Diseño e Implementación de un Sistema de Reconocimiento de Voz basado en Microcontroladores

Fritz Alexis Ariza Carrero y Juan Sebastián Román Paz

Trabajo de Grado para Optar al Título de Ingeniero Electrónico

Director

Jaime Guillermo Barrero Pérez

Magíster en Potencia Eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Ingeniería Electrónica

Bucaramanga

2025

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

Pág.

Resumen

Título: Diseño e Implementación de un Sistema de Reconocimiento de Voz basado en Microcontroladores[*]

Autor: Fritz Alexis Ariza Carrero y Juan Sebastián Román Paz[**]

Palabras Clave: Bases de datos, Reconocimiento de voz, Redes neuronales y Microcontroladores.

Descripción: Este proyecto aborda el desafío de implementar sistemas de reconocimiento de voz en español en microcontroladores de bajo costo, una tecnología ampliamente dominada por modelos en inglés, lo que limita su alcance en comunidades hispanohablantes. En el trabajo se diseña e implementa un sistema basado en redes neuronales convolucionales (CNN) integradas en la placa de desarrollo ESP32 Audio Kit V2.2, capaz de clasificar entre cinco comandos de voz en español con presencia de ruido. La base de datos creada, balanceada en sus etiquetas, incluye 1,548 audios recolectados en diferentes entornos. El diseño de la red neuronal alcanzó una precisión del 89,40%. La solución propuesta destaca por su accesibilidad económica y su potencial impacto en la inclusión tecnológica, beneficiando aplicaciones en entornos educativos y profesionales. Este enfoque representa una contribución en la universalidad del reconocimiento de voz, adaptado a las necesidades de comunidades con recursos limitados y entornos diversos.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Ingeniería Electrónica. Director: Jaime Guillermo Barrero Pérez. Magister en Potencia Eléctrica.

Abstract

Title: Design and Implementation of a Speech Recognition System based on Microcontrollers[*]

Author(s): Fritz Alexis Ariza Carrero y Juan Sebastián Román Paz [**]

Key Words: Databases, Neural Networks, Microcontrollers and Speech Recognition**.**

Description: This project addresses the challenge of implementing spanish speech recognition systems on low-cost microcontrollers, a technology largely dominated by english models, which limits its reach in Spanish-speaking communities. We design and implement a system based on convolutional neural networks (CNN) integrated on the ESP32 Audio Kit V2.2 development board, capable of classifying between five Spanish voice commands and noise. The database created perfectly balanced in its labels includes 1,548 audios collected in different environments. The neural network design achieved an accuracy of 89.40%. The proposed solution stands out for its economic accessibility and its potential impact on technological inclusion, benefiting applications in educational and professional environments. This approach represents a contribution in the universality of speech recognition, adapted to the needs of communities with limited resources and diverse environments.

[*] Degree Work

[**]School of Physicomechanical Engineering. School of Electrical, Electronic and Telecommunications Engineering. Electronic Engineering. Director: Jaime Guillermo Barrero Pérez. Master in Electrical Power

Introducción

Según Deng & Yu (2013) el aprendizaje profundo ha redefinido el campo de reconocimiento de voz, proporcionando soluciones más precisas y robustas frente a las variaciones del habla, ruido ambiental y diferentes idiomas. No obstante, la mayoría de estas soluciones están diseñados para operar en inglés, limitando su accesibilidad en comunidades hispanohablantes. Este proyecto propone la creación de un sistema de reconocimiento de voz en español basado en microcontroladores, buscando una solución efectiva y económica que amplíe el acceso a esta tecnología en contextos con recursos limitados.

Ahondando en este tipo de soluciones, Xie & Shen desarrollaron un sistema inteligente que utiliza asistentes de voz e IoT para monitorizar el estado de las plantas, esto con ayuda de un microcontrolador.

Por otra parte, el actual proyecto presenta cómo diseñar redes neuronales y construir una base de datos equilibrada para lograr un reconocimiento de voz aceptable, el cual sea adaptable a diferentes hablantes y condiciones variantes de ruido, además de ser implementado en un microcontrolador.

El impacto de este proyecto responde a la necesidad de herramientas accesibles en español, promoviendo el avance de la inteligencia artificial aplicada a plataformas portables de bajo costo, con un impacto tanto tecnológico como social, al beneficiar a usuarios en entornos educativos, médicos y profesionales mediante una interacción más inclusiva y eficiente con dispositivos electrónicos.

1. Objetivos

1.1 Objetivo General

Diseñar e implementar un sistema basado en un microcontrolador capaz de reconocer al menos tres comandos de voz en español de diferentes hablantes.

1.2 Objetivos Específicos

Elegir un micrófono que permita capturar sonidos para enviar esta información a un sistema de desarrollo, considerando aspectos tanto técnicos como económicos.

Seleccionar un sistema de desarrollo de bajo costo con capacidad de ejecutar redes neuronales, para el reconocimiento de al menos tres comandos de voz en español.

Consolidar una base de datos de comandos de voz en español, utilizando un conjunto de datos previamente seleccionado y muestras propias, para entrenar y validar el modelo de reconocimiento de voz pre entrenado elegido para el sistema.

Evaluar y validar el funcionamiento del sistema implementado, midiendo su precisión y eficiencia en el reconocimiento de comandos de voz en el idioma español.

2. Marco Teórico

A continuación, se abordan conceptos clave como redes neuronales, las herramientas y técnicas más relevantes en el campo; además, se examina la implementación mediante el uso de microcontroladores. Esto en busca de introducir las bases conceptuales para el desarrollo y cumplimiento de los objetivos planteados en el proyecto.

2.1 Deep Learning

El aprendizaje profundo (*Deep Learning*), una rama del aprendizaje automático (*Machine Learning*), es una técnica avanzada de la Inteligencia Artificial (IA) que utiliza redes neuronales las cuales emulan el aprendizaje del cerebro humano. Estas redes procesan grandes volúmenes de datos para identificar patrones complejos, permitiendo tareas como la clasificación, detección o predicción con alta precisión. A diferencia de otros enfoques como regresiones o árboles de decisión, el DL sobresale en escenarios donde la riqueza y cantidad de datos son cruciales, aunque también exige un mayor poder computacional. Este enfoque, central en la evolución de la IA, ha revolucionado campos como el reconocimiento de voz, visión por computadora y procesamiento del lenguaje natural.

2.2 Redes Neuronales Convolucionales

Una red neuronal convolucional (CNN) es un algoritmo de aprendizaje profundo capaz de analizar imágenes, asignar pesos y sesgos a sus elementos y diferenciarlos entre sí. A diferencia de otros métodos de clasificación, las CNN requieren menos preprocesamiento, ya que pueden

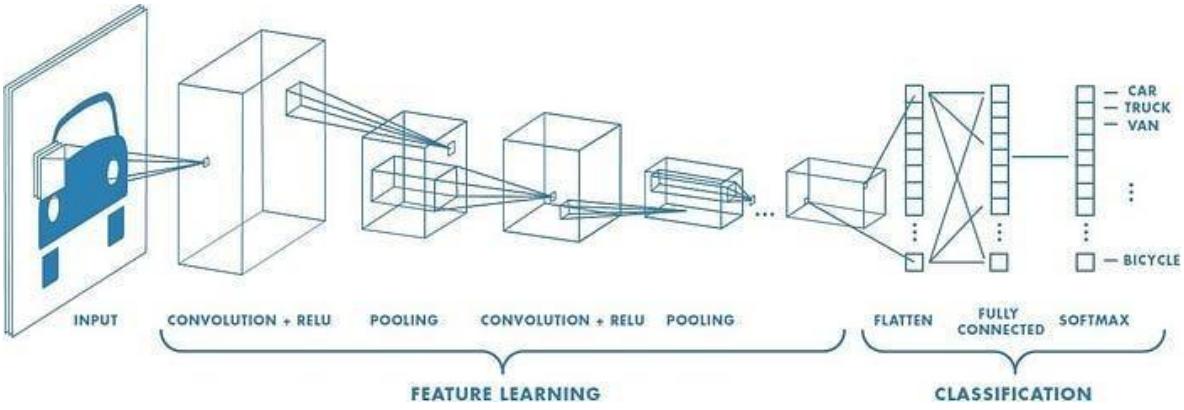
aprender automáticamente los filtros o características necesarios durante el entrenamiento, en lugar de diseñarlos manualmente.

