UACM

Universidad Autónoma de la Ciudad de México

NADA HUMANO ME ES AJENO

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS INDUSTRIALES

Motor de DC Controlado por Voz Utilizando Inteligencia Artificial

TESIS QUE PARA OBTENER EL TÍTULO DE

LICENCIADO EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS INDUSTRIALES

PRESENTA

Pedro Emmanuel Andrade Velázquez

Director de la Tesis

M. en I. Christian Agustín Vázquez Villanueva

Ciudad de México, diciembre de 2023

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

INTEGRACIÓN DEL JURADO:

Presidente:

M. en I. Víctor Manuel Macías Medrano, Casa Libertad

Secretario:

Dr. Marcos Ángel González Olvera, San Lorenzo Tezonco

Vocal:

M. en I. David Estrada Espinosa, Casa Libertad

1er Suplente:

2do Suplente:

Plantel de adscripción: Casa Libertad

DIRECTOR DE TESIS:

M. en I. CHRISTIAN AGUSTÍN VÁZQUEZ VILLANUEVA

UACM, Plantel San Lorenzo Tezonco.

Agradecimientos.

Quiero expresar mi más sincero agradecimiento por su invaluable orientación, apoyo, dedicación, sabiduría, paciencia y sobre todo su tiempo a mi director de tesis M. en I. Christian Agustín Vázquez Villanueva, Gracias por ser una fuente constante de inspiración y por contribuir significativamente a mi desarrollo académico.

Agradezco profundamente por su tiempo, dedicación y contribuciones durante la revisión de mi tesis a mis 3 lectores, Dr. Marcos Ángel González Olvera, M. en I. Víctor Manuel Macías Medrano y M. en I. David Estrada Espinosa. La orientación de cada uno de ustedes ha sido esencial para mi éxito académico. Aprecio sinceramente su valiosa colaboración en este importante proyecto.

Le agradezco a mis padres Juana Angelica Velázquez López y Pedro Andrade Soria, así como hermanos y familia en general, por su apoyo incondicional en este viaje académico. Este logro es nuestro, y celebro con gratitud el papel que cada uno ha desempeñado en mi éxito.

A mis compañeros, quiero expresar mi sincero agradecimiento por compartir conmigo este viaje académico. Sus risas, amistad y colaboración han hecho que este camino sea inolvidable. Gracias por ser parte de este capitulo en mi vida. En especial, quiero dedicar unas palabras de agradecimiento a Emma, cuya presencia ilumino este trayecto académico, haciendo más memorable esta etapa compartida.

Mi querida UACM, quiero expresar mi profundo agradecimiento por brindarme una experiencia educativa invaluable. Los conocimientos adquiridos y las conexiones realizadas han dejado una marca duradera en mi vida. Gracias por ser parte fundamental de mi crecimiento académico y personal.

Índice.

Resumen.

En esta tesis, se aplicó el reconocimiento de voz basado en Inteligencia Artificial (IA) a un circuito electrónico que controle una señal PWM (Modulación por Ancho de Pulsos) de un motor DC. El objetivo principal fue la de desarrollar un modelo basado en la Inteligencia Artificial capaz de reconocer cuatro comandos de voz: encender, apagar, aumentar velocidad y disminuir velocidad.

Para lograr esto, se utilizó el software Anaconda y su entorno Jupyter Notebook, junto con la biblioteca TensorFlow para la programación del reconocimiento de voz. Para la implementación del circuito se utilizó el módulo ESP32 DEVKIT V1 30 Pines Wifi + Bluetooth y el software Visual Studio Code con su extensión PlatformIO.

Una vez entrenado el modelo con IA, se procedió a diseñar el circuito que se usó en el programa que realizo las acciones correspondientes en función del comando recibido. Se espera que este sistema de control mediante la inteligencia artificial y reconocimiento de voz permita usar una interfaz más fácil para los operadores de un motor de alfarero, por ejemplo.

Capítulo 1.

Introducción.

La Inteligencia Artificial (IA) es esencial para muchas tecnologías que utilizamos a diario, desde el reconocimiento facial en los teléfonos móviles hasta los asistentes virtuales de voz como Alexa, Siri y Cortana. También se utiliza ampliamente en la publicidad para crear recomendaciones personalizadas para los clientes basados en sus búsquedas. Además, se utiliza en el comercio para optimizar procesos, planificar inventarios, manejar procesos logísticos y predecir comportamientos del mercado, entre otros aspectos [1].

La razón principal por la cual se está utilizando la IA en esta tesis es para resolver un problema mediante el uso de un circuito que contenga un motor DC, ya que gracias a una serie de piezas (Hardware/Circuito) se permite “emular” la inteligencia humana de tal manera que un circuito tenga la capacidad de ”pensar”, ” razonar” e “interactuar” con el entorno para llevar a cabo las tareas para las cuales fue diseñado y construido.

1.1 Motivación.

La motivación detrás del estudio e investigación en el campo de la Inteligencia Artificial aplicada a un circuito es contribuir al avance de la automatización y eficiencia de sistemas electrónicos, ya que esta tecnología está transformando significativamente a la sociedad y la industria. El uso de la IA en la automatización de procesos y la mejora de la precisión en el análisis de datos es fundamental, y su implementación en campos específicos como el control de maquinaria y procesos industriales es cada vez más común. En este caso, se busca aplicar el concepto de IA mediante el uso del reconocimiento de voz para controlar un motor de torno de alfarero. La implementación del reconocimiento de voz en un motor de torno de alfarero tiene el potencial de eliminar errores humanos al reducir las operaciones manuales, además de automatizar tareas complejas y mejorar la toma de decisiones en el proceso de producción. Este proyecto no solo tiene un enfoque práctico, sino que también tiene un gran potencial para contribuir al desarrollo y mejora de la industria alfarera.

En un mundo cada vez más digitalizado, el conocimiento y habilidades en IA se están convirtiendo en esenciales para competir en el mercado laboral y para entender y aprovechar las oportunidades que ofrece la tecnología. Por lo tanto, investigar y estudiar en el campo de la IA es una inversión valiosa en el futuro tanto personal como profesional.

1.2 Planteamiento del Problema.

En la industria alfarera, el uso de tornos para la elaboración de cerámica presenta dificultades en cuanto al costo elevado, peso y seguridad en el manejo. Actualmente, existen tres tipos de tornos en el mercado, el torno de rueda, el torno de pedal y el torno eléctrico. Sin embargo, cada uno de estos presenta sus propios desafíos. Por ejemplo, el torno de rueda y pedal son costosos y pesados, lo que dificulta su transporte de un lugar a otro. El uso de estos tornos requiere una gran cantidad de esfuerzo físico, lo que puede generar problemas en las rodillas de los alfareros debido al constante uso del volante o pedal. Por otro lado, el torno eléctrico es más fácil de usar, pero su costo es elevado, lo que lo hace inaccesible para muchos alfareros.

1.3 Justificación.

Esta investigación aborda desafíos significativos en la industria alfarera, específicamente en lo que respecta a la optimización de la eficiencia y seguridad en el proceso de elaboración de cerámica. Actualmente, la utilización de tornos tradicionales conlleva costos elevados, implica un manejo pesado y presenta riesgos para la seguridad de los alfareros. Además, el uso prolongado de tornos convencionales, como los de pedal, puede resultar en un desgaste físico significativo en las rodillas de los artesanos.

Para superar estas limitaciones, se propone la implementación de la Inteligencia Artificial (IA) y el reconocimiento de voz como medio para controlar el motor del torno alfarero. A través de esta innovación, se busca desarrollar un circuito de bajo costo que permita una gestión eficaz de la señal PWM del motor. Este enfoque no solo mejorará la seguridad y la eficiencia en la producción, sino que también reducirá la carga física asociada con el uso constante del torno, aliviando así la presión en las rodillas de los alfareros.

Es importante destacar que este proyecto se basa en una profunda investigación y análisis técnico, respaldado por pruebas exhaustivas a nivel de laboratorio.

1.4 Objetivos.

1.4.1 Objetivo General.

El objetivo general es aplicar los conocimientos adquiridos durante la carrera de Ingeniería en Sistemas Electrónicos Industriales impartida por la UACM, para construir un prototipo de circuito para aplicar a un torno de alfarero que utilice la tecnología de la IA y obtener nuevos conocimientos de dicha tecnología, este prototipo debe solucionar los problemas mencionados en el planteamiento del problema.

1.4.2 Objetivos Particulares.

a) Crear un Modelo matemático utilizando IA para reconocimiento de voz
    - Generar datos de entrenamiento
    - Entrenar el modelo con los datos de entrenamiento
    - Convertir el modelo ya entrenado a .cc (es un archivo de código fuente en lenguaje de programación C++) para poder subirlo al módulo ESP32

b) Diseñar el circuito que se encargara de hacer girar el plato o rueda del torno solo con la voz.
    - Diseñar su PCB

c) Adaptar el prototipo del circuito para aplicar a un torno de alfarero

1.5 Organización del Trabajo.

En el primer capítulo, además de la introducción, se discutirán varios temas importantes. Estos incluyen los motivos detrás del proyecto, la formulación del problema que se desea resolver, la justificación para llevar a cabo el proyecto y los objetivos específicos que se buscan alcanzar con la tesis. En resumen, se tratará de dar un panorama amplio de los fundamentos y contexto del proyecto.

En el segundo capítulo, se presentará la investigación previa y las consideraciones teóricas necesarias para llevar a cabo el proyecto. Este capítulo proporcionará una visión general de toda la investigación que se realizó para poder llevar a cabo el reconocimiento de voz mediante inteligencia artificial. Se detallan aspectos relevantes, como el software y librerías utilizadas, así como el lenguaje de programación que se utilizó en el desarrollo del proyecto. Este capítulo será una guía para entender las decisiones tomadas en el desarrollo del proyecto y las herramientas utilizadas.

En el tercer capítulo describe todo el proceso para crear un modelo de reconocimiento de voz mediante IA. Se explica cómo se generaron los datos de entrenamiento, que son audios, y cómo se prepararon para ser utilizados en el entrenamiento. Se detalla el proceso de entrenamiento del modelo y cómo se convirtió el modelo entrenado a un archivo .cc, todo esto a través de programación. Este capítulo proporciona una descripción detallada del proceso de creación del modelo de IA para el reconocimiento de voz.

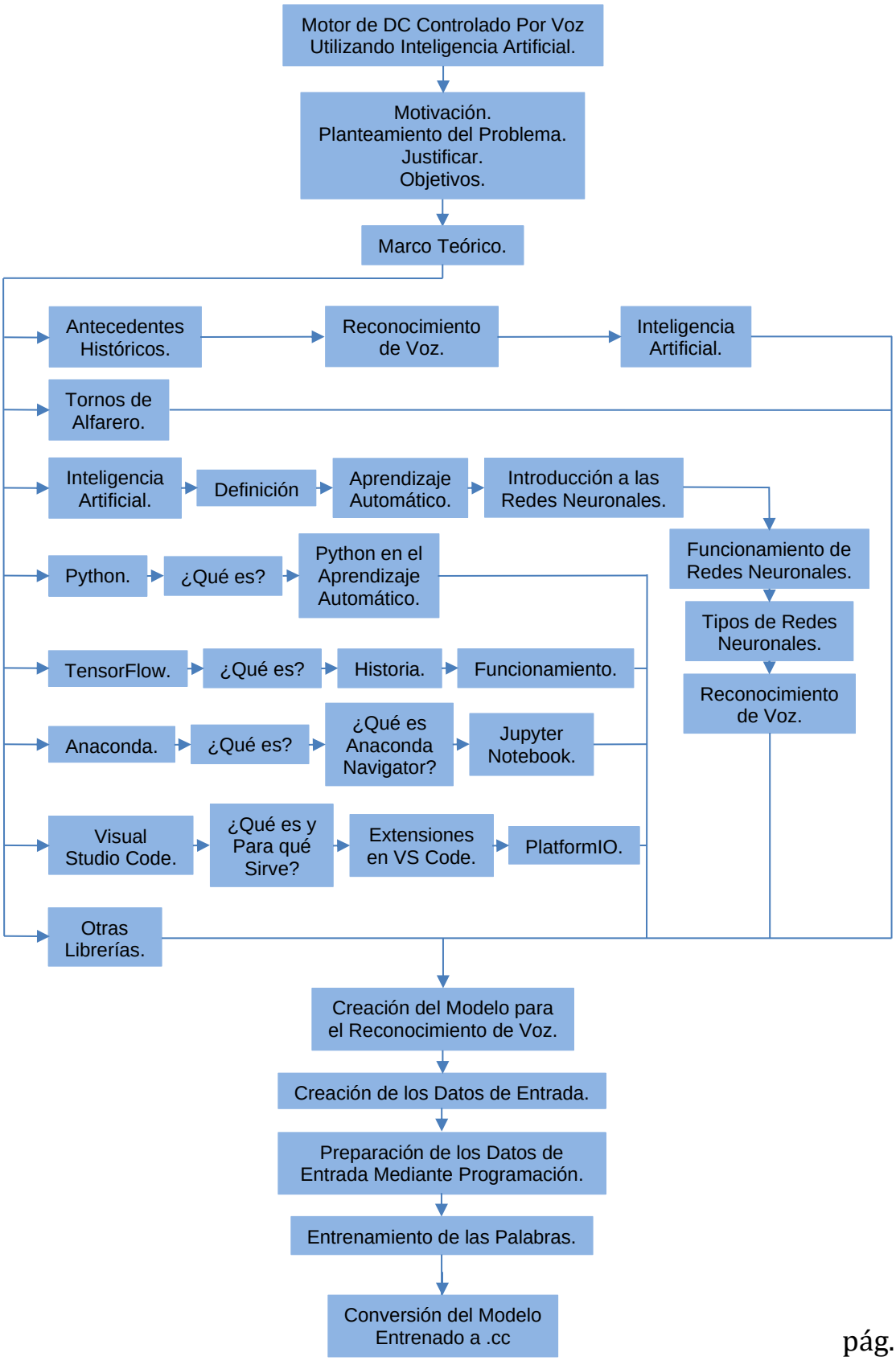
En el cuarto capítulo se diseñó y construyó el sistema electrónico utilizado en el proyecto. En este capítulo se describe en detalle los componentes utilizados en el proyecto, tales como el micrófono INMP441, el puente H L298N, el Motor DC y el Microcontrolador ESP32. Además, se presenta el diseño del circuito y se muestran los diagramas de conexión de cada componente, así como también el diseño del PCB utilizado para la construcción del sistema electrónico.

En el capítulo cinco se describe el proceso de implementación del modelo entrenado al sistema electrónico, mencionando el proceso de cargar el modelo de aprendizaje automático entrenado en el ESP32. En general, se explicará cómo se carga el código del modelo en el microcontrolador y se describe el programa que se encarga de controlar el motor.

En el capítulo seis se presentan los resultados obtenidos del proyecto. Se describen los resultados obtenidos en el proceso de reconocimiento de voz aplicado a un motor de torno de alfarero. Además, se incluyen fotos del prototipo del circuito final que se construyó, permitiendo al lector tener una visión general de cómo se ve el proyecto terminado.

Finalmente, en el último capítulo siente, se presentan las conclusiones finales del trabajo y se mencionan algunos posibles trabajos a futuro relacionados con el tema del proyecto.

En la Figura 1 se aprecia un diagrama que proporciona una visión general de este Trabajo, este diagrama ayuda a entender la estructura del trabajo desarrollado.



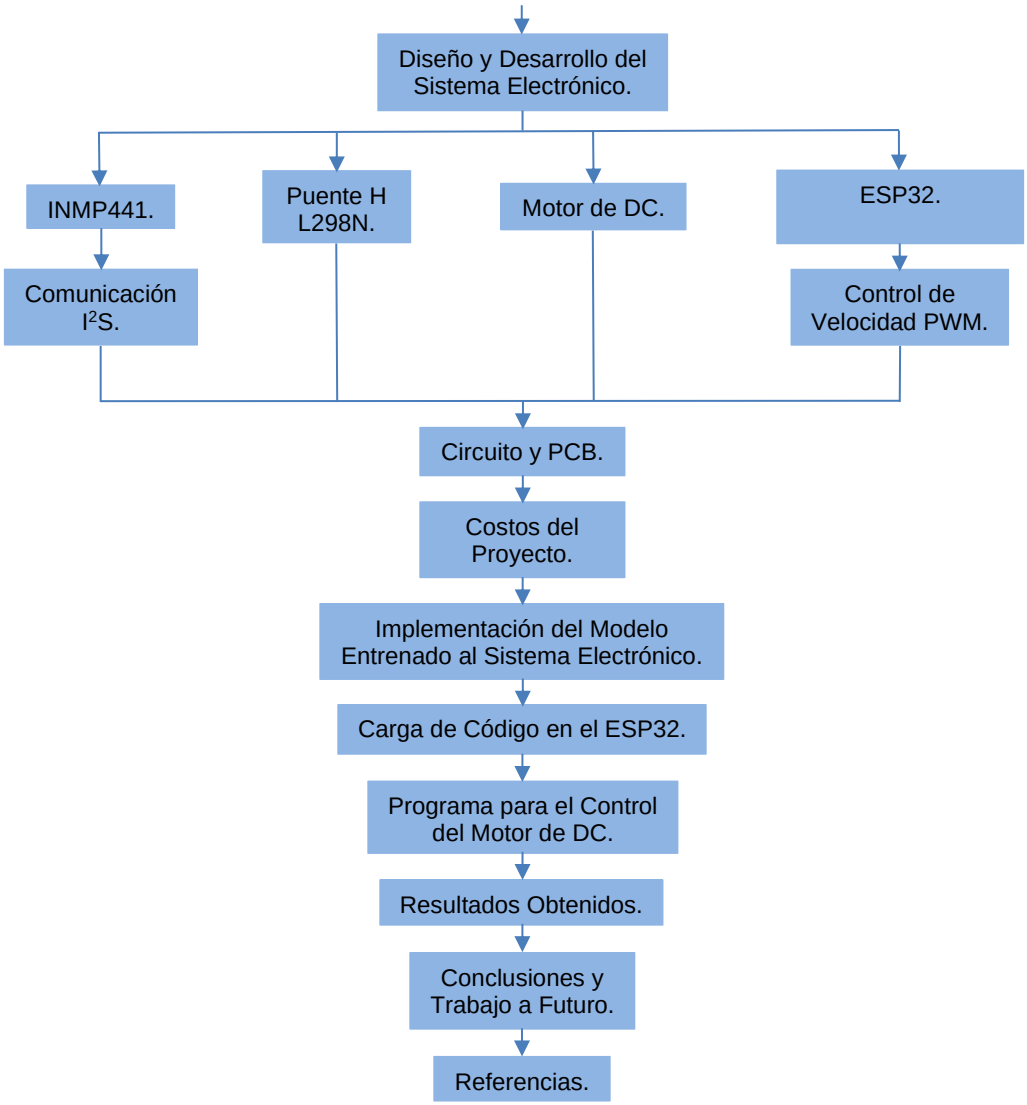


Figura 1. Diagrama General del Trabajo.

1.6 Diagrama General del Proyecto.

En la Figura 2, de una manera resumida se proporciona una visión general de los pasos y componentes clave involucrados en el desarrollo de este trabajo. Esta figura abarca los primeros pasos del trabajo realizado, iniciando con el Software, Material y librerias utilizadas para continuar con el entranamiento del modelo matematico creado en TensorFlow esto abarca desde la creacion de los datos de entrada a entrenar, su preparacion, el entrenamiento y la conversion del modelo a un archivo .cc para posteriormente utilizar PlatformIO en donde podremos subir el archivo .cc al ESP32 y finalmente crear su PCB.

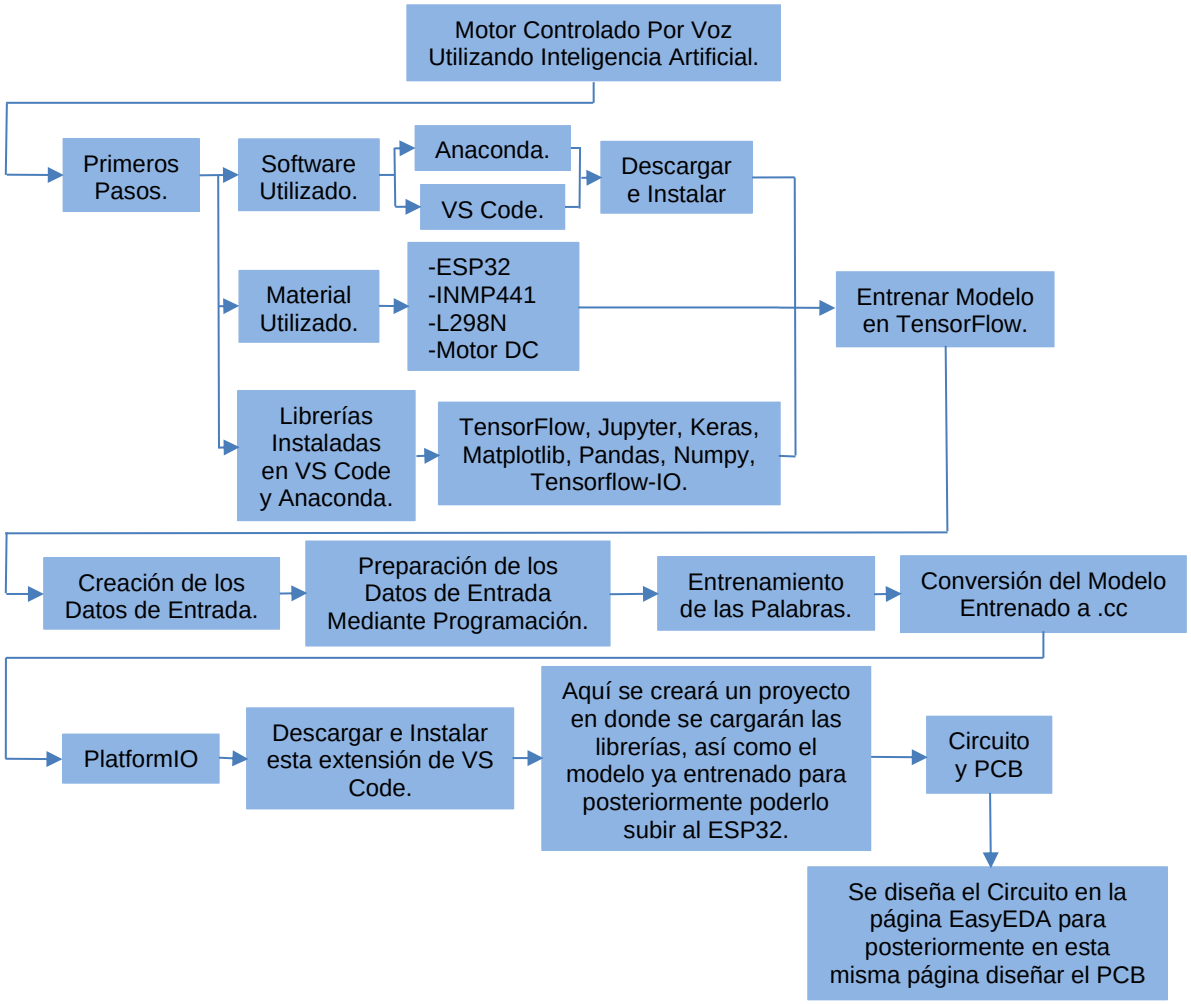


Figura 2. Diagrama General del Proyecto.

Capítulo 2.
Marco Teórico.

2.1 Antecedentes Históricos.

Estos antecedentes son fundamentales para entender el contexto y el desarrollo de estos temas en el pasado, y cómo han evolucionado para llegar a la tecnología actual. Con esta información se podrá entender cómo se ha llegado a la posibilidad de realizar el control por voz mediante el uso de inteligencia artificial y el reconocimiento de voz.

