo local de los microcontroladores se consiguió tener una aplicación funcional capaz de entrenar el modelo a través de las imágenes.

Una vez se realizaron las pruebas en la aplicación se observó que en ciertas situaciones el comportamiento de esta era correcto y funcionaba sin problemas, pero en otras no era capaz de conseguir los resultados esperados generando más error del esperado en las predicciones. Pese a ello, la aplicación es capaz de diferenciar en la mayoría de las veces las clases con casos sencillos como los colores, aunque como se ha comentado anteriormente algunos casos es posible que confunda dos clases en algunas predicciones. Este comportamiento sería debido a que la red neuronal escogida para la aplicación no era la óptima para este tipo de aplicaciones relacionadas con las imágenes. Tras investigar sobre diferentes tipos de redes neuronales se llegó a la conclusión que la mejor opción habría sido la utilización de una red convolucional, las cuales disponen de unas capas extras en la red encargadas de resaltar contornos y otras características más relevantes en las imágenes. Por tanto se ha decidido tener como trabajo futuro del proyecto la investigación y reproducción de una red convolucional la cual se espera que aumente la precisión de las predicciones de la aplicación.

Como último bloque del segundo objetivo encontramos la utilización de un microcontrolador como el servidor del aprendizaje federado encargado de comunicarse con todos los clientes y de obtener sus modelos locales para poder desarrollar uno global. Debido a que uno de los requisitos para poder investigar este bloque era la necesidad de disponer de una aplicación funcional que usará la tecnología de Bluetooth para transmitir la información. Al no disponer de este requisito no se pudo investigar a fondo este bloque y el alcance de esta posibilidad.

Por tanto, en cuanto a los objetivos genéricos del proyecto, es correcto afirmar que se ha logrado el primer objetivo satisfactoriamente, pero el segundo objetivo no, ya que solo un bloque de los tres planteados inicialmente ha conseguido dar resultados.

Como conclusión final del proyecto podemos afirmar que es posible crear aplicaciones de aprendizaje federado en TinyML que usen imágenes como data y que la tecnología de Bluetooth es posible de aplicar pero requiere de adaptaciones importantes en la fase de comunicación de las aplicaciones que se quieran adaptar para el uso de esta tecnología. Se espera que TinyML siga creciendo en el campo del aprendizaje automático y con ello la diversidad en las aplicaciones de este sector, con ello espero que el contenido de este proyecto sea de ayuda para la inicialización en el campo de las aplicaciones relacionadas con imágenes y BLE (Bluetooth Low Energy).

## 11. Referencias

---

- [1] Marc Monfort. TinyML: From Basic to Advanced Applications. Trabajo de fin de grado, UPC, Facultat de Informàtica de Barcelona, 2021 [Consulta: 27 de septiembre 2021]. Disponible en: <https://github.com/MarcMonfort/TinyML-FederatedLearning>
- [2] Reddi, V. J., Moroney Laurence, & Pete Warden. (n.d.). edX: Tiny Machine Learning (TinyML) Professional Certificate | edX. [Consulta: 27 de septiembre 2021]. Disponible en : <https://www.edx.org/professional-certificate/harvardx-tiny-machine-learning>
- [3] Jitsi: Free Video Conferencing Software for Web & Mobile | Jitsi. (2021) Disponible en : <https://jitsi.org>
- [4] GitHub: Where the world builds software | GitHub. (2021) Disponible en : <https://github.com>
- [5] Software Engineer Salary in Spain | PayScale. [Consulta: 09 de octubre 2021]. Disponible en: [https://www.payscale.com/research/ES/Job=Software\\_Engineer/Salary](https://www.payscale.com/research/ES/Job=Software_Engineer/Salary)
- [6] Project Manager Salary in Spain | PayScale. [Consulta: 09 de octubre 2021]. Disponible en: [https://www.payscale.com/research/US/Job=Project\\_Manager%2C\\_\(Unspecified\\_Type\\_%2F\\_General\)/Salary](https://www.payscale.com/research/US/Job=Project_Manager%2C_(Unspecified_Type_%2F_General)/Salary)
- [7] Calendario laboral 2020: festivos y puentes | ElMundo. [Consulta: 09 de octubre 2021]. Disponible en: <https://www.elmundo.es/economia/2020/04/10/5e907073fc6c8326248b45db.html>
- [8] Bases y tipos de cotización 2021 | Seguridad Social. [Consulta: 09 de octubre 2021]. Disponible en: <https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537#36538>
- [9] Arduino Tiny Machine Learning Kit | Arduino Store. [Consulta: 09 de octubre 2021]. Disponible en: <https://store.arduino.cc/products/arduino-tiny-machine-learning-kit>