Su arquitectura se inspira en la corteza visual del cerebro humano, donde las neuronas responden a estímulos en áreas específicas, conocidas como campos receptivos. Estos campos se superponen para abarcar toda la región visual, permitiendo una comprensión más completa de la imagen.

Durán (2017) plantea que una red neuronal convolucional es un tipo de red multicapa que consta de diversas capas convolucionales y de pooling (submuestreo) alternadas, y al final tiene una serie de capas full-connected como una red perceptrón multicapa.

Figura 1

Arquitectura de una CNN para clasificar vehículos



Nota. Figura 1 Modelo de red neuronal para la clasificación de vehículos (Saha, 2018)

2.2.1 Uso de las redes CNN para el reconocimiento de voz

Aunque hay otros modelos de redes neuronales, como las redes neuronales recurrentes (RNN), que procesan directamente la señal de audio para tareas de reconocimiento y clasificación de sonido, las CNN son ampliamente utilizadas debido a su eficiencia y versatilidad en este campo.

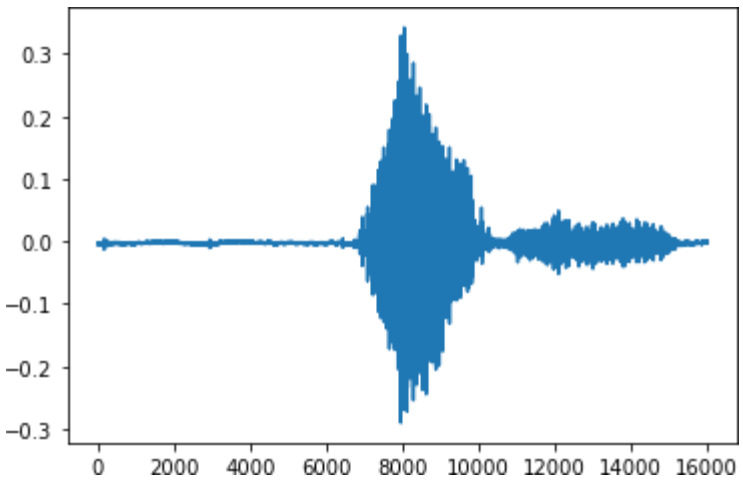
Al igual que ocurre con las imágenes, tenemos que convertir nuestro mundo físico en números o en una representación digital que un ordenador pueda entender. En el caso del audio, se utiliza un micrófono para captar el sonido y, a continuación, se convierte de sonido analógico a sonido digital mediante el muestreo a intervalos de tiempo constantes. Esto se denomina frecuencia de muestreo.

Cuanto mayor es la frecuencia de muestreo, mayor es la calidad del sonido; sin embargo, a partir de cierto punto, el oído humano no puede detectar la diferencia. La frecuencia de muestreo típica utilizada para aplicaciones de audio es de 48 kHz o 48.000 muestras por segundo. El audio puede grabarse en diferentes canales. Por ejemplo, las grabaciones estéreo tienen 2 canales, derecho e izquierdo (Microsoft, s.f.).

2.2.1.1 Forma de Onda. Al muestrear audio, se registra la amplitud de la señal a una frecuencia específica. Con esta información, se crea una forma de onda que puede representarse gráficamente como una señal en función del tiempo.

Figura 2

Ejemplo de forma de Onda para un audio diciendo la palabra: yes

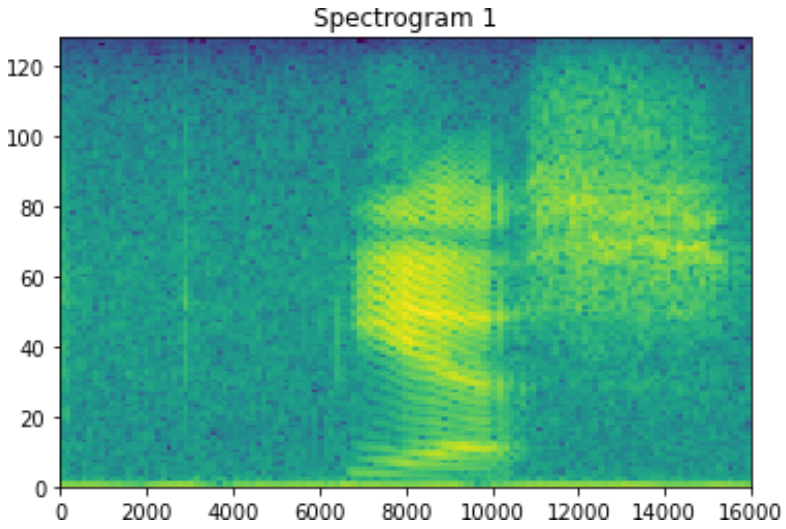


Nota. Figura 2 Ejemplo de forma de onda (Microsoft, s.f.).

2.2.1.2 Espectrograma. A partir de la forma de onda, es posible generar un espectrograma, que muestra una representación visual de la amplitud en función del tiempo y la frecuencia. En la Figura 3, el eje horizontal corresponde al tiempo, el eje vertical a la frecuencia, y los colores indican la amplitud.

Figura 3

Ejemplo de un espectrograma para un audio diciendo la palabra: yes



Nota. Figura 3 Ejemplo de espectrograma (Microsoft, s.f.).

2.3 Principales herramientas usadas en la creación de redes neuronales

Google (s.f.). define las características de Colab de la siguiente manera: Colab es un servicio alojado de Jupyter Notebook que no requiere configuración y que ofrece acceso sin coste económico a recursos de computación, como GPUs y TPUs. Colab es una solución especialmente adecuada para el aprendizaje automático, la ciencia de datos y la educación.

Por su parte, TensorFlow es una biblioteca de código abierto desarrollada por Google para realizar tareas de aprendizaje automático y aprendizaje profundo. Según TensorFlow (s.f.),

“TensorFlow facilita la creación de modelos de aprendizaje automático para computadoras de escritorio, dispositivos móviles, la web y la nube.” (p.1)

Mientas que Keras es una biblioteca de código abierto que se utiliza para el aprendizaje automático y DL, ofrece API consistentes y simples, minimiza la cantidad de acciones del usuario necesarias para casos de uso comunes y proporciona mensajes de error claros y prácticos. Keras también le da la máxima prioridad a la elaboración de una excelente documentación y guías para desarrolladores. (Keras, s.f.).

2.4 Uso de microcontroladores para el aprendizaje automático

Los microcontroladores ofrecen varias ventajas para el aprendizaje automático y la inteligencia artificial, destacando por su bajo consumo de energía, lo que los hace ideales para aplicaciones autónomas sin necesidad de estar conectados a una fuente de energía constante. Además, son económicos, lo que los convierte en una opción asequible en comparación con estaciones de trabajo de alto rendimiento.

Su flexibilidad es otra ventaja, ya que están integrados en dispositivos cotidianos, lo que permite agregar inteligencia artificial sin depender de la red. También, los microcontroladores permiten procesar datos localmente, protegiendo la privacidad al evitar el envío de información sensible a la nube.

2.4.1 Protocolo I2S

Es un protocolo de comunicación digital diseñado específicamente para la transferencia de datos de audio entre componentes electrónicos. Por ejemplo, en la ESP32, I2S facilita la captura y reproducción de audio en tiempo real al comunicarse con un codec, como el ES8388 o

directamente con sensores digitales. Su integración con microcontroladores permite aplicaciones diversas que van desde reconocimiento de voz hasta grabación de audio en proyectos embebidos.