2.1.1 Reconocimiento de Voz.

El reconocimiento de voz ha recorrido un fascinante camino a lo largo de la historia, marcando hitos significativos en su desarrollo y aplicaciones. Desde los primeros intentos de Alexander Graham Bell en 1870 por proporcionar una forma visible de comunicación para personas con discapacidad auditiva, hasta los avances notables en la década de 1970 con la creación del primer sistema comercial de reconocimiento de voz, hemos presenciado una evolución impresionante en esta tecnología. Algunos momentos clave en la historia del reconocimiento de voz son:

- 1870: Alexander Graham Bell intenta desarrollar un dispositivo para proporcionar la palabra visible para personas que no escuchan.
- 1930: Tihamer Nemes, científico húngaro, intenta patentar una máquina para la transcripción automática de la voz. Su petición fue negada y considerada poco realista.
- 1952: Primer esfuerzo para crear una máquina de reconocimiento de voz en los laboratorios de AT&T. Se desarrolla un sistema que reconoce dígitos del 0 al 9 con una exactitud del 98%.
- 1958: Dudley crea un clasificador que evalúa continuamente espectros. Este método se convirtió en el paradigma dominante en el reconocimiento de voz.

- 1959: Denes, del College of London, agrega probabilidades gramaticales a la información acústica. Esto permite que la probabilidad de una palabra no dependa únicamente de la entrada acústica.
- 1962: David y Selfridge realizan una tabla comparando una serie de experimentos de reconocimiento de voz. Se logra una buena precisión para múltiples hablantes.
- 1964: Martin implementa redes neuronales para el reconocimiento de fonemas.
- 1963: Widrow entrena redes neuronales para reconocer dígitos.
- 1971-1976: Proyecto ARPA de comprensión del habla financiado por la Agencia de Proyectos de Investigación Avanzada (ARPA) donde se desarrolló el primer sistema de reconocimiento de voz comercial con un vocabulario limitado y dependiente del locutor, pero con una precisión del reconocimiento de cerca del 90%. También se desarrollaron técnicas importantes para el reconocimiento de voz como el análisis cepstral, la codificación predictiva lineal y el modelo oculto de Markov.
- 1980: Con el crecimiento de las computadoras personales y el apoyo de ARPA, el reconocimiento de voz comenzó a ser más accesible y asequible para el público en general. Se desarrollaron sistemas de reconocimiento de voz independientes del locutor y con vocabularios más extensos.
- 1990: Los costos de las aplicaciones de reconocimiento de voz continuaron decreciendo y los vocabularios extensos se volvieron comunes. También se desarrollaron aplicaciones de reconocimiento de voz de flujo continuo (sin pausas significativas entre palabras y frases) y se mejoró la precisión del reconocimiento [2].

Para poder entender la voz humana, se utilizan las Redes Neuronales, como se observa en la historia. Estas surgieron como una forma atractiva de codificar el sonido. Desde entonces, se han utilizado en varios aspectos del reconocimiento de voz, como en la clasificación de sonidos, el reconocimiento de palabras individuales, el reconocimiento de voz con video, entre otros. Al rededor del 2000, el reconocimiento de voz se volvió cada vez más popular en dispositivos móviles y asistentes virtuales. Además, el uso de técnicas de aprendizaje automático y grandes conjuntos de datos para entrenar modelos de reconocimiento de voz ha llevado a una mayor precisión y capacidad para manejar diferentes acentos y dialectos.

2.1.2 Inteligencia Artificial.

Los antecedentes históricos de la IA se remontan a varios siglos atrás, con ideas y teorías sobre la posibilidad de crear máquinas inteligentes. Sin embargo, el campo de la IA como lo conocemos hoy en día se considera que tiene sus raíces en el siglo XX. Algunos de los eventos y desarrollos clave en la historia de la IA se resumen en:

- 1842: De los números a la poesía, La matemática Ada Lovelace fue la primera en ver el potencial de las computadoras más allá de las matemáticas.
- 1921: Se introduce la palabra “Robot”, Karel Capek, un dramaturgo checo, lanzó su obra de ciencia ficción “Rossum’s Universal Robots”, donde exploró el concepto de personas artificiales a las que llamó robots, que proviene de la palabra “robota” (esclavo).
- 1943: Las neuronas se vuelven artificiales, El primer modelo matemático de la neurona fue propuesto por Warren McCulloch y Walter Pitts.
- 1950: Turing Tes, Alan Turing propone un test para saber si una máquina exhibe un comportamiento inteligente.
- 1956: La “IA” nace, El término “inteligencia artificial” es acuñado en una conferencia en la Universidad de Dartmouth organizada por John McCarthy.
- 1956: El primer programa de IA, Allen Newell, Herbert Simon y Cliff Shaw fueron coautores de Logic Theorist, el primer programa informático de inteligencia artificial.
- 1961: Unimate, El robot industrial, Unimate, inventado por George Devol, se convirtió en el primero en trabajar en una línea de montaje de General Motors
- 1964: Eliza, Joseph Weizenbaum, científico informático de MIT, desarrolló Eliza, el primer chatbot que podía conversar funcionalmente en inglés con una persona.
- 1969: El problema del XOR, Marvin Minsky y Seymour Papert exploran en un libro las fortalezas y limitaciones de los perceptrones, la más importante siendo la incapacidad de implementar la función lógica XOR.
- 1974-1980: Invierno IA, Muchos comienzos en falso y callejones sin salida dejan a la investigación en IA sin fondos y con poco interés

- 1986: Aprendiendo a aprender con retropropagación, En un artículo muy influyente, Rumelhart, Hinton, y Williams, popularizan el algoritmo de retropropagación para entrenar redes neuronales multicapa [3].

Es importante señalar que estos eventos son solo algunos ejemplos de los muchos hitos importantes en la historia de la IA, y hay muchos otros eventos y desarrollos que también han contribuido al campo.

2.2 Tornos de Alfarero.

Actualmente existen tres tipos de tornos para alfarería:

1. El Torno de Rueda (Figura 3): Es uno de los más antiguos y suele ser utilizado por alfareros expertos, ya que requiere habilidad para su uso correcto. El torno de rueda se utiliza girando con el pie la base de madera llamada volante, lo que hace que gire el plato o rueda y permite moldear el barro.
2. El Torno de Pedal (Figura 4): Es similar al torno de rueda, pero en lugar de utilizar un volante, se utiliza un pedal que es pisado por el alfarero para hacer que gire el torno. Una desventaja de este tipo de torno es que no permite controlar la velocidad de giro.
3. El Torno Eléctrico: Puede tener un pedal que se presiona con el pie para comenzar a girar, pero también existen algunos que no necesitan de un pedal ya que, al conectarlos a la corriente eléctrica, comienzan a funcionar automáticamente [4].

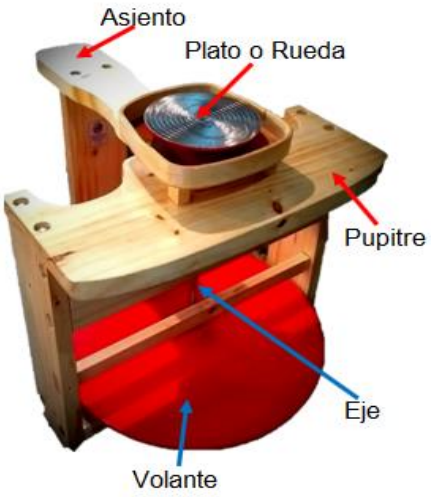


Figura 3. Torno de Rueda.

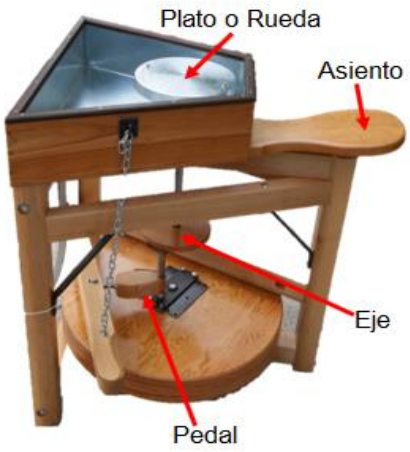


Figura 4. Torno de Pedal.

2.3 Inteligencia Artificial.

2.3.1 Definición.

Es difícil describir exactamente qué es la inteligencia artificial, pero en general se podría afirmar que se trata de la habilidad de las computadoras de utilizar algoritmos, aprender a partir de los datos y aplicar lo aprendido en la toma de decisiones, de manera similar a como lo hace una persona [5].

La IA cuenta con distintas ramas como lo son:

- Aprendizaje automático (Machine Learning)
- Aprendizaje profundo (Deep Learning)
- Base de datos conocimiento (Data base knowledge)
- Minería de datos (Data mining)

En este trabajo se estudiarán y aplicarán técnicas de aprendizaje automático y profundo, las cuales son una parte importante de la inteligencia artificial. La IA busca replicar la inteligencia humana en las computadoras, permitiéndoles medir, razonar, participar y aprender. El objetivo del aprendizaje automático es desarrollar métodos para que las computadoras puedan aprender, mejorando su rendimiento a medida que se les proporciona más datos. El aprendizaje profundo, también se utilizará en este trabajo, se enfoca en la creación de modelos abstractos de alto nivel como la detección de rostros, objetos, imágenes y el reconocimiento de voz.

2.3.2 Aprendizaje Automático.

El Aprendizaje Automático o Machine Learning busca que una computadora aprenda a partir de un conjunto de datos con los cuales es entrenada. El objetivo es identificar patrones en estos datos con los cuales se puedan realizar predicciones sobre nuevos datos. En comparación con la programación tradicional, donde se procesan datos de entrada y se genera una salida a través de una serie de reglas, en el Aprendizaje Automático, los datos y las salidas son utilizados para generar las reglas mediante un proceso de entrenamiento. Estas reglas son conocidas como modelos y son el resultado de detectar patrones o tendencias en los datos originales para poder hacer predicciones sobre datos nunca antes vistos. Este proceso es llevado a cabo por algoritmos, los cuales, si son bien diseñados, tienen buenos resultados y pueden mejorar a través de pruebas y errores para obtener mejores resultados [6].

2.3.3 Introducción a las Redes Neuronales.

Una red neuronal es un conjunto de algoritmos diseñados para simular el funcionamiento del cerebro humano. Estos algoritmos son capaces de reconocer patrones en la información, y pueden ser utilizados para reconocimiento de imágenes, sonidos, palabras, entre otros. En el campo de la inteligencia artificial, las redes neuronales son utilizadas para "aprender" a realizar tareas específicas. Esto se logra mediante el entrenamiento continuo de la red, en el cual se le proporciona información de qué es correcto y qué es incorrecto. Este proceso se conoce como aprendizaje profundo, ya que utiliza una estructura de capas conectadas, similar al cerebro humano, para crear un sistema adaptable. Una red neuronal básica está compuesta por varias capas, cada una de las cuales tiene un conjunto de nodos conectados entre sí, conocidos como neuronas. Cada neurona recibe información, la procesa y la transmite a las siguientes neuronas, siempre y cuando la información saliente supere cierta función de activación. En una red neuronal, existen distintos tipos de capas, como la capa de entrada, capas intermedias (ocultas) y la capa de salida como se observa en la Figura 5. Dependiendo de cómo estas capas y neuronas estén conectadas entre sí, existen

distintos tipos de redes neuronales. Por último, el método de entrenamiento es similar para la mayoría de las redes neuronales: se le proporciona información de entrada a la red, esta produce una salida, se le indica cuál es la salida correcta y la red se encarga de modificar y optimizar sus parámetros internos para lograr el resultado deseado [7].

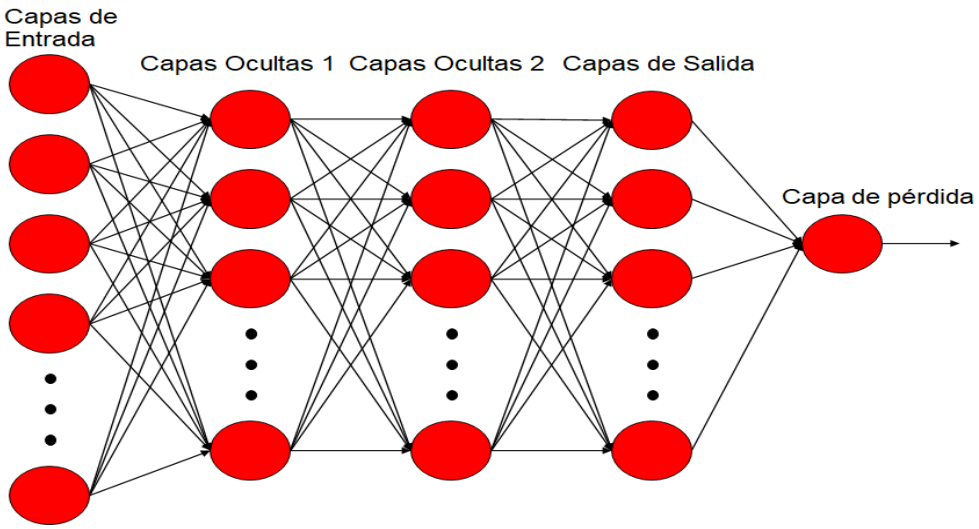


Figura 5. Arquitectura de una red neuronal.

2.3.4 Funcionamiento de las Redes Neuronales.

La arquitectura de las redes neuronales se basa en el cerebro humano. Las células del cerebro, conocidas como neuronas, forman una red compleja y altamente interconectada que se comunican mediante señales eléctricas para ayudar a los humanos a procesar la información. De manera similar, una red neuronal artificial está compuesta por "neuronas artificiales" o módulos de software llamados nodos, que trabajan juntos para resolver un problema específico. Estas redes neuronales artificiales son programadas en software o algoritmos que, en esencia, utilizan sistemas informáticos para resolver cálculos matemáticos.

Una red neuronal básica tiene tres capas de neuronas artificiales interconectadas:

- **Capas de entrada:** Es donde la información del mundo exterior entra a la red neuronal artificial. Los nodos de entrada procesan, analizan y clasifican los datos, y los pasan a la siguiente capa. Es la información que se evaluará para obtener un resultado.
- **Capas ocultas.** Toma su entrada de la capa de entrada o de otras capas ocultas. Una red neuronal puede tener varias capas ocultas. Cada capa oculta analiza la salida de la capa anterior, la procesa aún más y la pasa a la siguiente capa. Es donde se realiza todo el procesamiento de aprendizaje.
- **Capa de salida.** Proporciona el resultado final de todo el procesamiento de datos realizado por la red neuronal artificial. Puede tener uno o varios nodos. Por ejemplo, si se trata de un problema de clasificación binaria (sí/no), la capa de salida tendrá un nodo de salida que dará como resultado 1 o 0. Sin embargo, si se trata de un problema de clasificación multiclase, la capa de salida puede estar compuesta por más de un nodo de salida [8].

2.3.5 Tipos de redes neuronales.

Red Neuronal Monocapa – Perceptrón Simple.

La Red Neuronal Monocapa, también conocida como Perceptrón Simple, es la forma más básica de una red neuronal. Consiste en una sola capa de neuronas que reciben entradas y las transmiten a una capa de neuronas de salida (como se observa en la Figura 6) donde se llevan a cabo los cálculos necesarios para obtener un resultado.

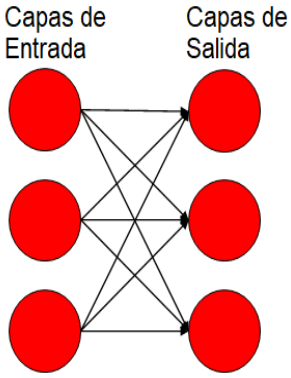


Figura 6. Red Neuronal Monocapa.

Red Neuronal Multicapa – Perceptrón Multicapa.

Es una extensión de la red neuronal monocapa, en la que se agrega un conjunto de capas intermedias u ocultas entre la capa de entrada y la capa de salida (como se observa en la Figura 7). En esta red, las entradas se proyectan a través de varias capas de neuronas antes de llegar a la capa de salida donde se realizan los cálculos finales.

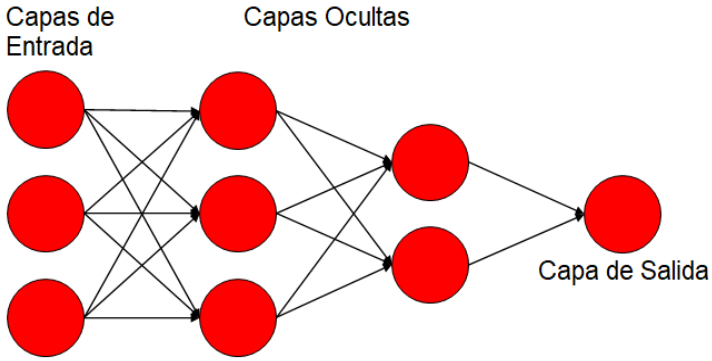


Figura 7. Red Neuronal Multicapa.

Red Neuronal Convolucional (CNN).

Esta red es diferente al perceptrón multicapa, ya que cada neurona no está conectada con todas las capas siguientes, sino solo con un subgrupo de ellas (Como se ve en la Figura 8). Esto ayuda a reducir el número de neuronas necesarias y la complejidad computacional requerida para su ejecución [9].

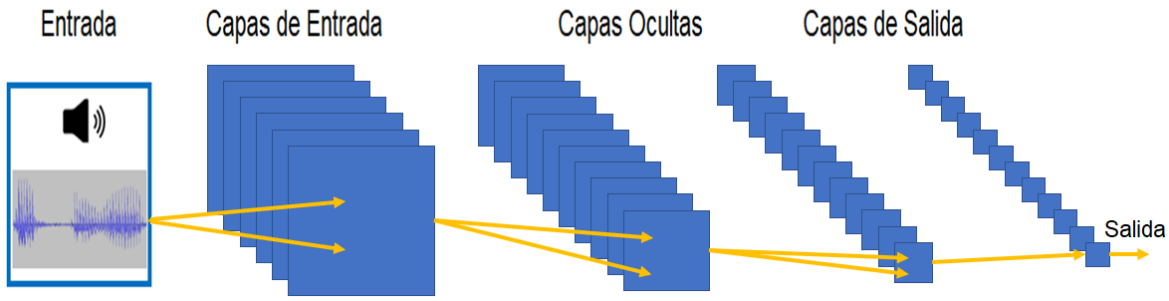


Figura 8. Red Neuronal Convolucional.

Una Red Neuronal Convolucional (CNN) es una herramienta poderosa en el campo del procesamiento de señales, particularmente en aplicaciones como el reconocimiento de voz. Se puede pensar en una CNN como un conjunto de filtros especializados que escanean una señal, como una forma de onda de audio, para detectar características específicas.

En el caso del reconocimiento de voz, estas características podrían ser patrones de frecuencia y duración de ciertos sonidos o fonemas. Una CNN busca patrones distintivos en la forma en que se pronuncian diferentes palabras.

El proceso comienza con la captura de la señal de voz y su conversión en una representación digital. Luego, la CNN aplica múltiples capas de filtros para analizar la señal en diferentes niveles de detalle. Estos filtros se entrenan para reconocer patrones específicos, como cambios en la frecuencia o la amplitud.

A medida que la CNN procesa la señal, las características detectadas se combinan y se utilizan para determinar la probabilidad de que la entrada de voz corresponda a una palabra o frase específica. Esto es similar a cómo, cuando escuchamos a alguien hablar, nuestro cerebro identifica ciertos sonidos y los relaciona con palabras y significados.

En el contexto del reconocimiento de voz, una CNN bien entrenada puede distinguir entre diferentes comandos de voz, como "encender", "apagar", "aumentar velocidad" o "disminuir velocidad". Esto tiene aplicaciones prácticas en el control de dispositivos electrónicos, como motores o sistemas de automatización que es lo que se busca en este trabajo.

Es importante destacar que el éxito de una CNN en el reconocimiento de voz depende en gran medida del conjunto de datos de entrenamiento y del diseño de la red. Una selección adecuada de datos y una arquitectura de red optimizada son fundamentales para lograr un alto nivel de precisión en la clasificación de comandos de voz.

Una Red Neuronal Convolucional (CNN) es preferible para el reconocimiento de voz sobre una red neuronal monocapa o multicapa por las siguientes razones:

1. **Extracción automática de características**: Las CNN pueden identificar automáticamente características relevantes de la señal de voz, evitando la necesidad de una extracción manual.
2. **Preservación de la estructura temporal**: Las CNN mantienen la estructura temporal de los datos, esencial para procesar señales secuenciales como el habla.
3. **Eficiencia en la detección de características locales**: Son altamente eficientes en la detección de patrones específicos en una señal de voz.

4. **Buena capacidad de generalización**: Las CNN pueden aprender a reconocer características relevantes en diferentes partes de una señal de voz y aplicar ese conocimiento a datos nuevos.

La elección de la arquitectura de red dependerá de la complejidad de la tarea y los datos disponibles, pero en el caso del reconocimiento de voz, las CNN ofrecen ventajas significativas.

2.3.6 Reconocimiento de voz.

Las redes neuronales son capaces de analizar la voz humana, independientemente de los diferentes patrones de habla, tonos, idiomas y acentos.

El proceso de reconocimiento de voz es complejo, pero, en resumen, el proceso de reconocimiento de voz implica recolectar audio mediante un micrófono, limpiar y preparar el audio, analizar características relevantes, entrenar un modelo de reconocimiento de voz, clasificar el audio y convertir la clasificación en una palabra o frase comprensible. Para lograr esto, se aplican varias disciplinas como la inteligencia artificial, la informática, la acústica y el procesamiento de señales.

2.4 Python.

En este trabajo se utilizarán varios programas y bibliotecas para realizar tareas específicas. Entre ellos se encuentran Anaconda con su entorno Jupyter Notebook y su biblioteca TensorFlow, así como Visual Studio Code como editor y su extensión PlatformIO. Es importante destacar que todos estos programas y bibliotecas utilizan el lenguaje de programación Python, por lo que es esencial tener conocimientos previos sobre Python para poder trabajar con ellos de manera eficiente.

2.4.1 ¿Qué es Python?

Python es un lenguaje de programación ampliamente utilizado en muchas aplicaciones, incluyendo el desarrollo web, software, análisis de datos y aprendizaje automático. Los programadores lo prefieren por su eficiencia y facilidad de aprendizaje, así como por su capacidad de funcionar en diferentes plataformas y entornos. Además, es gratuito y se adapta bien a todos los sistemas operativos [10].

2.4.2 Python en el Aprendizaje Automático (Machin Learning).

La ciencia de datos implica el proceso de obtener información valiosa a partir de los datos de entrada, mientras que el aprendizaje automático enseña a las computadoras a aprender por sí mismas a partir de los datos y a realizar predicciones precisas. Los científicos de datos utilizan Python para realizar tareas como limpiar y eliminar datos incorrectos, seleccionar características importantes de los datos, encontrar estadísticas significativas y visualizar los datos mediante gráficos y tablas. Utilizan las bibliotecas de Python para entrenar modelos de aprendizaje automático y clasificar datos con precisión, como imágenes, texto, tráfico de red, reconocimiento de voz y reconocimiento facial [10].

2.5 TensorFlow.

2.5.1 ¿Qué es TensorFlow?

La biblioteca TensorFlow es una herramienta de código abierto que permite llevar a cabo tareas relacionadas con el aprendizaje automático (también conocido como machin learning). Fue desarrollada por Google con el objetivo de facilitar la creación y entrenamiento de redes neuronales artificiales, las cuales son capaces de detectar patrones y razonamientos similares a los utilizados por los humanos [11].