- [10] ELEGOO Kit Componentes Electrónicos | Amazon. [Consulta: 09 de octubre 2021]. Disponible en: [https://www.amazon.es/gp/product/B01N03JEZ9/ref=ppx\\_yo\\_dt\\_b\\_asin\\_title\\_o00\\_s0\\_1?ie=UTF8&psc=1](https://www.amazon.es/gp/product/B01N03JEZ9/ref=ppx_yo_dt_b_asin_title_o00_s0_1?ie=UTF8&psc=1)
- [11] Aticco Urquinaona | Coworkingspain. [Consulta: 09 de octubre 2021]. Disponible en: <https://coworkingspain.es/espacios/coworking/barcelona/aticco-urquinaona>
- [12] Sistema Embebido | Wikipedia. [Consulta: 3 de abril 2022]. Disponible en: [https://es.wikipedia.org/wiki/Sistema\\_embebido](https://es.wikipedia.org/wiki/Sistema_embebido)
- [13] What is PlatformIO? | PlatformIO. [Consulta: 7 de abril 2022]. Disponible en: <https://docs.platformio.org/en/latest/what-is-platformio.html>
- [14] ¿En qué consiste un SDK? | IONOS. [Consulta: 8 de abril 2022]. Disponible en: [https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/software-development-kit/#:~:text=Un%20software%20development%20kit%20\(SDK,sistema%20en%20tiempo%20de%20ejecución.](https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/software-development-kit/#:~:text=Un%20software%20development%20kit%20(SDK,sistema%20en%20tiempo%20de%20ejecución.)
- [15] Edge Impulse | GitHub. [Consulta: 19 de marzo 2022]. Disponible en: <https://github.com/orgs/edgeimpulse/repositories>
- [16] 0.3MP: OV7675 | ArduCam. [Consulta 11 de abril 2022]. Disponible en : <https://www.arducam.com/products/camera-breakout-board/0-3mp-ov7675/>
- [17] ACK | Wikipedia. [Consulta 12 de abril 2022]. Disponible en: <https://es.wikipedia.org/wiki/ACK>
- [18] A neural network for arduino | hobbizine. [Consulta 11 de abril 2022]. Disponible en: <http://robotics.hobbizine.com/arduinoann.html>
- [19] Función Sigmoide | Wikipedia. [Consulta 13 de abril 2022]. Disponible en: [https://es.wikipedia.org/wiki/Función\\_sigmoide](https://es.wikipedia.org/wiki/Función_sigmoide)
- [20] Bleak | pypi. [Consulta 8 de abril 2022]. Disponible en: <https://pypi.org/project/bleak/>

## Anexo Código Bluetooth

```

class Connection:

    client: BleakClient = None

    def __init__(
        self,
        loop: asyncio.AbstractEventLoop,
        read_characteristic: str,
        write_characteristic: str,
        data_dump_handler: Callable[[str, Any], None],
        data_dump_size: int = 256,
    ):
        self.loop = loop
        self.read_characteristic = read_characteristic
        self.write_characteristic = write_characteristic
        self.data_dump_handler = data_dump_handler

        self.last_packet_time = datetime.now()
        self.dump_size = data_dump_size
        self.connected = False
        self.connected_device = None

        self.rx_data = []
        self.rx_timestamps = []
        self.rx_delays = []

    def on_disconnect(self, client: BleakClient, future: asyncio.Future):
        self.connected = False
        # Put code here to handle what happens on disconnect.
        print(f"Disconnected from {self.connected_device.name}!")

    async def cleanup(self):
        if self.client:
            await self.client.stop_notify(read_characteristic)
            await self.client.disconnect()

    async def manager(self):
        print("Starting connection manager.")
        while True:
            if self.client:
                await self.connect()
            else:
                await self.select_device()
                await asyncio.sleep(15.0, loop=loop)

    async def connect(self):
        if self.connected:
            return
        try:
            await self.client.connect()
            self.connected = await self.client.is_connected()
            if self.connected:
                print(f"Connected to {self.connected_device.name}")
                self.client.set_disconnected_callback(self.on_disconnect)
                await self.client.start_notify(
                    self.read_characteristic, self.notification_handler,
                )
            while True:
                if not self.connected:
                    break
                await asyncio.sleep(3.0, loop=loop)
        except Exception as e:
            print(e)

    async def select_device(self):
        print("Bluetooth LE hardware warming up...")
        await asyncio.sleep(2.0, loop=loop) # Wait for BLE to initialize.
        devices = await discover()

        print("Please select device: ")
        for i, device in enumerate(devices):
            print(f"{i}: {device.name}")

        response = -1
        while True:
            response = await ainput("Select device: ")
            try:
                response = int(response.strip())
            except:
                print("Please make valid selection.")

            if response > -1 and response < len(devices):
                break
            else:
                print("Please make valid selection.")

        print(f"Connecting to {devices[response].name}")
        self.connected_device = devices[response]
        self.client = BleakClient(devices[response].address, loop=self.loop)