2.5 Revisión del estado del arte

En los últimos años, el desarrollo de sistemas de reconocimiento de voz basados en microcontroladores ha tomado gran relevancia, impulsado por los avances en inteligencia artificial y la expansión de plataformas IoT. Macías Reyes (2024) diseñó un asistente de voz que, mediante un ESP32, delega tareas complejas a la nube utilizando servicios como Google Cloud Speech-to-Text y la API de ChatGPT para procesamiento semántico, lo que demuestra la viabilidad de una arquitectura híbrida que combina eficiencia local y capacidades avanzadas en la nube. De forma complementaria, Xie y Shen (2023) desarrollaron un sistema de monitoreo ambiental activado por voz, utilizando un ESP32-Lyrat para convertir comandos hablados en texto, acceder a datos vía ThingsBoard, y proporcionar respuestas mediante síntesis de voz, eliminando así la necesidad de interfaces gráficas tradicionales. Por su parte, López Neira (2023) propuso una solución domótica por voz basada en NodeMCU V3 (ESP8266), programada mediante la IDE de Arduino y una infraestructura distribuida con protocolos HTTP y MQTT, integrando además una habilidad personalizada para Alexa que permite la interacción natural con el entorno, y la implementación de un servidor central mediante una Raspberry Pi utilizando Node-Red para la gestión del sistema, junto con una base de datos MariaDB1.

Si bien estos proyectos evidencian importantes avances en la integración de sistemas embebidos con tecnologías de reconocimiento de voz, la mayoría dependen de soluciones en la nube para realizar tareas computacionalmente intensivas, lo que implica limitaciones en entornos con conectividad restringida. Esto resalta la necesidad de explorar alternativas locales, capaces de

ejecutar procesos de reconocimiento directamente en el dispositivo, especialmente en idiomas como el español, que aún cuentan con menor soporte. En este contexto, el desarrollo de sistemas autónomos, eficientes y adaptados lingüísticamente a nuestro idioma se plantea como un reto vigente y de alto valor para la democratización de la tecnología.

3. Desarrollo de la solución

En este capítulo se aborda el proceso empleado para lograr el objetivo de implementar un sistema para el reconocimiento de voz. Se divide principalmente en cuatro secciones: elección de componentes y sistemas de desarrollo, creación de la base de datos, diseño de la red neuronal e implementación en el microcontrolador. Buscando dar a entender el proceso de prototipado, las metodologías aplicadas, el porqué de cada decisión de diseño e implementación, así como los resultados obtenidos.

3.1 Elección de componentes

3.1.1 Tipos de micrófonos

En implementaciones con ESP32, como es el caso, se pueden utilizar diferentes tipos de micrófonos dependiendo del propósito del proyecto y las decisiones de diseño. En la Tabla 1 se muestra una breve comparación entre estos tipos de micrófonos.

Tabla 1

Comparación entre tipos de micrófonos

Característica	*Electret*	*MEMS*	*Carbón*	*Cinta/Diafragma*
Conexión a ESP	*Analógica*	*Digitales (I2S)*	*Analógica*	*Analógica*
Sensibilidad(dB)	*-45dB a -35dB*	*-46dB a -26dB*	*-60dB a -40dB*	*-60dB a -30dB*
SNR	*60dB - 70dB*	*60dB – 80dB*	*~40dB*	*70dB – 85dB*
Requiere ampl	*Sí*	*No (digitales)*	*Si*	*Si*
RF (Hz)	*20 Hz – 16kHz*	*20 Hz – 20kHz*	*300Hz – 3.5kHz*	*20 Hz – 20kHz*
Directividad	*Omnidireccional*	*Omnidireccional*	*Omnidireccional*	*Bidireccional*
Costo	*Bajo*	*Moderado*	*Bajo*	*Alto*

Nota. SNR: Relación señal/ruido, ampl: amplificador, RF: Respuesta en frecuencia.

Analizando la Tabla 1 podemos observar que las dos mejores opciones para esta implementación son los micrófonos de condensador Electret y los micrófonos MEMS digitales debido a su alta sensibilidad y buena respuesta en frecuencias de voz, lo que permite una captura clara y precisa del audio. Los Electret son económicos y fáciles de integrar con circuitos analógicos, mientras que los MEMS digitales ofrecen mayor reducción de ruido eléctrico y una salida digital directa, gracias a que se puede conectar directamente a la ESP32 a través de I2S.

Finalmente, en la siguiente sección se desarrollará el motivo por el cual se eligió la placa ESP32 Audio Kit V2.2 que incorpora micrófonos Electret, en el datasheet se observa el esquemático de conexiones de estos micrófonos a dicha placa de desarrollo.

3.1.2 Elección del sistema de desarrollo

La ESP32 Audio Kit V2.2 es una pequeña placa de desarrollo de audio desarrollada por Ai-Thinker basada en el microcontrolador ESP32 A1S. La mayoría de los periféricos de audio están distribuidos en ambos lados de la placa de desarrollo. Soporta tarjeta TF, salida de auriculares, dos entradas de micrófono y dos salidas de altavoz. (Tecnología Anxinke, s.f.). Donde

TF hace alusión a las tarjetas microSD. En la Tabla 2 se comparan algunas características de esta placa con otros microcontroladores.

Tabla 2

Comparación entre placas de desarrollo con microcontroladores

Características	ESP32 Audio Kit	ESP32-S3	Raspberry Pi PicoW	Arduino nano 33 BLE Sense
Procesador	Dual-core 240 a MHz	Dual-core a 240 MHz	Dual-core a 133 MHz	ARM Cortex a 64 MHz
Memoria RAM	520kB SRAM + 4MB PSRAM	512kB SRAM	264kB SRAM	256kB SRAM
Almacenamiento	4 MB + SD	16 MB	2 MB	1 MB
Conectividad	Wi-Fi, Bluetooth	Wi-Fi, Bluetooth	Wi-Fi	Bluetooth Low Energy
Soporte de Audio	Integrado (ES8388, I2S, DAC/ADC)	Compatible con I2S	Compatible con I2S	Micrófonos analógicos
Micrófonos	2 analógicos	No	No	Digital
Soporte para altavoces	Salidas estéreo amplificadas	Requiere módulos	Requiere módulos	Requiere módulos
Soporte de IA	Amplio (TF Lite)	Amplio	Limitado	Limitado
Costo	Moderado	Moderado	Bajo	Moderado

Se eligió la placa ESP32 Audio Kit V2.2 para la implementación del proyecto debido a sus características avanzadas que la hacen ideal para aplicaciones relacionadas con el procesamiento de audio y la inteligencia artificial. Esta placa cuenta con un procesador potente de doble núcleo,

integra el Codec ES8388, cuenta con capacidad Wi-Fi y Bluetooth integrados, así como interfaces de audio de alta calidad. En cuanto a sus micrófonos son de la rama Electret conectados directamente al codec, mientras que cuenta con conector para auriculares, así como dos salidas para altavoces con una potencia máxima de 3W por canal. Asimismo, su flexibilidad y soporte para múltiples bibliotecas de software facilitan la integración de modelos de IA, lo que la convierte en una opción excelente para el objetivo de este proyecto.

Por estas características la ESP32 Audio Kit V2.2 se destaca sobre las demás en procesamiento de audio y manejo de redes neuronales, gracias a su soporte para TensorFlow Lite. Además, contiene una ranura para tarjetas SD de hasta 64 GB, lo cual incrementa su almacenamiento. En la Figura 4 se observa la distribución de componentes en la ESP32 Audio Kit V2.2. Esta placa se adquirió mediante la página web de Mercado Libre y tuvo un costo de COP 100.000.

Figura 4

ESP32 Audio Kit V2.2

Nota. Figura 4 ESP32 Audio Kit V2.2. MercadoLibre. Recuperado 12 de enero de 2025.

3.1.3 Selección del ambiente de desarrollo de programación

Optamos por utilizar la IDE de Arduino en lugar de la plataforma de Espressif debido a su facilidad de uso, amplia documentación y sólida comunidad de soporte. Estas características simplifican la programación del microcontrolador y aceleran el desarrollo, especialmente en proyectos que integran redes neuronales y procesamiento de señales.

Aunque el entorno de Espressif (ESP-IDF) ofrece mayor control sobre el hardware y es ideal para aplicaciones avanzadas, elegimos la IDE de Arduino ya que contamos con mayor experiencia en el uso de esta misma. Además, la IDE de Arduino también es compatible con las características avanzadas de la ESP32.