2.5.2 Historia de TensorFlow.

El desarrollo inicial (2011-2016):

En el año 2011, el equipo de Google Brain creó una biblioteca de aprendizaje automático para su uso interno, llamada DistBelief. Esta biblioteca se utilizaba principalmente para negocios de Google, como su búsqueda o bien sus anuncios. Sin embargo, en el año 2015, Google decidió lanzar la biblioteca de TensorFlow como código abierto. Además, en 2016, Google anunció las unidades de procesamiento de tensores (TPU), las cuales son utilizadas para el aprendizaje profundo.

Primera Versión y avances en tecnologías multiplataforma (2017-2019):

En febrero de 2017, se lanzó TensorFlow 1.0 en su versión estable. Además, en mayo de 2017, se lanzó TensorFlow Lite, una biblioteca para desarrollo de aprendizaje automático en dispositivos móviles. En julio de 2018, Google anunció TPU Edge, diseñado para ejecutar modelos de aprendizaje automático utilizando TensorFlow Lite en teléfonos inteligentes. En enero de 2019, se anunció una nueva versión de TensorFlow, la 2.0.0, la cual se lanzó en septiembre de ese mismo año. También en mayo de 2019, se anunció TensorFlow Graphics para el renderizado gráfico y modelado 3D.

Nueva Vesion de TensorFlow 2.0.0 (2019-2020)

En septiembre de 2019, se lanzó TensorFlow 2.0.0, la cual agilizó inconvenientes en la construcción de redes neuronales. En esta versión, TensorFlow adoptó Keras como la principal interfaz de programación de aplicaciones (API) para construir,

entrenar y evaluar redes neuronales. Con esta nueva versión se agilizaron las herramientas de carga y procesamiento de datos, así como nuevas características. [12].

2.5.3 Funcionamiento.

TensorFlow es una herramienta que permite crear gráficos de flujo de datos que describen cómo los datos se mueven a través de una serie de nodos de procesamiento. Cada nodo representa una operación matemática, y cada conexión entre nodos es una matriz de datos multidimensional. A través de TensorFlow, se pueden programar estos gráficos en Python, ya que los nodos son objetos de Python y las aplicaciones de TensorFlow son a su vez aplicaciones de Python. Sin embargo, las operaciones matemáticas reales no se llevan a cabo en Python, sino en bibliotecas de transformaciones escritas en C++ de alto rendimiento. Python actúa como un intermediario que conecta estas piezas y proporciona abstracciones de programación de alto nivel.

TensorFlow permite ejecutar aplicaciones en casi cualquier plataforma, desde máquinas locales hasta clústeres en la nube, dispositivos móviles, CPU o GPU. Si se utiliza la nube de Google, se puede aprovechar la Unidad de Procesamiento TensorFlow (TPU) para aumentar el rendimiento. Los modelos resultantes creados con TensorFlow se pueden desplegar en la mayoría de los dispositivos para realizar predicciones [12].

2.6 Anaconda.

2.6.1 ¿Qué es Anaconda?

Anaconda es una plataforma de código abierto similar a TensorFlow, que permite escribir y ejecutar programas en Python y R. Es desarrollada por continuum.io, una compañía especializada en Python. Es muy utilizada para aprender a usar Python en áreas como la computación científica, el análisis de datos y el aprendizaje automático [13].

2.6.2 ¿Qué es Anaconda Navigator?

Anaconda Navigator es una herramienta gráfica incluida en la distribución Anaconda que facilita el uso de aplicaciones, paquetes, entornos y canales conda sin tener que usar comandos. Con Navigator se pueden buscar paquetes en Anaconda.org o en un repositorio local de Anaconda, y está disponible para los sistemas operativos Windows, macOS y Linux [14].

2.6.3 Jupyter Notebook.

El Jupyter Notebook es una herramienta gratuita y de código abierto que nos permite crear y compartir documentos y código. Es un ambiente interactivo que permite a los usuarios experimentar con diferentes lenguajes de programación y compartirlos con otros. El nombre Jupyter proviene de los tres lenguajes de programación con los que fue iniciado: Julia, Python y R, aunque actualmente soporta una amplia variedad de lenguajes [15].

2.7 Visual Studio Code.

2.7.1 ¿Qué es y para qué sirve?

VS Code es un editor de código desarrollado por Microsoft, es de código abierto y se puede utilizar en diferentes plataformas como Windows, macOS y Linux. Es una herramienta robusta que ofrece funciones de depuración y una gran variedad de extensiones, lo que permite escribir y ejecutar código en diferentes lenguajes de programación. Además, cuenta con una terminal integrada, que se activa fácilmente en el directorio de trabajo y puede utilizar diferentes shells como PowerShell, Bash o Ubuntu, lo que hace que sea fácil ejecutar los comandos necesarios al desarrollar código [16].

2.7.2 Extensiones en VS Code.

El gran poder de VS Code se debe principalmente a sus extensiones, que nos permiten personalizar y añadir funciones adicionales de manera modular y aislada. Por ejemplo, podemos programar en diferentes lenguajes, añadir nuevos temas al editor, y conectarnos con otros servicios. Estas extensiones mejoran la experiencia de usuario y no afectan el rendimiento del editor ya que se ejecutan en procesos independientes [16]. Para este trabajo se utilizará la extensión PlatformIO.

2.7.3 PlatformIO.

PlatformIO es una herramienta de programación de código abierto para C/C++ especialmente diseñada para trabajar con hardware. Se presenta como una extensión de Visual Studio Code [17].

2.8 Otras Librerías.

Keras.

La librería está diseñada para facilitar y agilizar el proceso de experimentación en el campo del aprendizaje automático y el aprendizaje profundo. Es una herramienta independiente que se utiliza de manera completa para crear modelos de aprendizaje automático y aprendizaje profundo, y ayuda a los ingenieros a desarrollar sus aplicaciones [18].

Matplotlib.

La librería es especialmente útil para crear representaciones visuales de datos en 2D con alta calidad gráfica, tales como histogramas, gráficos de error, gráficos de dispersión y gráficos de barras con solo unas pocas líneas de código. Permite exportar los gráficos generados en una variedad de formatos [18].

Pandas.

La librería Pandas se utiliza para el análisis de datos con una variedad de estructuras de datos flexibles y expresivas que son adecuadas tanto para datos relacionales como etiquetados. Es considerada como esencial en la resolución de problemas de análisis de datos en el mundo real en Python, debido a su estabilidad y rendimiento altamente optimizado [18].

Numpy.

La librería Numpy es una herramienta amplia para trabajar con matrices de diversos tipos. Ofrece una gran cantidad de funciones matemáticas de complejidad elevada, lo que la hace ideal para el manejo de vectores y matrices de gran tamaño. Es especialmente útil para realizar operaciones de álgebra lineal, transformadas de Fourier y generación de números aleatorios [18].

Tensorflow-IO.

Es una colección de sistemas de archivos y formatos de archivos que no están disponibles en el soporte integrado de tensorflow.

Capítulo 3.
Creación del Modelo para el Reconocimiento de Voz.

3.1 Diagrama del Modelo de Reconocimiento de Voz.

En la figura 9 se muestra un diagrama que representa el proceso de reconocimiento de voz, un área fundamental de la IA que permite a las maquinas interpretar y comprender el lenguaje hablado humano. El diagrama ilustra las etapas clave involucradas en el reconocimiento de voz, desde la ceación de los datos de entrada hasta la conversión del modelo entrenado a un archivo compilado en C++.

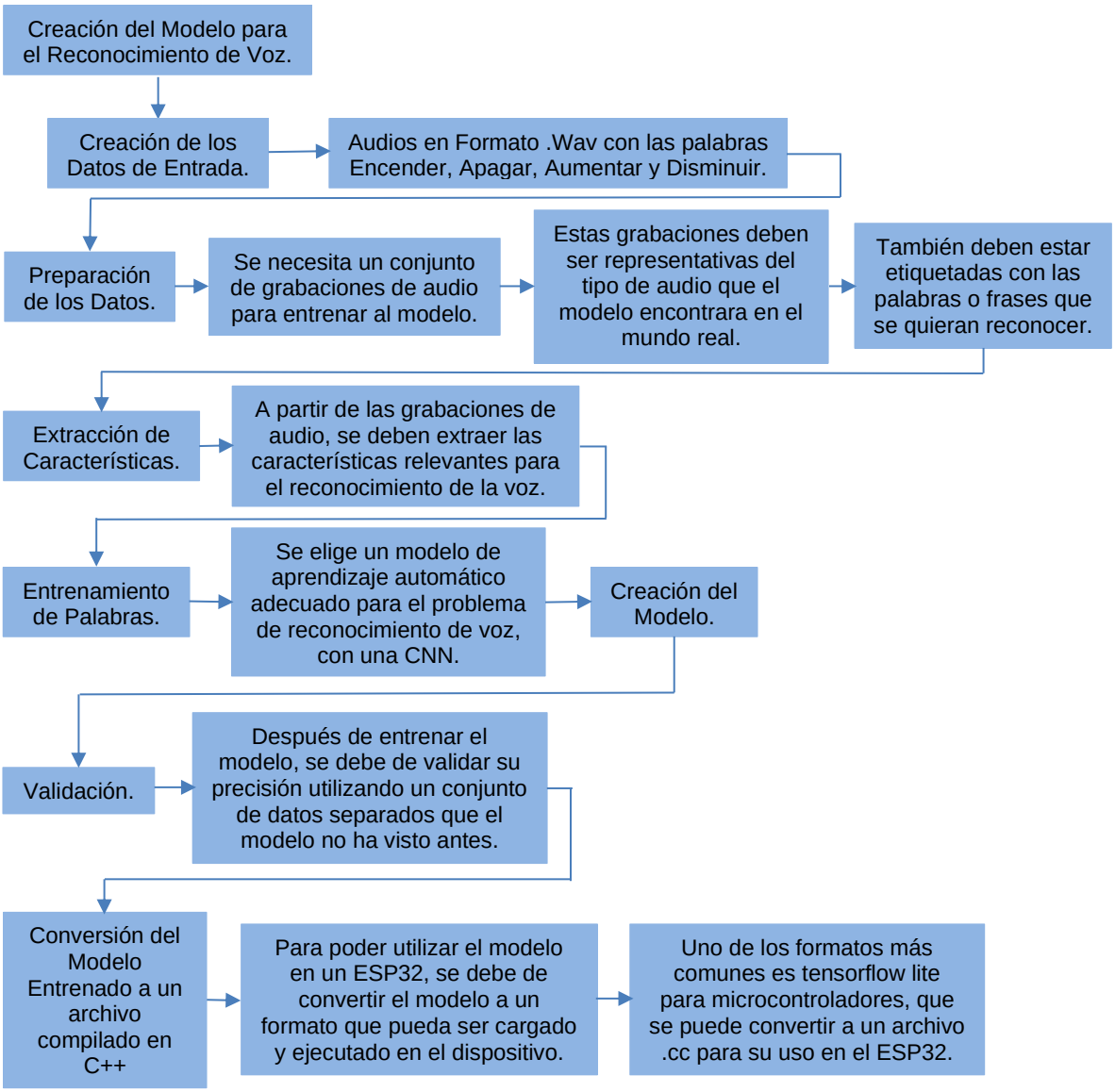


Figura 9. Diagrama del modelo de reconocimiento de voz.

Diagrama a Bloques del Proceso de Reconocimiento de Voz.

En el diagrama de la Figura 10 se detalla el proceso completo para llevar a cabo el reconocimiento de voz. El enfoque de este diagrama se centra exclusivamente en la etapa de reconocimiento de voz. Primeramente, se requiere la presencia de un sonido inicial, el cual es capturado por el micrófono INMP441. A fin de mejorar la calidad de la señal captada y mitigar posibles interferencias, se lleva a cabo un proceso de preprocesamiento. Seguidamente, se extraen características acústicas como la frecuencia y el espectro, las cuales servirán como datos de entrada para el modelo encargado de identificar y clasificar las palabras, asignándoles una acción correspondiente. Dichas acciones constituirán los comandos operativos del proyecto. Para culminar el proceso, se activará un indicador luminoso (LED) con el propósito de notificar al usuario que se ha reconocido y procesado un comando.

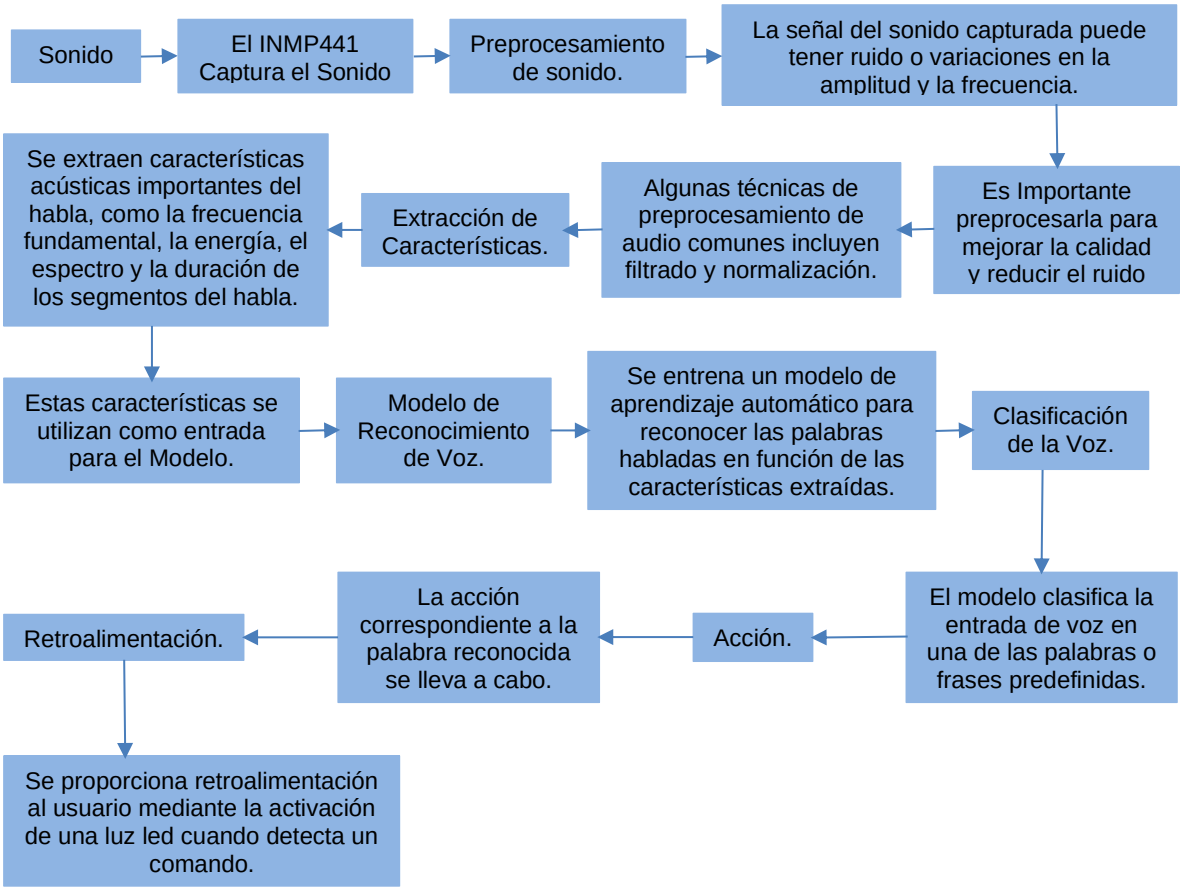


Figura 10. Diagrama del Proceso de Reconocimiento de Voz.

3.2 Creación de los Datos de Entrada.

Para que las redes neuronales puedan aprender a realizar una tarea, es necesario proporcionarles datos de entrada. Estos datos se analizan para poder procesar el aprendizaje y, finalmente, obtener las salidas deseadas.

Para lograr que un circuito ejecute acciones mediante comandos de voz, se debe crear un conjunto de datos de entrada. Estos datos son audios que contienen las palabras clave: Encender, Apagar, Aumentar y Disminuir, las cuales son los comandos que activarán las acciones del circuito. Además, se crearán otros audios con palabras no relevantes y ruido ambiental, con el fin de entrenar al modelo para que ignore estos sonidos y sólo responda a los comandos relevantes. Este conjunto de datos incluirá también palabras como: Abajo, Adelante, Aprender, Arriba, Atrás, Casa, Derecha, Gato, Izquierda, No, Pájaro, Parar, Perro, Seguir, Si, Vamos y los sonidos de ruido ambiente como por ejemplo ruido de gente.

Los datos de entrada en forma de audios se crearon utilizando una aplicación de grabación en el celular, pero se puede utilizar cualquier aplicación que grabe audio. Cada carpeta contiene un número de audios entre 1800 y 3420, con sus respectivas palabras, con la excepción de las carpetas llamadas Problemas_de_Ruido y Ruido_de_Fondo, las cuales contienen solo 5 y 11 archivos de audio respectivamente, con solo ruido ambiental. Estas carpetas tienen nombres con diagonal baja en lugar de espacios, esto es porque en la programación se trabaja con las carpetas, y algunas veces pueden generar errores al obtener los datos de las carpetas si se utilizan espacios en los nombres.

Una vez que se tiene el conjunto de audios, es necesario procesarlos para que cumplan con las especificaciones necesarias para el entrenamiento de la red neuronal. Esto incluye recortar cada audio a 1 segundo de duración y convertirlos al formato .wav (un formato de archivo de audio que almacena señales de onda). Ya que los audios grabados por los celulares suelen estar en otro formato. Para realizar esta conversión es necesario utilizar otro software y para este trabajo se eligió Audacity. Este software permite realizar el corte de audio y la conversión de formatos. Con este software, se importan todos los audios de cada carpeta, por ejemplo, primero se importan todos los audios de la palabra "apagar", y se observa en el software las señales de esta palabra (Figura 11).

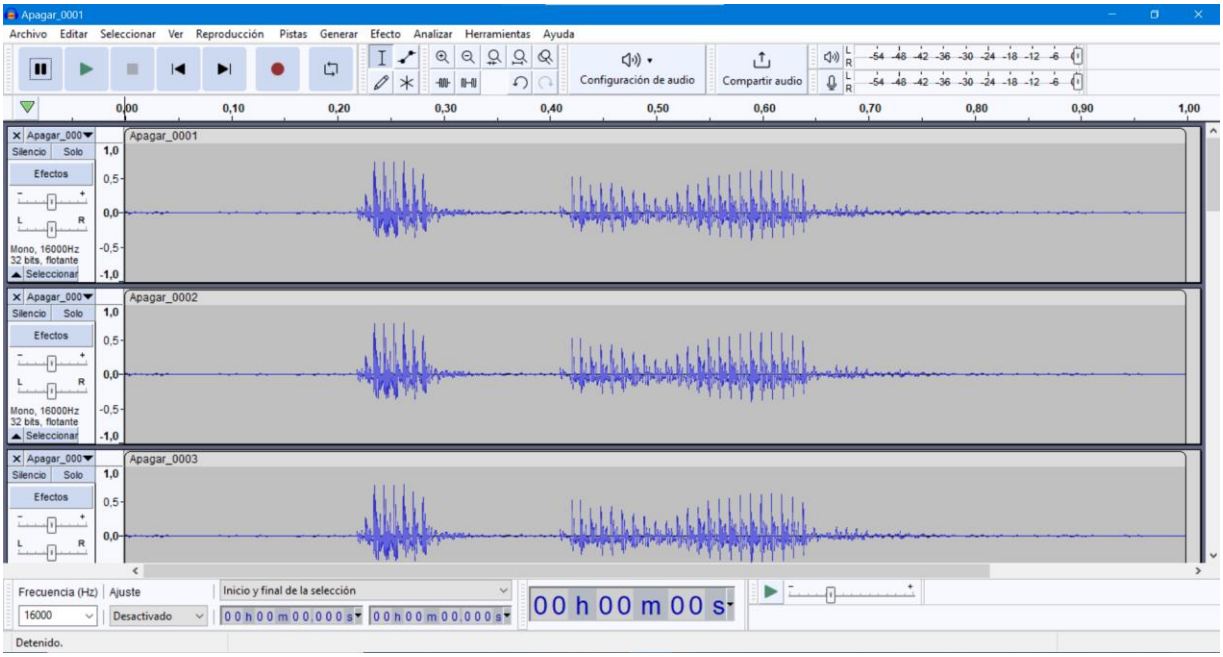


Figura 11. Señales de Audios de la Palabra Apagar y Visualización de Audacity.

Ya que se importaron todos los archivos de audio a Audacity, se cortan cada uno de los audios a 1 segundo y se selecciona la frecuencia, en la parte inferior izquierda de la Figura 11 dice Frecuencia (Hz) abajo debe de decir 16000 Hz, ya que esta frecuencia es comúnmente utilizada para el muestreo del audio de habla debido a que cumple con el teorema de muestreo de Nyquist-Shannon. Este teorema establece que, para reproducir fielmente una señal analógica, esta debe ser muestreada a una frecuencia dos veces mayor que la frecuencia más alta presente

en la señal.

En el caso del habla humana, la frecuencia máxima de ondas de sonido producidas durante la articulación de palabras es de aproximadamente 8000 Hz. Por lo tanto, una frecuencia de muestreo de 16000 Hz es suficiente para capturar todas las frecuencias relevantes del habla y permitir una reproducción de calidad.

Si se utilizara una frecuencia de muestreo menor, podrían producirse artefactos de aliasing, que son distorsiones no deseadas que ocurren cuando se submuestrea una señal. Por lo tanto, elegir una frecuencia de muestreo de 16000 Hz asegura una captura precisa y completa del espectro de frecuencias del habla.

Una vez teniendo la frecuencia antes mencionada, seleccionamos la pestaña que dice "archivos", "exportar" y finamente seleccionamos "exportar como wav".

Todos los audios deben ser procesados para convertirlos al formato .wav y organizarlos en sus respectivas carpetas. De esta manera, una vez que estén todos en el formato correcto y organizados en las carpetas correspondientes, se podrán utilizar en el programa que se está desarrollando.

Los datos se almacenaron en una carpeta llamada "Audios", dentro de la cual se encuentran subcarpetas que llevan el nombre de las palabras correspondientes, como se observa en la Figura 12.

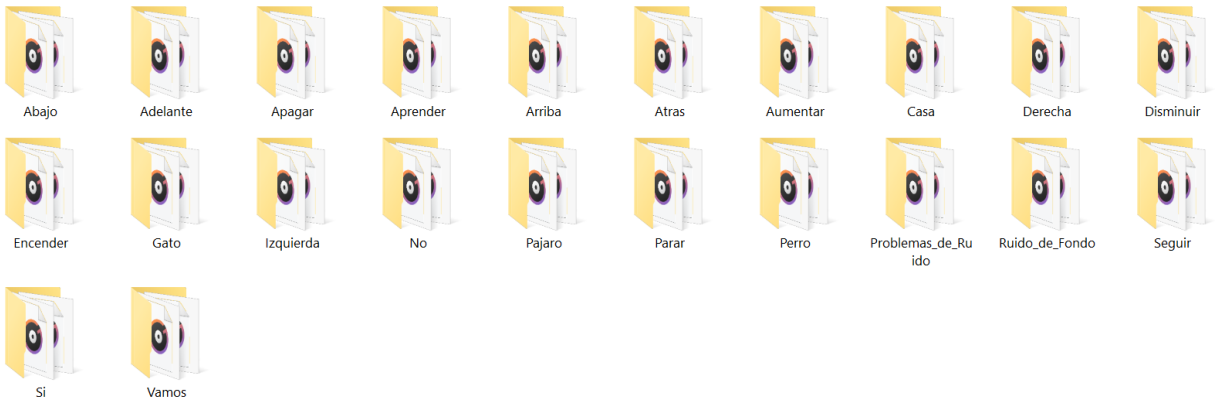


Figura 12. Subcarpetas con Datos de Audio de Palabras.

