Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

TRABAJO FIN DE GRADO

Reconocimiento de gestos y voz mediante inteligencia artificial incorporada en un microcontrolador

Gesture and voice recognition by using machine learning on an embedded processor

Autor:

Roberto Aldea Cebollo

Director:

Bonifacio Martin del Brío

Grado en Ingeniería Electrónica Industrial y Automática

Departamento de Ingeniería Electrónica y Comunicaciones

Escuela de Ingeniería y Arquitectura

Universidad de Zaragoza

Resumen

En la actualidad, la inteligencia artificial se puede encontrar cada vez en un mayor número de dispositivos sin que los usuarios sean conscientes de ello, un ejemplo es el uso de una palabra para activar un dispositivo y ejecutar acciones más complejas, tal como realizan '*Oye Siri*' o '*Alexa*'.

En este trabajo, por un lado, se estudia la implementación de algoritmos de inteligencia artificial en un dispositivo basado en un microcontrolador estándar de bajo coste y recursos de procesamiento limitados. En concreto se va a utilizar la placa Arduino Nano 33 BLE, desarrollada específicamente para este tipo de aplicaciones, la cual incluye diversos sensores de voz y movimiento. El objetivo es explorar nuevas posibilidades de interacción humano-computador, haciendo uso de un dispositivo de interacción para reconocer comandos de voz y / o movimientos gestuales. A modo de demostrador, este proyecto se ha centrado en realizar los controles más básicos a la hora de llevar a cabo una presentación usando un programa de presentaciones tipo Microsoft PowerPoint o similar.

Como segundo objetivo, se plantea el estudio y uso de la plataforma online *Edge Impulse* para el desarrollo del proyecto completo, desde la captura de datos y generación del *dataset*, extracción de características, desarrollo de modelos de aprendizaje automático, entrenamiento, evaluación e implementación en el microcontrolador. El proceso completo de desarrollo de sistemas inteligentes basados en microcontroladores se realiza desde dicha plataforma de forma totalmente online, desde un navegador de internet.

Como conclusiones podemos destacar que uno de los aspectos más importantes a la hora de crear un modelo de inteligencia artificial es tener una base de datos lo suficientemente amplia para entrenar adecuadamente el clasificador, en nuestro caso, hemos desarrollado nuestras propias bases de datos. Por otro lado, comprobamos que el tipo de extracción de características que se realice sobre los datos puede ser más importante que el modelo clasificador concreto para obtener los mejores resultados.

Abstract

Nowadays, artificial intelligence can be found in an increasing number of devices without users being aware of its integration, a little-known example is the use of a word to activate a device and perform more complex actions, as performed by 'Hey Siri' or 'Alexa'.

In this work it is proposed to combine artificial intelligence with a device that combines voice and/or motion sensors and with limited computing processing resources. The aim is to be able to control different elements of a computer (human machine interface) by distinguishing different commands (voice and gestures) through the mentioned sensors.

This project has focused on performing the most basic controls when conducting a presentation using Microsoft PowerPoint. The project has been done through the online platform Edge Impulse (EI), using the Arduino Nano 33 BLE device.

One of the main targets of this work is explore the possibilities of the online tool Edge Impulse, for developing through a web navigator intelligent system implemented on embedded devices.

One of the most important aspects when creating an artificial intelligence model is to have a large enough database for the model to be able to perform the classification correctly. The type of processing performed on the data and the characteristic of the model is also another key factor in order to obtain the best results.

Índice

Tabla de figuras

Capítulo 1: Introducción

1.1 Contexto

Según la RAE, la inteligencia artificial es la "disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico" [1]. Actualmente, la inteligencia artificial se encuentra en una gran variedad de equipos que se usan de forma cotidiana, incluyendo los dispositivos inteligentes, con el principal objetivo de contextualizar los datos obtenidos o automatizar acciones sin la necesidad de la intervención humana.

En este proyecto se va a trabajar con algoritmos de *Machine Learning* (ML), traducido al castellano como aprendizaje automático, rama de la inteligencia artificial de la que parten muchos otros conceptos como *Deep Learning* o redes neuronales. El ML es un tipo de aprendizaje automático centrado en el uso de datos y algoritmos para que una máquina imite la forma de aprender de los seres humanos, es decir de forma gradual.

Una variante dentro del ML es el *Deep Learning* y dado que es un concepto que se usa sin distinción con el ML, hay que aclarar las diferencias entre ellos. El *Deep Learning* es una mejora respecto al ML ya que, al basarse en redes neuronales mucho más complejas, permite automatizar el proceso de extracción de características (etapa que hay que realizar manualmente en el resto de los algoritmos de *machine learning*), permitiendo además el uso de bases de datos de un mayor volumen para entrenar el modelo.

En cuanto a otros conceptos que vamos a manejar en este trabajo, en inteligencia artificial existe el aprendizaje supervisado, el aprendizaje no supervisado, aprendizaje semi supervisado y el aprendizaje por refuerzo [2]. La principal diferencia entre el supervisado y el no supervisado está en los datos, ya que el primero cuenta con datos que han sido etiquetados, que es lo que se pretende predecir, mientras que los segundos no cuentan con un etiquetado, por lo que el algoritmo deberá extraer las diferencias entre los datos. El aprendizaje semi supervisado por lo tanto es una mezcla de los dos anteriores, ya que se etiqueta una pequeña parte de los datos para que después, a través de un modelo con aprendizaje supervisado, se obtenga la etiqueta del resto de datos; finalmente se entrena un algoritmo de aprendizaje supervisado que utilice las etiquetas generadas tanto manualmente como las generadas por el modelo anterior. Por último, está el aprendizaje por refuerzo que es un método de aprendizaje automático que se basa en recompensar los comportamientos deseados y penalizar los no deseados.

Sin duda alguna, la industria tecnológica está teniendo cada vez un impacto más grande en la vida cotidiana. Los cambios y avances crecen de forma constante y ocurren a nuestro alrededor, tal y como se puede observar en los teléfonos, coches, casi cualquier dispositivo calificado como inteligente y todos los *gadgets* que adquirimos y usamos para mejorar tanto la conectividad y la eficiencia como nuestro bienestar. Por lo que conseguir implementar un modelo de inteligencia artificial en un dispositivo lo suficientemente pequeño tiene un grandísimo potencial, que muchas empresas están tratando de explotar.

De esta idea surge el *Tiny Machine Learning* (TinyML), que se puede definir como el conjunto de tecnologías que incluyen ML y sistemas embebidos para hacer uso de aplicaciones inteligentes en dispositivos pequeños y de muy bajo consumo. En el TinyML se considera de muy bajo consumo a aquel dispositivo o equipo que no supere 1mW, con todo lo que ello supone [3], lo que equivale a disponer de un dispositivo operativo durante aproximadamente un año con una pila de tipo botón.

Estos dispositivos generalmente se basan en microcontroladores y supondría contar con un dispositivo para poder colocarlo en cualquier tipo de ubicación con nula intervención humana durante tiempos prolongados de tiempo. Este tipo de dispositivos perciben el entorno mediante sensores y ejecutan el algoritmo de ML en base a los datos recibidos, pero poseen unos recursos muy limitados en cuanto a memoria y capacidades computacionales.

Además de todo lo mencionado anteriormente, otra razón para tratar de incorporar un algoritmo con inteligencia artificial en un dispositivo como un microcontrolador es la popularidad que tienen estos en campos tan variados como la industria automovilística, telecomunicaciones, sanidad, aplicaciones culinarias y electrónica de consumo. Lo que demuestra que un microcontrolador podemos encontrarlo en cualquier sitio actualmente, casi invisibles en nuestro día a día.

En este trabajo vamos a experimentar con la implementación de algoritmos de ML en microcontroladores para realizar reconocimiento de voz y de gestos, para valorar su uso en el desarrollo de dispositivos de interacción humano-computador inteligentes, que faciliten al usuario dicha interacción.

El concepto de reconocimiento de voz se puede describir como el proceso por el cual un ordenador mapea una señal acústica formando un significado abstracto y cuyo objetivo principal es traducir el sonido a texto y/u órdenes. Las técnicas de reconocimiento de voz se pueden clasificar de varias maneras [4]:

- Basadas en el tipo de declaraciones: esto hace referencia al tipo de entradas que va a recibir el modelo, pudiendo ser un dictado de palabras sueltas, palabras con una pausa mínima entre ellas o un dictado continuo, siendo este último el más complejo.

- Basadas en el número de registros: si existe más de un registro, entendido como personas y sus respectivas voces, el modelo será capaz de ser mucho más robusto a la hora de capturar y clasificar las palabras, mientras que si solo hay un registro probablemente si se somete a un registro diferente o no sea capaz de clasificarlo correctamente o lo clasificara como desconocido.

- Basadas en el vocabulario: esto implica el número de palabras diferentes que se han registrado en la base de datos, teniendo 4 tamaños dependiendo del número.

En cuanto al reconocimiento de gestos, es una de las maneras más sencillas de realizar la interacción humano-ordenador, y más teniendo en cuenta que los gestos son comúnmente usados en el día a día. Además de los habituales teclado y ratón existen diferentes técnicas alternativas ("inteligentes") para implementar dicha interacción, como por ejemplo la visión por computador, a través de gestos táctiles o mediante el reconocimiento de movimientos, usando generalmente un acelerómetro.

1.2 Objetivos

Como se ha dicho antes, el objetivo fundamental de este trabajo es experimentar con la implementación de algoritmos de ML en microcontroladores (TinyML) para realizar reconocimiento de voz y de gestos, para valorar su uso en el desarrollo de dispositivos de interacción humano-computador inteligentes.

Y un segundo objetivo general es experimentar con las nuevas herramientas recientemente introducidas para el desarrollo de *machine learning* en microcontroladores, como es el caso de *Edge Impulse*, herramienta online (https://www.edgeimpulse.com/) accesible desde un navegador web, que permite el desarrollo completo de un proyecto de TinyML: captura de datos, extracción de características, entrenamiento del modelo, evaluación e implementación en el microcontrolador.

Como la interacción humano-computador es un campo muy amplio, en este trabajo, a modo de demostrador, nos vamos a centrar en los pasos iniciales que podrían llevar al desarrollo de un dispositivo que pueda ser capaz de sustituir a un ratón (en su uso básico) cuando un orador tiene que realizar una presentación ante el público con un programa tipo PowerPoint o similar. Para ello, se han planteado los siguientes objetivos más concretos:

- Generar una base de datos, tanto de voz como de gestos, lo suficientemente grande (dentro de las limitaciones temporales que un trabajo fin de grado conlleva) para que el algoritmo sea capaz de aprender y generalizar.
- Analizar las bases de datos realizando diferentes tipos de preprocesamiento, con el fin de obtener la mejor separación de los datos para que le resulte más sencillo al modelo encontrar las diferencias y, por tanto, obtener mejores resultados.
- Investigar los diferentes modelos disponibles para las diferentes bases de datos y que sea capaz de ejecutarse en el microcontrolador seleccionado con la mayor eficiencia y precisión posible.
- Implementar tanto el preprocesado como el modelo en el microcontrolador y comprobar que funciona tal y como se esperaba con buenos resultados.
- Realizar la comunicación entre el microcontrolador y el ordenador, con el propósito de realizar diferentes acciones en función del resultado obtenido mediante el microcontrolador.

1.3 Planificación

Para poder realizar los objetivos planteados anteriormente se han propuesto una serie de tareas para realizar de forma gradual, haciendo una división por temática.

- Etapas iniciales:

    1. Estudio y funcionamiento del dispositivo, que incluye microcontrolador y sensores: se pretende obtener el funcionamiento detallado de los sensores que se van a usar durante el proyecto, así como entender el entorno de trabajo.
    2. Estudio de los modelos de ML: se van a estudiar los diferentes modelos que funcionan mejor con cada estructura de datos y las diferentes posibilidades que se podrían implementar.
    3. Estudio de la plataforma de desarrollo: en paralelo a lo anterior, se estudian todas las opciones que tiene la plataforma en cuanto a procesado y modelado del algoritmo, así como las opciones de implementación en el dispositivo.
    4. Estudio de la comunicación entre equipos: dado que se va a realizar la clasificación de los gestos o comandos de voz en el dispositivo, estos van a tener que ser comunicados de alguna manera al ordenador para que este sea capaz de ejecutar las ordenes asociadas a dichos comandos, del tipo que sean, por lo que se realiza un estudio de las alternativas de comunicación y ejecución de los comandos.

- Desarrollo:

    1. Generación de la base de datos: se ha de generar una base de datos lo suficientemente grande, además se deben etiquetar datos para que el modelo sea capaz de aprender.
    2. Procesado, entrenamiento y testeo: realizando los procesamientos estudiados anteriormente y probando diferentes valores y tipos para obtener la mejor clasificación de los datos. Tras el procesamiento se realizan pruebas con diferentes modelos para obtener la mejor exactitud.
    3. Programación de la comunicación con el dispositivo: tras tener un modelo funcional en la aplicación web, se comprueba el funcionamiento real en el dispositivo realizando la comunicación con el ordenador.

4. Memoria: se documenta todo lo realizado anteriormente y se realiza una comparativa de los diferentes resultados.

- Comprobaciones:

1. Comprobación del modelo y predicciones: se realizan pruebas para verificar que el modelo seleccionado funciona lo mejor posible, además se intenta realizar alguna mejora en los modelos.

2. Comprobación de la comunicación: se comprueba que no haya fallos de comunicación entre el programa desarrollado y el dispositivo, mientras este último realiza las inferencias y envía por el puerto serie los resultados.

El tiempo empleado aproximado queda reflejado en el cronograma de la Figura 1, aunque las fechas eran orientativas cuando se empezó el proyecto.

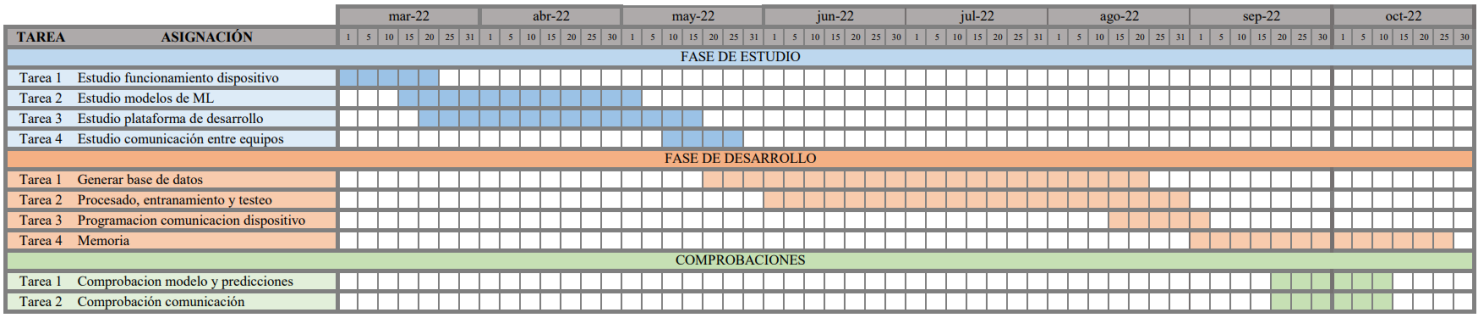


Figura 1. Diagrama de Gant del proyecto

1.4 Guía de la memoria

La memoria se estructura de la siguiente manera. En el segundo capítulo se describen brevemente las herramientas utilizadas, la placa Arduino Nano 33 BLE y la plataforma online *Edge Impulse*. En el tercer capítulo se explica el trabajo realizado en relación con el reconocimiento de comandos de voz, desde la generación de la base de datos hasta la realización del modelo, pasando por la extracción de parámetros de los datos. En el capítulo cuatro se realiza lo propio, pero en este caso para el reconocimiento de movimiento/gestos, con la diferenciación de los modelos usando movimientos continuos y movimientos individuales y por tanto más complejos. El capítulo quinto va a consistir en un análisis de los resultados obtenidos en cada modelo de reconocimiento, así como las ventajas y desventajas de cada uno. Por último, en el capítulo final se presentan las conclusiones, así como posibles líneas futuras y mejoras.

Capítulo 2: Hardware y Software para TinyML

2.1 Placa Arduino Nano 33 BLE Sense

En el mercado existen diversas placas que incluyen microcontroladores de 32 bits, de muy bajo consumo y coste relativamente reducido, que son adecuadas para experimentar con el TinyML y desarrollar prototipos [3].

Figura 2. Arduino Nano 33 BLE Sense

Algunas de estas placas son la Arduino Nano 33 BLE Sense (Figura 2), SparkFun Edge board y la STM32F746G Discovery kit. Estas placas disponen de ejemplos en internet, así como libros que guían a lo largo de diferentes proyectos para implementar algoritmos de TinyML y amplias comunidades en las que preguntar en caso de que surjan dudas. La ventaja de estas placas es que muchos fabricantes están empezando a construirlas con los sensores típicos utilizados en proyectos de TinyML junto con aplicaciones para *Internet of Things* (IoT), tal y como es el caso de las placas mencionadas de SparkFun y Arduino.