3.2 Creación de la base de datos

Las bases de datos son fundamentales en las redes neuronales, ya que proporcionan los ejemplos necesarios para entrenar y validar el modelo, permitiéndole aprender patrones y tomar decisiones precisas. Recopilar datos propios es crucial porque garantiza que el conjunto sea representativo del caso específico de uso, mejorando el rendimiento del sistema y adaptándolo a las necesidades particulares del proyecto.

En nuestro caso, esta recopilación propia logró crear una base de datos equilibrada y con comandos muy específicos, que buscando en bases de datos públicas no fue posible encontrar.

3.2.1 Recolección de datos

Para la recopilación de datos se pidió la ayuda de nuestro entorno familiar, laboral y educativo. En el Apéndice E se ahonda un poco más en la información de la población muestreada. A ellos se les pidió grabar mínimo 5 y hasta un máximo de 15 audios por persona, donde, en cada audio solo se mencionase el comando exacto y no superara un límite de 5 segundos.

Los comandos que se les pidió grabar fueron:

- Enciende la luz
- Apaga la luz
- Abre la puerta
- Cierra la puerta
- Dime la hora
- Comandos desconocidos, ruido de fondo o silencio

Además, de ser posible, se les solicitó la repetición de estos comandos en cada uno de los siguientes 3 entornos:

- Con ruido blanco
- Con ruido de fondo
- Con personas hablando o música de fondo

3.2.2 Conversión y etiquetado de datos

Al obtener los audios de diferentes fuentes y formatos, por ejemplo, a través de *WhatsApp* que se descargan en formato *.ogg*, se precisó de una etapa de conversión de estos mismos a formato *.wav*, ya que este es más comúnmente aceptado en las tareas de procesamiento de audio, adicionalmente, el microcontrolador seleccionado también trabaja con sonido en formato *.wav*.

Esta conversión se realizó con ayuda de la plataforma web 123apps en su apartado de convertidor de audio (Apéndice A). Dicha herramienta gratuita permite convertir hasta 75 audios al tiempo. En el Apéndice D se observa el proceso de cortar, convertir y etiquetar un audio.

La Tabla 3 presenta un resumen de las características técnicas de los audios después de la conversión y su promedio de duración en segundos.

Tabla 3

Características técnicas de los audios después del proceso de conversión

Comando	1 canal	2 canales	44,1 (kHz)	48 (kHz)	Promedio (s)
Apaga	15	243	176	82	2,4238
Enciende	15	243	176	82	2,4822
Dime	15	243	176	82	2,3096
Cierra	15	243	176	82	2,4813
Abre	15	243	176	82	2,4086
Ruido	188	70	188	70	2,3994

En cuanto al etiquetado de estos audios, se optó por usar solo la primera palabra de cada comando, seguido de la letra A para los audios sin ruido, la letra B para los audios con ruido ambiental de fondo y la letra C para los audios con música o personas hablando. Luego, se usó la palabra persona seguida de un número para identificarla. Todo esto separado por guiones bajos.

Así, el audio enviado por la primera persona cargada en el dataset, diciendo el comando *enciende la luz* y sin ruido de fondo, corresponde a la etiqueta: **enciende_A_persona_1**.

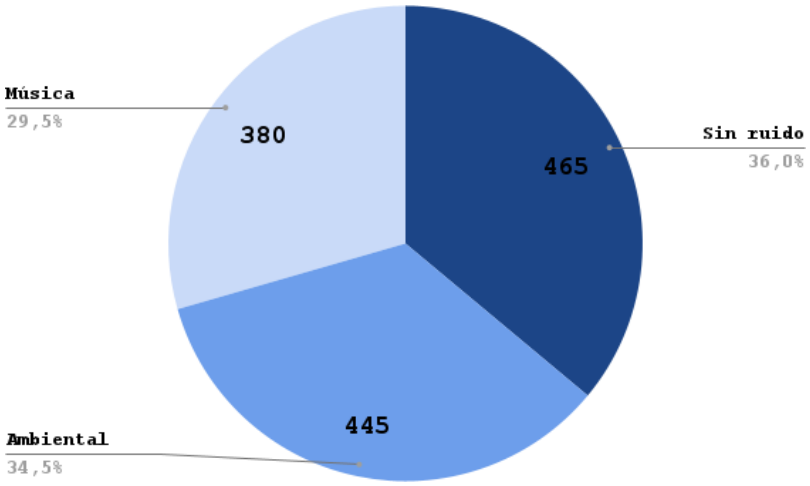
3.2.3 Información sobre la base de datos

Finalizada la recolección de datos, se obtuvo un total de 1,290 audios, de 106 personas diferentes, con 258 muestras asignadas a cada uno de los comandos evaluados, además de 258 audios de ruido. Esto permitió configurar una base de datos completamente balanceada en cuanto a la distribución de las etiquetas (*labels*), garantizando equidad en el entrenamiento y evaluación del sistema. En el Apéndice G se encuentra el dataset creado y de libre acceso.

En la Figura 5 se muestra la proporción de los audios de los 5 comandos obtenidos según el entorno de ruido presente durante su grabación. Los audios categorizados como "Sin ruido" corresponden a grabaciones realizadas con la indicación de hacerlo sin ruido de fondo, mientras que los clasificados como "Ambiental" incluyen grabaciones con ruido de fondo. Por último, la categoría "Música" agrupa los audios grabados en entornos con personas hablando o música de fondo.

Figura 5

Proporción de audios según el entorno en el que fueron grabados



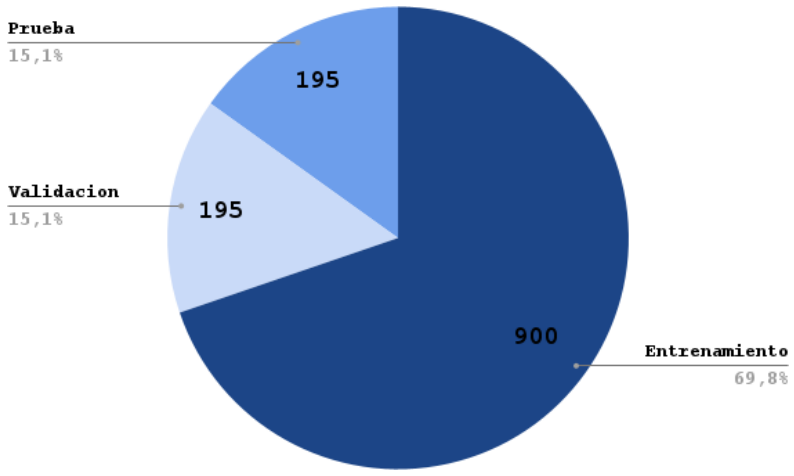
3.2.4 Distribución del dataset

Antes de proceder al diseño de la red neuronal, se dividió el *dataset* creado en 3 subconjuntos de entrenamiento, validación y prueba.

La Figura 6 presenta la partición utilizada para este *dataset*. Siendo muy cercana a una proporción 70-15-15, una de las particiones más comunes en bases de datos empleadas para el desarrollo de modelos de aprendizaje automático.

Figura 6

Partición del dataset



3.3 Diseño de la red neuronal

El curso "Introducción a la clasificación de audio con TensorFlow" fue clave en el diseño y desarrollo de la red neuronal, especialmente en la etapa de conversión de audios a representaciones de imágenes como espectrogramas. Este recurso proporcionó una guía práctica y comprensible para aplicar técnicas clave de preprocesamiento de audio, facilitando así la preparación de los datos y el desarrollo eficiente del modelo de la red neuronal.

3.3.1 Preprocesamiento de audio e imágenes

El proceso de preparación de los datos de audio para la red neuronal comienza con la carga y estandarización de los archivos de sonido, los cuales son remuestreados a 16 kHz y convertidos a un formato compatible con TensorFlow. A partir del nombre de cada archivo se extrae información clave como el comando de voz, el tipo de ruido de fondo y el identificador del hablante, lo que permite etiquetar correctamente los datos. Posteriormente, las señales de audio son transformadas en espectrogramas mediante la Transformada de Fourier de Tiempo Corto

(STFT), asegurando que todas las representaciones tengan la misma longitud y estructura. Estas representaciones espectrales, que reflejan cómo varía la energía del audio en el tiempo y la frecuencia, se convierten en imágenes en escala de grises que serán utilizadas como entradas para entrenar y evaluar el modelo. Finalmente, dichas imágenes son organizadas en conjuntos de entrenamiento y prueba, permitiendo el desarrollo eficiente de la red neuronal.