Es importante destacar que cada audio tiene un patrón de señal único relacionado con la palabra que se está grabando. A continuación, se presentan las señales de cada uno de los comandos Apagar (Figura 13) Aumentar (Figura 14) Disminuir (Figura 15) y Encender (Figura 16).

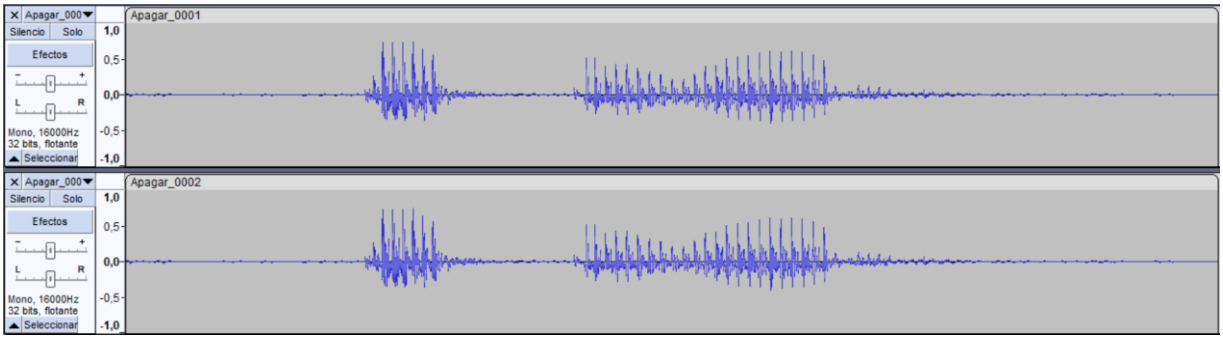


Figura 13. Señal del Comando Apagar.

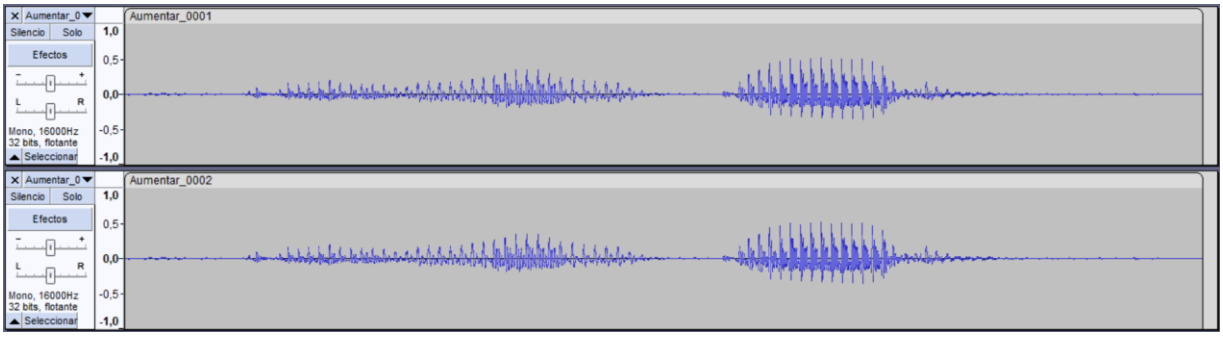


Figura 14. Señal del Comando Aumentar.

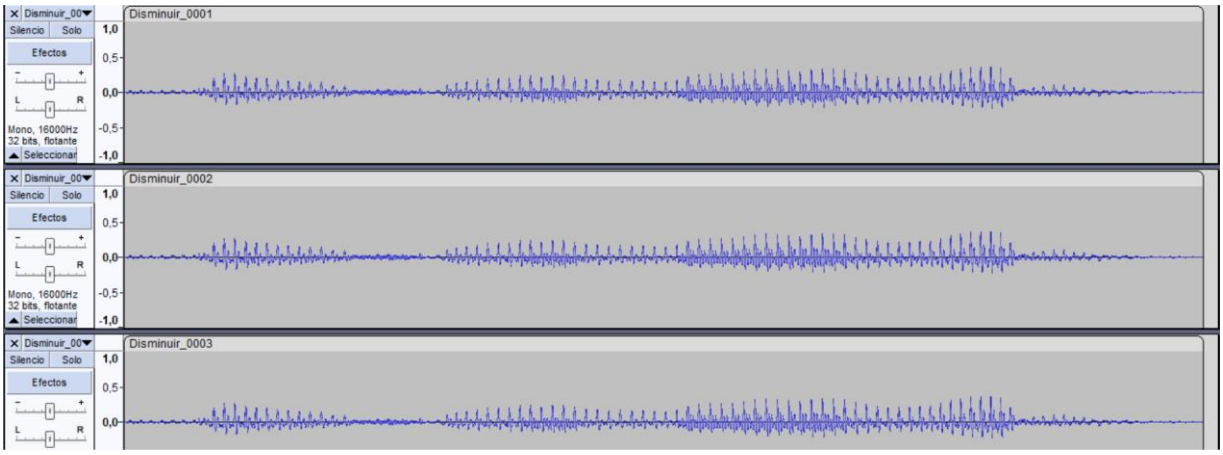


Figura 15. Señal del Comando Disminuir.

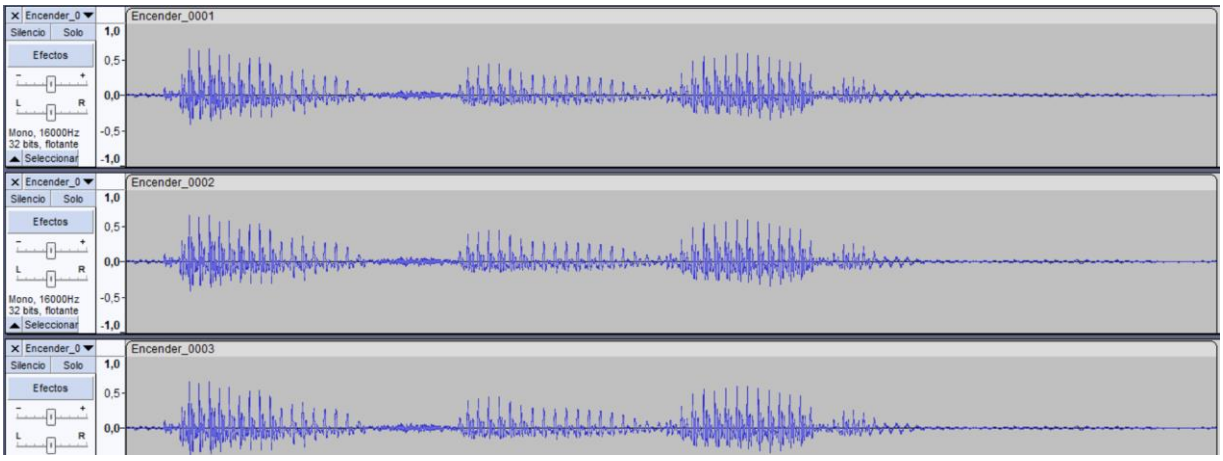


Figura 16. Señal del Comando Encender.

3.3 Preparación de los Datos de Entrada Mediante Programación.

Para comenzar a preparar los datos de entrada mediante programación, se abre el software Anaconda Navigator y se procede a descargar las librerías necesarias para el procesamiento de los datos, estas incluyen Keras, Matplotlib, Pandas, Numpy y Tensorflow. Es posible que algunas de estas librerías ya estén instaladas, pero las que no estén disponibles se tendrán que instalar. También se instala Tensorboard, Tensorboard-data-server y Tensorboard-plugin-wit para poder visualizar los datos de manera efectiva. Para la instalación, se selecciona "Environmen", "base (root)", y en la opción "Search packages", se buscan las librerías.

Instaladas las librerías, se selecciona el entorno Jupyter Notebook en donde se realizará la programación creando un archivo llamado "DatosDeEntrenamiento.ipynb".

En Jupyter Notebook, cada celda es un espacio donde se escriben líneas de código. En este caso, se explicará qué hace cada una de las celdas. En la Celda de Código N°1 (Figura 17) se importan algunas librerías necesarias para el proceso de entrenamiento, incluyendo TensorFlow, numpy, tensorflow-io, para acceder a archivos de audio y para manejar audio dentro de TensorFlow, tqdm para mostrar una barra de progreso en Jupyter Notebook, matplotlib para crear gráficos en python.

```
import tensorflow as tf
import numpy as np
from tensorflow.io import gfile
import tensorflow_io as tfio
from tensorflow.python.ops import gen_audio_ops as audio_ops
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
from tensorflow.python.ops import gen_audio_ops as audio_ops
```

Figura 17. Celda de Código N°1.

La Celda de Código N°2 (Figura 18) establece una variable.

- "SPEECH_DATA" en ella se le asigna el valor "Audios", En otras palabras, es una variable que almacena la dirección donde están los archivos de audio para ser usado en el código, algo así como la “ruta"

```
SPEECH_DATA='Audios'
```

Figura 18. Celda de Código N°2.

En la Celda de Código N°3 (Figura 19) se establecen 3 variables.

- “EXPECTED_SAMPLES” refiere el número de muestras de audio esperadas en cada archivo de audio.
- “NOISE_FLOOR” es el nivel mínimo de ruido permitido en un archivo, por lo que cualquier señal de audio con una amplitud menor a 0.1 será considerada como ruido y será eliminada.
- “MINIMUM_VOICE_LENGTH” en esta variable se guarda el tiempo mínimo permitido para una señal de voz en un archivo de audio, es decir, 4000 ms.

```
EXPECTED_SAMPLES=16000
NOISE_FLOOR=0.1
MINIMUM_VOICE_LENGTH=EXPECTED_SAMPLES/4
```

Figura 19. Celda de Código N°3.

La Celda de Código N°4 (Figura 20) establece dos listas.

- “command_words” contiene una serie de palabras, las cuales representan comandos válidos para el sistema de reconocimiento de voz.
- “nonsense_words” contiene una serie de palabras las cuales no tienen relación con los comandos válidos, estas palabras son consideradas ruido.

Estas dos listas serán utilizadas en una especie de etiquetado, para clasificar los audios en los comandos válidos y los no válidos.

```
command_words = [
    'Encender',
    'Apagar',
    'Aumentar',
    'Disminuir',
    '_invalid',
]
nonsense_words = [
    'Abajo',
    'Adelante',
    'Aprender',
    'Arriba',
    'Atras',
    'Casa',
    'Derecha',
    'Gato',
    'Izquierda',
    'No',
    'Pajaro',
    'Parar',
    'Perro',
    'Seguir',
    'Si',
    'Vamos',
]
```

Figura 20. Celda de Código N°4.

La Celda de Código N°5 (Figura 21) define varias funciones que se utilizan para procesar y validar los archivos de audio antes de ser utilizados para entrenar el modelo de aprendizaje automático.

- “get_files(word)” esta función toma como parámetro una palabra y devuelve una lista de archivos de audio en formato .wav que se encuentran en una subcarpeta con el nombre de esa palabra dentro de la carpeta audio.
- “get_voice_position(audio, noise_floor)” en esta función se toma un archivo de audio y un nivel de ruido como parámetro y devuelve las posiciones en las que comienza y termina la señal de voz en el archivo, descartando cualquier ruido.
- "get_voice_length(audio, noise_floor)" : esta función toma un archivo de audio y un nivel de ruido como parámetros y devuelve la longitud de tiempo de la señal de voz en el archivo, descartando cualquier ruido.
- "is_voice_present(audio, noise_floor, required_length)" : esta función toma un archivo de audio, un nivel de ruido y una longitud de tiempo requerida como parámetros y devuelve un valor booleano que indica si hay una señal de voz presente en el archivo.
- is_correct_length: Se utiliza para determinar si el archivo de audio tiene la longitud esperada, es decir la cantidad de muestras esperadas. La función toma un archivo de audio y la longitud esperada como parámetros y devuelve un valor booleano que indica si el archivo tiene la longitud esperada.

is_valid_file: Se utiliza para validar si un archivo de audio es válido, esto es si el archivo de audio tiene el formato esperado y si la señal de voz presente en el archivo cumple con las condiciones requeridas (duración y nivel de ruido). La función toma como parámetro el nombre del archivo, comprueba si tiene la longitud esperada, limpia el audio y finalmente verifica si hay una señal de voz presente en el archivo de audio con una duración mayor o igual a la requerida y si cumple con el nivel de ruido esperado, devuelve un valor booleano (True o False) si el archivo es válido o no.

```
def get_files(word):
    return gfile.glob(SPEECH_DATA + '/'+word+'/*.wav')

def get_voice_position(audio, noise_floor):
    audio = audio - np.mean(audio)
    audio = audio / np.max(np.abs(audio))
    return tfio.audio.trim(audio, axis=0, epsilon=noise_floor)

def get_voice_length(audio, noise_floor):
    position = get_voice_position(audio, noise_floor)
    return (position[1] - position[0]).numpy()

def is_voice_present(audio, noise_floor, required_length):
    voice_length = get_voice_length(audio, noise_floor)
    return voice_length >= required_length

def is_correct_length(audio, expected_length):
    return (audio.shape[0]==expected_length).numpy()


def is_valid_file(file_name):
    audio_tensor = tfio.audio.AudioIOTensor(file_name)
    if not is_correct_length(audio_tensor, EXPECTED_SAMPLES):
        return False
    audio = tf.cast(audio_tensor[:], tf.float32)
    audio = audio - np.mean(audio)
    audio = audio / np.max(np.abs(audio))
    if not is_voice_present(audio, NOISE_FLOOR, MINIMUM_VOICE_LENGTH):
        return False
    return True
```

Figura 21. Celda de Código N°5.

La Celda de Código N°6 (Figura 22) contiene una función para calcular el espectrograma de un archivo de audio. Un espectrograma es una representación gráfica de la frecuencia de una señal de audio a lo largo del tiempo. Realizando los siguientes pasos:

1. La señal se normaliza para que se encuentre en un rango de -1 a 1.
2. Se calcula el espectrograma con la función “audio_spectrogram”. Esta función divide la señal de audio en trozos y calcula la transformada de Fourier para cada trozo.

El análisis de Fourier es una herramienta matemática que nos permite describir ondas mediante fórmulas. Estas ondas están presentes en diversos fenómenos como ondas de radio, rayos X, ultrasonidos, infrasonidos, sísmicas, e incluso en el sonido que producimos con nuestra voz.

En el contexto del reconocimiento de voz, se utiliza la transformada de Fourier, una técnica matemática aplicada en el procesamiento de señales. Su función es descomponer una señal en sus componentes de frecuencia. En un programa de reconocimiento de voz, se emplea principalmente para extraer características importantes de la señal de audio. Estas características luego se utilizan en el análisis y procesamiento posteriores para reconocer y comprender los comandos de voz.

Para este trabajo, el analizador TRF (Transformada Rápida de Fourier) será una herramienta valiosa, ya que nos permite visualizar de manera gráfica el mundo, generalmente invisible, del sonido. Es importante recordar que las Series de Fourier nos permiten expresar cualquier onda periódica, sin importar su complejidad, como la combinación de ondas simples. Por otro lado, el Coeficiente de Fourier nos proporciona la amplitud de cada una de esas ondas simples. Esto significa que podemos utilizar el análisis de Fourier para describir cualquier tipo de onda, por muy complicada que sea, siempre y cuando se repita de manera periódica.

En las Figuras 13, 14, 15 y 16 se pueden observar las señales en forma de ondas de sonido para las palabras 'Apagar', 'Aumentar', 'Disminuir' y 'Encender'. Estas ondas muestran un patrón de repetición, lo que nos permite realizar un análisis de Fourier para cada una de estas palabras.
Para comenzar, se utiliza el análisis de Fourier para identificar la naturaleza de las ondas asociadas a las palabras. Una alternativa a utilizar las fórmulas de los coeficientes de Fourier para determinar el tamaño o amplitud de las ondas senoidales y cosenoidales que componen cada una de las palabras es mediante el uso del espectro, una forma más sencilla de visualizar los componentes de una onda compleja.

Es esencial tener en cuenta que, para obtener estos espectros, se requiere tener conocimiento de las frecuencias fundamentales de las ondas. Por lo tanto, es importante recordar la siguiente fórmula:

$$f = \frac{1}{T}$$

Donde *f* representa la frecuencia, es decir, cuántas veces la onda oscila en un segundo, mientras que *T* se refiere al periodo, que indica cuánto tiempo dura una oscilación.

Sin embargo, todo este estudio lo realiza el programa, ya que hacerlo uno mismo sería bastante complicado. En este punto, entra en juego la TRF (Transformada Rápida de Fourier), que podríamos decir que hace visible la voz humana para nosotros, representándola en forma de ondas. Esta herramienta tiene otra ventaja: es capaz de calcular los coeficientes de Fourier de las ondas en un instante determinado. Una vez obtenidos estos coeficientes, la TRF proporciona el espectro de la onda.

Otro aspecto importante son los filtros que se aplican a las ondas. Al aplicar estos filtros, podemos eliminar picos en las ondas, lo que nos permite descubrir qué tipo de sonido se obtiene como resultado. Estos resultados nos revelan las características distintivas del sonido asociado a cada palabra, lo que nos permite determinar con mayor precisión a qué sonido se refiere [19].

A continuación, se explica lo que hace la transformada de Fourier en el programa de reconocimiento de voz:

- **Descomposición en frecuencia:** La Transformada de Fourier toma una señal de audio, que es una onda en el dominio del tiempo, y la descompone en sus componentes de frecuencia. Esto significa que separa la señal en sus diferentes frecuencias, lo que permite analizar qué frecuencias están presentes en la señal y en qué cantidad.

- **Espectrograma:** Uno de los resultados comunes de aplicar la Transformada de Fourier a una señal de voz es la creación de un espectrograma. Un espectrograma es una representación gráfica en dos dimensiones que muestra cómo varía la energía de diferentes frecuencias en función del tiempo. En un espectrograma, el eje horizontal representa el tiempo, el eje vertical representa las frecuencias y el color o intensidad de cada punto en el gráfico indica la energía en esa frecuencia y tiempo específicos.

- **Características relevantes:** A partir del espectrograma o de la información de frecuencia generada por la Transformada de Fourier, se pueden extraer características relevantes para el reconocimiento de voz. Estas características pueden incluir la distribución de energía en diferentes bandas de frecuencia, la frecuencia fundamental (tono de la voz), cambios en el espectro a lo largo del tiempo y otros descriptores que ayudan a identificar patrones en la señal de voz.

- **Entrada para el modelo:** Las características extraídas se utilizan como entrada para un modelo de reconocimiento de voz, como una red neuronal, un clasificador, o cualquier otro algoritmo de aprendizaje automático. El modelo utiliza esta información para aprender a reconocer patrones en los datos y realizar la tarea de reconocimiento de voz, que podría ser la identificación de palabras, comandos o hablantes.

La Transformada de Fourier es una herramienta esencial en el reconocimiento de voz porque permite descomponer una señal de audio en información de frecuencia que luego se utiliza para extraer características relevantes y entrenar modelos de reconocimiento de voz. Esto ayuda a la máquina a entender y procesar la información acústica en el habla humana, esta información es importante, sin embargo, para este trabajo no se tocará el desarrollo de la transformada de Fourier.

3. La función “pooling” reduce la resolución espacial del espectrograma.
4. Se aplica una función para calcular el logaritmo base 10 del espectrograma.
5. Finalmente, la función devuelve el espectrograma calculado.

En resumen, la función se encarga de transformar la señal de audio en una representación visual de las frecuencias que componen el audio, se le aplica un pooling y se logaritma para reducir el tamaño de los datos, y se retorna esta presentación.

```
def get_spectrogram(audio):
    audio = audio - np.mean(audio)
    audio = audio / np.max(np.abs(audio))
    spectrogram = audio_ops.audio_spectrogram(audio,
                                              window_size=320,
                                              stride=160,
                                              magnitude_squared=True).numpy()
    spectrogram = tf.nn.pool(
        input=tf.expand_dims(spectrogram, -1),
        window_shape=[1, 6],
        strides=[1, 6],
        pooling_type='AVG',
        padding='SAME')
    spectrogram = tf.squeeze(spectrogram, axis=0)
    spectrogram = np.log10(spectrogram + 1e-6)
    return spectrogram
```

Figura 22. Celda de Código N°6.

La Celda de Código N°7 (Figura 23) contiene la función "process_file" se utiliza para preparar un archivo de audio para su uso como entrada para un modelo de aprendizaje automático. La función limpia el archivo de audio, lo desplaza aleatoriamente y lo mezcla con ruido de fondo, se calcula el espectrograma de este audio resultante y se devuelve este espectrograma para ser utilizado como entrada para el modelo de aprendizaje automático. El objetivo es lograr que el modelo sea capaz de identificar los comandos de voz a pesar del ruido ambiental.

```
def process_file(file_path):
    audio_tensor = tfio.audio.AudioIOTensor(file_path)
    audio = tf.cast(audio_tensor[:], tf.float32)
    audio = audio - np.mean(audio)
    audio = audio / np.max(np.abs(audio))
    voice_start, voice_end = get_voice_position(audio, NOISE_FLOOR)
    end_gap=len(audio) - voice_end
    random_offset = np.random.uniform(0, voice_start+end_gap)
    audio = np.roll(audio,-random_offset+end_gap)
    background_volume = np.random.uniform(0, 0.1)
    background_files = get_files('Ruido_de_Fondo')
    background_file = np.random.choice(background_files)
    background_tensor = tfio.audio.AudioIOTensor(background_file)
    background_start = np.random.randint(0, len(background_tensor) - 16000)
    background = tf.cast(background_tensor[background_start:background_start+16000], tf.float32)
    background = background - np.mean(background)
    background = background / np.max(np.abs(background))
    audio = audio + background_volume * background
    return get_spectrogram(audio)
```

Figura 23. Celda de Código N°7.

En la Celda de Código N°8 (Figura 24) se están declarando las variables "train", "validate" y "test" como listas vacías que se utilizarán para almacenar los datos de entrenamiento, validación y prueba respectivamente, y también se están estableciendo las variables TRAIN_SIZE, VALIDATION_SIZE y TEST_SIZE como números flotantes que indican la proporción de datos que se usarán para cada uno de estos grupos.

```
train = []
validate = []
test = []

TRAIN_SIZE=0.8
VALIDATION_SIZE=0.1
TEST_SIZE=0.1
```

Figura 24. Celda de Código N°8.

En la Celda de Código N°9 (Figura 25) se procesa un conjunto de archivos de audio para su uso en un modelo de aprendizaje automático. El procesamiento incluye: la validación de la longitud y la presencia de voz en cada archivo, la adición de ruido de fondo a cada archivo, la generación de espectrogramas para cada archivo y la división de los archivos en conjuntos de entrenamiento, validación y prueba (train, validate, test).

```
def process_files(file_names, label, repeat=1):
    file_names = tf.repeat(file_names, repeat).numpy()
    return [(process_file(file_name), label) for file_name in tqdm(file_names, desc=f"{word} ({label})", leave=False)]

def process_word(word, label, repeat=1):
    file_names = [file_name for file_name in tqdm(get_files(word), desc="Comprobando", leave=False) if is_valid_file(file_name)]
    np.random.shuffle(file_names)
    train_size=int(TRAIN_SIZE*len(file_names))
    validation_size=int(VALIDATION_SIZE*len(file_names))
    test_size=int(TEST_SIZE*len(file_names))
    train.extend(
        process_files(
            file_names[:train_size],
            label,
            repeat=repeat
        )
    )
    validate.extend(
        process_files(
            file_names[train_size:train_size+validation_size],
            label,
            repeat=repeat
        )
    )
    test.extend(
        process_files(
            file_names[train_size+validation_size:],
            label,
            repeat=repeat
        )
    )

for word in tqdm(command_words, desc="Procesando Palabras"):
    if '_' not in word:
        repeat = 40 if word in ('Encender','Apagar') else 20
        process_word(word, command_words.index(word), repeat=repeat)

for word in tqdm(nonsense_words, desc="Procesando Palabras"):
    if '_' not in word:
        process_word(word, command_words.index('_invalid'), repeat=1)

print(len(train), len(test), len(validate))
```

Figura 25. Celda de Código N°9.