```

Fig 44 y 45: Clase Connection usada en el servidor para conectar microcontroladores por Bluetooth Low Energy

```
if __name__ == "__main__":
    # Create the event loop.
    loop = asyncio.get_event_loop()

    read_characteristic = "00001143-0000-1000-8000-00805f9b34fb"
    write_characteristic = "00001142-0000-1000-8000-00805f9b34fb"

    data_to_file = DataToFile(output_file)
    connection = Connection(
        loop, read_characteristic, write_characteristic, data_to_file.write_to_csv
    )

    devices = [connection]

    size_hidden_layer = (650+1)*16
    size_output_layer = (16+1)*3

    np.random.seed(12345)
    hidden_layer = np.random.uniform(-0.5, 0.5, (650+1)*16).astype('float32')
    output_layer = np.random.uniform(-0.5, 0.5, (16+1)*3).astype('float32')

    # To load a Pre-trained model
    # hidden_layer = np.load("./hidden_montserrat.npy")
    # output_layer = np.load("./output_montserrat.npy")

    for d in devices:
        init_network(hidden_layer, output_layer, d)

    devices_connected = devices

    try:
        asyncio.ensure_future(connection.manager())
        asyncio.ensure_future(main())
        loop.run_forever()
    except KeyboardInterrupt:
        print()
        print("User stopped program.")
    finally:
        print("Disconnecting...")
        loop.run_until_complete(connection.cleanup())
```

Fig 46: Código main de la aplicación Bluetooth del servidor

```

// Device name
const char* nameOfPeripheral = "MicrophoneMonitor";
const char* uuidOfService = "00001101-0000-1000-8000-00805f9b34fb";
const char* uuidOfRxChar = "00001142-0000-1000-8000-00805f9b34fb";
const char* uuidOfTxChar = "00001143-0000-1000-8000-00805f9b34fb";

// BLE Service
BLEService microphoneService(uuidOfService);

// Setup the incoming data characteristic (RX).
const int WRITE_BUFFER_SIZE = 256;
bool WRITE_BUFFER_FIZED_LENGTH = false;

// RX / TX Characteristics
BLECharacteristic rxChar(uuidOfRxChar, BLEWriteWithoutResponse | BLEWrite, WRITE_BUFFER_SIZE, WRITE_BUFFER_FIZED_LENGTH);
BLEByteCharacteristic txChar(uuidOfTxChar, BLERead | BLENotify | BLEBroadcast);

```

Fig 47: variables necesarias en el arduino para usar el Bluetooth Low Energy

```

// Start BLE.
startBLE();

// Create BLE service and characteristics.
BLE.setLocalName(nameOfPeripheral);
BLE.setAdvertisedService(microphoneService);
microphoneService.addCharacteristic(rxChar);
microphoneService.addCharacteristic(txChar);
BLE.addService(microphoneService);

// Bluetooth LE connection handlers.
BLE.setEventHandler(BLEConnected, onBLEConnected);
BLE.setEventHandler(BLEDDisconnected, onBLEDDisconnected);

// Let's tell devices about us.
BLE.advertise();

// Print out full UUID and MAC address.
Serial.println("Peripheral advertising info: ");
Serial.print("Name: ");
Serial.println(nameOfPeripheral);
Serial.print("MAC: ");
Serial.println(BLE.address());
Serial.print("Service UUID: ");
Serial.println(microphoneService.uuid());
Serial.print("rxCharacteristic UUID: ");
Serial.println(uuidOfRxChar);
Serial.print("txCharacteristics UUID: ");
Serial.println(uuidOfTxChar);

Serial.println("Bluetooth device active, waiting for connections...");

```

Fig 48: fase de inicialización del Bluetooth Low Energy en el microcontrolador