Para este trabajo se ha escogido la Arduino Nano 33 BLE Sense ya que es un dispositivo de bajo coste y fácil adquisición. Además, incluye todos los sensores necesarios a la hora de desarrollar los modelos que se van a plantear, como son un micrófono y un acelerómetro, tal y como se indica en la Figura 3.

Las características propias de la Arduino Nano 33 BLE Sense, además de los sensores, incluyen uno de los últimos estándares en Bluetooth, el 5.0. En cuanto al microcontrolador, incluye un NINA-b3 de 32 bits que funciona a 64MHz e incluye 1MB de Flash y 256KB de SRAM. También ofrece compatibilidad con las interfaces típicas como son SPI, I2C, I2S, UART y USB, así como pines digitales, analógicos y compatibles con salida PWM.

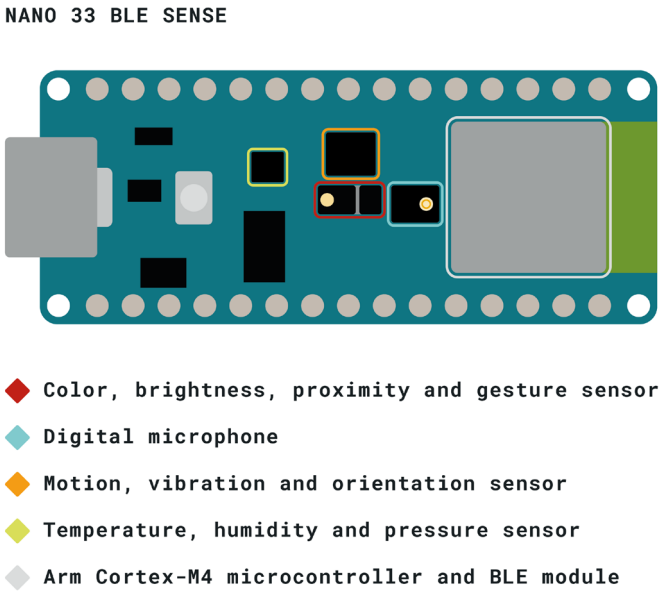


Figura 3. Foto y diagrama de bloques simplificado de Arduino Nano 33 BLE Sense

Otro motivo de elegir esta placa es que la plataforma donde se va a desarrollar los proyectos es compatible con ella, y también ofrece ayudas y diferentes ejemplos a realizar específicamente en esta placa. Para más información acerca de los dispositivos soportados por la plataforma recurrir al Anexo I.

2.2 Plataforma Edge Impulse

Los algoritmos de *machine learning* hoy en día se desarrollan a menudo en entorno Python. Las famosas librerías TensorFlow de Google son unas de las más utilizadas y cuentan con una versión (TensorFlow Lite) que soporta diversos dispositivos para poder realizar proyectos de TinyML [3].

No obstante, en este trabajo vamos a experimentar con la plataforma Edge Impulse (https://www.edgeimpulse.com/), una herramienta online que permite desarrollar sistemas inteligentes TinyML directamente desde un navegador web, cubriendo todas las etapas requeridas: captura de datos, extracción de características, entrenamiento del modelo e implementación en el microcontrolador.

Edge Impulse (EI) está pensado para facilitar el uso de aprendizaje automático en aplicaciones empotradas, al permitir a los desarrolladores crear y optimizar soluciones con datos del mundo real, de una forma relativamente intuitiva gracias a un entorno que integra todas las etapas de diseño. Ello permite que el proceso de creación, despliegue y escalado de aplicaciones de ML empotradas sea más fácil y rápido, facilitando a los ingenieros el desarrollo de dispositivos inteligentes [5].

Al crear un proyecto, este solicita que se especifique a que está destinado, o lo que es lo mismo, que tipo de datos va a usar como entrada, con 4 opciones: datos de un acelerómetro, datos de audio, imágenes y otros. En este proyecto se han manejado tanto datos del acelerómetro como de audio.

La herramienta dispone en todo momento de un menú lateral en el que aparecen todos los accesos directos y en el orden que hay que seguir para conseguir un modelo y poder implementarlo en un dispositivo.

La herramienta ha denominado *impulse* a los proyectos que se generan desde su servicio e incluyen el preprocesamiento y el algoritmo. Una particularidad sobre EI es que a la hora de generar el modelo, aunque a primera vista se realiza a través de bloques visuales, en realidad estamos generando código Python y usando desde un nivel de bloques las librerías mencionadas anteriormente, como TensorFlow y TensorFlowLite, pero con la ventaja de tener todo en un mismo entorno y ofrecer resultados que son mucho más fácil de visualizar.

En el Anexo I se explica más ampliamente la herramienta online Edge Impulse y se describe su uso básico, así como una guía paso a paso para obtener un modelo de forma rápida.

Capítulo 3: Reconocimiento de voz

3.1 Desarrollo de la base de datos

Las etapas en el desarrollo del reconocedor de comandos de voz son las siguientes: desarrollo de la base de datos (Apartado 3.1), análisis y extracción de características (Apartado 3.2), selección del modelo clasificador, evaluación del modelo e implementación en el microcontrolador (Apartado 3.3).

La base de datos (*dataset*) es fundamental a la hora de realizar un buen modelo de inteligencia artificial. Una de las maneras de obtener una base de datos robusta es tener un equilibrio de todas las clases que se desean clasificar (*balanced clases*).

A pesar de que se trata de un proceso muy laborioso, en este trabajo hemos optado por desarrollar nuestra propia base de datos en vez de utilizar una de las muchas disponibles en internet. Para generar nuestra base de datos para reconocer voz hemos obtenido muestras generalmente de un único sujeto (el autor). Dado que el resultado final iba a realizarse con la placa de Arduino Nano 33 BLE, tomamos las muestras con el micrófono incluido en dicha placa. Por otro lado, comprobamos que si se tomaban muestras con otro dispositivo (smartphone), éstas eran muy diferentes, al tener un mejor micrófono y amplificador, las señales obtenidas eran mucho menos ruidosas. Ello se muestra en la Figura 4 y Figura 5, siendo respectivamente un audio obtenido mediante Arduino y mediante un teléfono de la misma palabra repetida varias veces con pausas entre una repetición y otra.

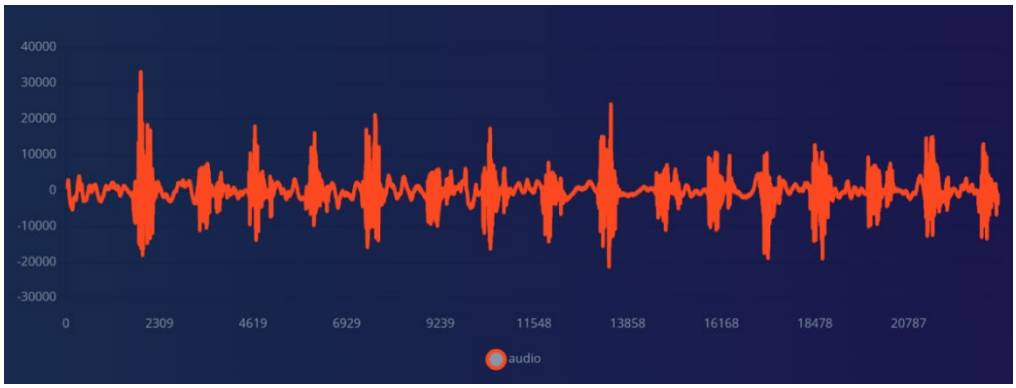


Figura 4. Audio grabado mediante Arduino

Figura 5. Audio grabado mediante iPhone

En caso de haber generado la base de datos a partir de un teléfono móvil, se habrían podido obtener muestras de un mayor número de sujetos y, por tanto, sería mucho más robusta (más independiente del hablante), pero dado que posteriormente se habría implementado en la placa de Arduino, que tiene un micrófono diferente y genera audios de características diferentes, el algoritmo hubiera tenido mucho peor rendimiento.

En definitiva, como el objetivo era generar un prototipo-demostrador de las posibilidades de reconocer audio en dispositivos tipo TinyML, se ha optado por generar una base de datos exclusivamente con los recursos hardware integrados en la placa Arduino Nano 33 BLE, de manera que buena parte de las muestras pertenecen al mismo sujeto (el autor del TFG).

Tal y como se indica en muchos libros [3], a la hora de hacer una clasificación simple entre dos sonidos, disponer de tres y cinco minutos de cada clase es suficiente. Pero a medida que se desea realizar una clasificación con una mayor cantidad de clases, aumenta de igual manera la dificultad y por tanto debe aumentar el número de muestras que se dispone de cada clase.

Esta base de datos está compuesta por 4 palabras una para cada comando que se va a usar en el modelo. Pensando en el manejo de un programa de presentaciones, como PowerPoint, dichas palabras van a ser: Inicio, Avanza, Blanco y Volver. Además, también es necesaria una clase adicional, calificada como Ruido, en la que hay muestras tanto de sonidos ambiente, dentro de una casa y de la calle, así como de otras palabras aleatorias, lo cual mejorará la futura clasificación.

Para facilitar la creación de la base de datos se han tomado muestras de aproximadamente treinta segundos y posteriormente se cortaban los audios en segmentos de un segundo, ya que todas las palabras son cortas y es tiempo más que suficiente. Exceptuando los audios clasificados como Ruido, en los que no era necesario realizar cortes ya que posteriormente también se recorren todos los audios y se cortan en tramos de un segundo, con desplazamientos de un segundo entre corte y corte. En la Figura 6 se muestra el número de muestras que existen por etiqueta, con la excepción de la etiqueta ruido dado que algunos audios no han sido cortados. En general salen unas 540 muestras para entrenamiento y unas 130 para test.

	Train	Test
Avanza	548	133
Inicio	542	134
Blanco	550	134
Volver	544	135
Ruido	218	126

Figura 6. Número de muestras de cada etiqueta de voz

Otro punto importante es la división de la base de datos en subconjuntos de entrenamiento y prueba, con el fin de utilizar muestras diferentes para entrenar y para evaluar el rendimiento de los algoritmos de aprendizaje automático. En este proyecto la división, tal y como recomiendan en los artículos de guía de la plataforma *Edge Impulse* [5], es aproximadamente de un 80% para entrenamiento y un 20% para evaluación respectivamente. No obstante, tal y como se explicará más adelante, a la hora de entrenar se volverá a realizar una división de los datos adicional, de manera que en realidad serán tres los subconjuntos (entrenamiento, validación y evaluación). El objetivo es generar un modelo que generalice bien, para lo cual se debe entrenar (ajustar los parámetros o pesos del modelo) con un conjunto de entrenamiento, ajustar sus hiperparámetros (parámetros de configuración de su estructura) con un conjunto de validación y realizar la evaluación final con un conjunto de test diferente a los dos anteriores. Como se ha dicho, el conjunto de entrenamiento y el de validación suele comprender el 80% de las muestras y el de test final el 20% restante. Además, las muestras deben repartirse entre ellos de forma aleatoria respetando la proporción de cada clase, de tal forma que se mantenga dentro de cada una de las clases la relación anteriormente mencionada de 80/20 (muestreo estratificado).

La plataforma *Edge Impulse* donde se desarrolla el proyecto también permite hacer una visualización de los datos mediante dos técnicas de análisis y proyección. La primera es una PCA (*Principal Component Analysis*) que trata de comprimir y clasificar los datos con el fin de reducir las dimensiones de la base de datos, esto se realiza encontrando mediante técnicas estadísticas unas nuevas variables no correlacionadas, que son menos que las variables de entrada originales y que contienen la mayor parte de la información útil.

La segunda técnica es la t-SNE (*t-distributed Stochastic Neighbour Embedding*) que mediante la creación de una distribución de probabilidad que represente las similitudes entre vecinos en un espacio de gran dimensión y en un espacio de menor dimensión, posteriormente se convierten las distancias en probabilidades.

Otro parámetro a la hora de realizar la visualización es seleccionar la forma de generarlo, usando los bloques de procesamiento que vaya a usar el modelo, el modelo que se haya creado ya entrenado o en el caso de las palabras de menos de un segundo, permite usar un modelo ya entrenado. Usando este último, obtenemos una clara diferencia entre ambas técnicas de visualización.

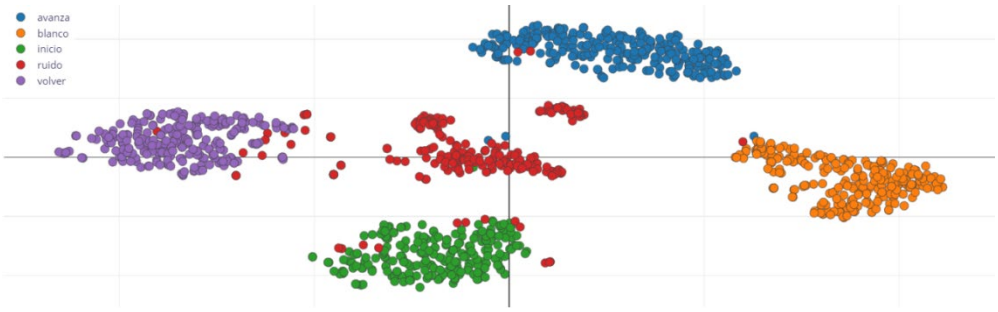


Figura 7. Visualización de audio usando t-SNE y un modelo pre-entrenado

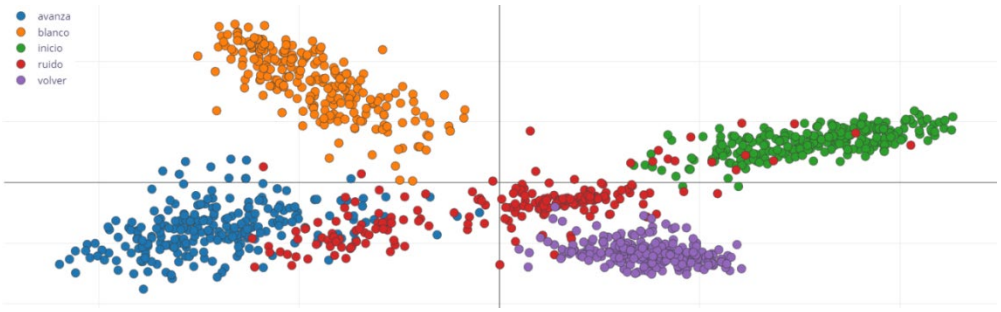


Figura 8. Visualización de audio usando PCA y un modelo pre-entrenado

En las figuras cada punto representa una muestra de audio pintado en el color que identifica su clase, además se proyectan las muestras multidimensionales sobre un plano para la visualización y análisis de la base de datos. Tal y como se ve en la Figura 7, la técnica de t-SNE es capaz de separar de forma mucho más clara las cinco clases, y además están separadas unas de otras. Por el contrario, mediante la PCA de la Figura 8, las clases están mucho más cercanas las unas a las otras y los datos menos agrupados entre sí. Aunque, al usar un modelo especializado en audios de menos de un segundo para un número de clases reducido, ambas técnicas dan resultados bastante favorables.

3.2 Extracción de características (*feature extraction*)

Antes de comenzar con la extracción de parámetros, en *Edge Impulse* (EI) hay que crear lo que denominan *impulse,* que se puede traducir como "el algoritmo", las etapas (*pipeline*) de preprocesamiento más el modelo de aprendizaje automático.

A la hora de crear el *impulse* hay que seleccionar la duración de las muestras que se van a usar como entrada para el procesamiento y el modelo, así como el intervalo entre muestras, de tal forma que, si se impone un segundo de duración y un intervalo de cien milisegundos, para un audio de dos segundos se obtendrán diez muestras distintas. Una vez se han seleccionado los tiempos, es cuando se puede elegir los bloques de procesamiento, que en el caso de datos de audio permite la posibilidad de elegir entre no aplicar ningún tipo e introducir las muestras crudas, realizar el espectrograma de cada muestra, realizar un procesado con la técnica de *Mel-filterbank energy* (MFE) o de *Mel Frequency Cepstral Coefficients* (MFCC) y, por último, el *Syntiant*.

Hay que destacar que no todos los tipos de procesamiento son compatibles con el dispositivo donde se desea implementar el modelo, como es el caso de *Syntiant,* que sí que da mejores resultados, pero consume muchos más recursos de los que dispone el dispositivo, en este caso la placa Arduino Nano 33 BLE y, por tanto, no es compatible, además de que es un bloque específico para la placa con el mismo nombre. En este caso, pese a que EI indica que MFCC es útil para voz humana se ha seleccionado el MFE (*Mel-filterbank energy*), ya que se obtenían mejores resultados.

El MFE es una técnica que funciona de forma similar a la percepción del sonido por parte del oído humano, que discrimina (y pondera) más las bajas frecuencias y menos las altas, por ello se usa en procesamiento de señales de audio, ya que una matriz de filtros de paso banda separa la señal de entrada en múltiples componentes, cada uno de los cuales lleva una única subbanda de frecuencia de la señal original y a la hora de configurar el

bloque de MFE, se pueden modificar una serie de parámetros, entre los que están el límite inferior y superior de frecuencias del filtro Los parámetros con los que más se ha experimentado para obtener un mejor resultado han sido: la duración del marco, el paso entre marcos y el número de bancos de filtros.