En la Celda de Código N°10 (Figura 26) en resumen, se utiliza para procesar archivos de audio que contienen palabras específicas y señales de ruido de fondo. Se divide en tres etapas: primero se revisan y seleccionan los archivos de audio válidos para cada palabra y se asignan etiquetas a cada uno. Luego, se procesan cada archivo de audio seleccionado mediante el cálculo de su espectrograma y se agrupan en conjuntos de entrenamiento, validación y prueba. Finalmente, se procesan los archivos de ruido de fondo generando muestras simuladas con el fin de agregar más datos al conjunto de pruebas.

```
def process_background(file_name, label):
    audio_tensor = tfio.audio.AudioIOTensor(file_name)
    audio = tf.cast(audio_tensor[:], tf.float32)
    audio_length = len(audio)
    samples = []
    for section_start in tqdm(range(0, audio_length-EXPECTED_SAMPLES, 16000), desc=file_name, leave=False):
        section_end = section_start + EXPECTED_SAMPLES
        section = audio[section_start:section_end]
        spectrogram = get_spectrogram(section)
        samples.append((spectrogram, label))

    for section_index in tqdm(range(1000), desc="Palabras simuladas", leave=False):
        section_start = np.random.randint(0, audio_length - EXPECTED_SAMPLES)
        section_end = section_start + EXPECTED_SAMPLES
        section = np.reshape(audio[section_start:section_end], (EXPECTED_SAMPLES))

        result = np.zeros((EXPECTED_SAMPLES))
        voice_length = np.random.randint(MINIMUM_VOICE_LENGTH/2, EXPECTED_SAMPLES)
        voice_start = np.random.randint(0, EXPECTED_SAMPLES - voice_length)
        hamming = np.hamming(voice_length)
        result[voice_start:voice_start+voice_length] = hamming * section[voice_start:voice_start+voice_length]
        spectrogram = get_spectrogram(np.reshape(section, (16000, 1)))
        samples.append((spectrogram, label))


    np.random.shuffle(samples)

    train_size=int(TRAIN_SIZE*len(samples))
    validation_size=int(VALIDATION_SIZE*len(samples))
    test_size=int(TEST_SIZE*len(samples))

    train.extend(samples[:train_size])

    validate.extend(samples[train_size:train_size+validation_size])

    test.extend(samples[train_size+validation_size:])


for file_name in tqdm(get_files('Ruido_de_Fondo'), desc="Procesamiento de Ruido de Fondo"):
    process_background(file_name, command_words.index("_invalid"))

print(len(train), len(test), len(validate))
```

Figura 26. Celda de Código N°10.

La Celda de Código N°11 (Figura 27) se encarga de procesar archivos de audio que contienen problemas de ruido (carpeta "Problemas_de_Ruido"), y los divide en conjuntos de entrenamiento, validación y prueba (train, validate, test). El procesamiento incluye obtener el espectrograma de cada sección de audio, y etiquetarlos como _invalid.

```
def process_problem_noise(file_name, label):
    samples = []
    # Cargar el archivo de audio
    audio_tensor = tfio.audio.AudioIOTensor(file_name)
    audio = tf.cast(audio_tensor[:], tf.float32)
    audio_length = len(audio)
    samples = []
    for section_start in tqdm(range(0, audio_length-EXPECTED_SAMPLES, 800), desc=file_name, leave=False):
        section_end = section_start + EXPECTED_SAMPLES
        section = audio[section_start:section_end]
        # Obtener el espectrograma
        spectrogram = get_spectrogram(section)
        samples.append((spectrogram, label))

    np.random.shuffle(samples)

    train_size=int(TRAIN_SIZE*len(samples))
    validation_size=int(VALIDATION_SIZE*len(samples))
    test_size=int(TEST_SIZE*len(samples))

    train.extend(samples[:train_size])
    validate.extend(samples[train_size:train_size+validation_size])
    test.extend(samples[train_size+validation_size:])


for file_name in tqdm(get_files("Problemas_de_Ruido"), desc="Ruido de Problemas de Procesamiento"):
    process_problem_noise(file_name, command_words.index("_invalid"))
```

Figura 27. Celda de Código N°11.

La Celda de Código N°12 (Figura 28) es para procesar un conjunto de datos de audio y agruparlos en tres conjuntos: entrenamiento, pruebas y validación y para imprimir el número de elementos en cada conjunto.

```
print(len(train), len(test), len(validate))
```

```
305030 38141 38124
```

Figura 28. Celda de Código N°12.

En la Celda de Código N°13 (Figura 29) se encuentra la función “np.random.suffle(train)” la cual se utiliza para mezclar aleatoriamente los elementos de la lista train. Esto para evitar que el algoritmo se ajuste de manera excesiva a un orden específico de los datos de entrenamiento.

```
np.random.shuffle(train)
```

Figura 29. Celda de Código N°13.

La Celda de Código N°14 (Figura 30) está separando las características y las etiquetas para los conjuntos de datos de entrenamiento, validación y prueba, utilizando la función zip() y el operador de desempaquetado (*). X_train, X_validate, y X_test son las características (espectrogramas de audio) mientras que Y_train, Y_validate, y Y_test son las etiquetas (índice de palabras comando o etiqueta inválida).

```
X_train, Y_train = zip(*train)
X_validate, Y_validate = zip(*validate)
X_test, Y_test = zip(*test)
```

Figura 30. Celda de Código N°14.

La Celda de Código N°15 (Figura 31) guarda los datos de entrenamiento, validación y prueba en archivos con extensión .npz utilizando la función np.savez_compressed. los nombres de los archivos son “Entrenamiento_Espectrograma.npz”, “Espectrograma_Validacion.npz”, “Prueba_Espectrograma.npz” respectivamente. Estos archivos contienen los datos de las variables X y Y que son el espectrograma y las etiquetas correspondientes para cada uno de los conjuntos de datos (entrenamiento, validación y prueba)

```
np.savez_compressed(
    "Entrenamiento_Espectrograma.npz",
    X=X_train, Y=Y_train)
print("Datos de Entrenamiento Guardados")
np.savez_compressed(
    "Espectrograma_Validacion.npz",
    X=X_validate, Y=Y_validate)
print("Datos de Validación Guardados")
np.savez_compressed(
    "Prueba_Espectrograma.npz",
    X=X_test, Y=Y_test)
print("Datos de Prueba Guardados")
```

Figura 31. Celda de Código N°15.

La Celda de Código N°16 (Figura 32) obtiene el ancho y la altura de la imagen de los espectrogramas.

```
IMG_WIDTH=X_train[0].shape[0]
IMG_HEIGHT=X_train[0].shape[1]
```

Figura 32. Celda de Código N°16.

En la Celda de Código N°17 (Figura 33) se encuentra la función "plot_images2" la cual es para graficar un arreglo de imágenes, recibe el arreglo de imágenes, el ancho y alto de la imagen, crea una Imagen con 2 filas y 5 columnas.

```
def plot_images2(images_arr, imageWidth, imageHeight):
    fig, axes = plt.subplots(2, 5, figsize=(10, 10))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(np.reshape(img, (imageWidth, imageHeight)))
        ax.axis("off")
    plt.tight_layout()
    plt.show()
```

Figura 33. Celda de Código N°17.

En las Celdas de Código N°18, 19, 20 y 21 (Figura 34) se grafican imágenes de los espectrogramas en formato específico (2 filas y 5 columnas)

```
word_index = command_words.index("Encender")

X_left = np.array(X_train)[np.array(Y_train) == word_index]
plot_images2(X_left[:10], IMG_WIDTH, IMG_HEIGHT)
```

```
word_index = command_words.index("Apagar")

X_left = np.array(X_train)[np.array(Y_train) == word_index]
plot_images2(X_left[:10], IMG_WIDTH, IMG_HEIGHT)
```

```
word_index = command_words.index("Aumentar")

X_left = np.array(X_train)[np.array(Y_train) == word_index]
plot_images2(X_left[:10], IMG_WIDTH, IMG_HEIGHT)
```

```
word_index = command_words.index("Disminuir")

X_left = np.array(X_train)[np.array(Y_train) == word_index]
plot_images2(X_left[:10], IMG_WIDTH, IMG_HEIGHT)
```

Figura 34. Celdas de Código N°18,19,20 y 21.

Los espectrogramas que muestran las celdas anteriores Encender (Figura 35) Apagar (Figura 36) Aumentar (Figura 37) y Disminuir (Figura 38) son los siguientes:

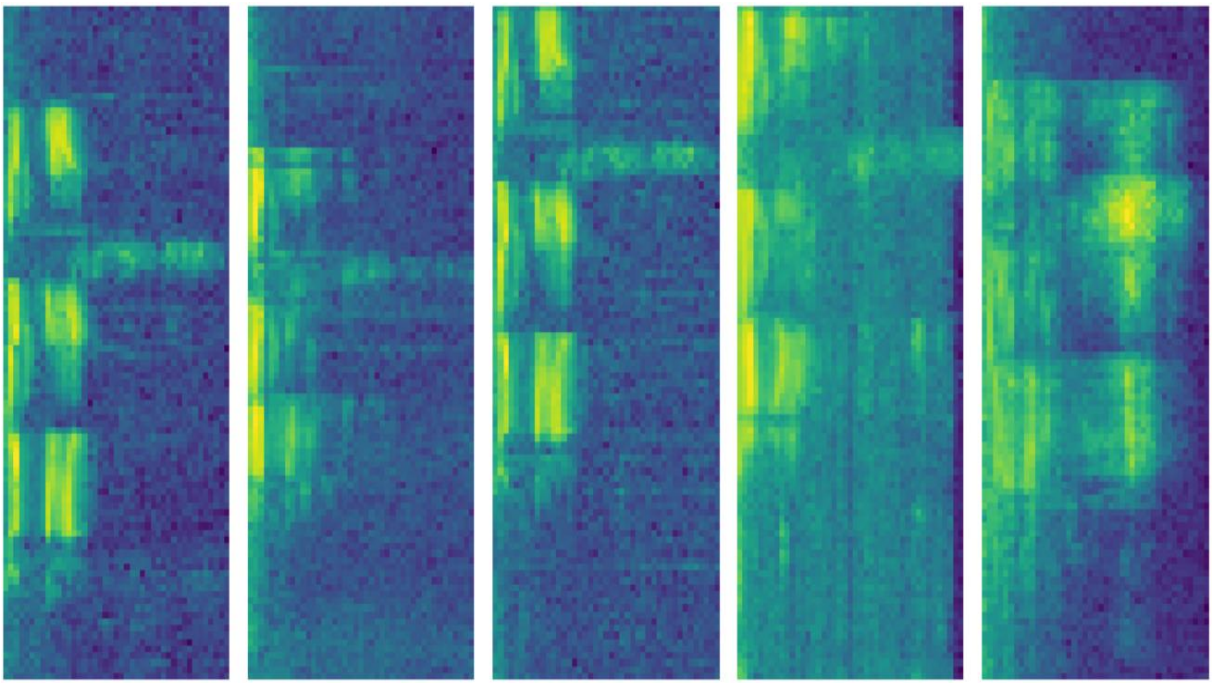


Figura 35. Espectrograma del comando Encender.

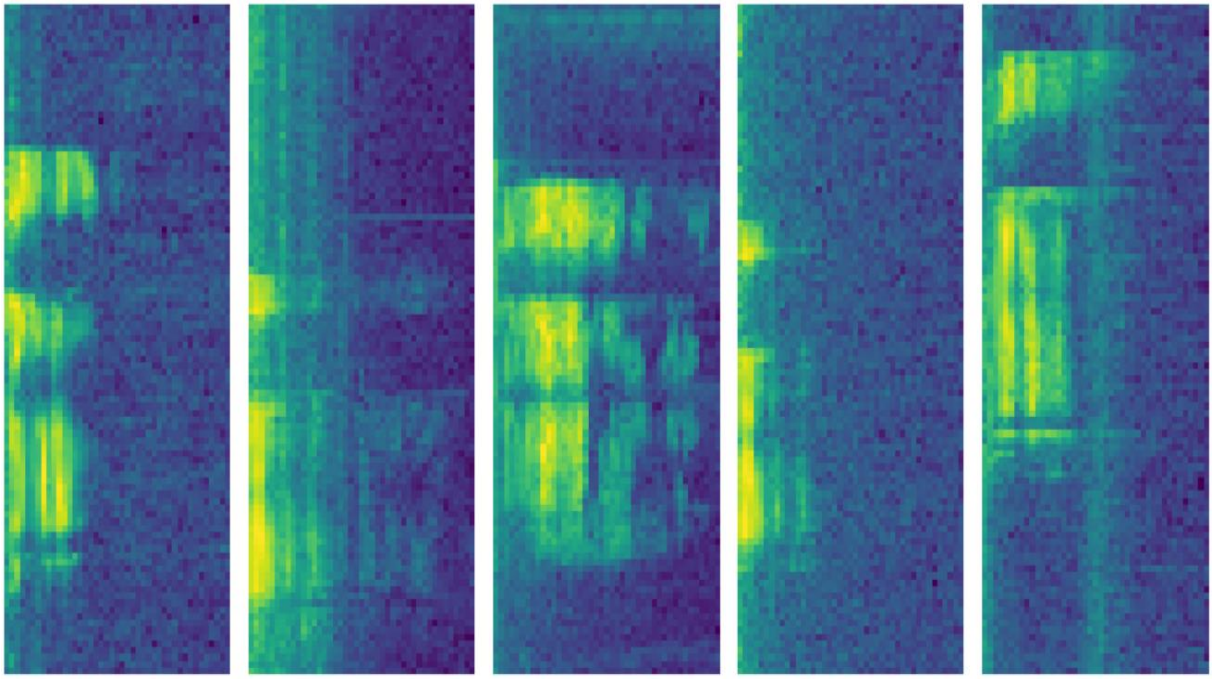


Figura 36. Espectrograma del comando Apagar.

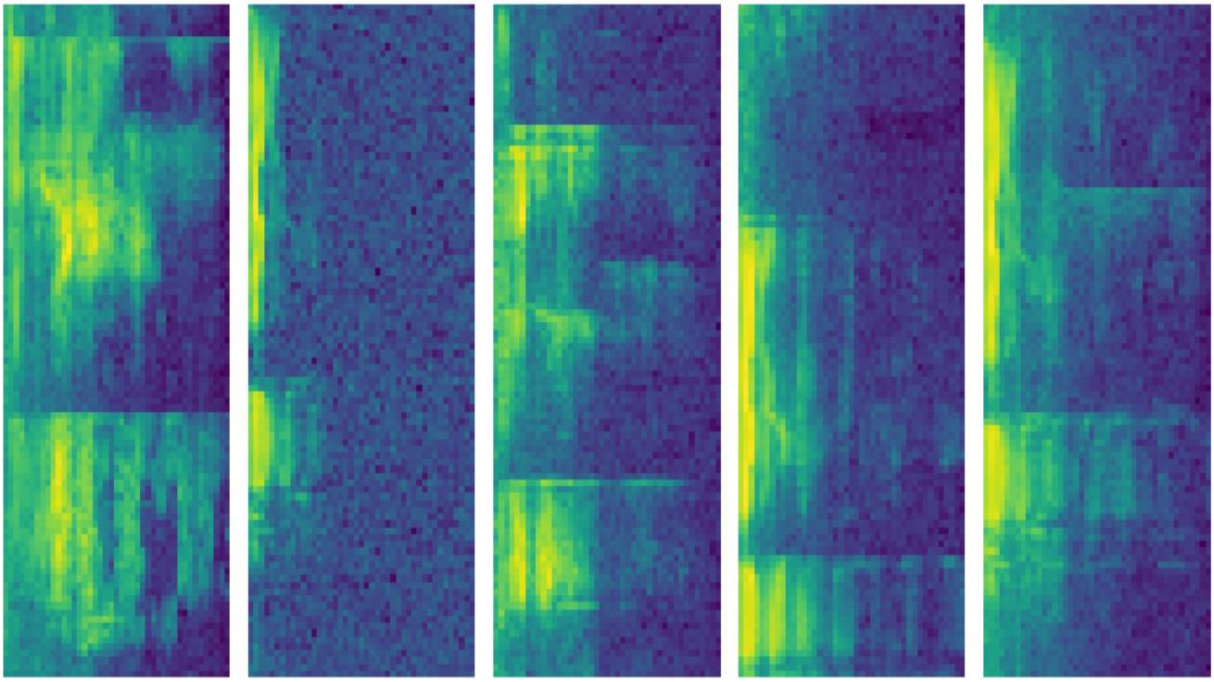


Figura 37. Espectrograma del comando Aumentar.

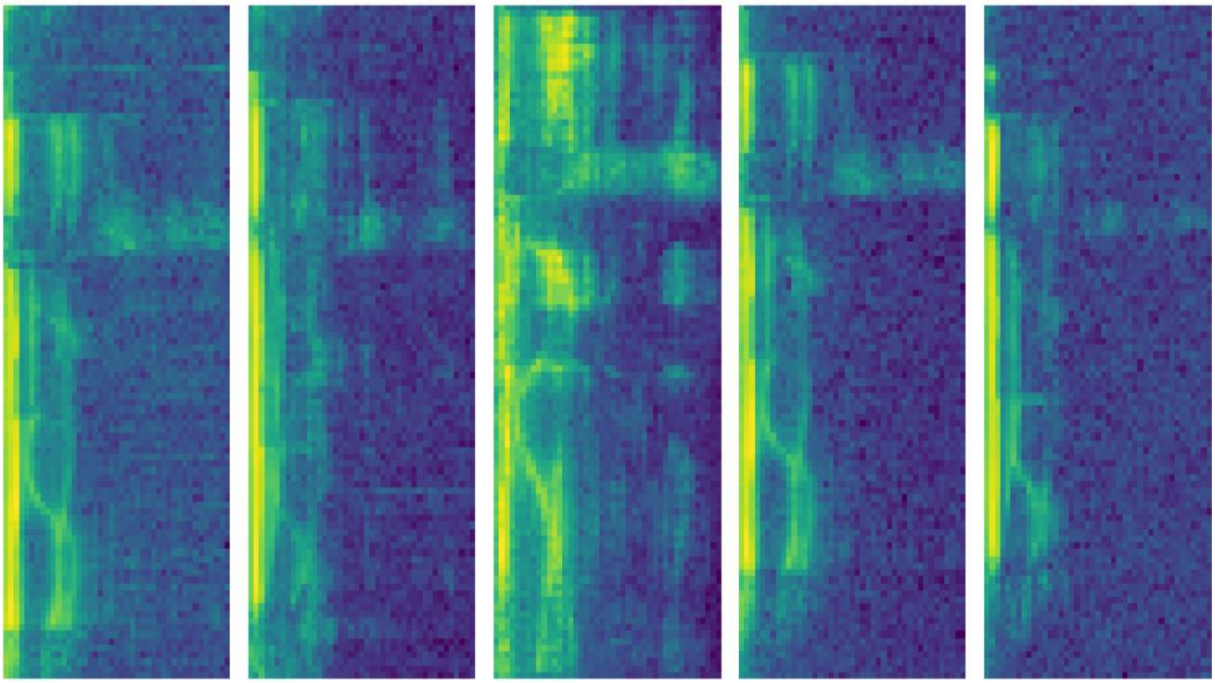


Figura 38. Espectrograma del comando Disminuir.

3.4 Entrenamiento de Palabras.

Ahora que los archivos de audio han sido procesados y almacenados en el formato de espectrograma, es hora de entrenar un modelo para poder reconocer las palabras. Para esto, se crea un nuevo archivo de programación llamado "EntrenandoTodasLasPalabras.ipynb" a continuación se describirá cada una de las celdas de programación.

La Celda de Código N°1 (Figura 39) importa varias librerías necesarias para el entrenamiento de los datos, entre ellas: datatime, temsorflow, numpy, keras, etc.

```
import datetime
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D, BatchNormalization
from tensorflow.data import Dataset
import matplotlib.pyplot as plt
```

Figura 39. Celda de Código N°1.

La Celda de Código N°2 (Figura 40) carga la librería tensorboard para poder ser utilizada en este Notebook

```
%load_ext tensorboard
%reload_ext tensorboard
```

Figura 40. Celda de Código N°2.

En la Celda de Código N°3 (Figura 41) se borra una carpeta llamada “logs”, esto para que los registros antiguos no interfieran con los nuevos.

```
!rm -rf ./logs/
```

Figura 41. Celda de Código N°3.

En la Celda de Código N°4 (Figura 42) se inicia tensorboard.

```
%tensorboard --logdir logs
```

Figura 42. Celda de Código N°4.

La Celda de Código N°5 (Figura 43) te muestra el número de GPU (Unidad de Procesamiento Grafico) disponibles en el sistema.

```
print("Num GPUs Available: ", len(tf.config.experimental.list_physical_devices('GPU')))
```

```
Num GPUs Available:  0
```

Figura 43. Celda de Código N°5.

En la Figura 43 se muestra que el pc no cuenta con GPU (lo que hace que el entrenamiento sea más rápido), por lo que se utilizara CPU (Unidad Central de Procesamiento, cuando se utiliza el CPU el entrenamiento tarda más tiempo).

La Celda de Código N°6 (Figura 44) llama de nuevo a la lista de palabras que se convertirán en comandos.

```
command_words = [
    'Apagar',
    'Aumentar',
    'Disminuir',
    'Encender',
    '_invalid',
]
```

Figura 44. Celda de Código N°6.

La Celda de Código N°7 (Figura 45) carga los archivos de entrenamiento, validación y prueba, que se guardaron anteriormente en los archivos comprimidos con la extensión npz. Los archivos contienen los espectrogramas y las etiquetas correspondientes para cada conjunto de datos.

```
training_spectrogram = np.load('Entrenamiento_Espectrograma.npz')
validation_spectrogram = np.load('Espectrograma_Validacion.npz')
test_spectrogram = np.load('Prueba_Espectrograma.npz')
```

Figura 45. Celda de Código N°7.

En la Celda de Código N°8 (Figura 46) se cargan los datos de entrenamiento, validación y prueba que se guardaron anteriormente en archivos npz en variables específicas. X_train, X_validate y X_test son arreglos de numpy que contienen los espectrogramas de audio. Y_train_cats, Y_validate_cats y Y_test_cats son arreglos de numpy que contienen las etiquetas de cada espectrograma. IMG_WIDTH y IMG_HEIGHT son variables que contienen el ancho y alto de los espectrogramas cargados.

```
X_train = training_spectrogram['X']
Y_train_cats = training_spectrogram['Y']
X_validate = validation_spectrogram['X']
Y_validate_cats = validation_spectrogram['Y']
X_test = test_spectrogram['X']
Y_test_cats = test_spectrogram['Y']

IMG_WIDTH=X_train[0].shape[0]
IMG_HEIGHT=X_train[0].shape[1]
```

Figura 46. Celda de Código N°8.

La Celda de Código N°9 (Figura 47) crea un histograma, en donde se muestra la distribución de los datos como se ve en la Figura 48, esto para verificar que no exista un desequilibrio en el número de ejemplos.

```
plt.hist(Y_train_cats, bins=range(0,len(command_words)+1), align='left')
```

Figura 47. Celda de Código N°9.