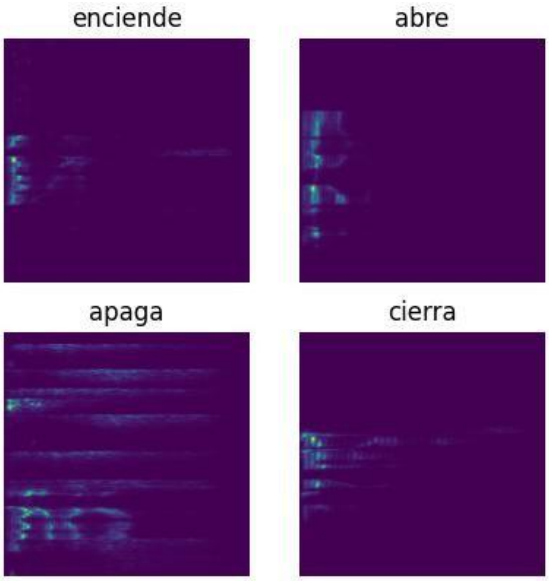
En la Figura 7 se muestra a modo de diagrama de flujo el proceso para el preprocesamiento de audio e imágenes. Mientras que en el Apéndice M se profundiza en el proceso que lleva a cabo cada función. En la Figura 8 se muestran algunos espectrogramas creados y que son usados en el conjunto de entrenamiento de la red neuronal.

Figura 7

Funciones para el preprocesamiento de audio e imágenes

Figura 8

Imágenes de algunos espectrogramas del conjunto de entrenamiento



3.3.2 Métodos usados en el entrenamiento de la red neuronal

En el Apéndice F se introducen brevemente algunos elementos claves para el diseño y entrenamiento de la red neuronal. Cada uno de estos componentes jugó un papel esencial en el rendimiento, la estabilidad y la eficiencia del modelo. A modo de resumen, el inicializador usado fue HeNormal, se usaron regularizadores L2 y Dropout, la función de activación predominante fue ReLu, pero se usó Softmax en la última capa, pues es la indicada para procesos de clasificación. El optimizador elegido fue Adam y se usó un callback compuesto por EarlyStopping y ModelCheckpoint.

3.3.3 Proceso de modelado de la red neuronal

En este proceso se realizaron múltiples pruebas y ajustes, tanto en el modelo de la red neuronal como en diferentes parámetros o variables que influían al momento de obtener los mejores resultados posibles, ya sea para las métricas obtenidas, o en el proceso de implementación en el microcontrolador. Estos cambios fueron clave para garantizar un equilibrio entre rendimiento y compatibilidad con las limitaciones del hardware. Cabe aclarar que se partió haciendo uso del

modelo empleado en el curso "Introducción a la clasificación de audio con TensorFlow", y con base en este se fueron realizando los cambios detallados a continuación.

Después de múltiples pruebas se decidió ajustar el *image_size* de las imágenes en 32x32 píxeles, pues al momento de la implementación nos producía el menor consumo de memoria.

La tasa de aprendizaje (*learning rate*), que controla el tamaño de los pasos que da el optimizador en la dirección del gradiente para minimizar la función de perdida, también fue modificada en diferentes ocasiones. Inicialmente, se trabajó con valores fijos, pero posteriormente se exploraron estrategias para ajustarla dinámicamente durante el entrenamiento. Entre estas, se intentó con la técnica conocida como *Decay* (decaimiento) que disminuye el learning rate progresivamente a lo largo del entrenamiento. También se probó la estrategia conocida como *Reduce on Plateau*, que reduce el learning rate si no hay mejora en la métrica seleccionada, ya sea la precisión o la función de pérdida, durante un número definido de épocas. Sin embargo, tras varios experimentos, se observó que estas técnicas no solo no mejoraban significativamente las métricas, sino que, en algunos casos, las perjudicaban. Por ello, se decidió fijar el ***learning rate*** en un valor constante **de 0.0005**, pues fue con el valor que encontramos mejores resultados a lo largo de las diferentes pruebas y cambios.

En algunos intentos se compiló el modelo con los optimizadores SGD (*Stochastic Gradient Descent*) y RMSprop (*Root Mean Square Propagation*), pero se evidenciaba de inmediato una desmejora respecto al optimizador Adam, el cuál fue el seleccionado.

Evidentemente, el mayor foco de cambios y pruebas fue el modelo de la red neuronal, en este se probó el inicializador HeNormal, el cual producía una leve mejora al usarlo, por lo que se decidió incluirlo en el modelo final. También se ajustó el número de capas convolucionales, la cantidad de filtros que tenían las mismas, así como el valor del regularizador L2, cuyos detalles se

muestran en la Tabla 4. Asimismo, la capa de BatchNormalization se añadió en varias configuraciones, pero su uso ocasionaba un aumento en la función de pérdida, por lo que se decidió no implementarla.

En cuanto a las capas densas, inicialmente se eliminó la primera de ellas que contaba con 128 filtros, pues representaba un coste computacional alto, pero al hacerlo el *accuracy* del entrenamiento no superaba valores de 0.7, motivo por el cual se conservó en el modelo final. Finalmente, también se experimentó con diferentes porcentajes de Dropout, encontrando los mejores resultados con el 20% en cada capa añadida.

3.3.3.1 Modelo final de la red neuronal y parámetros. Tras el proceso de modelado descrito, la Tabla 4 presenta las capas utilizadas en el modelo junto con información relevante. Cabe recordar que se usó el inicializador HeNormal, y la función de activación ReLu en todas las capas convolucionales y densas, excepto en la capa final, que hace uso de Softmax.

Tabla 4

Capas utilizadas en el modelo de la red neuronal

Capa (tipo)	Información	Forma de salida	Parámetros
Conv2D_1	16 neuronas, L2=0.01	(32, 32, 16)	160
Conv2D_2	16 neuronas, L2=0.01	(32, 32, 16)	2,320
MaxPooling2D_1		(16, 16, 16)	0
Conv2D_3	32 neuronas, L2=0.0001	(16, 16, 32)	4,640
Conv2D_4	32 neuronas, L2=0.001	(16, 16, 32)	9,248
MaxPooling2D_2		(8, 8, 32)	0
Conv2D_5	64 neuronas, L2=0.0001	(8, 8, 64)	18,496

Conv2D_6	64 neuronas, L2=0.001	(8, 8, 64)	36,928
MaxPooling2D_3		(4, 4, 64)	0
Flatten		(1024)	0
Dense_1	128 neuronas	(128)	131,200
Dropout_1	20%	(128)	0
Dense_2	64 neuronas	(64)	8,256
Dropout_2	20%	(64)	0
Dense_3	32 neuronas	(32)	2,080
Dropout_3	20%	(32)	0
Dense_4	5 neuronas	(6)	198

En la Tabla 5 se muestra el resumen del número de parámetros y su tamaño.

Tabla 5

Resumen de parámetros

Tipo	**Total de parámetros**	**Tamaño**
Parámetros totales	640,580	2.44 MB
Parámetros entrenables	213,526	834.09 KB
Parámetros no entrenables	0	0 B
Parámetros del optimizador	427,054	1,63 MB

3.3.4 Preparación del código para su ejecución en el microcontrolador

Tras seleccionar el mejor modelo con base en las métricas y la matriz de confusión, se crearon dos funciones clave para su implementación en la ESP32. La primera,

representative_dataset, permite realizar la cuantización del modelo, reduciendo su tamaño y mejorando su velocidad al convertir los valores de punto flotante a enteros, lo cual es fundamental dada la limitada capacidad del microcontrolador. La segunda, convert_to_tflite, transforma el modelo .Keras a TensorFlow Lite y aplica optimizaciones adicionales; el resultado se convierte luego a formato .h con la herramienta xxd, declarando las variables como constantes para su uso eficiente en el entorno embebido.