La duración del marco y el paso entre marcos han sido los parámetros más importantes, ya que tomar una mayor duración implica más tiempo de procesamiento y uso de los recursos, ya que el marco que se ha de procesar es mucho más grande. Por el contrario, un menor tiempo entre marcos también aumenta el tiempo de procesamiento, ya que el número total de marcos dentro de una muestra aumenta. Dado que se desea implementar en un dispositivo con recursos limitados cuantos menos sean desperdiciados mejores resultados se obtendrán, que se verán reflejados en una mejor clasificación por parte del modelo.

Realizando pruebas con una duración del marco y tiempo entre marcos de 0.01s y de 0.05s se han obtenido los resultados que se muestran en las Figura 9 y Figura 10, respectivamente. Se observa que al tomar marcos mucho más grandes disminuye de forma considerable el tiempo de procesamiento y, a su vez, al tomar un menor número de marcos también se reduce el pico de memoria utilizada. Por el contrario, eso implica que, a la hora de extraer información, esta es más pobre y se observa que algunas muestras esta algo más dispersas que en el caso de una menor duración del marco.

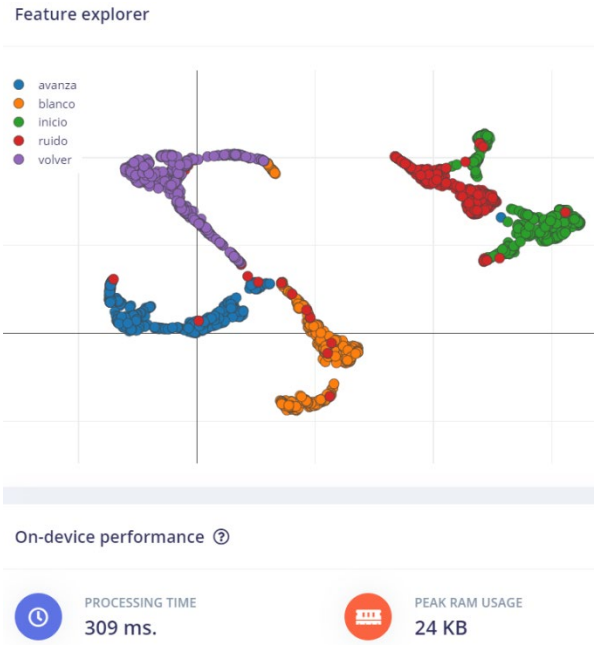


Figura 9. Distribución con MFE y duración del marco de 0.01s

Pese a que hay una diferencia de tiempo de procesamiento de casi el triple entre un caso y otro, como posteriormente se utilizará este procesamiento para entrenar el modelo, cualquier pequeña mejora a la hora de clasificar los datos puede suponer una mejor precisión entre un modelo y otro. Ya que como es el caso de la palabra avanza, con un marco mucho más pequeño (Figura 9) las muestras están mucho más agrupadas que en el caso de un marco más largo (Figura 10), y de igual manera ocurre con el grado de desorden del conjunto en general.

Figura 10. Distribución con MFE y una duración del marco de 0.05s

También se ha probado a variar el número de bancos durante el procesamiento, lo que supondrá un aumento de la memoria usada a mayor número de bancos, ya que se extrae más cantidad de datos de cada marco. Tal y como se refleja en la Figura 11 y Figura 12, donde se han usado respectivamente 20 y 60 bancos de filtros. Además, se observa, al igual que ocurre variando la duración del marco, que al aumentar el banco aumenta de forma considerable el tiempo de procesamiento.

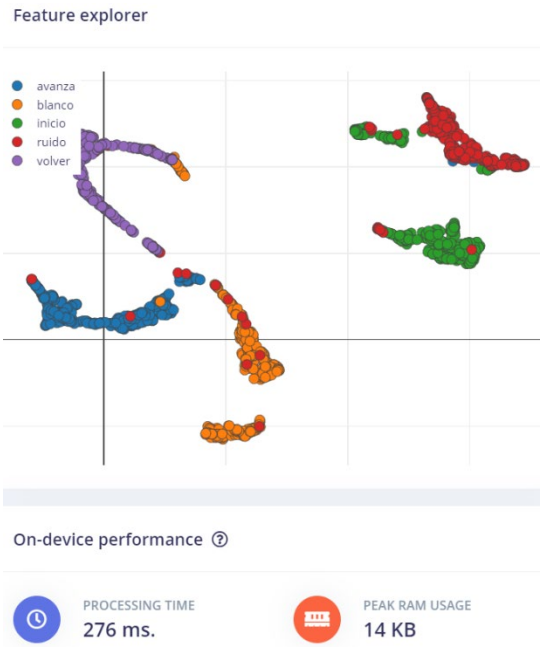


Figura 11. Distribución con MFE y 20 bancos

Una apreciación importante es que se supone que al tener un mayor número de bancos se debería observar una mejor ordenación de las muestras de acuerdo con su clase, pero se evidencia que no hay una gran diferencia entre las clasificaciones, siendo tal y

como era de esperar un poco más ordenada la distribución con 60 bancos. Pero dado que sí que hay una diferencia importante en los recursos pico que va a necesitar en el dispositivo donde se desea implementar, se ha optado por hacer un término medio, por lo que se han escogido 40 bancos.

Figura 12. Distribución con MFE y 60 bancos

Dado que el bloque de procesamiento recomendado por EI era el MFCC, técnica que al igual que el MFE, realiza un análisis del espectro de frecuencias usando la escala Mel, pero en este caso los coeficientes contienen información sobre los cambios de ritmo en las diferentes bandas del espectro. Utilizando los mejores valores para el caso del MFE, se ha replicado para el caso del MFCC donde se debería obtener un resultado cuanto menos similar a la hora de realizar la distribución, pero tal y como se observa en la Figura 13 las clases no están agrupadas de forma tan clara como en cualquiera de las figuras usando el bloque de MFE, además de que el tiempo de procesamiento ha aumentado acercándose peligrosamente a casi un segundo, lo que no sería adecuado ya que tendría que ir almacenando el dispositivo en memoria el audio durante ese tiempo, por el contrario, cuantas más inferencias se puedan realizar por segundo más posible es que sea capaz de clasificar de forma correcta las muestras.

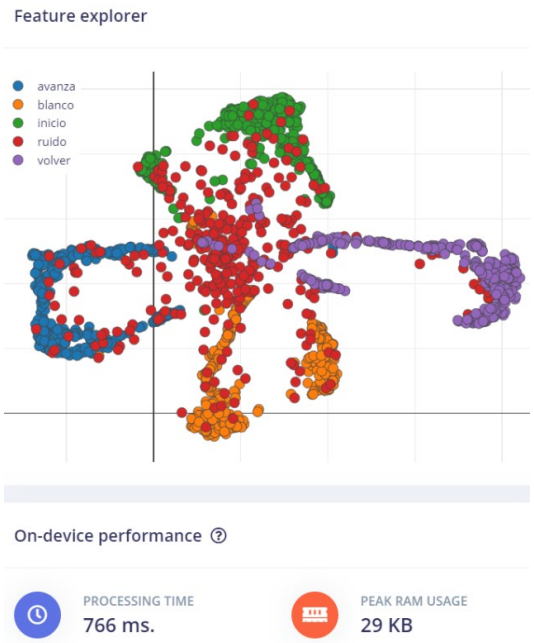


Figura 13. Distribución con MFCC y con una duración de marco de 0.01s

3.3 Desarrollo del modelo de aprendizaje automático

Una vez analizada la base de datos y realizada la extracción de características, se procede al desarrollo del modelo de *machine learning* que se ha seleccionado cuando se ha creado el "impulso". Muchos de dichos modelos se basan en redes neuronales, las cuales al entrenar con una base de datos (aprendizaje), ajustan sus pesos para conseguir la relación entrada-salida deseada. La red neuronal una vez entrenada se utiliza en modo inferencia, es decir, ante un vector de entradas responde con la salida o predicción que genera.

Los modelos disponibles que se pueden seleccionar aparecen en tres grupos: clasificador, regresor o si se desea implementar un detector de anomalías (o incluso uno propio, creado por uno mismo). Aunque el algoritmo es una parte importante, si se han realizado los pasos anteriores de forma correcta y se ha obtenido una forma muy robusta de realizar una diferenciación entre los datos, hasta el algoritmo menos optimizado será capaz de ofrecer unos resultados relativamente aceptables. La etapa de extracción de características puede influir tanto o más en el rendimiento final que el modelo de aprendizaje automático seleccionado.

Hay diferentes opciones a la hora de crear un modelo destinado a reconocer datos de audio, en la plataforma hay una opción grafica donde pueden irse añadiendo diferentes bloques para obtener un modelo completo, aunque la propia aplicación recomienda implementar tanto un algoritmo basado en capas convolucionales de una dimensión como capas convolucionales de dos dimensiones. Además de capas convolucionales de una y dos dimensiones, también está la posibilidad de realizar una capa de neuronas densa, el resto de las opciones no son para realizar una clasificación de los datos, sino para estructurar los datos y ajustar como entra o sale de las capas (del tipo que sea).

Una capa convolucional [6], como su nombre indica, realiza una convolución, que es una operación matemática que desliza una función sobre otra y mide la integral de su multiplicación puntual. No obstante, para no realizar integrales, en la práctica calculan correlaciones cruzadas (muy similares a las convoluciones) sobre los datos de entrada para pasar el resultado a la siguiente capa como un vector. En definitiva, en una capa convolucional, no todas las neuronas están conectadas con todas, sino con las vecinas y compartiendo los pesos que corresponden a cada filtro.

Por otra parte, una capa densa está conformada por neuronas en las que todas están conectadas a las de la capa anterior (todas con todas, en vez de conexiones parciales) [6].

Finalmente, indicar que se denomina neurona a un nodo sencillo de procesamiento que recoge un conjunto de números de entrada (vector de entrada), en la que cada entrada tiene un peso, realiza la suma ponderada entrada por peso y se le aplica una función generando un único valor numérico como salida [6]. Normalmente las funciones aplicadas son no lineales, siendo el tipo ReLU las más habituales en la actualidad.

Se denomina hiperparámetros a los parámetros independientes que definen la estructura y entrenamiento del modelo, como el número de pasos que se van a realizar durante el entrenamiento y el ritmo de entrenamiento (con un ritmo o ratio menor, mejor aprenderá el modelo, pero a coste de tardar más en completar el entrenamiento). A su vez está el porcentaje de datos que se va a destinar a la validación, estos datos son extraídos de los datos utilizados para entrenar y la principal función es evitar que el modelo sea únicamente valido con los datos de entrenamiento, lo que se denomina *overfitting,* en castellano sobreajuste, un valor en torno al veinte por ciento es lo típico que se suele utilizar.

Además de los hiperparámetros mencionados anteriormente, al tratar con señales de audio la aplicación permite aumentar las muestras (*data augmentation*) añadiendo diferentes niveles de ruido, así como deformar el eje temporal, enmascarar las bandas de tiempo (enmascara bloques aleatorios del eje de tiempo en cada espectrograma), enmascarar las bandas de frecuencia (enmascara bloques aleatorios del eje de frecuencias de cada espectrograma) y deformar el eje de tiempo de cada espectrograma en un punto aleatorio. Todas estas formas de aumentar los datos es recomendable aplicarlas una vez se ha obtenido un modelo y se quiere evitar el sobreajuste, dado que permite diferentes niveles de aumentar los datos se recomienda aplicar de uno en uno.

Para obtener el mejor modelo, EI dispone de una herramienta que busca el procesado y el modelo más adecuado al dispositivo donde se quiere implementar, especificando también la latencia máxima que se desea, esta herramienta se denomina EON *Tuner* (*Edge Optimized Neural*).El EON Tuner analiza los datos de entrada, los posibles bloques de procesamiento de señales y las arquitecturas de las redes neuronales, y ofrece una visión general de las posibles arquitecturas de los modelos que se ajustarán a los requisitos de latencia y memoria del dispositivo elegido. Esta herramienta permite obtener un modelo con resultados aceptables y a partir del cual se pueden realizar modificaciones para optimizar aún más y mejorar si es posible el algoritmo.

Se han realizado pruebas con diferentes modelos, uno de ellos sigue una estructura convolucional de una dimensión y tiene la siguiente estructura. Primero se realiza una reestructuración de las columnas de los datos, con el fin de poder introducirlo posteriormente a una capa convolucional (tiene que tener el mismo número de columnas que el número de bancos que se han seleccionado en el procesado). Tras esto, las salidas se introducen en otra capa convolucional y, para finalizar, se incluye una capa de *dropout*, en la que se introducen ceros de forma aleatoria para evitar el sobreajuste (y mejorar la generalización del modelo), seguido de una capa de aplanado, que limita la salida a una dimensión, tal y como se muestra en la Figura 14.

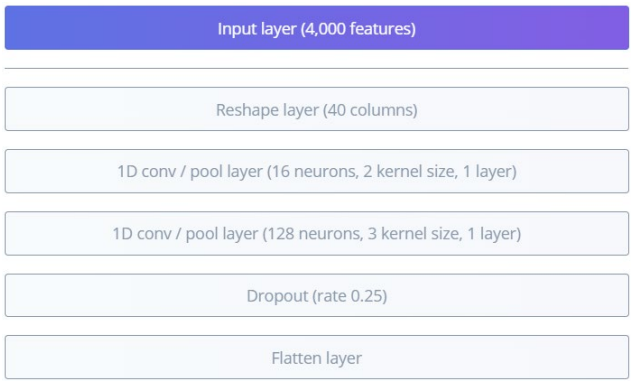


Figura 14. Modelo de convolución para voz

Se ha comparado este modelo con otro en el que en vez de utilizar capas convolucionales se emplean capas densas, la cual tiene la siguiente estructura. La salida del bloque de procesamiento se introduce directamente en una capa densa con un número pequeño de neuronas para que hagan una primera clasificación de los datos, en este caso hemos escogido dieciséis. Posteriormente, se introduce en una capa intermedia de *dropout* y la salida de esta va a una capa densa con un número mucho mayor de neuronas, en este caso se ha probado con 256. Finalmente, y replicando los pasos del anterior modelo, se introducen sendas capas de *dropout* y de aplanado, Figura 15.

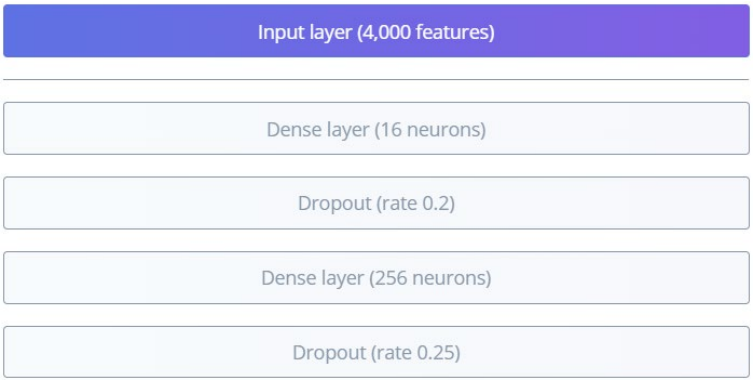


Figura 15. Modelo de capas densas para voz

Los resultados que se obtienen con ambos modelos son similares en cuanto a la precisión en el entrenamiento, sin embargo, dado que la convolución es un proceso computacional costoso, el modelo que involucra dicha operación tiene un tiempo de procesamiento mucho mayor, así como un mayor uso de RAM, Figura 16. Además, se observa que los resultados no son mucho mejores a la hora de realizar la predicción.

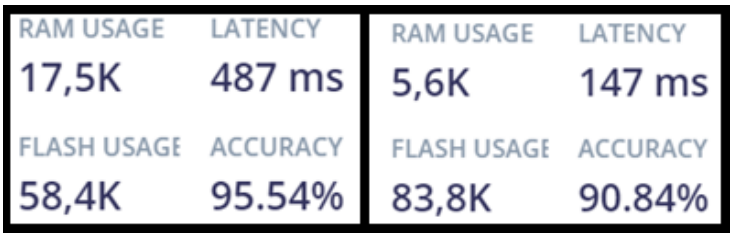


Figura 16. Requisitos y exactitud del modelo de convolución (izq.) y del modelo capas densas (drcha.)

Finalmente, hay que tener en cuenta que cuantas más capas se introduzcan en un modelo y más complejas sean estás, más tiempo le costara al modelo realizar una inferencia, pero no tiene por qué significar que vaya a obtener un mejor resultado a la hora de clasificar los datos. De hecho, a menudo un modelo con muchas capas sobreajusta y generaliza peor.

Una observación importante es que una cosa son los valores de precisión obtenidos a la hora de entrenar y/o realizar las pruebas en computador, y otra diferente cuando el modelo se reestructura para que pueda incorporarse en un microcontrolador de recursos computacionales limitados, lo cual puede degradar la precisión obtenida sobre computador. Esto se estudiará en el capítulo 5.