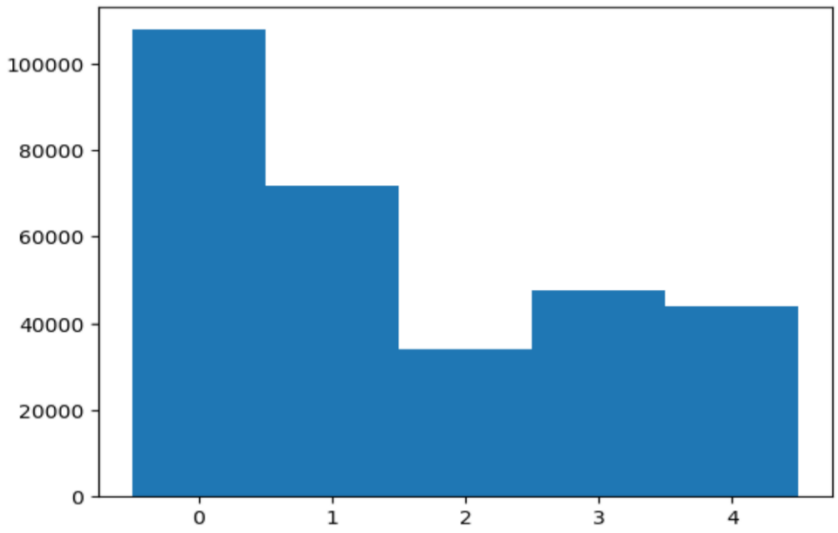


Figura 48. Distribución de Palabras Comando

La Celda de Código N°10 (Figura 49) imprime los valores únicos y sus frecuencias en el conjunto de entrenamiento. Los valores únicos son los índices de las palabras de comando y las frecuencias son el número de veces que ese índice aparece en el conjunto de entrenamiento. Es decir, se está contando cuantas veces aparece cada comando en el conjunto de entrenamiento.

```
unique, counts = np.unique(Y_train_cats, return_counts=True)
print(unique, counts)
dict(zip([command_words[i] for i in unique], counts))
```

```
[0 1 2 3 4] [107840  71680  34080  47520  43910]

{'Apagar': 107840,
 'Aumentar': 71680,
 'Disminuir': 34080,
 'Encender': 47520,
 '_invalid': 43910}
```

Figura 49. Celda de Código N°10.

En la Celda de Código N°11 (Figura 50) está convirtiendo las etiquetas de los conjuntos de entrenamiento, validación y prueba en una representación "one hot" con el uso de la función one_hot de tensorflow.

```
Y_train = tf.one_hot(Y_train_cats, len(command_words))
Y_validate = tf.one_hot(Y_validate_cats, len(command_words))
Y_test = tf.one_hot(Y_test_cats, len(command_words))
```

Figura 50. Celda de Código N°11.

La Celda de Código N°12 (Figura 51) crea los conjuntos de datos para el entrenamiento.

```
batch_size = 32

train_dataset = Dataset.from_tensor_slices(
    (X_train, Y_train)
).repeat(
    count=-1
).shuffle(
    len(X_train)
).batch(
    batch_size
)

validation_dataset = Dataset.from_tensor_slices((X_validate, Y_validate)).batch(X_validate.shape[0]//10)

test_dataset = Dataset.from_tensor_slices((X_test, Y_test)).batch(len(X_test))
```

Figura 51. Celda de Código N°12.

La Celda de Código N°13 (Figura 52) crea un modelo de la red neuronal secuencial en Keras, utilizando varias capas de Convolución 2D.

```
model = Sequential([
    Conv2D(4, 3,
           padding='same',
           activation='relu',
           kernel_regularizer=regularizers.l2(0.001),
           name='conv_layer1',
           input_shape=(IMG_WIDTH, IMG_HEIGHT, 1)),
    MaxPooling2D(name='max_pooling1', pool_size=(2,2)),
    Conv2D(4, 3,
           padding='same',
           activation='relu',
           kernel_regularizer=regularizers.l2(0.001),
           name='conv_layer2'),
    MaxPooling2D(name='max_pooling3', pool_size=(2,2)),
    Flatten(),
    Dropout(0.1),
    Dense(
        80,
        activation='relu',
        kernel_regularizer=regularizers.l2(0.001),
        name='hidden_layer1'
    ),
    Dropout(0.1),
    Dense(
        len(command_words),
        activation='softmax',
        kernel_regularizer=regularizers.l2(0.001),
        name='output'
    )
])
model.summary()
```

Figura 52. Celda de Código N°13.

En la Celda de Código N°14 (Figura 53) se compila el modelo utilizando el optimizador “adam” y los “epochs” son el número de veces que el modelo se va a entrenar con el conjunto de entrenamiento.

```
epochs=10

model.compile(optimizer='adam',
              loss=tf.keras.losses.CategoricalCrossentropy(),
              metrics=['accuracy'])
```

Figura 53. Celda de Código N°14.

La Celda de Código N°15 (Figura 54) establece un directorio de registro para el tensorboard.

```
log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
```

Figura 54. Celda de Código N°15.

En la Celda de Código N°16 (Figura 55) entrena el modelo con los datos de entrenamiento y validación utilizando el optimizador “adam”.

```
model_checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath="checkpoint.model",
    monitor='val_accuracy',
    mode='max',
    save_best_only=True)

history = model.fit(
    train_dataset,
    steps_per_epoch=len(X_train) // batch_size,
    epochs=epochs,
    validation_data=validation_dataset,
    validation_steps=10,
    callbacks=[tensorboard_callback, model_checkpoint_callback]
)
```

Figura 55. Celda de Código N°16.

La Celda de Código N°17 (Figura 56) Guarda el modelo en un archivo de nombre “trained.model”

```
model.save("trained.model")
```

Figura 56. Celda de Código N°17.

La Celda de Código N°18 (Figura 57) carga el modelo anterior previamente entrenado.

```
model2 =  keras.models.load_model("trained.model")
```

Figura 57. Celda de Código N°18.

La Celda de Código N°19 (Figura 58) evalúa el modelo cargado “model2” devolviendo una lista de resultados, que incluyen perdidas y la precisión del modelo.

```
results = model2.evaluate(X_test, tf.cast(Y_test, tf.float32), batch_size=128)
```

Figura 58. Celda de Código N°19.

La Celda de Código N°20 (Figura 59) evalúa el desempeño del modelo y hace predicciones.

```
predictions = model2.predict(X_test, 128)
```

Figura 59. Celda de Código N°20.

En la Celda de Código N°21 (Figura 60) se crea una matriz de confusión en matplotlib. La matriz de confusión es una herramienta para visualizar el rendimiento de un modelo de clasificación.

```
import itertools

def plot_confusion_matrix(cm, class_names):
    """
  Returns a matplotlib figure containing the plotted confusion matrix.

  Args:
    cm (array, shape = [n, n]): a confusion matrix of integer classes
    class_names (array, shape = [n]): String names of the integer classes
  """
    cm = cm.numpy()
    cm = np.around(cm.astype("float") / cm.sum(axis=1)[:, np.newaxis], decimals=2)

    figure = plt.figure(figsize=(8, 8))
    plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
    plt.title("Matriz de confusión")
    plt.colorbar()
    tick_marks = np.arange(len(class_names))
    plt.xticks(tick_marks, class_names, rotation=45)
    plt.yticks(tick_marks, class_names)

    threshold = cm.max() / 2.0
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        color = "white" if cm[i, j] > threshold else "black"
        plt.text(j, i, cm[i, j], horizontalalignment="center", color=color)

    plt.tight_layout()
    plt.ylabel("Etiqueta verdadera")
    plt.xlabel("Etiqueta predicha")
    plt.show()
```

Figura 60. Celda de Código N°21.

La Celda de Código N°22 (Figura 61) es utilizada para poder graficar y visualizar la matriz de confusión.

```
cm = tf.math.confusion_matrix(
    labels=tf.argmax(Y_test, 1), predictions=tf.argmax(predictions, 1)
)

plot_confusion_matrix(cm, command_words)
```

Figura 61. Celda de Código N°22.

En la Celda de Código N°23 (Figura 62) se está creando un nuevo conjunto de entrenamiento con los datos de entrenamiento, validación y prueba existentes, se mezcla y se divide en lotes de tamaño 30. Este conjunto de entrenamiento completo se utilizará para entrenar otro modelo.

```
batch_size = 30
complete_train_X = np.concatenate((X_train, X_validate, X_test))
complete_train_Y = np.concatenate((Y_train, Y_validate, Y_test))

complete_train_dataset = Dataset.from_tensor_slices((complete_train_X, complete_train_Y)).repeat(count=-1).shuffle(len(complete_train_X)).batch(batch_size)
```

Figura 62. Celda de Código N°23.

En la Celda de Código N°24 (Figura 63) se entrena nuevamente el modelo durante 5 épocas.

```
history = model2.fit(
    complete_train_dataset,
    steps_per_epoch=len(complete_train_X) // batch_size,
    epochs=5
)
```

Figura 63. Celda de Código N°24.

La Celda de Código N°25 (Figura 64) Guarda el modelo en un archivo de nombre “fully_trained.model”

```
model2.save("fully_trained.model")
```

Figura 64. Celda de Código N°25.

La Celda de Código N°26 (Figura 65) evalúa el modelo entrenado (model2)

```
results = model2.evaluate(complete_train_X, tf.cast(complete_train_Y, tf.float32), batch_size=128)
```

```
2979/2979 [==============================] - 701s 235ms/step - loss: 0.0535 - a
ccuracy: 0.9979
```

Figura 65. Celda de Código N°26.

La Celda de Código N°27 (Figura 66) mide la precisión y el rendimiento del modelo en los datos de entrenamiento.

```
predictions = model2.predict(complete_train_X, 128)
```

```
2979/2979 [==============================] - 708s 237ms/step
```

Figura 66. Celda de Código N°27.

La Celda de Código N°28 (Figura 67) muestra la matriz de confusión. Como se observa en la Figura 68 se ve gráficamente esta matriz, esta es útil para evaluar el rendimiento del modelo y ver por dónde está cometiendo errores. También se observa que en la diagonal de la matriz de confusión solo se ven 1, esto significa que el modelo está haciendo una gran cantidad de predicciones correctas, por lo que se puede decir que el modelo ya se encuentra entrenado.

```
cm = tf.math.confusion_matrix(
    labels=tf.argmax(complete_train_Y, 1), predictions=tf.argmax(predictions, 1)
)

plot_confusion_matrix(cm, command_words)
```

Figura 67. Celda de Código N°28.

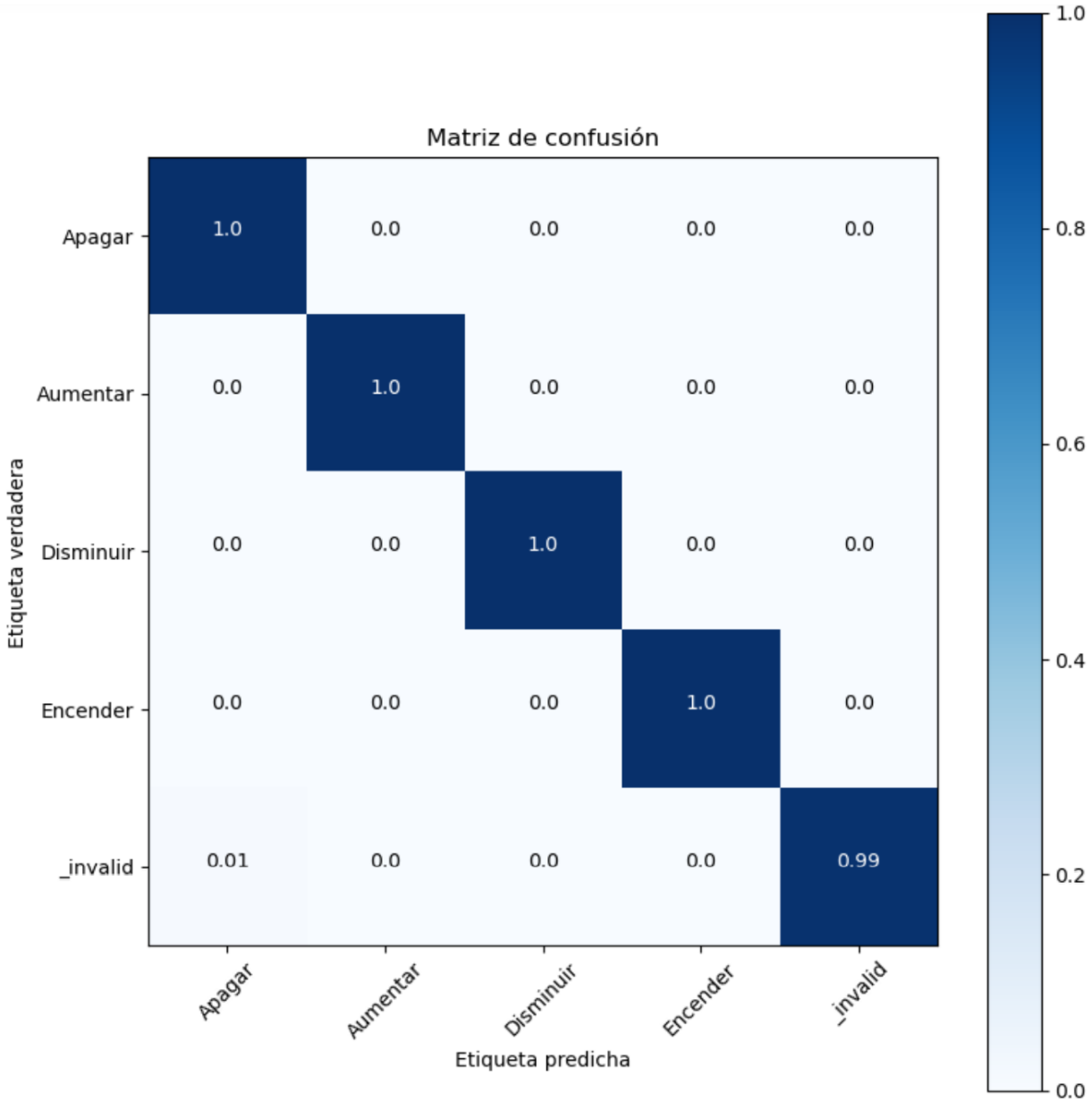


Figura 68. Matriz de Confusión.

3.5 Conversión del Modelo Entrenado a un archivo compilado en C++.

El modelo ha sido entrenado y está funcionando correctamente, pero para poder usarlo en un dispositivo ESP32, es necesario convertirlo a un formato de archivo .cc. Esto se debe a que el ESP32 utiliza un microcontrolador que no es compatible con el lenguaje de programación Python, por lo tanto, se debe convertir el modelo a un archivo compilado en C++ (formato .cc) para que el ESP32 pueda entender y utilizarlo. Para eso se creará otro archivo de nombre “ConvertirModelo” y se explicará que hace cada celda de código.

La Celda de Código N°1 (Figura 69) importa las librerías que se utilizaran como lo son: TensorFlow y numpy

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

Figura 69. Celda de Código N°1.

En la Celda de Código N°2 (Figura 70) se cargan tres archivos de datos de entrenamiento.

```
training_spectrogram = np.load('Entrenamiento_Espectrograma.npz')
validation_spectrogram = np.load('Espectrograma_Validacion.npz')
test_spectrogram = np.load('Prueba_Espectrograma.npz')

X_train = training_spectrogram['X']
X_validate = validation_spectrogram['X']
X_test = test_spectrogram['X']

complete_train_X = np.concatenate((X_train, X_validate, X_test))
```

Figura 70. Celda de Código N°2.

La Celda de Código N°3 (Figura 71) utiliza la clase TFLiteConverter de TensorFlow Lite para convertir el modelo guardado de nombre “fully_trained.model” a un modelo TFLite llamado “converted_model.tflite”

```
converter2 = tf.lite.TFLiteConverter.from_saved_model("fully_trained.model")
converter2.optimizations = [tf.lite.Optimize.DEFAULT]
def representative_dataset_gen():
    for i in range(0, len(complete_train_X), 100):
        yield [complete_train_X[i:i+100]]
converter2.representative_dataset = representative_dataset_gen
converter2.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
tflite_quant_model = converter2.convert()
open("converted_model.tflite", "wb").write(tflite_quant_model)
```

Figura 71. Celda de Código N°3.

La línea de código N°4 (Figura 72) convierte el archivo "converted_model.tflite" en un archivo .cc llamado "model.cc" que puede ser utilizado en el ESP32. Es importante mencionar que esta línea de código debe ser ejecutada en la terminal de la computadora en lugar de en el Jupyter Notebook, y en este caso se utiliza la terminal de Ubuntu.

```
xxd -i converted_model.tflite > model.cc
```

Figura 72. Línea de Código N°4.

Para la programación en la parte de Preparación de los datos de entrada mediante programación, entrenamiento de palabras y conversión del modelo entrenado a un archivo compilado en C++ se basó en el código de atomic14 [23].

Capítulo 4.
Diseño y Desarrollo del Sistema Electrónico.

La siguiente tarea es diseñar y desarrollar el Sistema Electrónico para el reconocimiento de voz en un circuito. Para ello se requieren los siguientes dispositivos: el micrófono INMP441, el puente H L298N, el motor DC y el ESP32. Es importante conocer las especificaciones de estos dispositivos para poder armar correctamente el circuito.

4.1 INMP441.

El INMP441 (Figura 73) es un micrófono omnidireccional de baja potencia con un alto rendimiento, es ideal para las aplicaciones de reconocimiento de voz y se usa en este circuito ya que es necesario capturar el sonido de la voz para que el modelo de inteligencia artificial pueda reconocerlo y tomar una acción. Además, el INMP441 es compatible con la comunicación I^2S, lo que facilita su conexión con el ESP32, ya que este microcontrolador también cuenta con un periférico de I^2S, lo que permite una comunicación directa entre los dos dispositivos, sin necesidad de utilizar un ADC (Conversor Analógico-Digital) externo.

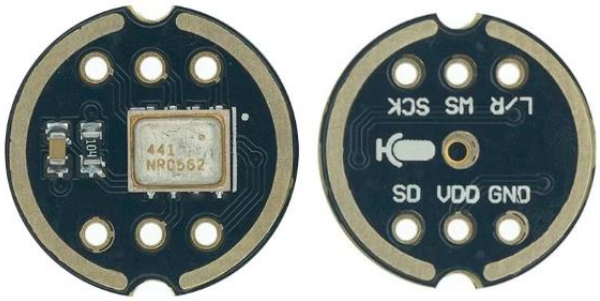


Figura 73. Micrófono INMP441.

4.1.1 Comunicación I^2S.

La comunicación I^2S (Inter-IC Sound) es un protocolo de comunicación desarrollado específicamente para el traslado de datos de audio entre dispositivos electrónicos, tiene características específicas para el manejo de datos de audio como la alta velocidad de transferencia de datos y la capacidad de transmitir datos de audio simultáneamente por varios canales.

- SCK: Serial-Data Clock para interfaz I^2S
- SD: Salida de datos en serie para la interfaz I^2S.
- WS: Selección de la palabra de datos seriales para la interfaz I^2S
- L/R: Selección de canal izquierdo / Derecho. Cuando su nivel es bajo, la señal de salida estará en el canal izquierdo del marco I^2S, cuando se configura en alto, la salida estará en el canal derecho.
- GND: Tierra
- VDD: Voltaje de alimentación, máximo 3.3 V [20]

4.2 Puente H L298N.

El puente H (Figura 74) se utiliza en este circuito para controlar el motor, ya que permite controlar la velocidad de forma precisa. El L298N es un controlador de motores que utiliza una señal PWM (Modulación por Ancho de Pulsos) para controlar la velocidad del motor.

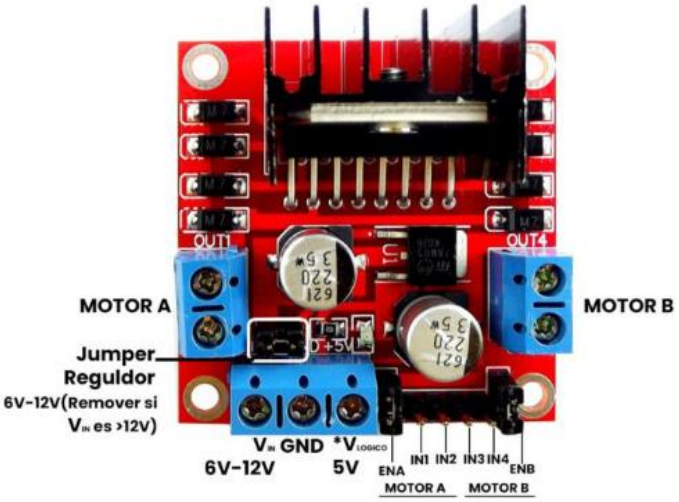


Figura 74. Puente H L298N.

4.3 Motor de DC.

Se utilizará un motor DC (Corriente Continua), se tendrá en cuenta:

- Tensión de operación: debe ser compatible con la tensión de operación del L298N, generalmente entre 5V y 35V.
- Corriente de Operación: debe ser compatible con la corriente continua máxima que el L298N pueda suministrar, generalmente entre 2A y 4A.
- Velocidad: se mide en RPM y en este caso es de 230 RPM
- Torque: es la relación entre el par motor y la corriente que el motor recibe, generalmente en mNm/A o ozin/A
- Voltaje: es la relación entre la velocidad del motor y la tensión que recibe, generalmente en RPM/V

Se utilizará un motor (Figura 75) con las siguientes especificaciones:

- Tipo: Motorreductor
- Dirección del eje: Bidireccional
- Reducción: 48:1
- Voltaje de alimentación: 3V a 6V DC
- Corriente sin carga:
  3V: 80mA
  5V: 90mA
  6V: 120mA
  Corriente máxima de paro (Eje detenido):
  3V: 500mA
  5V: 850mA
  6V: 950mA
- Velocidad sin carga Aproximado: 230 RPM
- Torque máximo: 0.7 kg.cm (Depende del voltaje y la carga que se aplique al eje) [21]

Figura 75. Motorreductor.

4.4 ESP32.

El ESP32 (Figura 76) es un microcontrolador de bajo costo y bajo consumo de energía desarrollado por Espressif Systems. Este módulo tiene una serie de características que lo hacen adecuado para utilizarlo en este trabajo. Algunas de las razones por las que se podría usar el ESP32 en este circuito de reconocimiento de voz son:

- Potencia de procesamiento: El ESP32 tiene dos núcleos de 32 bits, lo que lo hace capaz de manejar tareas complejas de procesamiento de señales, como el procesamiento de señales de audio.
- Conectividad: Cuenta con una variedad de opciones de conectividad, incluyendo Wifi y Bluetooth, lo que lo hace adecuado para aplicaciones que requieran conectividad inalámbrica.
- Memoria: Tiene una gran cantidad de memoria flash y RAM, 448 KByte ROM, 520 KByte SRAM, lo que permite almacenar grandes cantidades de datos y programas.
- Bajo consumo de energía: El ESP32 tiene un bajo consumo de energía, lo que lo hace adecuado para aplicaciones que requieran una larga duración de la batería.
- Comunidad activa: Tiene una gran comunidad activa de desarrolladores, lo que significa que hay una gran cantidad de ejemplos de código y bibliotecas disponibles para ayudar en el desarrollo del proyecto.
- PWM y I^2S: En este proyecto, es esencial utilizar las salidas PWM del ESP32 para controlar la velocidad del motor. Además, el ESP32 también cuenta con una comunicación I^2S que se utilizara para conectar el micrófono INMP441.