3.4 Implementación en el microcontrolador

3.4.1 Configuración inicial de la ESP32 Audio Kit

Inicialmente, se establecieron los parámetros de grabación, como la tasa de muestreo, los canales y la profundidad de bits, y se configuró el módulo de audio para capturar sonido desde el micrófono. También, se inicializaron las bibliotecas de TensorFlow Lite, se cargó el modelo de red neuronal y se asignó memoria para los tensores. Adicionalmente, se asignaron funciones a los botones de la placa: el botón 1 para escuchar el audio grabado anteriormente, el botón 3 para grabar y clasificar un nuevo audio, el botón 4 para procesar el archivo grabado mediante el modelo y el botón 6 para escuchar el manual de uso. En la Tabla 6 se muestra la información de los botones existentes en la placa.

Tabla 6

Botones de la ESP32 Audio Kit V2.2

Botón	GPIO	Función	Modo de activación
Boot	GPIO 0	Modo de carga (flash) o uso como entrada	Activo en bajo (LOW)

Reset	N/A (Reset)	Reinicia la ESP32	Pulsador físico
Botón 1	GPIO 36 (ADC1_CH0)	Entrada analógica/digital	Activo en bajo (LOW)
Botón 2	GPIO 39 (ADC1_CH3)	Entrada analógica/digital	Activo en bajo (LOW)
Botón 3	GPIO 34 (ADC1_CH6)	Entrada analógica/digital	Activo en bajo (LOW)
Botón 4	GPIO 13	Entrada digital	Activo en bajo (LOW)
Botón 5	GPIO 19	Entrada digital	Activo en bajo (LOW)
Botón 6	GPIO 23	Entrada digital	Activo en bajo (LOW)

Por configuraciones internas la ESP32 cuenta con un módulo de 6 switches, los cuales controlan el uso de la SD y, eventualmente, el switch 1 impide usar el botón 2, debido a que este también activa la SD. Por su parte, el switch 2 y 3 ayudan a realizar la carga de información a la SD.

3.4.2 Librerías usadas

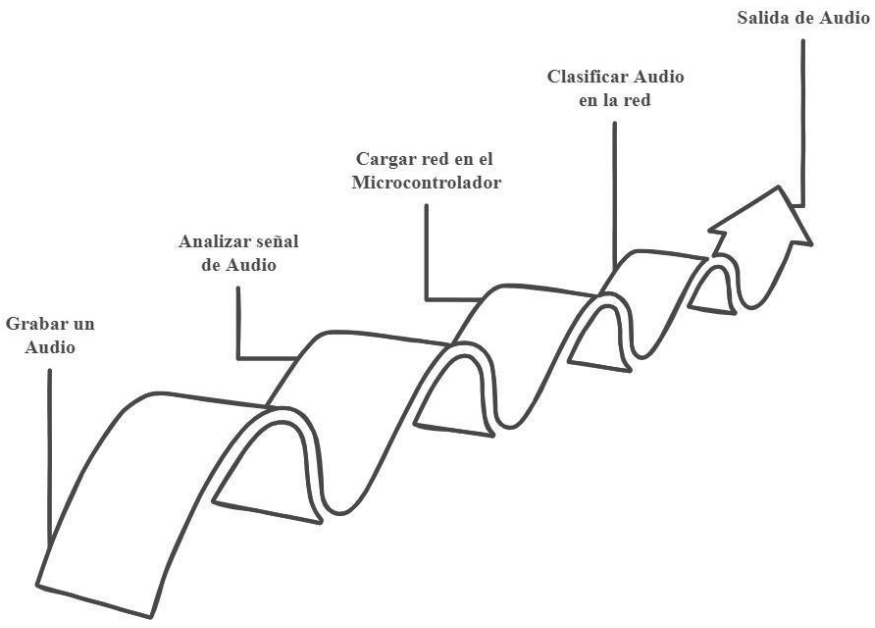
Las librerías SD.h y SPI.h permiten gestionar la comunicación con la tarjeta SD, esencial para almacenar las grabaciones. La librería AudioTools ofrece herramientas avanzadas para la manipulación de audio y la configuración del hardware. Para la ejecución del modelo de TensorFlow Lite, se emplearon TensorFlowLite_ESP32 y componentes de TFLite Micro, como all_ops_resolver, micro_interpreter y micro_error_reporter, que gestionan la interpretación del modelo, las operaciones disponibles y los reportes de error. Mientras que mejor_modelo.h contiene el modelo optimizado y compilado para su ejecución en el microcontrolador.

3.4.3 Proceso de implementación en el microcontrolador

El proceso implementado en el microcontrolador inicia con la grabación de un nuevo audio, activada al presionar el botón 3 y gestionada mediante la verificación e inicialización de la tarjeta SD para almacenar los datos capturados durante 2.5 segundos. Posteriormente, la señal se preprocesa, asegurando una longitud mínima de 8000 muestras, generando un espectrograma a través de la Transformada Directa de Fourier (DFT) y almacenándolo en la memoria PSRAM, lo que permite su análisis posterior. A continuación, el modelo previamente entrenado en Colab se integra en el microcontrolador utilizando TensorFlow Lite, donde se asigna un búfer de 40 KB para ejecutar la inferencia sobre la señal cuantizada, determinando la clasificación más probable. Finalmente, el sistema traduce esta predicción en una salida de audio reproducida mediante la selección de un archivo pregrabado correspondiente al comando identificado.

Figura 9

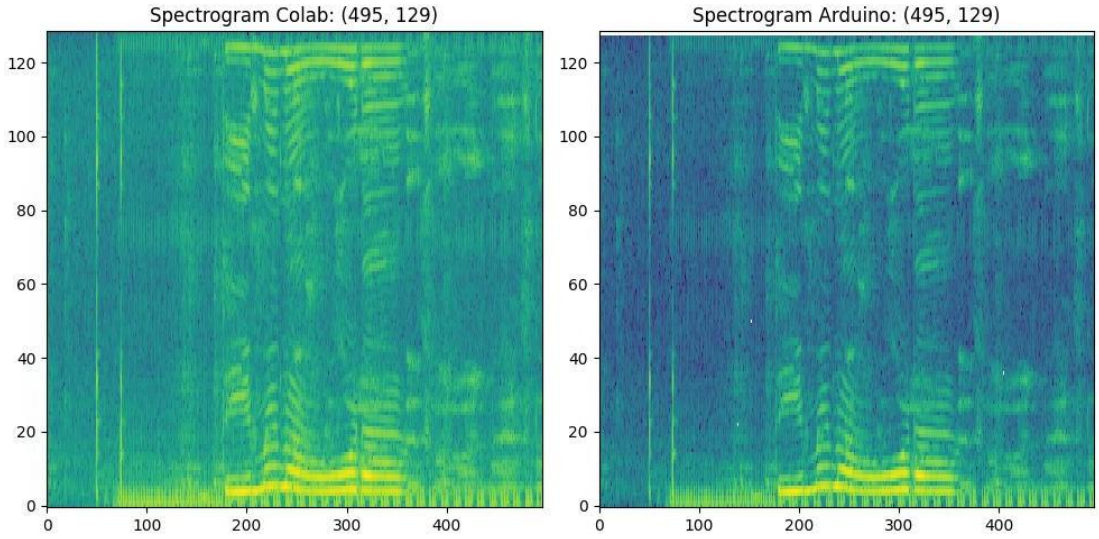
Procesos llevados a cabo en la implementación



En la Figura 9 se presenta mediante diagrama de bloques los procesos llevados a cabo en la implementación, los cuales se profundizan en el Apéndice M. En la Figura 10 se visualiza la comparación entre los espectrogramas creados en Colab y Arduino de uno de los audios del dataset.

Figura 10

Comparación de espectrogramas



Para finalizar, en el Apéndice K se encuentra el enlace al repositorio de GitHub con el código usado para la implementación en Arduino, así como el archivo con el mejor modelo usado.