Capítulo 4: Reconocimiento de gestos

4.1 Generación de la base de datos

De forma completamente paralela a lo realizado para el reconocimiento de voz, el proceso de obtener la base de datos de movimientos gestuales realizados con una mano se ha realizado a través de la herramienta que ofrece EI.

En este caso se han construidos dos bases de datos con el fin de diferenciar dos clases de movimientos. El primer tipo son movimientos continuos de la mano, en los que no se realizan paradas, como es el caso de un movimiento horizontal o vertical. El segundo tipo son movimientos específicos, como realizar un movimiento hacia la derecha o la izquierda. La duración se ha limitado a un segundo por movimiento para el primer caso y dos segundos para el segundo. A diferencia de la voz, para el movimiento no se puede realizar una clasificación de los datos a menos que se disponga ya de bloques en el impulso o un impulso completo entrenado. En la Figura 17 se describen los movimientos y como se agarra el dispositivo para el caso de los gestos continuos.

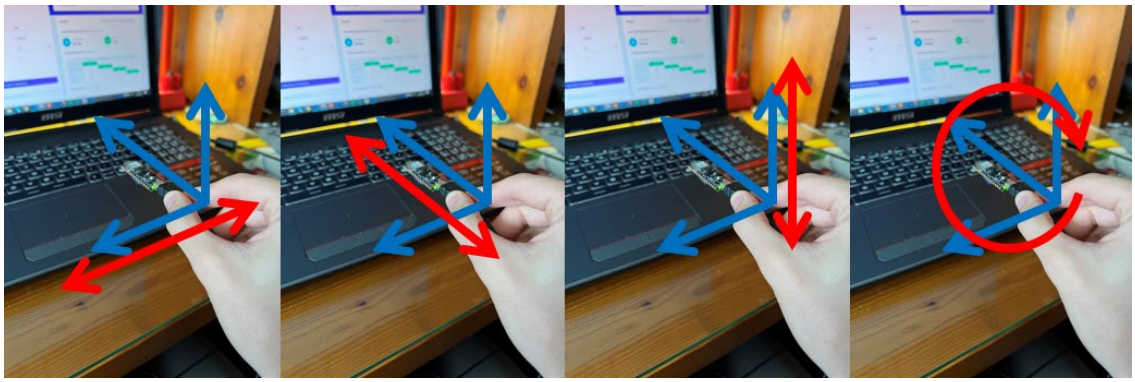


Figura 17. Ejemplos de movimientos, indicados con flechas rojas: Horizontal, Push, Vertical y Circle

Antes de empezar a crear la base de datos hay que estudiar el acelerómetro que incluye la placa de Arduino Nano 33 BLE e identificar como están orientados los ejes en el sensor. Empezamos por la base de datos con movimientos continuos, en la que se han diferenciado 4 clases de movimientos de la mano: Horizontal, Vertical, Push y Circle. Se pueden ver las diferencias en la Figura 18 de los movimientos característicos de cada eje, quedando el eje X para el movimiento Push, el eje Y para el Horizontal, el eje Z para el Vertical y Circle es una combinación de los tres ejes o mínimo de dos de ellos.

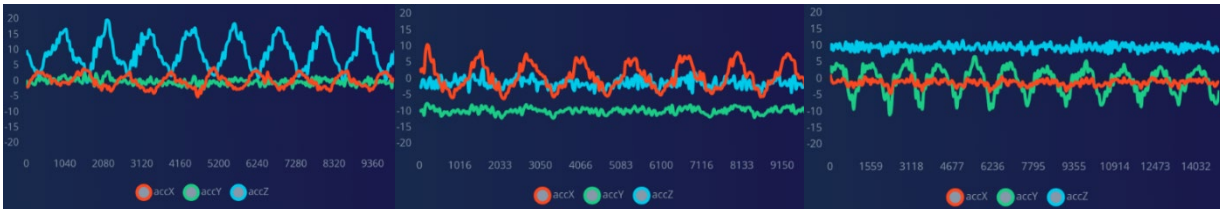


Figura 18. Muestras movimiento Vertical, Push y Horizontal

En cuanto a los movimientos individuales (específicos), para generar los datos se han espaciado con el fin de poder realizar los cortes. Además, tal y como se ve en la

Figura 19, se aprecian las diferencias entre los movimientos que involucran el mismo eje como es la clase *Left-Right* y *Right-Left,* en el que ambos muestran señal en el eje Y, pero el primero de ellos captura un pico negativo seguido de uno positivo y otro negativo de nuevo, mientras que el segundo movimiento realiza la secuencia de forma opuesta, positivo-negativo-positivo.

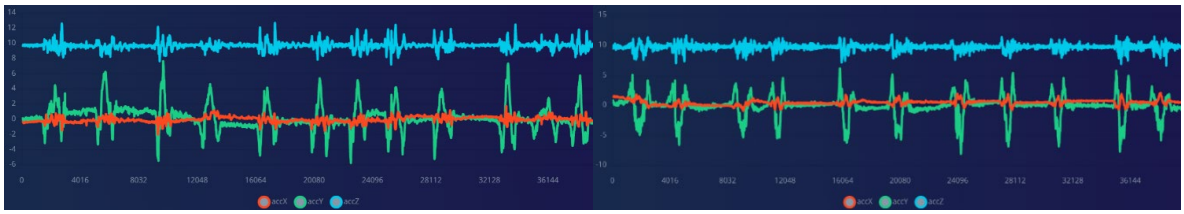


Figura 19. Muestras movimiento Left-Right y Right-Left

Dado que todos los movimientos, ya sean continuos o individuales, se han realizado manualmente por lo que, aunque presentan señales más intensas en uno o más de los ejes, el resto de los ejes también muestra algo de señal, aunque menos intensas. Esto se debe a que no se pueden realizar los movimientos con absoluta precisión, lo que otorga la ventaja de su similitud con los movimientos en una situación real.

En la Figura 20 se muestra el número total de muestras para cada uno de los modelos, con la particularidad que en los gestos continuos las muestras tienen una duración aproximada de 20s y será el propio impulso el encargado de recortar y generar las muestras que llegan al modelo, por lo que, aunque a priori sea un numero mucho menor comparado con las muestras de gestos individuales es posible que tras los recortes se obtenga un número superior de muestras.

Gestos Individuales			Gestos Continuos		
	Train	Test		Train	Test
Circle	292	72	Circle	28	9
Up	282	73	Push	30	6
Left-Right	286	72	Horizontal	27	10
Right-Left	292	73	Vertical	31	9

Figura 20. Número de muestras de cada etiqueta para los modelos de gestos

4.2 Extracción de características

Para procesar los datos del acelerómetro se dispone en la herramienta EI de bloques similares al caso de la voz, aunque en esta ocasión están dirigidos al movimiento. Pese a que la herramienta recomienda dos bloques, el bloque de Syntiant está dedicado a la placa con el mismo nombre. El otro bloque es un análisis espectral que extrae las características de frecuencia y potencia de una señal a lo largo del tiempo, aunque como se ha visto en el apartado anterior, los movimientos son fácilmente diferenciables, por lo que es posible que los propios datos crudos generen una clasificación aceptable.

Se va a realizar una comparación entre el procesamiento mediante el análisis espectral modificando los parámetros vistos en el apartado 3.2 con los datos en crudo.

Como punto de partida se va a realizar dicha comparación utilizando la base de datos con los movimientos continuos, escogiendo como valores iniciales una ventana de 1500ms y lapso entre ventanas de 100ms. Por otra parte, el bloque del análisis espectral incluirá un filtro paso bajo de orden seis y se aplicará una transformada de Fourier con 16 puntos. Con todos estos parámetros se obtiene una agrupación de los datos tal y como se muestran en la Figura 21.

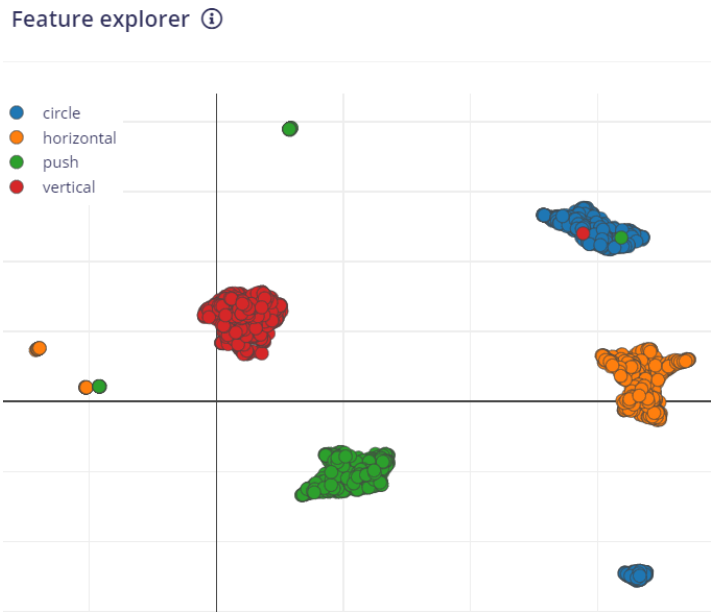


Figura 21. Clasificación con análisis espectral de movimientos continuos. Cada punto representa una muestra de la base de datos proyectada sobre un plano

El análisis espectral ha generado una agrupación bastante clara de las cuatro clases existentes, salvo alguna muestra que se distancia un poco (pero entra dentro de lo aceptable). En cambio, si se implementara el mismo bloque, pero sin el filtro paso bajo, se obtendría una clasificación como la de la Figura 22, en la que, si bien sí se aprecia que se realiza una clasificación por etiqueta de los datos, estos aparecen mucho más cerca unos de otros. Además, en el caso del círculo (*circle*) algunas muestras se acercan mucho a las de otras etiquetas. Por lo que, en este caso, parece que incluir el filtro paso bajo mejora el preprocesamiento de los datos para conseguir que al modelo clasificador le sea más fácil clasificar correctamente las muestras.

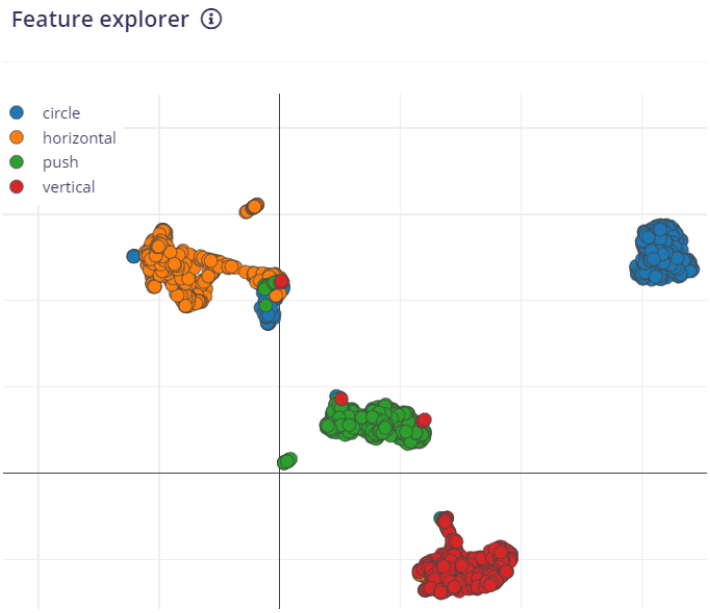


Figura 22. Clasificación con análisis espectral sin filtro de movimientos continuos

Por otra parte, si no se realizase ningún procesamiento de los datos y se desease introducir directamente los datos crudos al bloque de aprendizaje por los recursos limitados del dispositivo, ello no representa un problema tan grave porque tal y como se muestra en la Figura 23, los datos crudos ya se agrupan de manera clara, obteniendo casi mejores resultados que aplicando el bloque de análisis espectral.

Y en lo que respecta a los recursos que consume cada uno de los bloques de procesamiento, ambos consumen la misma RAM. El análisis espectral necesita 10ms, mientras que los datos crudos únicamente usan un milisegundo, por lo que no tiene uno sobre el otro ninguna ventaja.

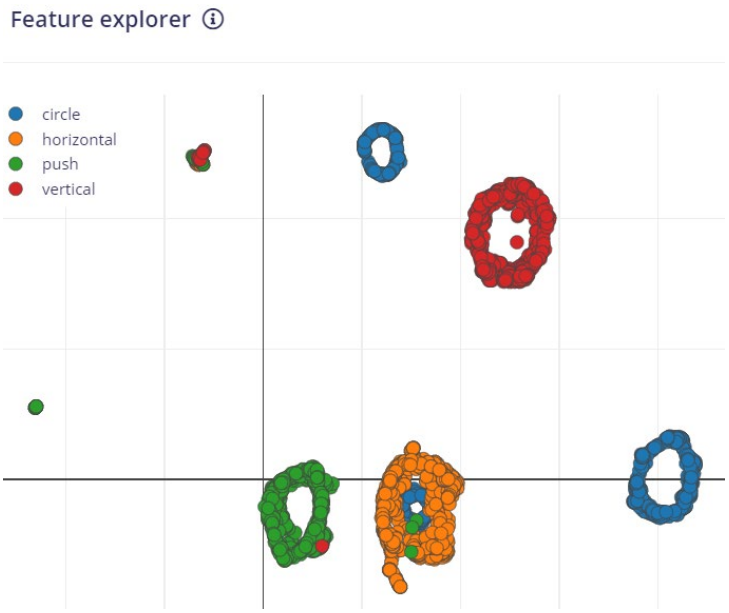


Figura 23. Clasificación de los datos crudos para movimiento continuo

Si analizamos los mismos bloques de procesamiento para el caso de la base de datos con movimientos individuales, se obtienen cosas diferentes. Empezando por el bloque de análisis espectral y aplicando un filtro paso bajo, tal y como se ha realizado con el movimiento continuo, la clasificación que realiza el bloque es mejorable, ya que, pese a que sí que agrupa de manera correcta los movimientos de diferentes ejes, los movimientos horizontales de derecha-izquierda y viceversa, los agrupa demasiado cerca unos de otros, tal y como se muestra en la Figura 24.

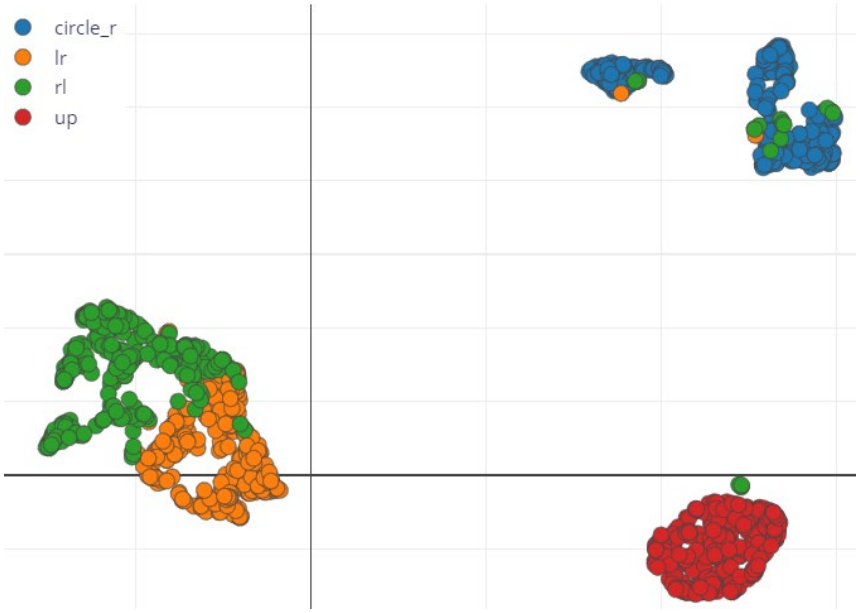


Figura 24. Análisis espectral de movimientos individuales

Por otra parte, si no se realiza ningún análisis y se clasifican directamente los datos crudos, tal y como se ha realizado en la Figura 25. Los datos con las mismas etiquetas aparecen agrupados de manera diferente a realizar un análisis espectral, con la ventaja de que en este caso aparecen los datos distanciados unos de otros, con la excepción de circulo y arriba que, aunque están más distanciados que los movimientos horizontales del caso anterior, en este caso están cercanos entre sí. Otro aspecto que comentar sobre usar los datos crudos es que en la clasificación aparecen algunas muestras que se alejan de otros datos con las mismas etiquetas, como es el caso del círculo, en el que los datos están agrupados, pero en pequeñas agrupaciones.