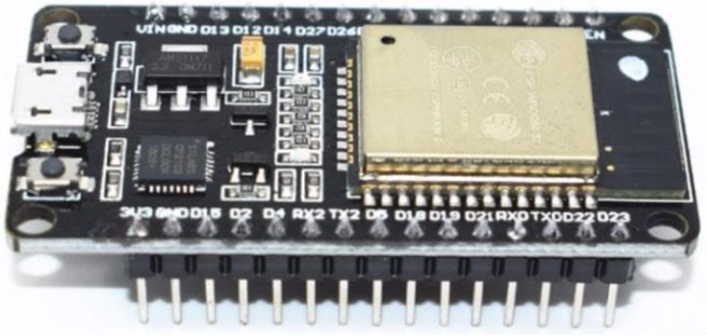


Figura 76. ESP32.

4.4.1 Control de Velocidad con PWM.

Para controlar la velocidad del motor usando el PWM del ESP32, es necesario conocer cómo configurar y controlar un pin de salida PWM. El proceso para hacer esto es el siguiente:

- Paso 1: Decidir el canal PWM_Canal que se va a utilizar [0 - 15]
- Paso 2: Seleccionar el pin GPIO al que enrutar esta señal PWM
- Paso 3: Asignar ese PWM_Canal al pin GPIO del ESP32, usando la siguiente función: ledcAttachPin(GPIO, PWM_Canal)
- Paso 4: Elegir la resolución PWM requerida para el canal seleccionado [1Bit – 16Bits]. Establecer la resolución en 8Bits, da un rango de ciclo de trabajo [0 - 255]. Al configurarlo en 10Bits, da un rango de [0 - 1023] y así sucesivamente.
- Paso 5: Escoger la frecuencia PWM requerida para el canal seleccionado. En este caso se opta por una frecuencia de 1kHz o 1000Hz que es la óptima para el motor elegido.
- Paso 6: Configurar el canal PWM con la frecuencia y resolución seleccionadas utilizando la función: ledcSetup(PWM_Canal, Frecuencia, Resolucion) o bien ledcSetup(PWM_Canal, 1000Hz, 8bits)
- Paso 7: Ahora se puede controlar ese pin PWM cambiando el ciclo de trabajo utilizando la función: ledcWrite(PWM_Canal, DutyCycle [0-255]) [22]

4.5 Circuito y PCB.

Ahora que ya se cuenta con la información de cada uno de los dispositivos que se utilizaran, se procede a diseñar el circuito que conecta los diferentes componentes del proyecto. El diseño del circuito debe conectar el ESP32, el micrófono INMP441, el motor y cualquier otro componente necesario, de acuerdo a la Figura 77.

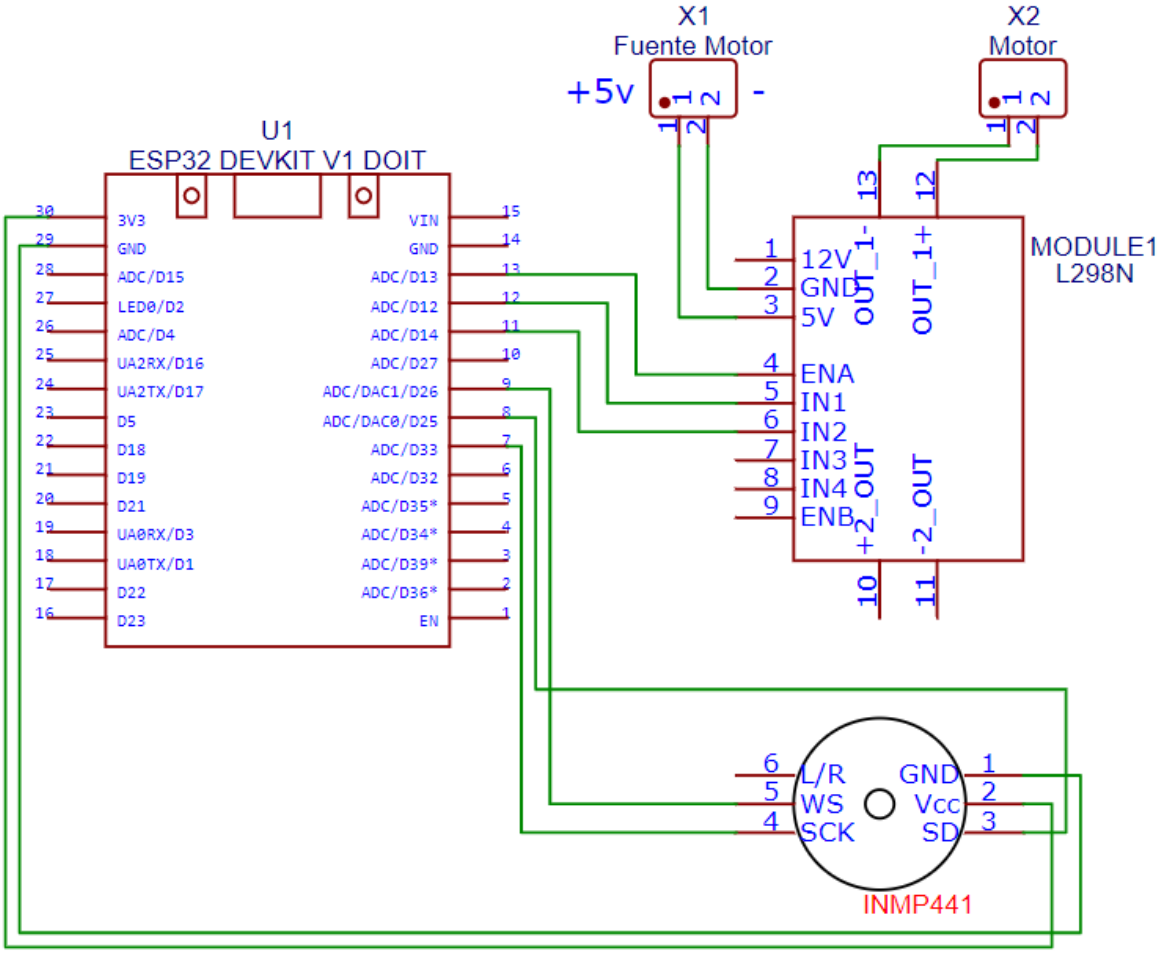


Figura 77. Circuito.

En cuanto al diseño del PCB (circuito impreso) se refiere, se ha diseñado como se observa en las Figuras 78 y 79

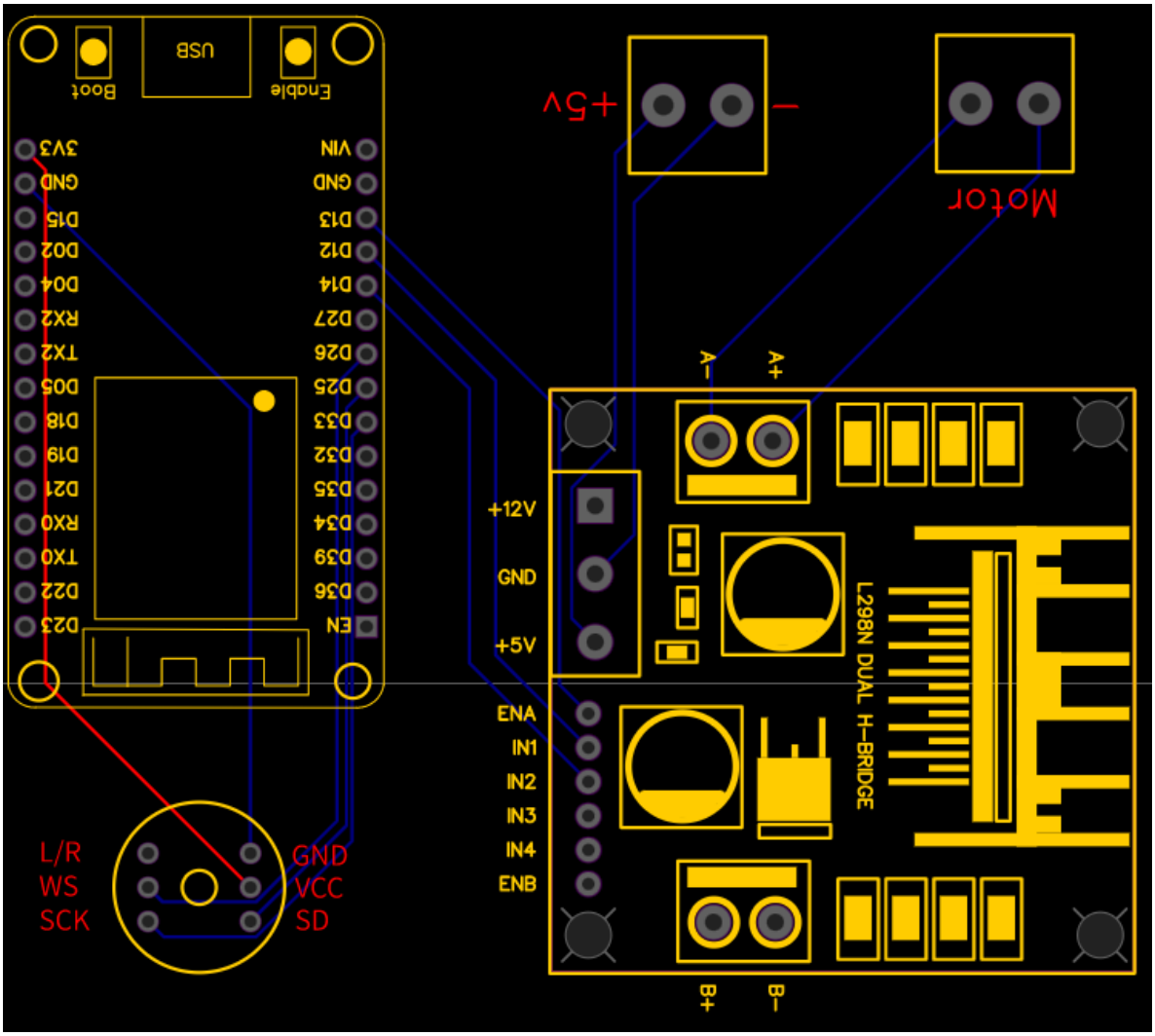


Figura 78. PCB.

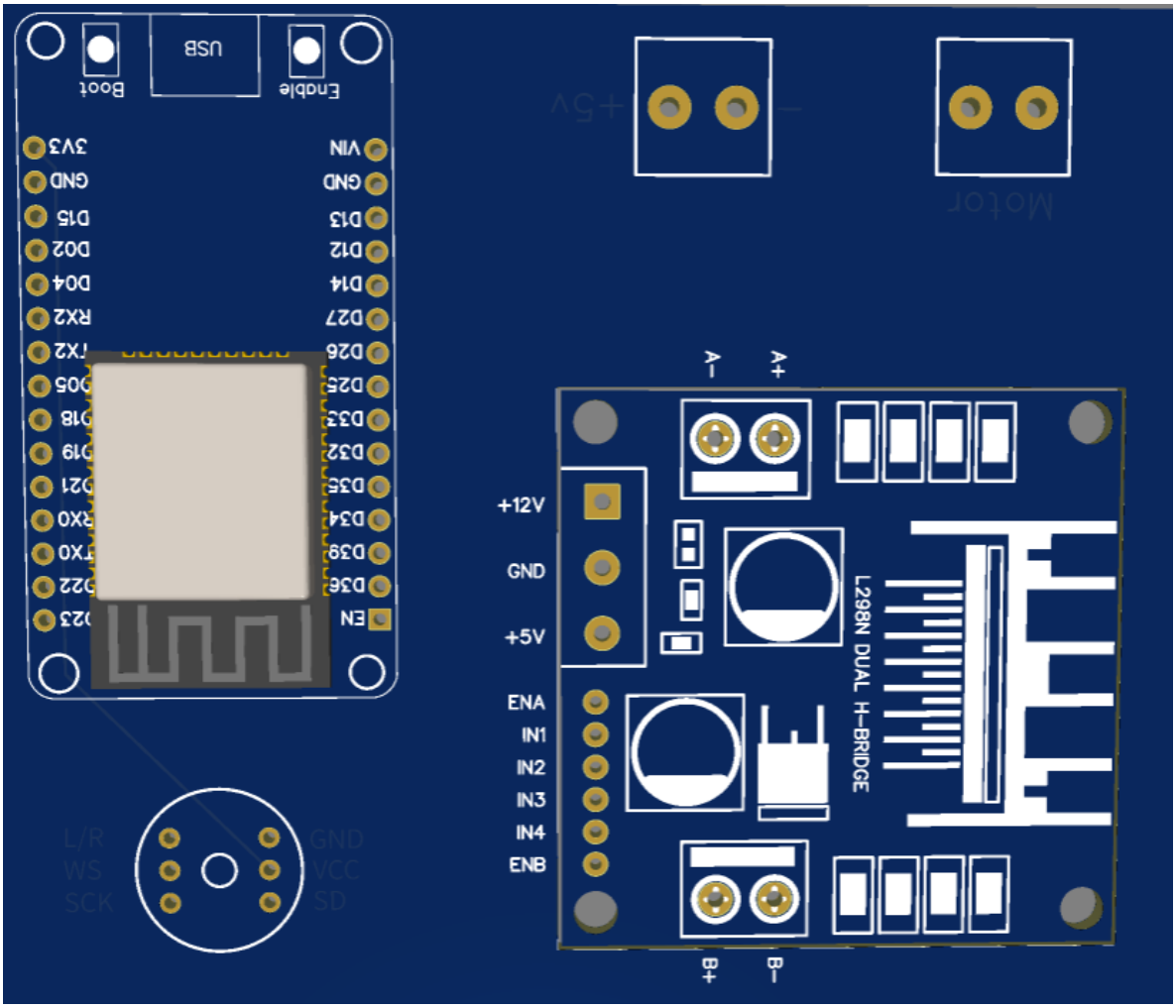


Figura 79. PCB en 3D.

4.6 Costo del Proyecto

Precios 2023 (Tabla 1).

Dispositivo	Precio
INMP441	$54.14
Puente H L298N	$60.00
Motor de DC	$25.00
ESP32	$149.00
PCB	Entre $30 a $1500
Total:	Sin tomar en cuenta la PCB el costo del Proyecto seria de: $288.14

Tabla 1. Precios 2023.

Capítulo 5.
Implementación de Modelo Entrenado al Sistema Electrónico.

5.1 Diagrama del Programa para el control del Motor de DC

El diagrama que se muestra en la Figura 80, describe el proceso que se realiza al recibir los datos de audio que se convierten en comandos para posteriormente obtener una respuesta a cada uno de los comandos recibidos en este caso serían las acciones que se realizaran.

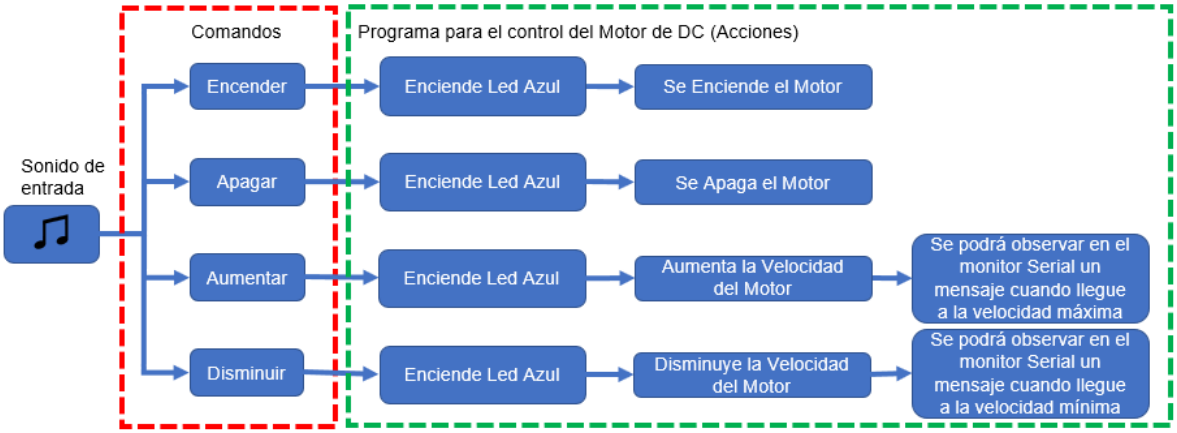


Figura 80. Control del Motor de DC.

5.2 Carga de Código en el ESP32.

Para subir el modelo de reconocimiento de voz entrenado al ESP32, se usará el software Visual Studio Code. En este programa, se instalará una extensión llamada PlatformIO, la cual es utilizada para trabajar con dispositivos como el ESP32. Una vez instalada la extensión, se creará un nuevo proyecto. Este proyecto se llamará con un nombre específico, se seleccionará la tarjeta que se usará, en este caso es el Espressif ESP32 Dev Module y en framework se elegirá Arduino.

Una vez creado el proyecto, se podrá ver una estructura de archivos y carpetas como se observa en la Figura 81.

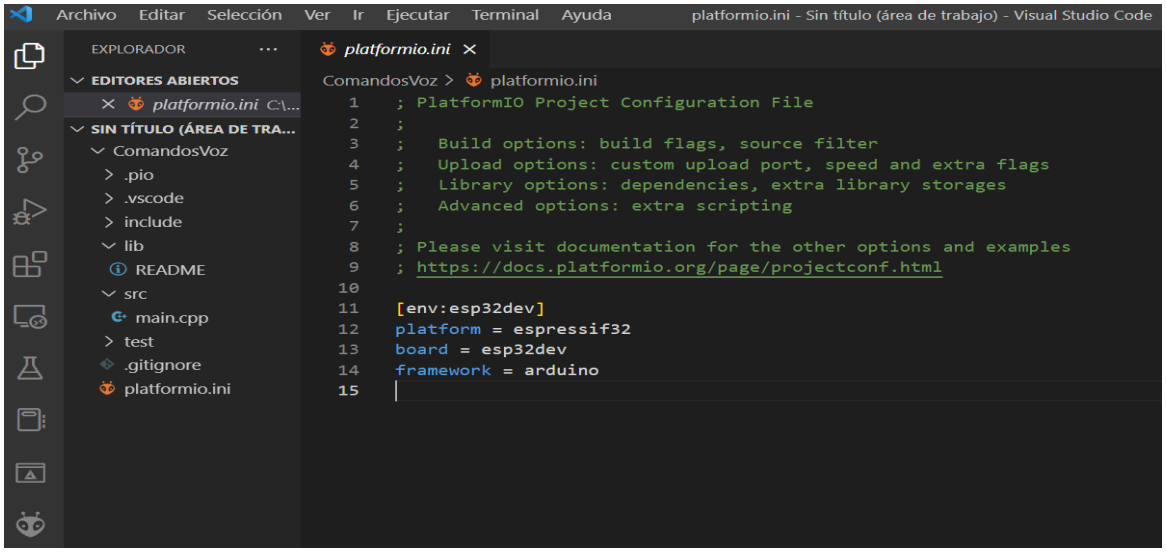


Figura 81. Estructura de Archivos y Carpetas.

En la estructura del proyecto creado se incluirán los archivos necesarios para el proyecto. Estos archivos incluirán el modelo de reconocimiento de voz entrenado, la configuración del proyecto, las librerías necesarias para la comunicación entre el micrófono INMP441 y el ESP32, entre otros. Es importante mencionar que se tendrán que descargar algunas librerías de código abierto adicionales que se encuentran en repositorios de Github para poder realizar la comunicación entre los dispositivos [23].

En la Figura 82 se muestra la estructura de archivos y carpetas del proyecto, y se indica dónde deben colocarse las librerías descargadas desde Github. Estas librerías son las siguientes: audio_input, audio_processor, neuronal_network (en esta carpeta se colocará el modelo de reconocimiento de voz entrenado y se creará un archivo .h para el modelo) y tfmicro. En la carpeta src se encuentran otras librerías que igual se buscan en repositorios, sin embargo, el archivo CommandProcessor.cpp se modificará para crear el programa de control del motor que realizará acciones dependiendo de los comandos de voz reconocidos por el modelo.

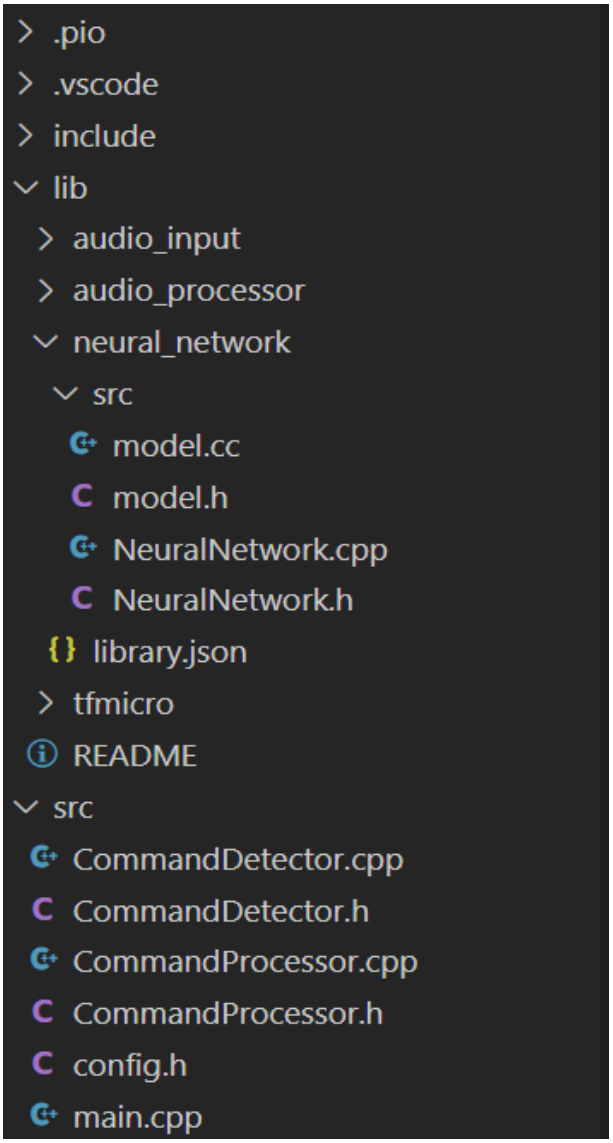


Figura 82. Librerías.

En resumen, las librerías que se utilizan son para:

- audio_input se utiliza para proporcionar acceso a los dispositivos de entrada de audio, como micrófonos, para lectura de datos de audio.
- audio_processor se utiliza para procesar los datos de audio capturados, como el filtrado, la caracterización y la normalización.
- neuronal_network se utiliza para cargar y utilizar el modelo de red neuronal especificado en reconocimiento de voz, y proporciona funciones para realizar inferencias con el modelo y obtener resultados de reconocimiento de voz.
- tfmicro se utiliza para ejecutar modelos de tensorflow en dispositivos con recursos limitados, como el ESP32.
- La clase CommandDetector se utiliza para detectar comandos de voz específicos en la entrada de audio.
- La clase CommandProcessor se utiliza para procesar los comandos de voz detectados y realizar acciones espesificas en respuesta a esos comandos, como el control del motor.

El archivo main es el archivo principal de ejecución del programa, que inicializa las librerías, configura el dispositivo y proporciona un bucle de ejecución principal para el programa.