3.5 Evaluación y resultados obtenidos.

3.5.1 Resultados de la red neuronal

Para el entrenamiento del modelo de la red neuronal se usaron 100 épocas, ya que, como se adelantó anteriormente, se empleó EarlyStopping configurado para detenerse después de 3 épocas en las que no notara mejoría en la función de pérdida de la validación. Esto permitió evitar que el modelo cayera en *overfitting* y se ahorrara tiempo si no era necesario seguir ajustando el

modelo. También hay que recordar que, gracias al ModelCheckpoint, se guardó el mejor modelo encontrado en función del *accuracy* evaluado en la validación del entrenamiento.

3.5.1.1 Resultados en el entrenamiento. Durante el entrenamiento, se monitorearon dos métricas clave, el *accuracy* (precisión) que mide la proporción de predicciones correctas sobre el total de predicciones realizadas y el *loss* (función de pérdida) que mide la diferencia entre las predicciones del modelo y las etiquetas reales. La función de pérdida utilizada fue SparseCategoricalCrossentropy. Por su parte, en la validación nos interesa monitorear las mismas métricas, pero ahora con el 15% de datos del dataset reservados anteriormente.

En la Figura 11 se muestra la evolución de la precisión a lo largo de las épocas en el conjunto de entrenamiento y validación, mientras que en la Figura 12 se muestra la pérdida vs épocas, también en ambos conjuntos.

Figura 11

Precisión de entrenamiento y validación vs épocas

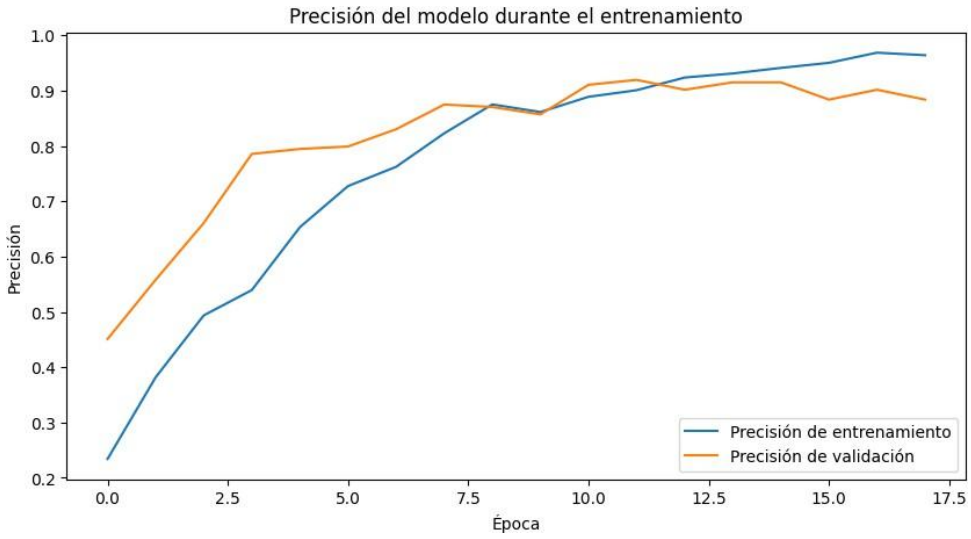
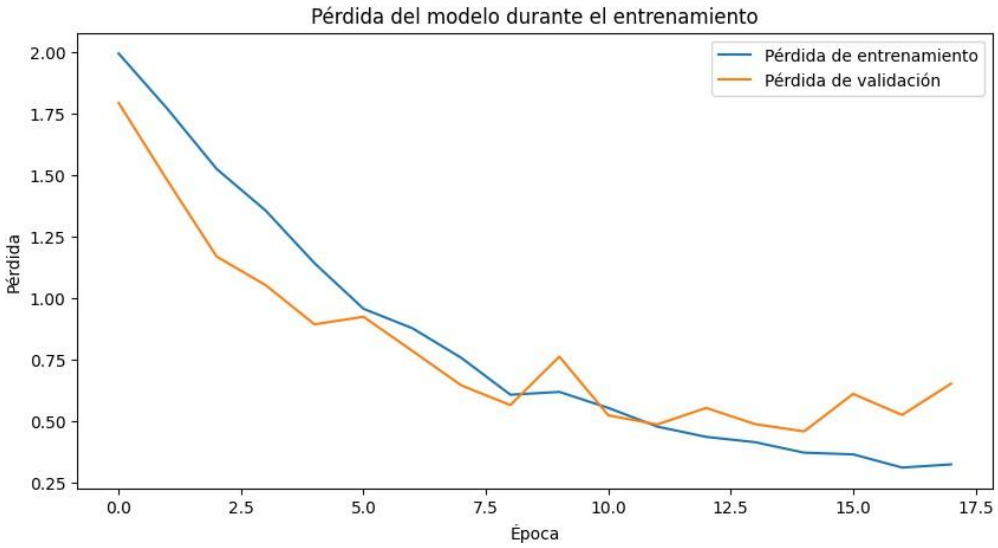


Figura 12

Pérdida de entrenamiento y validación vs épocas



3.5.1.2 Resultados en la prueba. Una vez guardado el mejor modelo se procedió a evaluarlo en los datos de *test* previamente guardados para este propósito. Al ser datos nuevos que el modelo no ha visto jamás, las métricas obtenidas en este apartado constituyen el verdadero resultado de la red neuronal. Cabe aclarar que esto se realiza con el modelo antes de cuantizar.

Nuevamente, las principales métricas de interés y sus respectivos resultados obtenidos se muestran en la Tabla 7.

Tabla 7

Métricas obtenidas en el conjunto de prueba

Precisión	Pérdida
0.8931	0.5636

En la Tabla 8 se presentan los resultados asociados a las métricas de interés para la prueba de la red neuronal.

Tabla 8

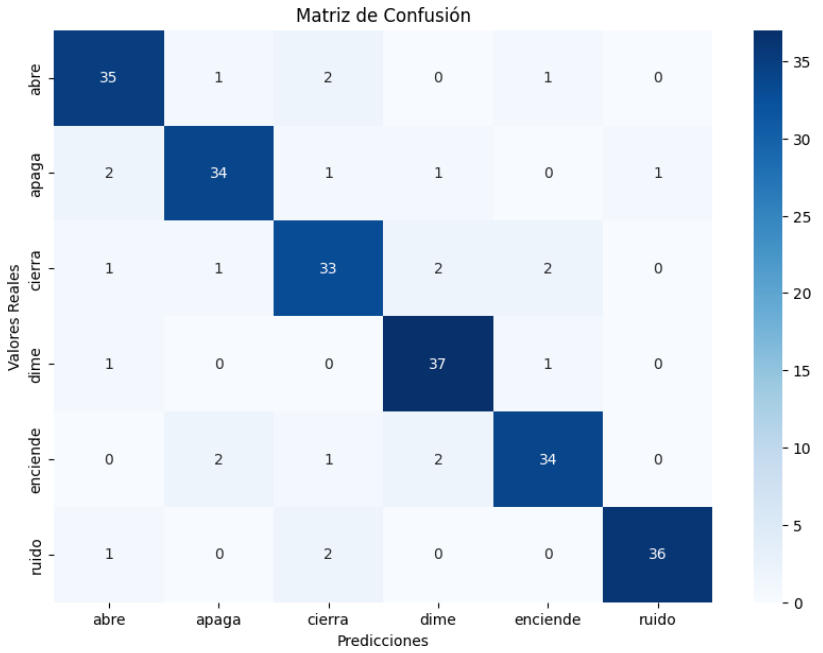
Métricas adicionales

Accuracy	Precision	Recall	F1 Score
0.8931	0.8940	0.8931	0.8932

Finalmente, la matriz de confusión es una herramienta que permite visualizar el desempeño del algoritmo. Donde cada columna de la matriz representa el número de predicciones de cada clase y cada fila representa a las instancias en la clase real. En la Figura 13 se muestra la matriz de confusión obtenida para la red neuronal realizada.

Figura 13

Matriz de confusión



3.5.1 Resultados en la implementación

Una vez cuantizado el mejor modelo se realiza el proceso mostrado en la Figura 9.

Para obtener la precisión del sistema se hicieron un total de 100 pruebas en el microcontrolador, 20 por cada comando de voz. El resumen de los resultados de estas pruebas y la

precisión lograda como el tiempo promedio de ejecución, desde que se oprime el botón para grabar hasta que se escucha el resultado de la predicción se muestran en la Tabla 9.