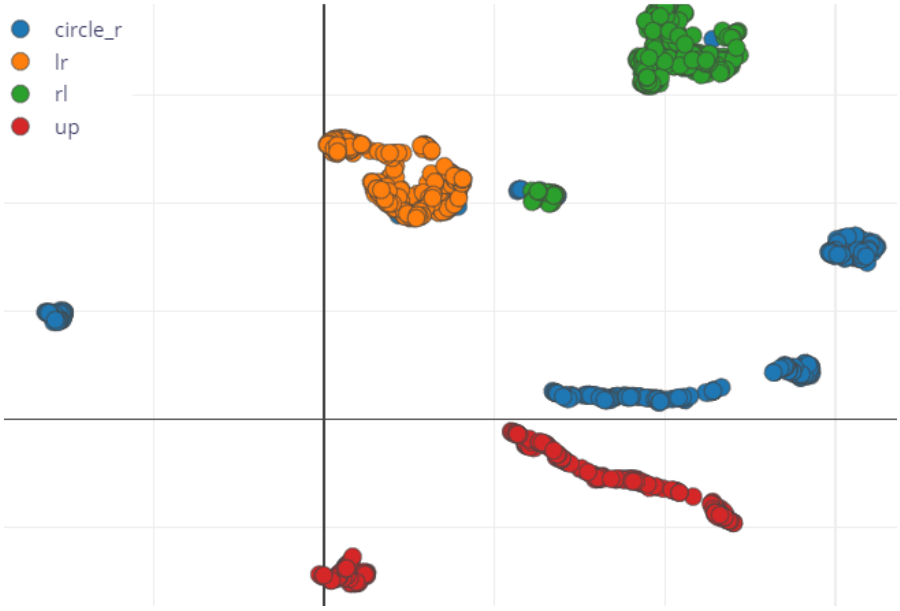


Figura 25. Análisis de datos crudos para movimientos individuales

4.3 Desarrollo del modelo

De igual manera que para el modelo de reconocimiento de voz, una vez se ha seleccionado el bloque de procesamiento y este ha generado las características de entrada (*features*), el siguiente paso es seleccionar el o los bloques para implementar el modelo y su entrenamiento. En ambos tipos de movimiento se ha elegido un clasificador (un regresor no tiene sentido en este caso).

Siguiendo el patrón de los apartados anteriores, y comenzando por el caso de movimientos continuos, se va a realizar una comparación de diferentes modelos, aunque dado que se obtenían resultados comparables realizando el análisis espectral con filtro paso bajo y con los datos crudos, en este caso se va a comparar tanto diferentes modelos como el mismo modelo con diferentes procesamientos.

Primero se va a crear un modelo formado por capas de neuronas densas y capas de *dropout* intercaladas. Se obtiene el modelo de la Figura 26, se empieza introduciendo un mayor número de neuronas para que obtengan el mayor número de diferencias posibles y las siguientes capas son las que realizan propiamente dicha la clasificación, y las capas de *dropout* evitan que se sobreajuste a los datos de entrenamiento.

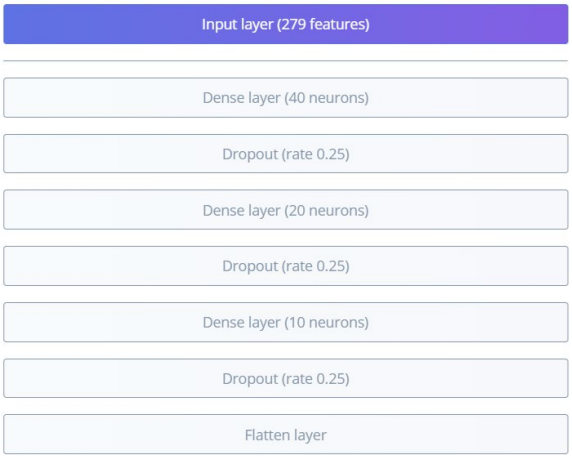


Figura 26. Modelo de capas densas para movimiento continuo

Los resultados que se obtienen mediante ambos tipos de procesamiento son similares en cuanto al valor numérico de la precisión, y pese a eso los valores obtenidos usando el análisis espectral son inferiores en cuanto a tiempo y memoria Flash usada. Por otra parte, las clasificaciones que se obtienen sí se pueden decir que mejoran cuando se realiza el análisis espectral, ya que las etiquetas están mejor agrupadas y hay una mayor separación entre unas etiquetas y otras, tal y como se muestra en la Figura 27.

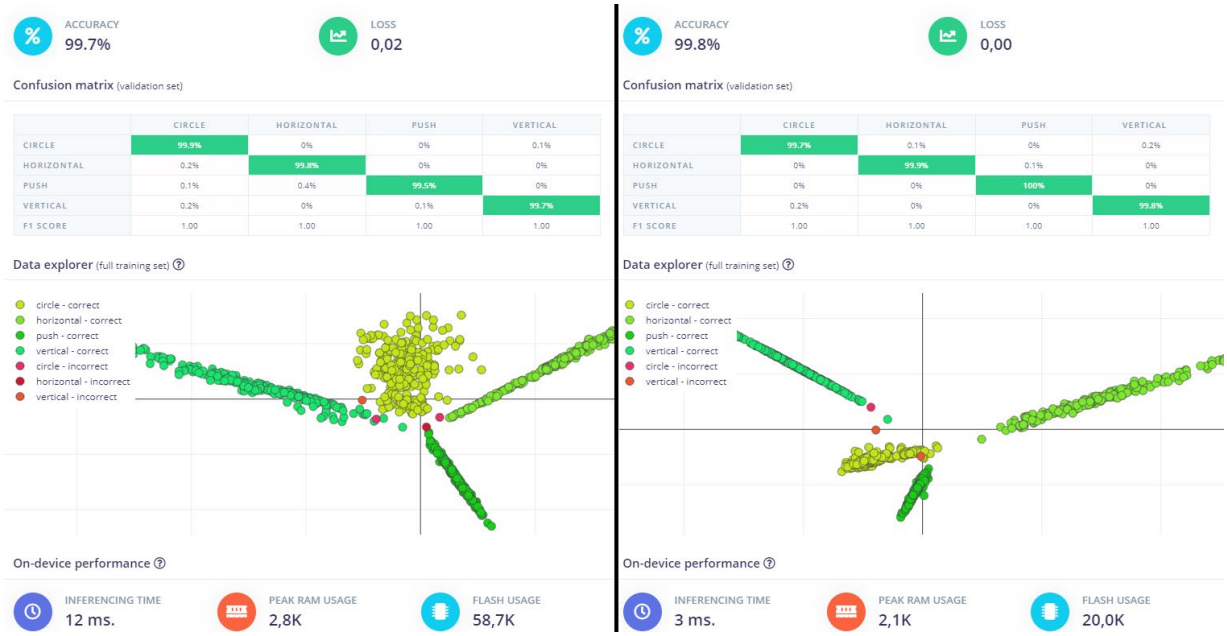


Figura 27. Resultados del modelo con datos crudos (izq.) y análisis espectral previo con 40 neuronas (drcha.)

Tal y como se muestra, ambos modelos serian adecuados a primera vista, aunque al ser más rápido y tener una mejor distribución, quizás uno se decanta por el análisis espectral, pero aún se puede comparar este con otro modelo que incluya un mayor número de neuronas para ver cómo se comporta, como es el caso de los resultados que se observan en la Figura 28, que corresponden a un modelo semejante al anterior con la diferencia que el número de neuronas es 60,30 y 10 respectivamente. Los números que se obtienen son algo superiores al caso del análisis espectral del modelo anterior, pero tiene la ventaja de que la precisión se mejora y la agrupación de los datos es el punto con mejor resultado.

Figura 28. Resultados del modelo con 60 neuronas y análisis espectral previo

Dado que no se ha incluido una "clase extra" para el resto de los movimientos que se realicen y así se pueda evitar clasificar cualquier movimiento como uno de los etiquetados, se ha implementado un modelo secundario denominado detector de anomalías. Este detector de anomalías trata de agrupar los datos de los que se dispone en clusters de vecinos que, mediante la distancia, en los ejes que se elijan, creará las agrupaciones; de tal manera que cualquier muestra que se aleje demasiado de estos clusters será considerada como una anomalía. En este proyecto se han realizado muestras con el dispositivo completamente estático de forma que los ejes del acelerómetro aparezcan sin ninguna señal, así como muestras con movimientos aleatorios y gesticulando como haría una persona. En la Figura 29, se ha introducido una de estas muestras en el detector de anomalías y se observa que la mayoría de las ventanas que ha obtenido de la muestra las sitúa fuera de las agrupaciones que realiza.

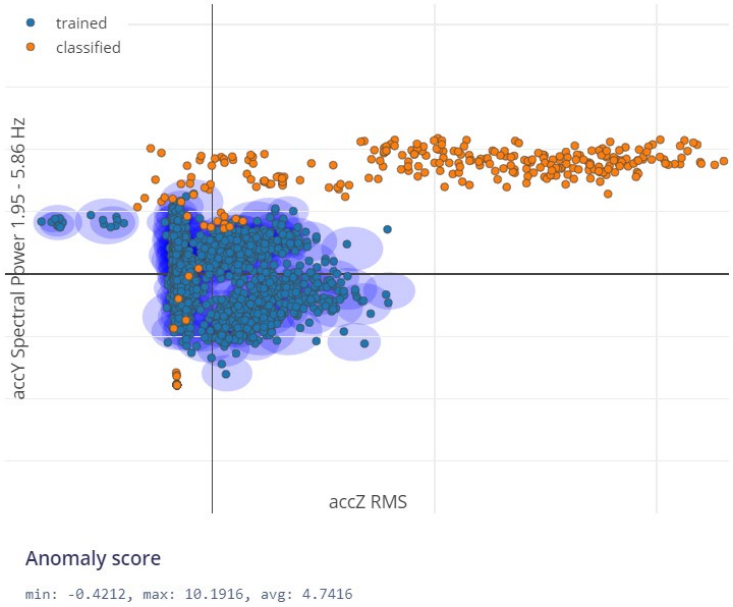


Figura 29. Detector de anomalías en movimiento continuo

Respecto a la creación de un modelo para la base de datos con movimientos individuales, dado que el procesamiento de los movimientos individuales no ha mostrado un claro vencedor a la hora de realizar la clasificación, ya se no realizando ningún procesamiento y dejando los datos crudos o procesando los datos con un análisis espectral, se van a comparar modelos ambos procesamientos con diferentes modelos.

Reproduciendo lo realizado con los movimientos continuos, se ha implementado un modelo con capas de neuronas densas igual que el realizado anteriormente (Figura 26), con el fin de evaluar si el modelo anterior funciona de manera similar y, por tanto, es capaz de ofrecer resultados similares.

Empezando por los resultados obtenidos con este modelo, y sin realizar ningún tipo de procesamiento, se obtiene una exactitud muy elevada y el tiempo que necesita para realizar la inferencia es pequeño, todo ello se indica en la Figura 30. Además, la agrupación de los datos que realiza la red neuronal mejora respecto a la que se realiza simplemente con el procesado de los datos, también hay que considerar que las agrupaciones están más separadas, lo que indica que el modelo debería ser capaz de clasificar futuras muestras correctamente.

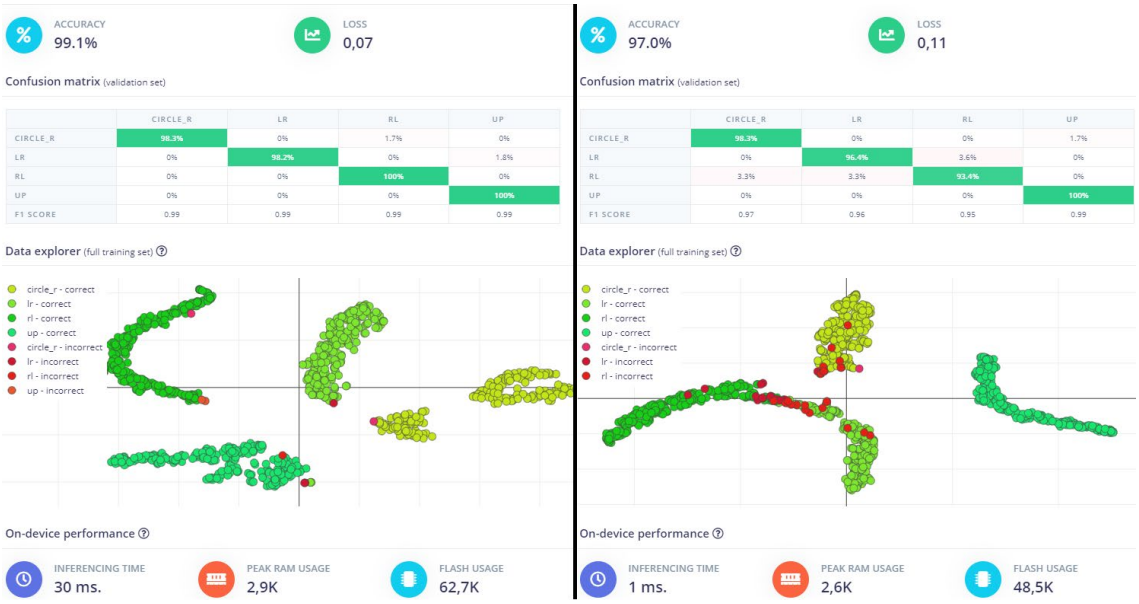


Figura 30. Resultados de los movimientos individuales con datos crudos(izqda.) y con análisis espectral (drcha.)

Respecto al caso de aplicar el análisis espectral, los resultados son algo inferiores que, al realizar el entrenamiento con datos crudos, con la excepción de que el tiempo de procesamiento necesario para realizar las inferencias se ha visto reducido considerablemente. Por otro lado, la agrupación que realiza el modelo de los datos es algo peor, ya que se observa que en más ocasiones de las deseadas el modelo clasifica de forma incorrecta los movimientos horizontales de *Left-Right* y *Right-Left*.

Por último, se ha realizado una comparación adicional entre modelos usando datos crudos. Se ha creado un modelo que en lugar de tener tres capas densas únicamente tiene dos de ellas, eliminando la capa de 10 neuronas y la siguiente a esta, la capa de *Dropout* al 0.25.

Capítulo 5: Análisis de los resultados

Una vez se han generado los modelos en el entorno EI, se puede comprobar en la propia aplicación web la exactitud de estos a diferentes niveles, sin tener que implementar de forma completa el modelo en el dispositivo elegido.

Desde la propia aplicación web se pueden recoger datos del micrófono o del acelerómetro y que el algoritmo realice una clasificación (inferencia) en el momento y ver los resultados obtenidos. De la misma forma, hay un apartado donde se pueden realizar pruebas con los datos seleccionados para analizar el modelo. Por último, en la herramienta online hay un apartado de implementación, donde a partir del modelo desarrollado se puede crear una librería específica para una lista abundante de dispositivos, y desplegar (implementar) el modelo directamente en el dispositivo.

En este proyecto se ha creado una librería (desarrollada en C++) para Arduino Nano 33 BLE que incluye un ejemplo de funcionamiento continuo tanto del micrófono como del acelerómetro, el cual va capturando y realizando las inferencias en tiempo real. Posteriormente dicha inferencia es comunicada por puerto serie. En la Figura 31, se incluye una comparativa de la RAM y Flash que utiliza cada firmware generado por Edge Impulse para la placa de Arduino, expresado como porcentaje respecto de la RAM y Flash integrada en el microcontrolador (1MB de Flash, 256KB de SRAM), tal y como aparece cuando se implementa desde el Arduino IDE. También se ha incluido el tiempo entre inferencias, aunque este parámetro es modificable.

Voz		Gestos Continuos		Gestos Individuales	
RAM (%)	18	RAM (%)	19	RAM (%)	19
Flash (%)	20	Flash (%)	15	Flash (%)	14
Inference (ms)	1000	Inference (ms)	200	Inference (ms)	100

Figura 31. Comparación entre modelos del uso de Flash, RAM y tiempo entre inferencias

Con el fin de realizar las acciones que se corresponden a los comandos, se ha diseñado un script en Python usando la librería *pyserial*, para que, en función de la respuesta de la inferencia realizada en el dispositivo, el ordenador realice la acción correspondiente. Dicha acción es una simulación de lo que sería utilizar los atajos de teclado que permite PowerPoint, todo ello a través de la librería de Python para el teclado y el ratón *pykeyboard*. El diagrama de flujo que realiza el script es sencillo, ya que únicamente tiene que encontrar el puerto serie en el que esta la placa Arduino, se selecciona el modelo que se va a ejecutar, modificando internamente las clases que ha de detectar y en qué orden salen los datos. Posteriormente, continua en bucle leyendo y decodificando los mensajes que se envían por el puerto. Una vez leído, verifica si ha detectado un comando un instante anterior y, en función del mensaje, realiza la acción correspondiente.