5.3 Programa para el Control de la señal PWM del Motor de DC.

El programa para el control del motor se crea en el archivo llamado CommandProcessor.cpp y es el siguiente:

```
#include <Arduino.h>
#include "CommandProcessor.h"

const char *words[] = {
    "Encender",
    "Apagar",
    "Aumentar",
    "Disminuir",
    "_nonsense",
};

void commandQueueProcessorTask(void *param)
{
    CommandProcessor *commandProcessor = (CommandProcessor *)param;
    while (true)
    {
        uint16_t commandIndex = 0;
        if (xQueueReceive(commandProcessor->m_command_queue_handle,
&commandIndex, portMAX_DELAY) == pdTRUE)
        {
            commandProcessor->processCommand(commandIndex);
        }
    }
}

int calcDuty(int ms)
{
    return (65536 * ms) / 20000;
}

const int Frecuencia = 1000;
const int CanalMotor = 0;
const int Resolucion = 8;
int VelocidadMotor=255;
```

```
void CommandProcessor::processCommand(uint16_t commandIndex)
{
    switch (commandIndex)
    {
    case 0: // Encender
        digitalWrite(GPIO_NUM_2, HIGH);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        digitalWrite(GPIO_NUM_2, LOW);
        VelocidadMotor=255;
        digitalWrite(GPIO_NUM_12, HIGH);
        digitalWrite(GPIO_NUM_14, LOW);
        break;
    case 1: // Apagar
        digitalWrite(GPIO_NUM_2, HIGH);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        digitalWrite(GPIO_NUM_2, LOW);
        VelocidadMotor=0;
        digitalWrite(GPIO_NUM_12, HIGH);
        digitalWrite(GPIO_NUM_14, LOW);
        break;
    case 2: // Aumentar
        digitalWrite(GPIO_NUM_2, HIGH);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        digitalWrite(GPIO_NUM_2, LOW);
        if(VelocidadMotor>=255){
        VelocidadMotor=255;
        Serial.println("**Llegaste a la Velocidad Máxima, Disminúyela si gustas**");
        }
        else{
            VelocidadMotor=VelocidadMotor+51; //Aquí ponemos los submúltiplos de 255
como 1,3,5,15,17,51,85,255. dependiendo de la exactitud que queramos
        }
        break;
    case 3: // Disminuir
        digitalWrite(GPIO_NUM_2, HIGH);
        vTaskDelay(500 / portTICK_PERIOD_MS);
        digitalWrite(GPIO_NUM_2, LOW);
        if(VelocidadMotor<=0){
        VelocidadMotor=0;
        Serial.println("**Llegaste a la Velocidad Minima, Auméntala si gustas**");
        }
        else{
```

```
            VelocidadMotor=VelocidadMotor-51; //Aqui ponemos los submultiplos de 255
como 1,3,5,15,17,51,85,255. dependiendo de la exactitud que queramos
        }
    }
    ledcWrite(CanalMotor, VelocidadMotor);
}

CommandProcessor::CommandProcessor()
{
    //Configuracion de Motores
    pinMode(GPIO_NUM_12, OUTPUT); // IN1
    pinMode(GPIO_NUM_14, OUTPUT); // IN2

    pinMode(GPIO_NUM_13, OUTPUT); // ENA (PWM)
    ledcSetup(CanalMotor, Frecuencia, Resolucion);
    ledcAttachPin(GPIO_NUM_13, 0);

    // Permite que hasta 5 comandos estén en ejecutándose a la vez
    m_command_queue_handle = xQueueCreate(5, sizeof(uint16_t));
    if (!m_command_queue_handle)
    {
        Serial.println("No se pudo crear la cola de comandos");
    }
    // iniciar la tarea del procesador de comandos
    TaskHandle_t command_queue_task_handle;
    xTaskCreate(commandQueueProcessorTask, "Command Queue Processor", 1024,
this, 1, &command_queue_task_handle);
}

void CommandProcessor::queueCommand(uint16_t commandIndex, float best_score)
{
    // Sin firmar larga ahora = milis ();
    if (commandIndex != 5 && commandIndex != -1)
    {
        Serial.printf("***** %ld Comando Detectado: %s(%f)\n", millis(),
words[commandIndex], best_score);
        if (xQueueSendToBack(m_command_queue_handle, &commandIndex, 0) !=
pdTRUE)
        {
            Serial.println("No más espacio para el comando");
        }
    }
}
```

Capítulo 6.
Resultados Obtenidos.

En este capítulo se muestran los resultados del sistema de control de un motor de corriente directa (DC) utilizando comandos de voz que se aplicara a un torno de alfarero. Se llevaron a cabo pruebas para evaluar el rendimiento del sistema en la detección de los cuatro comandos disponibles (Encender, Aumentar, Disminuir y Apagar) usando la voz de cualquier usuario.

En la Figura 83 se muestra el prototipo del circuito en el que se puede observar la presencia de un LED azul encendido. Este LED se enciende cuando se detecta uno de los cuatro comandos utilizados en el sistema.

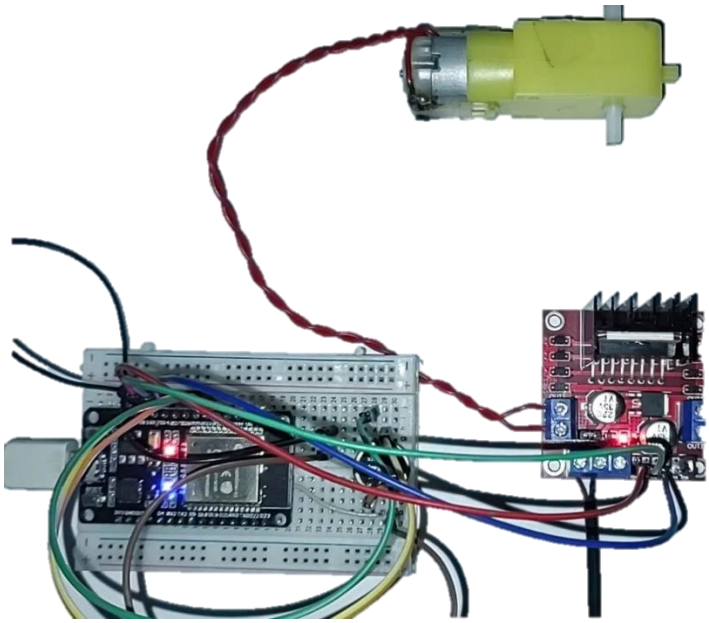


Figura 83. Prototipo de Circuito Electrónico.

Para comprobar que el circuito y el modelo están funcionando correctamente, es necesario usar el "PlatformIO: Serial Monitor". El programa está diseñado para imprimir mensajes en el monitor serial cuando se detecta algún comando. A continuación, se muestran los resultados de las pruebas realizadas en las siguientes imágenes, donde se puede observar que el proyecto funciona correctamente.

Prueba 1 Comando Encender (Figura 84):

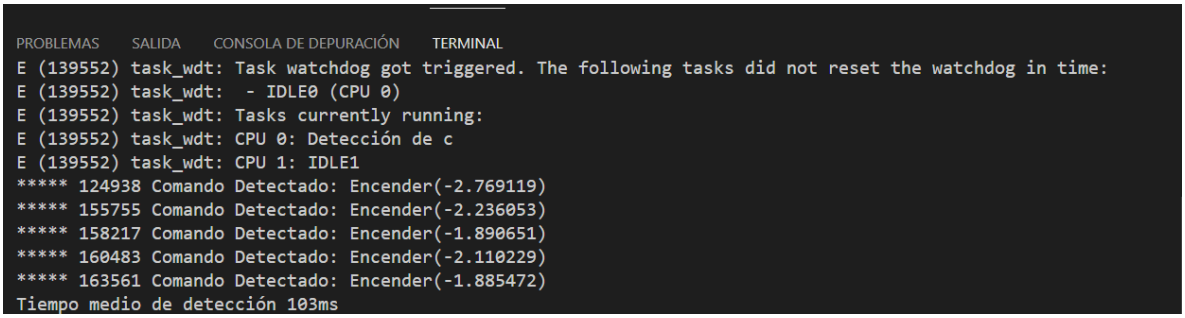


Figura 84. Comando Encender en el Monitor Serial.

Prueba 2 Comando Apagar (Figura 85):

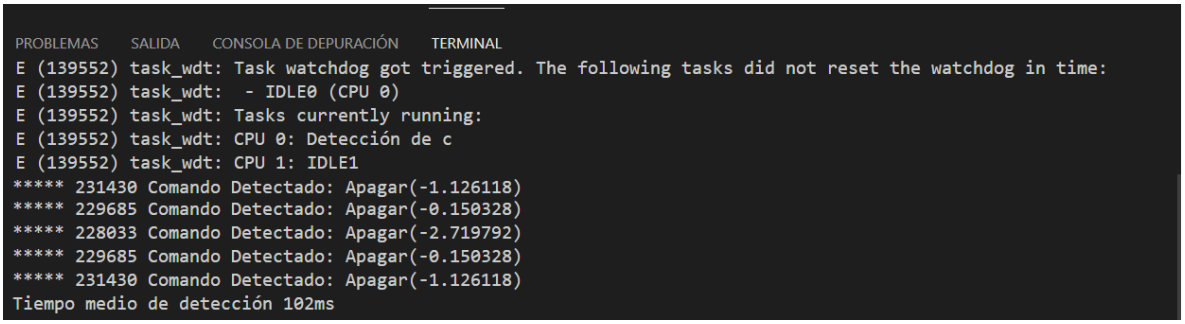


Figura 85. Comando Apagar en el Monitor Serial.

Prueba 3 Comando Aumentar (Figura 86), en esta se agrega un mensaje en el programa para que en el monitor serial se muestre que ha llegado a la velocidad máxima:

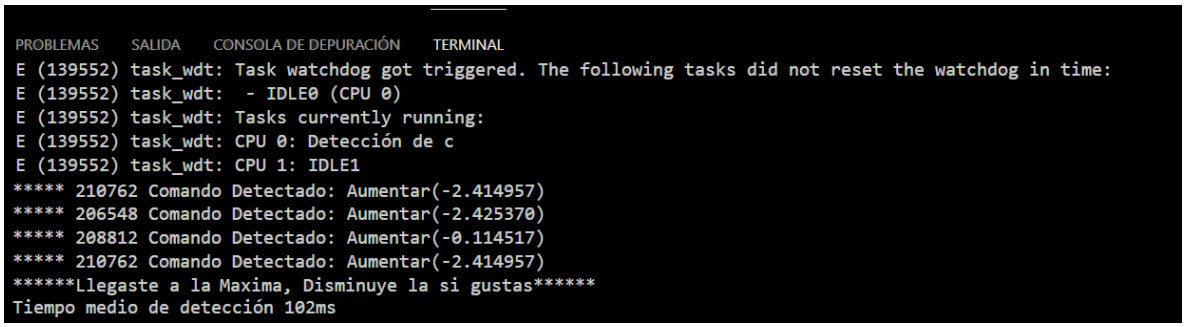


Figura 86. Comando Aumentar en el Monitor Serial.

Prueba 4 Comando Disminuir (Figura 87), en esta se agrega un mensaje en el programa para que en el monitor serial se muestre que ha llegado a la velocidad mínima:

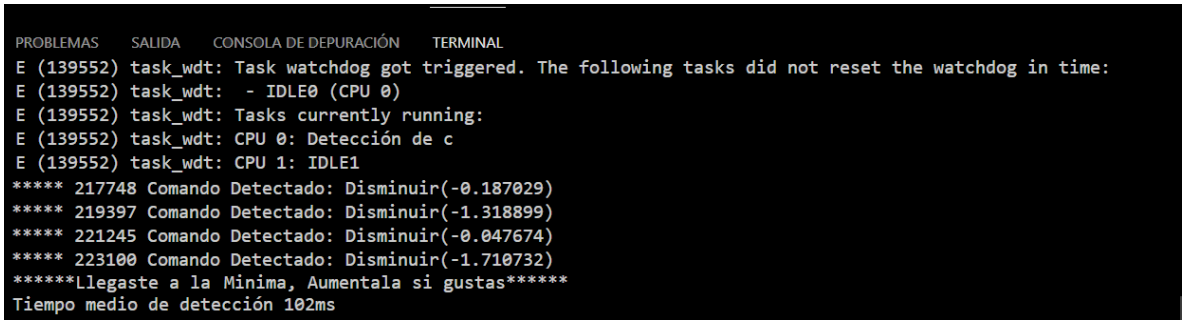


Figura 87. Comando Disminuir en el Monitor Serial.

Como se observa en las imágenes anteriores de las pruebas que se hicieron con el circuito, se puede observar que el modelo funciona correctamente en los cuatro comandos y el circuito reacciona correctamente a través de la voz.

La efectividad del sistema de reconocimiento de voz, tiene como objetivo evaluar la efectividad del reconocimiento de voz desarrollado para reconocer una variedad de comandos, se utilizaron 22 carpetas con sus archivos de audio correspondientes a diferentes comandos, cada uno con una duración de 1 segundo y una frecuencia de muestreo de 16000 Hz. Se evaluó el desempeño del sistema en condiciones de presencia y ausencia de ruido ambiente.

Conjunto de Datos: Se crearon 22 carpetas con audios diferentes, Abajo (1800 archivos de Audio), Adelante (1800 archivos de Audio), Apagar (2290 archivos de Audio), Aprender (1800 archivos de Audio), Arriba (1800 archivos de Audio), Atras (1800 archivos de Audio), Aumentar (2420 archivos de Audio), Casa (1800 archivos de Audio), Derecha (1880 archivos de Audio), Disminuir (3020 archivos de Audio), Encender (3420 archivos de Audio), Gato (1980 archivos de Audio), Izquierda (2160 archivos de Audio), No (1800 archivos de Audio), Pajaro (1800 archivos de Audio), Parar (1800 archivos de Audio), Perro (1800 archivos de Audio), Problemas de Ruido (5 archivos de Audio), Ruido de Fondo (11 archivos de Audio), Seguir (2400

archivos de Audio), Si (1800 archivos de Audio), Vamos (1800 archivos de Audio).

En las Pruebas Experimentales:

1. **Condiciones Sin Ruido de Fondo:**
   Se realizaron pruebas en un entorno controlado sin presencia de ruido de fondo o música.
2. **Condiciones Con Ruido de Fondo:**
   Se realizaron Pruebas en un entorno con ruido de fondo, simulando condiciones más realistas.

Resultados.

Sin Ruido de Fondo: 100% (También se puede Observar en la matriz de confusión)

Con Ruido de Fondo: $\frac{80\%+86\%+92\%+96\%}{4}$ = 88.5 %

Tasa de Precisión por comandos al tener ruido de fondo.

Se utilizará la siguiente Formula:

$$Tasa\ de\ Precisión = \frac{Número\ de\ Identificaciones\ Correctas}{Total\ de\ Pruebas}\ x\ 100\%$$

- Aumentar: $\frac{40}{50} x\ 100\%$ = 80%
- Disminuir: $\frac{43}{50} x\ 100\%$ = 86%
- Encender: $\frac{46}{50} x\ 100\%$ = 92%
- Apagar: $\frac{48}{50} x\ 100\%$ = 96%

Análisis de Errores

- Se observa una disminución en la tasa de precisión en condiciones de ruido ambiente.
- Los comandos “Aumentar” y “Disminuir” muestran un mayor nivel de confusión en presencia de ruido.

El sistema de reconocimiento de voz demostró ser altamente efectivo en condiciones sin ruido de fondo, con una tasa de precisión del 100%. Sin embargo, su desempeño se ve afectado en entornos ruidosos, donde la tasa de precisión disminuye al 88.5%.

Cabe mencionar que en el programa para el control del motor de DC se ha elegido una resolución de 8 bits, lo que significa que el valor máximo que se puede obtener para la señal PWM es 255. Esto se debe a que hay 256 valores posibles desde 0 hasta 255, con 2 elevado a la 8 como la cantidad total de valores posibles. De acuerdo a las especificaciones del motor, el valor PWM de 255 dará una velocidad de 230 RPM. En el programa, se le restará o sumará 51 en 51 para aumentar o disminuir la velocidad del motor. La tabla 2 muestra las velocidades del motor resultantes cuando se aumenta o disminuye la velocidad en incrementos de 51.

Valor máximo para PWM	Velocidad del Motor (RPM)
255	230
204	184
153	138
102	92
51	46
0	0

Tabla 2. Valor Máximo y Velocidad PWM.

Capítulo 7.
Conclusiones.

Con este proyecto se ha demostrado que es posible aplicar la tecnología de inteligencia artificial para mejorar los procesos productivos en la industria alfarera. El uso de un torno de alfarero que funciona mediante comandos de voz puede reducir los problemas físicos que experimentan los alfareros al utilizar los tornos de rueda o pedal, al mismo tiempo que se elimina la necesidad de un motor eléctrico costoso y pesado. Además, este proyecto también puede tener aplicaciones en otros sectores industriales donde se requiera una interacción con máquinas y equipos de una manera más intuitiva y natural.

En conclusión, la implementación exitosa de un prototipo de circuito para controlar un torno de alfarero a través de comandos de voz y basado en tecnología de inteligencia artificial, se presenta como una solución innovadora y práctica a los problemas actuales que enfrenta la industria alfarera. Este proyecto no solo evidencia el potencial de la inteligencia artificial para la mejora de procesos productivos, sino que también puede servir como punto de partida para la creación de nuevas soluciones que optimicen la seguridad, eficiencia y calidad de la producción industrial. La ejecución de este proyecto fue posible gracias a la aplicación efectiva del conocimiento adquirido durante la carrera de Ingeniería en Sistemas Electrónicos Industriales impartida por la UACM, en combinación con el nuevo conocimiento investigado. Se lograron cumplir los objetivos particulares del proyecto, entre los cuales se destaca la creación de un modelo de reconocimiento de voz a través de la inteligencia artificial, para lo cual se generaron los datos de entrenamiento, se entrenó el modelo y se convirtió el modelo ya entrenado a .cc para su integración al ESP32. Posteriormente, se logró el diseño del circuito y su PCB.

Trabajo a Futuro.

En cuanto a algún trabajo a futuro se puede considerar lo siguiente:

- Mejoras en la interfaz de usuario: Se puede trabajar en la mejora de la interacción entre el usuario y el sistema, por ejemplo, la inclusión de nuevos comandos de voz, una mayor precisión en el reconocimiento de voz, y la integración de una pantalla para proporcionar información en tiempo real.
- Integración de sensores: La integración de sensores puede proporcionar información útil sobre el proceso de elaboración de la cerámica, como la velocidad del motor, la temperatura y la presión en tiempo real, lo que permitiría ajustes más precisos y mejoras en la calidad de la producción.
- Aplicación a otros sectores productivos: La tecnología IA y los comandos de voz pueden ser aplicados en otros sectores productivos que requieren control de motores, como la industria automotriz o la industria de la construcción.
- Investigación y desarrollo de nuevas tecnologías: El proyecto de torno de alfarero controlado por comandos de voz y tecnología IA puede ser una base para la investigación y el desarrollo de nuevas tecnologías que mejoren los procesos productivos en la industria en general.
- Aplicaciones para el control de prótesis mediante comandos de voz.

Referencias.

[1] Claro. (2022, Agosto 30). *Importancia actual de la Inteligencia Artificial*. Claro. [En línea]. Disponible en: Importancia actual de la Inteligencia Artificial (claro.com.co) Accedido: Marzo 20, 2023

[2] Gold, B., Morgan, N., & Ellis, D. (2011). *Speech and Audio Signal Processing*. Hoboken, New Jersey: John Wiley & Sons.

[3] *Historia y evolución de la inteligencia artificial*. [En línea]. Disponible en: https://revistasdex.uchile.cl/index.php/bits/article/download/2767/2700 Accedido: Marzo 25, 2023

[4] Frank. (2021, Febrero 03). *Tipos de torno de alfarero y cómo elegir el tuyo*. Club de Cerámica. [En línea]. Disponible en: Tipos de torno de alfarero y cómo elegir el tuyo – Club de Cerámica (clubdeceramica.com) Accedido: Abril 02, 2023

[5] Rouhiainen, L. (2018). *Inteligencia Artificial 101 Cosas que debes saber hoy sobre nuestro futuro*. Barcelona: Alienta

[6] Pertuz Pineda, C. M. (2021). *Aprendizaje Automático y Profundo en Python.* Bogotá, Colombia: Ediciones de la U.

[7] Conde Ortiz, D. (2018). *Inteligencia Artificial con TensorFlow para predicción de comportamientos*. Sevilla, España: Universidad de Sevilla.

[8] Amazon. *¿Qué es una red neuronal?*. Amazon. [En línea]. Disponible en: ¿Qué es una red neuronal? Guía de IA y ML - AWS (amazon.com) Accedido: Abril 04, 2023

[9] D. Calvo. (2017, Julio 13). *Clasificación de redes neuronales artificiales*. diegocalvo. [En línea]. Disponible en: Clasificación de redes neuronales artificiales - Diego Calvo Accedido: Abril 06, 2023

[10] Amazon. *¿Qué es Python?*. Amazon. [En línea]. Disponible en: ¿Qué es Python? | Guía de Python para principiantes de la nube | AWS (amazon.com) Accedido: Abril 08, 2023

[11] J. Larkin. (2022, Junio 15). *¿Qué es TensorFlow y para qué sirve?*. incentro. [En línea]. Disponible en: ¿Qué es TensorFlow y para qué sirve? (incentro.com) Accedido: Abril 10, 2023

[12] L. González. (2021, Junio 01). *¿Qué es TensorFlow? ¿Cómo funciona?*. aprendeia. [En línea]. Disponible en: ¿Qué es TensorFlow? ¿Cómo funciona? - Aprende IA Accedido: Abril 20, 2023

[13] D. Ellis. (2022, Noviembre 03). *What is Anaconda for Python y Why Should You Learn it?*. blog.hubspot. [En línea]. Disponible en: ¿Qué es Anaconda para Python y por qué deberías aprenderlo? (hubspot.com) Accedido: Abril 25, 2023

[14] I. Rondón. (2022, Febrero 04). *¿Qué es Anaconda?*. eiposgrados. [En línea]. Disponible en: ¿Qué es Anaconda? -Escuela Internacional de Posgrados (eiposgrados.com) Accedido: Mayo 05, 2023

[15] *¿Conoces Jupyter Notebook?*. ceupe. [En línea]. Disponible en: ¿Conoces Jupyter Notebook? (ceupe.mx) Accedido: Mayo 10, 2023

[16] F. Flores. (2022, Julio 22). *Visual Studio Code*. openwebinars. [En línea]. Disponible en: Qué es Visual Studio Code y qué ventajas ofrece | OpenWebinars Accedido: Mayo 15, 2023

[17] D. Hinojosa Córdova. (2021, Octubre 21). *PlatformIO: Introducción*. Linkedin. [En línea]. Disponible en: PlatformIO: Introducción (linkedin.com) Accedido: Mayo 17, 2023

[18] L. González. (2022, Mayo 03). *Herramientas de Python para Machine Learning*. aprendeia. [En línea]. Disponible en: Herramientas de Python para Machine Learning - Aprende IA Accedido: Mayo 20, 2023

[19] Transnational College of LEX (2009). *Aventuras con Fourier*. 1ra ed. En español.

[20] InvenSense. *Omnidirectional Microphone with Bottom Port and I^2S Digital Output*. uelectronics. [En línea]. Disponible en: AR3262-INMP441-Modulo-de-Microfono-Omnidireccional-I2S-Datasheet.pdf (uelectronics.com) Accedido: Junio 03, 2023

[21] UNIT ELECTRONICS. *Motores*. uelectronics. [En línea]. Disponible en: Motorreductor Amarillo Para Carrito 48:1 y 120:1 (uelectronics.com) Accedido: Junio 05, 2023

[22] K. Magdy. (2021, Agosto 09). *ESP32 PWM Tutorial & Examples (AnalogWrite) - Arduino*. deepbluembedded. [En línea]. Disponible en: ESP32 PWM Tutorial y ejemplos (AnalogWrite) - Arduino – DeepBlue (deepbluembedded.com) Accedido: Junio 10, 2023

[23] github. *Voice-controlled-robot*. github. [En línea]. Disponible en: GitHub - atomic14/voice-controlled-robot: A voice-controlled robot using the ESP32 and TensorFlow Lite Accedido: febrero 20, 2023