Finalmente, en el vídeo de el Apéndice J se visualiza el funcionamiento de la implementación.

Tabla 9

Resultados en la implementación

Comando	**Precisión**	**Tiempo promedio de ejecución [s]**
Abre	70%	14.52
Apaga	75%	13.97
Cierra	90%	14.32
Dime	85%	14.06
Enciende	85%	14.91

4. Conclusiones

Inicialmente los audios fueron convertidos utilizando la calidad máxima disponible, que es de 96kHz. Sin embargo, tras realizar pruebas de procesamiento se detectaron errores que afectaban el reconocimiento de los audios por parte del modelo de la red neuronal. Por este motivo, se optó por convertirlos con tasas de muestreo de 44.1kHz y 48kHz para garantizar mayor compatibilidad.

Mientras que el uso de regularizadores (L2 y Dropout) mejoraron la generalización del modelo, evitando el sobreajuste, y la combinación del inicializador HeNormal y la función de activación ReLU optimizó la precisión del sistema, el uso constante de capas de BatchNormalization generaban un aumento significativo en la pérdida del sistema.

Redimensionar las imágenes de 256x256 a 32x32 píxeles, permitió disminuir notablemente el tamaño del mejor modelo y acelerar el proceso de entrenamiento, a costa de sacrificar un porcentaje de precisión en el modelo entrenado.

La red neuronal diseñada logró reconocer los comandos con una precisión promedio cercana al 90% durante las pruebas. Este resultado evidencia la efectividad del enfoque basado en redes neuronales convolucionales.

La cuantización del modelo para adaptarlo a las limitaciones del microcontrolador y los procesos en busca de disminuir las dimensiones y el tamaño de la red generaron un decrecimiento notable en la precisión del modelo al momento de implementarlo en el microcontrolador.

El proyecto presenta una solución accesible para el reconocimiento de voz en español, fomentando la inclusión tecnológica en comunidades hispanohablantes. Su implementación en un microcontrolador económico promueve su uso en entornos educativos y profesionales, impactando positivamente el desarrollo local y regional.

5. Recomendaciones

Se recomienda incrementar el tamaño y la diversidad del dataset, incluyendo más comandos, entornos de grabación y acentos, pues el idioma español es muy variado. Esto mejoraría la robustez del sistema frente a variaciones lingüísticas y contextuales, permitiendo aplicaciones más generalizadas en entornos hispanohablantes diversos.

Explorar los límites que puede alcanzar el sistema de desarrollo en cuanto a su almacenamiento, para así lograr una implementación más general con un mayor número de comandos.

La meta a largo plazo es llevar esta detección de comandos de voz a entornos y situaciones del diario vivir, por lo que se debería considerar la integración del sistema con plataformas de Internet de las Cosas (IoT). Por ejemplo, conectar el sistema a dispositivos domóticos podría crear un ecosistema completo que aproveche el reconocimiento de voz para automatizar tareas en el hogar o entornos laborales.

Es de provecho cursar asignaturas relacionadas al entendimiento del Deep Learning, el procesamiento digital de imágenes y el diseño e implementación con microcontroladores, ya sea en la Universidad Industrial de Santander u otros entornos educativos, ya que en nuestro caso aportaron conocimientos fundamentales y específicos que enriquecieron el desarrollo del proyecto.

Referencias Bibliográficas

Barrios, J. (2019). La matriz de confusión y sus métricas. Health big data. https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/

Deng, L., & Yu, D. (2013). *Deep learning: Methods and applications*. Foundations and Trends in Signal Processing, 7(3-4), 197–387. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/DeepLearning-NowPublishing-Vol7-SIG-039.pdf

Durán, J. (2017). Redes Neuronales Convolucionales en R Reconocimiento de caracteres escritos a mano [Tesis de pregrado, Universidad de Sevilla].

Everest Semiconductor. (s.f.). *ES8388 low power stereo audio codec with headphone amplifier [Datasheet]*. http://www.everest-semi.com/pdf/ES8388%20DS.pdf

Google. (s.f.). Conceptos básicos. https://research.google.com/colaboratory/intl/es/faq.html

K Keras. (s.f.). Keras: The Python deep learning API. https://keras.io/

López Neira, G. (2023). Diseño e implementación de un sistema domótico controlado mediante comandos de voz [Trabajo de Fin de Grado, Universitat Politècnica de Catalunya].

Macías Reyes, A. J. (2024). Implementación de un prototipo de asistente de voz con un SoC y ChatGPT [Trabajo de integración curricular, Escuela Politécnica Nacional].

Mercado Libre. (s.f.). Audio Kit V2.2 ESP32 A1S WiFi Bluetooth Audio [Fotografía]. Mercado Libre. https://articulo.mercadolibre.com.co/MCO-908821889-audio-kit-v22-esp32-a1s-wifi-bluetooth-audio-_JM

Microsoft. (s.f.). Introducción a la clasificación de audio con TensorFlow. Microsoft 365. https://learn.microsoft.com/es-es/training/modules/intro-audio-classification-tensorflow/

Saha, S. (2018, December 15). *A comprehensive guide to convolutional neural networks—the ELI5 way. Towards Data Science*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Schatzmann, P. (s.f.). ESP32 Audio Kit – Proyectos y Tutoriales. Phil Schatzmann's Blog. https://www.pschatzmann.ch/home/tag/esp32audiokit/

Shenzhen Ai-Thinker Technology Co., Ltd. (2021). *ESP32-A1S Specification* (Version 2.3). https://docs.ai-thinker.com/_media/esp32-a1s_v2.3_specification.pdf

Tecnología Anxinke. (2020). Kit de audio ESP32. https://docs.ai-thinker.com/en/esp32-audio-kit

TensorFlow. (s.f.). Aprende AA. https://www.tensorflow.org/?hl=es-419

Xie, H., & Shen, D. (2023). Asistente de voz para la monitorización del estado de una planta mediante un sistema basado en IoT. [Proyecto Fin de Grado, Grado en Ingeniería de Computadores, Universidad Politécnica de Madrid].

Apéndices

Apéndice A. Convertidor de audio. https://online-audio-converter.com/sp/

Apéndice B. Generador de audio desde texto https://www.narakeet.com/app/text-to-audio/?projectId=172a7b86-1f4b-46f1-bcbb-2af10131c62e

Apéndice C. Perfil de Phil Schatzmann, del cual se emplearon múltiples repositorios. https://github.com/pschatzmann/

Apéndice D. Vídeo del proceso de cortar, convertir y etiquetar un audio. https://www.youtube.com/watch?v=xDaxl7sOC6U

Apéndice E. Información sobre la población muestreada. https://github.com/FritzAriza/TDG_Ariza_Roman/blob/main/InfoPoblacional_BD.pdf

Apéndice F. Métodos usados para entrenamiento de la red neuronal. https://github.com/FritzAriza/TDG_Ariza_Roman/blob/main/MetodosUsados_RN.pdf

Apéndice G. Dataset. https://github.com/FritzAriza/TDG_Ariza_Roman/tree/main/dataset

Apéndice H. CNN con el mejor modelo creado en Colab. https://github.com/FritzAriza/TDG_Ariza_Roman/blob/main/Red_CNN_TG.ipynb

Apéndice I. Comparación entre espectrogramas. https://github.com/FritzAriza/TDG_Ariza_Roman/blob/main/comparacion_espectrogramas.ipynb

Apéndice J. Vídeo de la implementación. https://www.youtube.com/watch?v=HVI-O2AaETc

Apéndice K. Código usado para la implementación en Arduino. https://github.com/FritzAriza/TDG_Ariza_Roman/blob/main/model/modelo_TG.ino

Apéndice L. Código en Colab para convertir el mejor modelo. https://github.com/FritzAriza/TDG_Ariza_Roman/blob/main/convertidor_keras_h.ipynb

Apéndice M. Funciones de la red neuronal y proceso de la implementación.

https://github.com/FritzAriza/TDG_Ariza_Roman/blob/main/Funciones_y_procesos.pdf