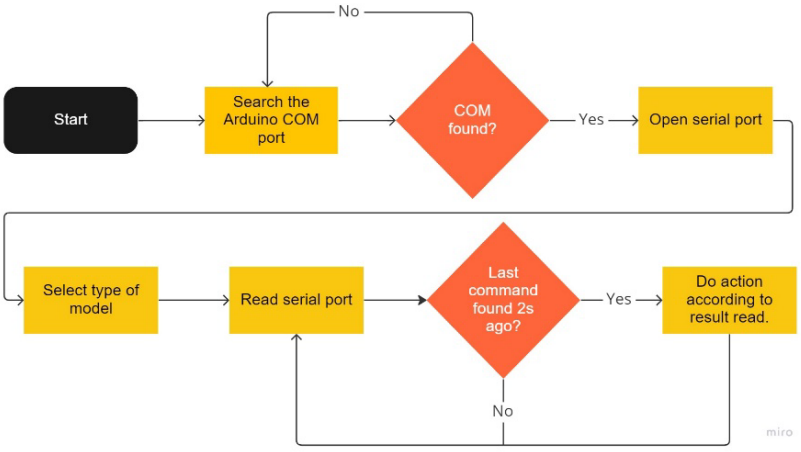
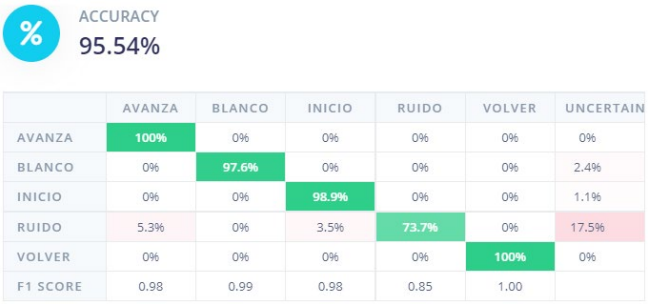


Figura 32. Diagrama de flujo script comunicación con Arduino

5.1 Reconocimiento de voz

Se han probado diferentes palabras (comandos de voz) para cada acción que se deseaba ejecutar y se ha tratado de seleccionar aquellas que mostraban una mayor diferenciación, así como ser acordes a la reacción esperada en una presentación PowerPoint.

Dicho esto, los modelos que se han creado han otorgado buenos resultados tanto a la hora de entrenar como en las pruebas internas de la aplicación, tal y como se muestra en la matriz de confusión de la Figura 33, y en las pruebas que posteriormente se realizaban al implementar el modelo en el dispositivo Arduino.



	AVANZA	BLANCO	INICIO	RUIDO	VOLVER	UNCERTAIN
AVANZA	100%	0%	0%	0%	0%	0%
BLANCO	0%	97.6%	0%	0%	0%	2.4%
INICIO	0%	0%	98.9%	0%	0%	1.1%
RUIDO	5.3%	0%	3.5%	73.7%	0%	17.5%
VOLVER	0%	0%	0%	0%	100%	0%
F1 SCORE	0.98	0.99	0.98	0.85	1.00	

Figura 33. Matriz de confusión con los resultados para el conjunto de test del modelo de reconocimiento de voz

Es de destacar que para implementar diferentes modelos de voz se han realizado diferentes pruebas modificando algunas variables definidas en la librería, entre las que se encuentran el número de ventanas en las que va a realizar inferencias y el tiempo aproximado entre inferencias (aunque este solo es aplicable si la duración es menor al tiempo impuesto). Posteriormente, dado que por el puerto serie se indica el porcentaje de acierto que ha tenido cada comando en cada inferencia, en el script de comunicación se puede introducir el nivel de confianza mínimo requerido para realizar la acción. Así como indicar que, si se detecta el mismo comando en un intervalo inferior a un tiempo establecido (en este proyecto se ha cogido 2 segundos), no realice la acción de forma duplicada. Esto se puede deber a que el tiempo entre inferencias es muy pequeño.

El parámetro que describe el tiempo entre inferencias es muy importante ya que realizando un mayor número de inferencias es posible aumentar las probabilidades de realizar una inferencia en los datos que contienen el comando a realizar, con la desventaja ya mencionada de tener resultados duplicados (aunque se ha visto que tiene fácil solución). En la Figura 34, se observa un ejemplo de los datos que envía el dispositivo Arduino por el puerto serie y que el script va leyendo.

```
14:08:58.430 ->     avanza: 0.00000
14:08:58.430 ->     blanco: 0.99609
14:08:58.430 ->     inicio: 0.00000
14:08:58.430 ->     ruido: 0.00000
14:08:58.430 ->     volver: 0.00000
14:08:59.408 ->     avanza: 0.04688
14:08:59.408 ->     blanco: 0.16016
14:08:59.408 ->     inicio: 0.08203
14:08:59.408 ->     ruido: 0.63281
14:08:59.408 ->     volver: 0.08203
```

Figura 34. Salida puerto serie modelo voz.

5.2 Reconocimiento de movimiento

Analizamos los dos modelos que se encargan de reconocer los dos movimientos que hemos distinguido en el capítulo 4, movimientos continuos y movimientos individuales. A la hora de realizar las pruebas de los modelos se observa que ambos proporcionan valores de acierto en la clasificación de movimientos gestuales elevados, aunque en el caso de los movimientos continuos es algo, superior alcanzando casi el 100% de exactitud, tal y como se observa en la Figura 35.

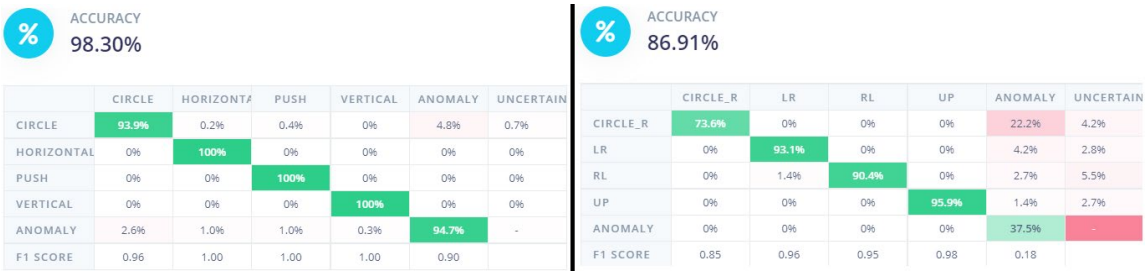


Figura 35. Matrices de confusión con los resultados para el conjunto de test para movimiento continuo (izqda.) y para movimiento individual (drcha.)

De igual manera, una vez los modelos se implementaban en la placa de Arduino, los resultados que se observaban inclinaban la balanza hacia el modelo de movimientos continuos, ya que pese a que el otro tenía un nivel aceptable de exactitud a la hora de realizar la clasificación de movimientos reales, en muchos de los casos primero lo clasificaba de forma incorrecta y luego de forma correcta, o por el contrario se tenía que repetir varias veces el movimiento que se deseaba reconocer antes de que lo reconociese finalmente. De esta repetición surgía el inconveniente de que al haber diferenciado el

movimiento de izquierda-derecha y derecha-izquierda, en algunos casos se realizaba uno, pero reconocía el opuesto.

Por su parte, el modelo que trata de clasificar movimientos continuos presenta la desventaja de que algún movimiento que se realizase de forma involuntaria podía ser clasificado como uno de los movimientos a reconocer, pese a introducir el sistema de detección de anomalías, ya que este no es perfecto de manera absoluta. Aunque, a diferencia del anterior modelo para movimientos individuales, en este caso el reconocimiento de movimientos era mucho más preciso y no era necesario estar realizan el movimiento durante más de uno o dos segundos con el fin de que se reconociese.

Con los resultados obtenidos para los modelos de movimiento resulta evidente que los movimientos continuos, pese a que pueden parecer menos intuitivos a la hora de ser consecuentes con la acción que realizan en la aplicación de PowerPoint, son mucho más fiables a la hora de ser reconocidos por un modelo de aprendizaje automático. De manera similar, tal y como se ha mencionado anteriormente, el modelo para movimientos individuales es mucho menos preciso a la hora de discernir cualquier movimiento frente a uno de los que se deseaba, mostrando la mayoría de las veces un movimiento como desconocido o detectándolo como un movimiento anómalo.

Similarmente a lo realizado en el modelo de reconocimiento de voz, en la librería que se implementa en el dispositivo se han probado a modificar diferentes parámetros con el fin de mejorar la precisión de los modelos de reconocimiento de gestos. En este caso, el modelo realiza una serie de inferencias y, según el número que acierte con un grado de confianza, devuelve el comando reconocido. Por lo que se tienen tres parámetros distintos para poder modificar, además del tiempo entre inferencias. Si se aumenta el número de inferencias, será necesario aumentar el número de aciertos si se desea mantener el porcentaje total de aciertos y no interpretar los resultados como desconocidos. Los datos que se muestran por el puerto serie se han limitado al resultado más probable, así como un vector con el número de aciertos para cada comando, tal y como se observa en la Figura 36.

```
17:13:27.415 -> uncertain  [ 0, 4, 0, 4, 0, 0, ]
17:13:27.696 -> uncertain  [ 0, 3, 0, 5, 0, 0, ]
17:13:27.974 -> vertical   [ 0, 2, 0, 6, 0, 0, ]
17:13:28.254 -> vertical   [ 0, 1, 0, 6, 1, 0, ]
17:13:28.533 -> vertical   [ 0, 0, 0, 6, 2, 0, ]
17:13:28.766 -> uncertain  [ 0, 0, 0, 5, 3, 0, ]
17:13:29.047 -> uncertain  [ 0, 0, 0, 4, 4, 0, ]
```

Figura 36. Salida del puerto serie para el modelo de reconocimiento de movimientos

Capítulo 6: Conclusiones y mejoras

6.1 Conclusiones.

En este trabajo se han desarrollado y construido sendos prototipos de reconocimiento de voz y de gestos implementados en un microcontrolador de bajo coste, como paso inicial a lo que podrían ser en el futuro los nuevos dispositivos de interfaz humano computador, por ejemplo, para su uso en presentaciones tipo PowerPoint o similar. Este trabajo es solamente una pequeña contribución en esa dirección.

Por otro lado, hemos comprobado el interés y potencia de la herramienta online *Edge Impulse*, que mediante un simple navegador de internet permite el desarrollo de sistemas inteligentes implementados en microcontroladores, permitiendo el desarrollo de sistemas TinyML de una forma mucho más simple y rápida que utilizando otras herramientas, como TensorFlow y Tensor Flow Lite para microcontroladores. *Edge Impulse* permite desarrollar todo el sistema en un único entorno, desde la captura de datos, su preprocesamiento y extracción de características, desarrollo del modelo de aprendizaje automático, entrenamiento, evaluación e implementación en el dispositivo final (despliegue). Hemos comprobado la potencia y facilidad de uso de *Edge Impulse*, incluso por parte de personal sin grandes conocimientos de *machine learning*, como era el caso. Hay que destacar que esta herramienta no permite la implementación en cualquier dispositivo del mercado, solo de aquellos soportados por ella que, por otra parte, son muchos dispositivos comerciales. En concreto hemos comprobado que Arduino Nano 33 BLE es una placa de desarrollo barata, pero muy potente y con un montón de posibilidades gracias a la cantidad de sensores que contiene.

Finalmente, analizando lo realizado durante el proyecto y comparando todos los modelos generados, el de reconocimiento de voz y los dos de reconocimiento de movimientos gestuales, se puede llegar a las siguientes conclusiones.

Empezando por la generación de una base de datos, y como se ha ido recalcando a lo largo de toda la memoria, es de extrema importancia tener una base de datos lo más grande posible, con muestras variadas y con un etiquetado lo más homogéneo que se pueda (clases equilibradas).

Siguiendo con el procesado de los datos, se ha de trabajar en la extracción de características hasta alcanzar una buena separabilidad visible con técnicas como PCA y t-SNA, de tal manera que el algoritmo de *machine learning* posterior sea capaz de diferenciar con mayor eficacia las diferentes etiquetas. Y en función de los datos que se vayan a utilizar, hay que escoger aquel preprocesado en el que haya un equilibrio entre extracción de parámetros clasificables y tiempo empleado en realizar dicho procesado, pues no hay que olvidar que al final todo el algoritmo se integrará en un dispositivo de recursos computacionales limitados.

Por último, respecto a los modelos desarrollados a lo largo de este proyecto, alguien sin experiencia en inteligencia artificial podría pensar *a priori* que reconocer de movimientos gestuales debería ser menos complicado que reconocer comandos de voz. Por el contrario, se ha podido concluir que el reconocedor de voz presenta mejores

resultados tanto a la hora de reconocer los comandos como de omitir aquellos datos que sean diferentes de los comandos. No obstante, hay que tener en cuenta que, en nuestro caso, por las limitaciones temporales asociadas a un trabajo fin de grado, las muestras incluidas en la base de datos desarrollada para reconocer voz pertenecen en gran parte a un solo hablante, las cosas podrían ser diferentes en el caso de bases de datos de múltiples y variados hablantes.

Las desventajas que se han encontrado en el modelo de reconocimiento de voz son propias del modo en el que se ha generado la base de datos, ya que en función de la entonación que se utilice para realizar los comandos, estos se reconocen de mejor o de peor manera. Además, en caso de que no haya reconocido el comando, para poder volver a realizar una inferencia de manera satisfactoria hay que esperar unos segundos para volver a pronunciar un comando. Además, el modelo puede fallar ante palabras que se asemejen a los comandos que ha de reconocer.

Por su parte, el modelo de reconocimiento de gestos desarrollado tiene el inconveniente de que es dependiente de la forma en que se sujeta el dispositivo y, por tanto, la colocación de los ejes, por lo que puede clasificar incorrectamente un movimiento por el simple hecho de haber realizado dicho movimiento de manera diferente a como se realizó a la hora de generar la base de datos. No obstante, hay que tener en cuenta que en este trabajo se ha pretendido desarrollar un prototipo demostrador, en una aplicación real de interfaz humano-máquina el dispositivo podría estar forzado a ser usado en una posición concreto incorporándolo en algún tipo de pulsera (como una pulsera de actividad) o incluso anillo. Una segunda posibilidad sería trabajar en el preprocesamiento para conseguir una extracción de rasgos que haga el sistema de reconocimiento independiente de la posición espacial. Una parte positiva es que, al implementar un detector de anomalías en este modelo, es capaz de desechar de forma muy eficaz todos aquellos datos que se alejen de los etiquetados, pero siempre sujetando de manera correcta el dispositivo.

Finalmente hay que remarcar que la herramienta *Edge Impulse* cuando realiza el "despliegue" (*deployment*) del modelo en el dispositivo, es decir, cuando genera el software (firmware) a implementar en el microcontrolador, debe realizar ciertas adaptaciones del modelo en función de los recursos computacionales del microcontrolador (bits de resolución y otros), por lo que el rendimiento del modelo implementado en el dispositivo hay que volver a evaluarlo y, si el rendimiento se degrada, realizar ajustes. En nuestra experiencia podemos decir que efectivamente el rendimiento sufre una cierta degradación en su "despliegue" (implementación), en gran medida debido a que los resultados obtenidos a través de la herramienta son obtenidos mediante muestras discretizadas. Por ello, hay que intentar obtener la mayor precisión en los resultados obtenidos en la herramienta y, posteriormente, realizar las oportunas correcciones ajustando los parámetros disponibles en la librería para mejorar los resultados cuando se realizan inferencias sobre datos obtenidos de forma continua.

6.2 Mejoras

Debido a la limitación temporal propia de un trabajo fin de grado, el proyecto aún está en una fase inicial respecto a todo lo que podría llegar a ser en un futuro, aunque ya

es funcional con algunas limitaciones. Se pueden plantear muchas mejoras en varios aspectos para poder obtener un dispositivo que sea completamente funcional y comercializable.

Empezando por la parte más estética, habría que introducir el dispositivo en una carcasa donde fuese posible visualizar si el dispositivo este encendido y realizando inferencias sería un punto a favor. Además, habría que incluir alimentación a pilas y comunicación inalámbrica haciendo uso de la comunicación bluetooth que ya contiene la placa. En un producto comercial podría incluirse todo ello, por ejemplo, en forma de pulsera de actividad.

En cuanto a la parte propiamente de inteligencia artificial, un aspecto fundamental seria ampliar lo máximo posible las bases de datos, tomando muestras variadas para que la red pueda realizar el entrenamiento con datos tan diversos como pueden ser en la realidad.

En el caso concreto del reconocedor de comandos de voz, habría que incluir en la base de datos voces de diferentes tonos y volúmenes, tanto masculinas como femeninas. En un producto real se podría implementar un doble modelo en secuencia, con el fin de distinguir cuando se va a pronunciar un comando y cuando no tiene que realizar ninguna acción. De esta forma el primer clasificador estaría pendiente de recibir una palabra clave y, en caso de escucharla, entraría en funcionamiento el segundo modelo que realiza la clasificación de los diferentes comandos que se desean, de forma similar a como lo hace Apple con “Oye Siri, reproduce música” o Amazon con “Alexa, apaga la luz”.

Por parte de los modelos de reconocimiento de gestos, el mayor inconveniente del modelo desarrollado es que se ve influenciado por la forma en la que cada persona coge el dispositivo, lo que hace que los ejes del acelerómetro puedan estar en otra disposición. El ensayo con otras formas de preprocesamiento que lo insensibilicen a la orientación espacial concreta y su inclusión en una carcasa tipo pulsera de actividad sería la solución, pero se escapa ya al alcance de este trabajo.

Referencias

[1] «RAE,» [En línea]. Available: https://dle.rae.es/.

[2] «Datos.Gob.es,» [En línea]. Available: https://datos.gob.es/es/blog/como-aprenden-las-maquinas-machine-learning-y-sus-diferentes-tipos.

[3] P. Warden y D. Situnayake, TinyML Machine Learning with TensorFlow Lite on, O'Reilly, 2020.

[4] S. K. SAKSAMUDRE, P. P. SHRISHRIMAL y R. R. DESHMUKH, «A review on different approaches for speech recognition system.,» vol. 115, nº 22, 2015.

[5] «EdgeImpulse,» [En línea]. Available: https://docs.edgeimpulse.com/docs/.

[6] A. Géron, Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow, O'Reilly Media, Inc., 2019.

[7] G. M. Iodice, TinyML Cookbook Combine artificial intelligence and ultra-low-power embedded devices to make the world smarter, Packt, 2022.

[8] «Tinyml.org,» [En línea]. Available: https://www.tinyml.org/#:~:text=Tiny%20machine%20learning%20is%20broadly,variety%20of%20always%2Don%20use%2D.

[9] «IBM,» [En línea]. Available: https://www.ibm.com/es-es/cloud/learn/machine-learning.

[10] «Columbia Engineering,» [En línea]. Available: https://ai.engineering.columbia.edu/ai-vs-machine-learning/#:~:text=Put%20in%20context%2C%20artificial%20intelligence,and%20improve%20themselves%20through%20experience.

[11] Z. HE, «Accelerometer Based Gesture Recognition Using Fusion Features and SVM.,» vol. 6, nº 6, 2011.

[12] «Wikipedia,» [En línea]. Available: https://en.wikipedia.org/wiki/Main_Page.

Anexo I
Introducción a Edge Impulse

Edge Impulse (EI) es una plataforma online que facilita la creación de modelos de inteligencia artificial con la intención de implementarlos en microcontroladores o microprocesadores, además de estar permanentemente en evolución con nuevas incorporaciones de placas compatibles y añadiendo nuevas opciones. Es una plataforma que incluye un pequeño tutorial con información básica con la que empezar a trabajar, así como diversos ejemplos, todos ellos muy bien explicados y guiados paso a paso, además de disponer recomendaciones en todas acciones que se quieren realizar a lo largo de la creación e implementación del impulso.

Al crear un proyecto, este solicita que se especifique a que está destinado el mismo, o lo que es lo mismo, que tipo de datos va a usar como entrada, con 4 opciones: datos de un acelerómetro, datos de audio, imágenes u otro no especificado. En este proyecto se han manejado tanto datos del acelerómetro como datos de audio, para esta introducción a EI se va a suponer que son datos de audio.

Se dispone en todo momento de un menú lateral en el que aparecen todos los accesos directos y en el orden que hay que seguir para conseguir un modelo y poder implementarlo en un dispositivo.

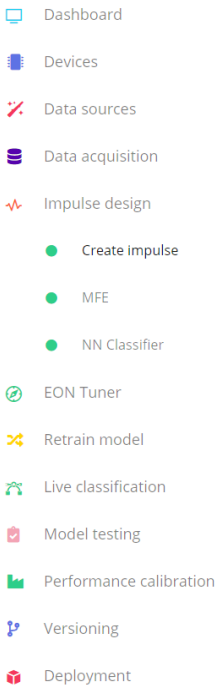


Figura 37. Anexo I, Pestañas en Edge Impulse

Los pasos que seguir a la hora de empezar con Edge Impulse son sencillos, incluso sin tener un microcontrolador, ya que se puede implementar el algoritmo final en un dispositivo móvil o en el propio ordenador.

Si se desea trabajar con una placa con microcontrolador como Arduino Nano 33 Sense, es necesario instalar una serie de softwares para que la aplicación sea capaz de comunicarse con la placa para, por ejemplo, recabar datos de los sensores: el CLI tanto de Arduino como de Edge Impulse, posteriormente se descarga la última versión del software de Edge Impulse para Arduino Nano 33 Sense (que es el dispositivo que nos ocupa en este caso).

Si se desea recabar datos con un dispositivo móvil, ya sea smartphone o tablet, es tan sencillo como entrar en la pestaña dispositivos y añadir un dispositivo móvil que hace aparecer un código QR, tras escanearlo ya podemos recabar datos desde el móvil.

Tras conectar el dispositivo se puede generar la base de datos. Con la placa de Arduino se ha de seleccionar el sensor con el que se quiere recabar datos, que en este caso es el micrófono, la frecuencia y la duración de la muestra, así como la etiqueta que se desea poner. No hay que preocuparse si la duración es demasiado larga ya que posteriormente se pueden realizar cortes, además de que, si se han capturado las palabras a distinguir con una pausa entre ellas, dicha pausa se puede eliminar tal y como se ve en la siguiente figura.

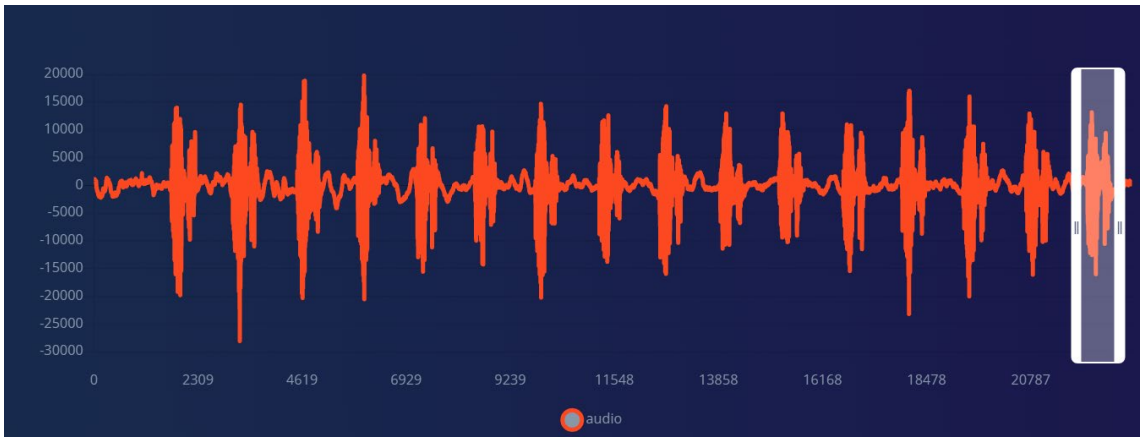


Figura 38. Anexo I, Ejemplo de datos y corte.

Una vez capturada la muestra, se puede ver y reproducir para ver si satisface nuestras necesidades. Algo que resulta muy útil es que en todo momento en el apartado de *Data Adquisition* aparece un diagrama de sectores con el porcentaje de datos que tiene cada una de las etiquetas, y de forma complementaria se puede visualizar también el porcentaje de datos destinados al entrenamiento y al testeo.

En esta pestaña también se puede realizar una visualización de los datos, teniendo la posibilidad de obtenerlo a través de los bloques seleccionados para el denominado "impulso" (el algoritmo), del impulso ya entrenado o en el caso de tener muestras de audio de menos de un segundo, permite obtenerlo mediante un modelo propio de la herramienta. En la siguiente figura se muestra las opciones que existen.

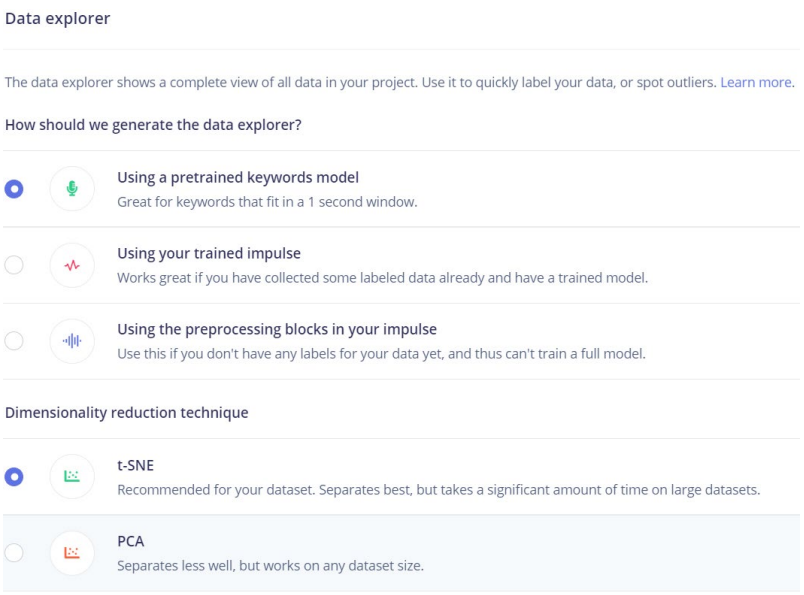


Figura 39. Anexo I, Formas de explorar los datos

Una vez se tiene creada la base de datos, pasamos a la pestaña de *Impulse*, que está dividida en tres secciones. La primera de ellas es especificar las características de los datos de entrada, dichas especificaciones incluyen el tamaño de la ventana, el tiempo entre ventanas, la frecuencia de los datos y qué hacer si se encuentran datos con una duración menor a la ventana (en la herramienta aparece como *Zero-Pad Data*), sin descartar dichos datos o rellenar los datos con valor cero.

El segundo sector está destinado a los bloques de procesamiento y se dispone de un amplio abanico de opciones para procesar los datos antes de usarlos, aunque si no se desea realizar ningún procesamiento existe un bloque para los datos crudos. Por otra parte, también se puede crear un bloque personalizado. Por último, está la sección de los bloques de aprendizaje entre los que se incluyen clasificadores, regresores y detector de anomalías; con posibilidad de incluir uno propio de igual manera.

En el caso de tener datos recabados por un sensor que proporciona más datos de los necesarios y solo necesitar uno de ellos (o solo necesitar los datos de un solo eje, en el caso de un acelerómetro), en el apartado de procesamiento se pueden seleccionar los datos que se desean incluir. Un ejemplo sería tener datos inerciales que incluye datos de un acelerómetro y un giróscopo y solo necesitar los datos del primero de ellos.

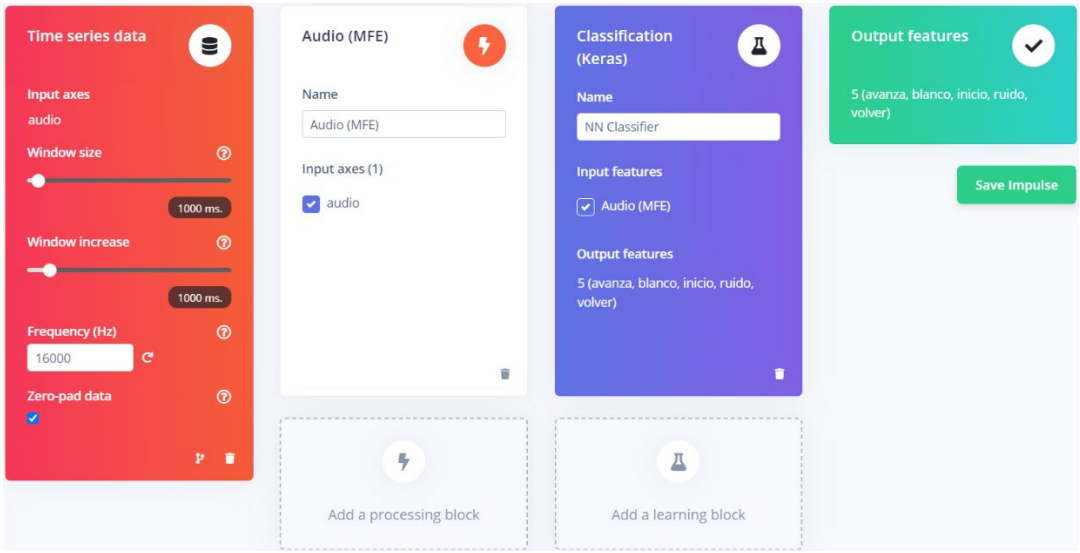


Figura 40. Anexo I, Ejemplo de impulso

Tras esto se procede a añadir los bloques de procesamiento y los bloques de aprendizaje. Generalmente, los bloques que recomienda la propia aplicación suelen dar resultados muy positivos, además de que para generar un modelo rápido se pueden implementar bloques aleatorios y posteriormente ejecutar el sintonizador EON (*Edge Optimized Neural*), con el fin de realizar una búsqueda del mejor modelo modificando pequeños parámetros y tras este ya realizar una búsqueda más profunda de mejoras.

Siguiendo el ejemplo de un modelo de reconocedor de voz, al añadir un bloque de procesamiento aparecen los listados en la imagen inferior, los que recomienda la aplicación son los marcados con una estrella, además de que todos incluyen una breve descripción para lo que deberían funcionar mejor.

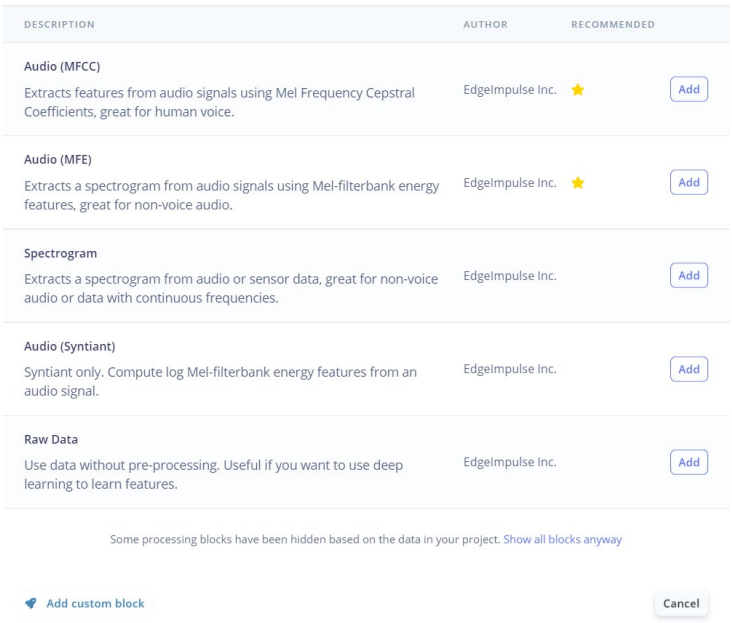


Figura 41. Anexo I, Tipos de bloques de procesamiento

Y realizando lo propio para los bloques de aprendizaje, también aparecen diferentes bloques en función de los datos que se haya introducido y aquello que se desee obtener, dado que las etiquetas son texto, no cabe usar un regresor cuya función principal es predecir variables numéricas.

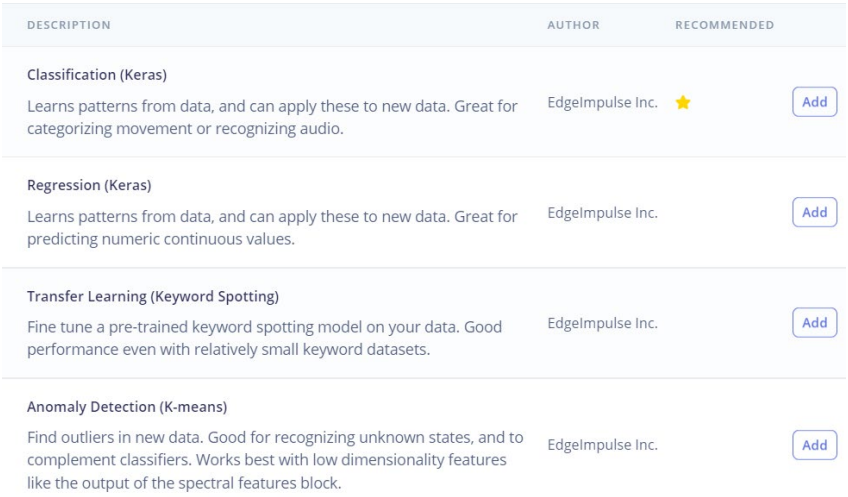


Figura 42. Anexo I, Tipos de bloques de aprendizaje

Una vez se han elegido tanto los bloques de procesamiento como de aprendizaje, se guarda el impulso y ya se pueden seleccionar los parámetros de estos bloques. Empezando por el bloque de procesamiento, que en este caso va a ser el bloque de audio MFCC, aparecen los diferentes parámetros que se pueden modificar, así como un pequeño ejemplo de lo que genera dicho bloque de procesamiento.

Una parte positiva es que al ir variando los diferentes parámetros se puede ver cómo va modificándose la salida del bloque en los diferentes datos. Una vez se han seleccionado los parámetros, se pulsa generar parámetros, tras esto aparece la clasificación que ha realizado el bloque de procesamiento con los parámetros impuestos.

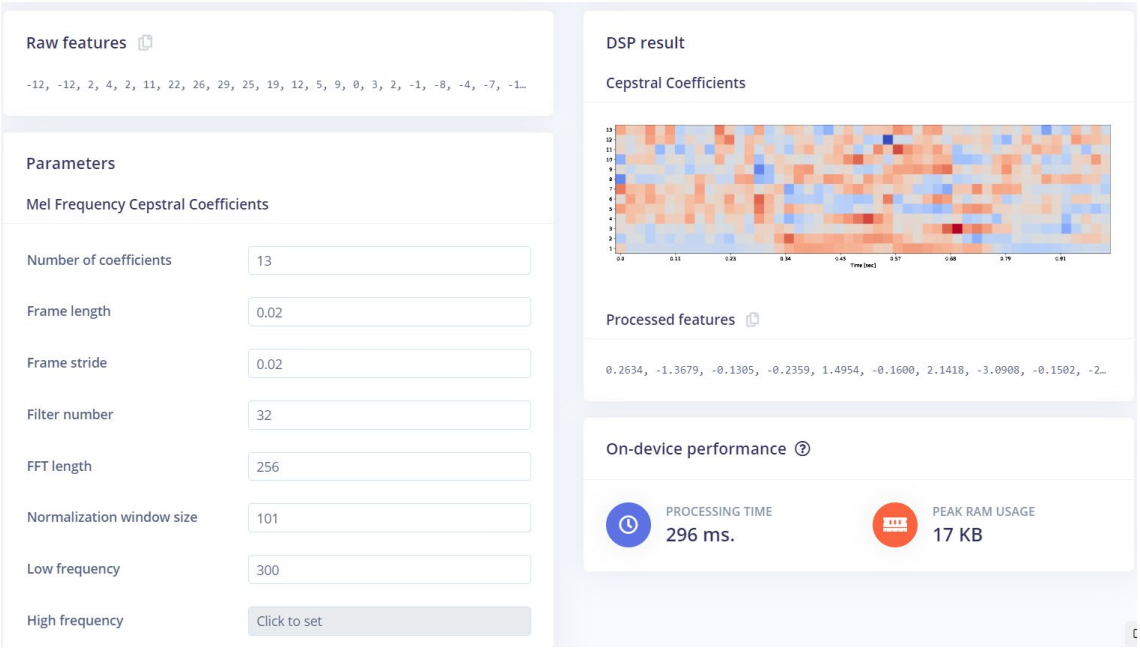


Figura 43. Anexo I, Parámetros del bloque de procesamiento

Figura 44. Anexo I, Ejemplo de resultado del bloque de procesamiento

Una vez se han obtenido los resultados del bloque de procesamiento, se puede continuar ajustando las diferentes opciones a la hora de entrenar en el bloque de aprendizaje. Así como crear un modelo personalizado, tanto con los diferentes bloques que permite la aplicación, como entrar en un modo avanzado y escribir el código en Python para personalizar y generar modelos más complejos. Empezando por los ajustes del entrenamiento, están el ritmo de entrenamiento y el número total de iteraciones que va a realizar el entrenamiento antes de darse por concluido, así como el porcentaje de datos usado para validar los datos y que los resultados sean más robustos. También existe la posibilidad de activar diferentes parámetros extra para implementar más funciones.

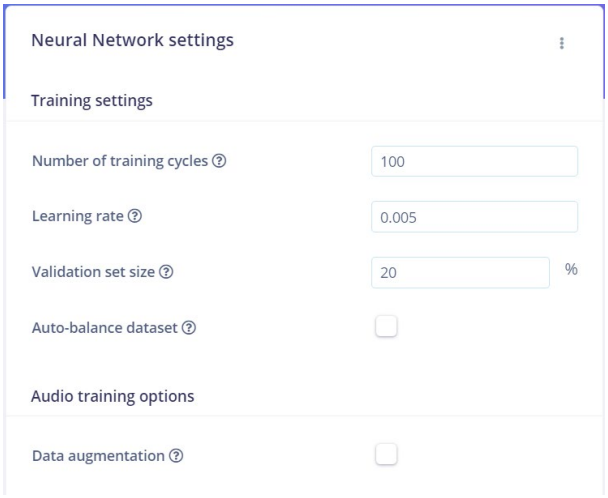


Figura 45. Anexo I, Ajustes generales del bloque de aprendizaje

El siguiente paso es la generación del modelo, la propia aplicación ofrece dos modelos convolucionales de forma predeterminada, aunque se pueden implementar tal y como se ha dicho cualquier modelo. Hay una limitación, la aplicación dispone de una versión de pago que permite realizar entrenamientos sin límite, mientras que la versión gratuita está limitada a entrenamientos que no superen los 20 minutos, ya que si es superior indica que son más complejos. Una vez se haya generado el modelo deseado, se pulsa el botón de comenzar entrenamiento, tras lo cual aparece una matriz de confusión, así como un gráfico donde se puede visualizar como ha realizado la clasificación el modelo y aquellos puntos que ha errado al realizar dicha clasificación.

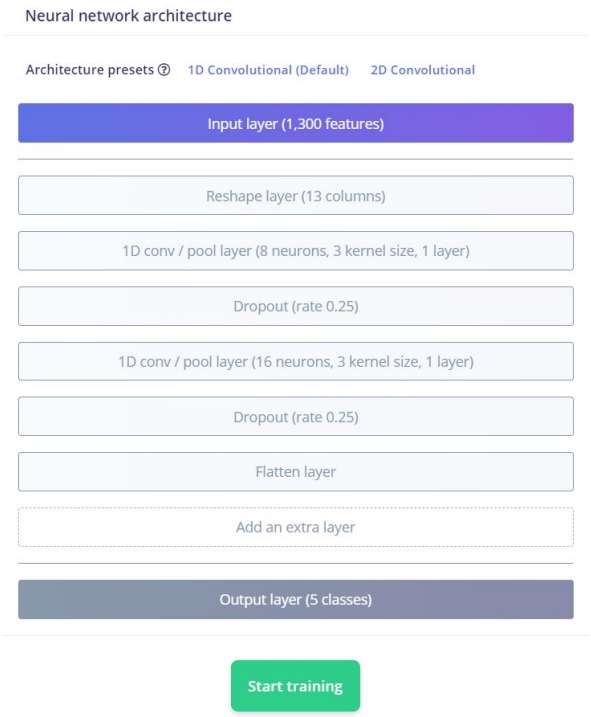


Figura 46. Anexo I, Ejemplo de modelo de aprendizaje

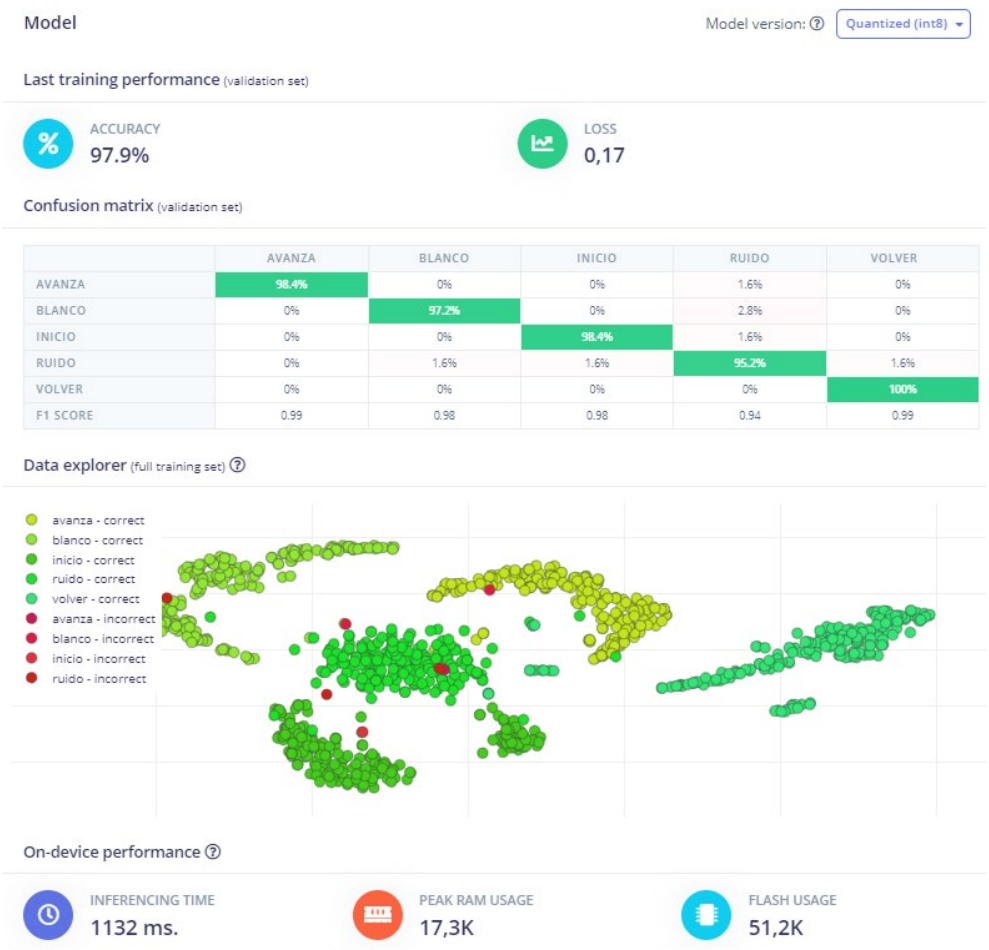


Figura 47. Anexo I, Resultados del bloque de aprendizaje

Una vez se ha obtenido el modelo, se puede comprobar su funcionamiento con datos que se pueden generar en ese mismo momento a través de la pestaña *Live Classification*. Los datos generados se guardan en la base de datos destinada a la prueba, también permite seleccionar una muestra de los datos de prueba y ver como interactúa con el modelo y si se predice correctamente. Una ventaja de esta pestaña es que se puede grabar un audio con las diferentes palabras a reconocer y se muestran aquellas que ha reconocido y en qué momento de la pista grabada ocurre, por lo que se puede comprobar de manera rápida si el modelo funciona correctamente.

En la pestaña de *Model testing*, se puede realizar la clasificación de todas las muestras destinadas a la prueba de manera rápida. También se indica el número de veces que identifica cada muestra con los resultados, aunque no se puede identificar en que instante de la muestra se ha identificado cada resultado. En esta pestaña también aparece una matriz de confusión donde se pueden visualizar el número de veces que se ha identificado correctamente cada muestra y cuantas veces se ha identificado cada muestra de manera incorrecta o si no se ha podido identificar la muestra y por tanto es desconocido.

La siguiente pestaña, *Performance Calibration*, está aún en fase de pruebas y, por lo tanto, no está disponible para todos los proyectos, pero está destinada a incorporar un filtro para mejorar los resultados una vez haya sido implementado el modelo en el dispositivo. En los diferentes modelos realizados durante este proyecto no se ha implementado en ninguno.

Siguiendo con las pestañas, *Versioning* es donde se puede ir guardando diferentes versiones del modelo a implementar y en caso de querer recuperar una versión anterior es posible realizarlo desde esta pestaña. A la hora de guardar cada versión es posible añadir diferentes comentarios para facilitar la descripción de la misma.

El paso final es la pestaña de *Deployment*, en esta pestaña aparecen disponibles todas las librerías en las que se pueden exportar el modelo generado, de igual manera aparecen los dispositivos donde se puede implementar de manera directa, entre los que se ubican diferentes dispositivos, así como el ordenador o el teléfono móvil.

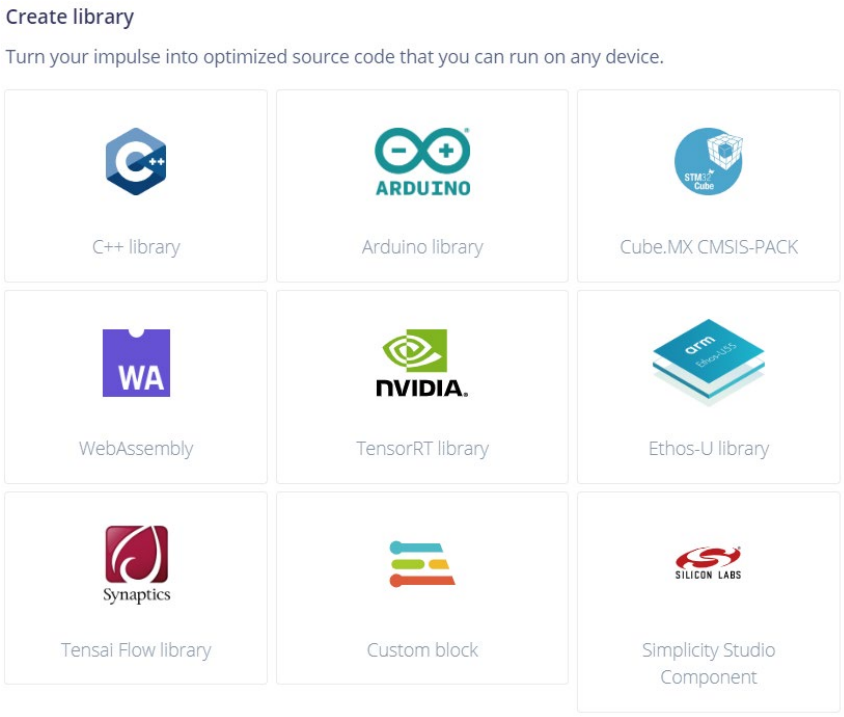


Figura 48. Anexo I, Librerías disponibles

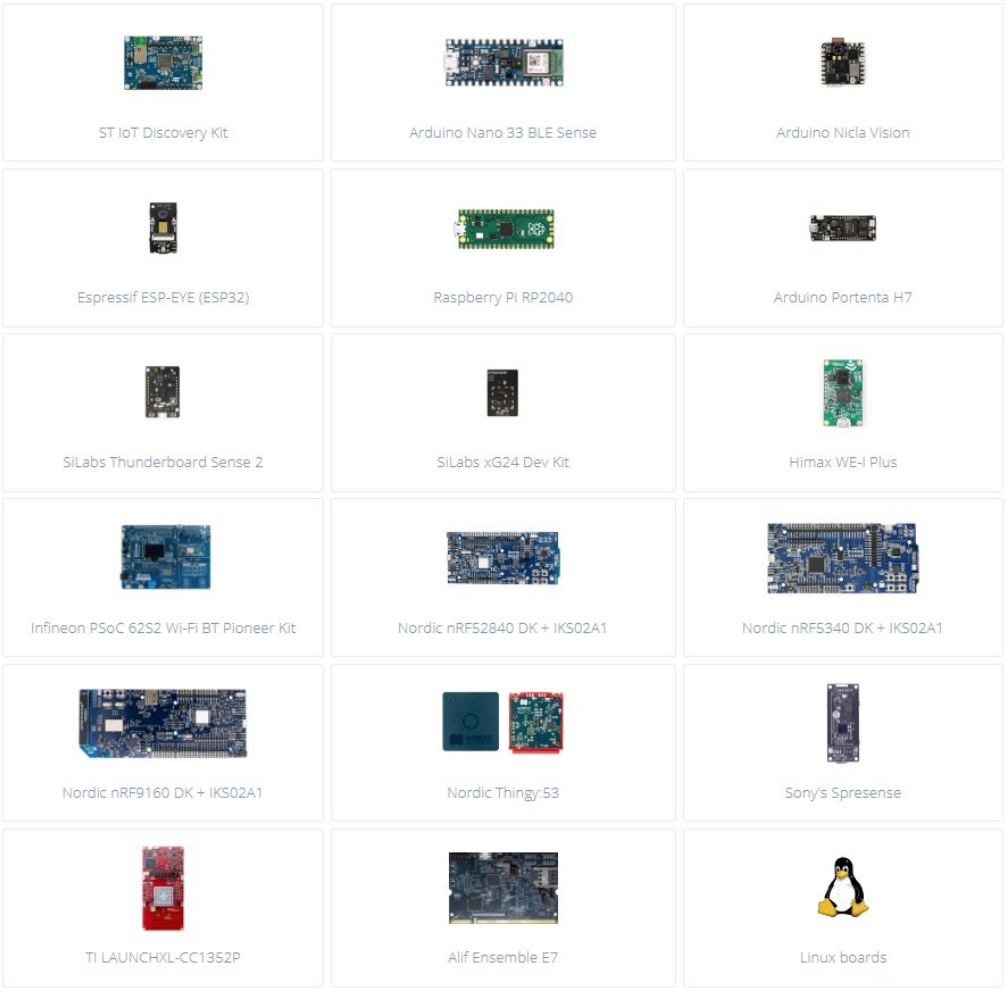


Figura 49. Anexo I, Dispositivos de implementación directa

En el caso de exportar el proyecto en formato de librería para Arduino, el proceder es sencillo. Una vez se ha exportado y sin llegar a descomprimir el archivo .zip, se implementa la librería directamente desde el Arduino IDE. Una vez se ha añadido la librería, en la sección de ejemplos se puede encontrar códigos de demostración para diferentes dispositivos. En el caso de usar la placa de Arduino Nano 33 Sense y en modelo de reconocimiento de sonido, se selecciona la misma y el ejemplo que incluye micrófono continuo. Los códigos que se incluyen a modo de ejemplo son completamente operativos, una vez se ha cargado al dispositivo va mostrando por el puerto serie los resultados que va obteniendo.

Como información adicional, aunque el modelo haya mostrado valores muy elevados de exactitud, es posible que una vez implementado en el dispositivo no funcione de la manera esperada. Algunos parámetros que se pueden modificar para mejorar el funcionamiento en el dispositivo son el número de ventanas y el tiempo entre inferencias. Además de eso será necesario realizar iteraciones con pequeñas variaciones en el impulso para mejorar el funcionamiento real en el dispositivo.