Sintetización de una red neuronal convolucional en microcontrolador para la identificación de gestos mediante el procesamiento de señales electroencefalográficas

Brayan Portugal Código:20142005008
Gabriel Guerrero Código:20142005115

Proyecto de trabajo de grado para optar a título de Ingeniero Electrónico.

Modalidad: Monografía
Director: MSc. Ing. Cesar Andrey Perdomo.

Universidad Distrital Francisco José De Caldas
Facultad de Ingeniería
Ingeniería Electrónica
Bogotá, 2023

Resumen

La capacidad de interpretar señales electroencefalográficas (EEG) para la identificación de gestos representa un avance significativo en la interfaz cerebro-computadora y las aplicaciones de control biométrico. Este trabajo de grado titulado "Sintetización de una Red Neuronal Convolucional en Microcontrolador para la Identificación de Gestos mediante el Procesamiento de Señales Electroencefalográficas". Explora la viabilidad de miniaturizar y optimizar una red neuronal convolucional (CNN) para su ejecución en un microcontrolador de baja potencia, el Arduino Nano 33 BLE, un dispositivo emergente en el campo de la computación embebida y el IoT.

En la Fase 1, se desarrolló un software en Python para la adquisición de datos EEG, utilizando el casco Ultracortex Mark IV y la placa Open BCI, de donde se extrajeron y almacenaron 1600 registros de gestos de 20 sujetos. La Fase 2 se centró en el diseño de una CNN, adaptada para la clasificación de EEG, que se entrenó con el 80 % de los datos y se validó con el 20 % restante. Utilizando Keras y TensorFlow, se logró una precisión del 64.5 % con el mejor modelo de 16000 modelos, destacando las limitaciones y desafíos de procesamiento en hardware con recursos limitados.

La Fase 3 culminó con la implementación de la CNN en el Arduino Nano 33 BLE, utilizando TensorFlow Lite para convertir el modelo de Python a C++. La solución implementada demostró la clasificación de gestos mediante datos preexistentes.

Este trabajo proporciona una comprensión profunda de los desafíos asociados con la integración de tecnologías de inteligencia artificial en microcontroladores y ofrece una metodología para abordar problemas de clasificación en tiempo real en plataformas embebidas, abriendo nuevas posibilidades para aplicaciones de bajo consumo en el campo de la neurotecnología.

Abstract

The ability to interpret electroencephalographic (EEG) signals for gesture identification represents a significant advance in brain-computer interface and biometric control applications. This degree project entitled "Synthesizing a Convolutional Neural Network on Microcontroller for Gesture Identification by Processing Electroencephalographic Signals". It explores the feasibility of miniaturizing and optimizing a convolutional neural network (CNN) for execution on a low-power microcontroller, the Arduino Nano 33 BLE, an emerging device in the field of embedded computing and IoT.

In Phase 1, software was developed in Python for EEG data acquisition, using the Ultracortex Mark IV helmet and Open BCI board, from which 1600 gesture recordings from 20 subjects were extracted and stored. Phase 2 focused on the design of a CNN, adapted for EEG classification, which was trained on 80% of the data and validated on the remaining 20%. Using Keras and TensorFlow, 64.5% accuracy was achieved with the best model out of 16000 models, highlighting the limitations and challenges of processing on resource-limited hardware.

Phase 3 culminated with the implementation of the CNN on the Arduino Nano 33 BLE, using TensorFlow Lite to convert the model from Python to C++. The implemented solution demonstrated gesture classification using pre-existing data.

This work provides an in-depth understanding of the challenges associated with integrating artificial intelligence technologies on microcontrollers and offers a methodology for addressing real-time classification problems on embedded platforms, opening up new possibilities for low-power applications in the field of neurotechnology.

Índice

1 Introducción **8**

2 Objetivos **10**
2.1 Objetivo General . . . 10
2.2 Objetivos Específicos . . . 10

3 Marco de referencia **11**
3.1 Marco Teórico . . . 11
3.1.1 Electroencefalografía . . . 11
3.1.2 Distribución de electrodos . . . 11
3.1.3 Tipos de señales . . . 15
3.1.4 Open BCI Cyton . . . 16
3.1.5 Red neuronal convolucional . . . 16
3.1.6 TinyML . . . 18
3.1.7 Ventajas de TinyML . . . 18
3.1.8 Aplicaciones de TinyML . . . 19
3.2 Estado del arte . . . 19
3.2.1 Implementation of an active-filtering circuit for electroencephalographic signal acquisition using an 8-bit microcontroller. [Perez and Tarrillo, 2020] . . . 19
3.2.2 Prótesis de mano controlada con señales EEG.[Cruz and Sánchez, 2021] . . . 21
3.2.3 BCI Wheelchair Control Using Expert System Classifying EEG Signals Based on Power Spectrum Estimation and Nervous Tics Detection.[Pawuś and Paszkiel, 2022] . . . 23
3.2.4 Multi-functional system for persons with disabilities using electroencephalography signals of eye blink.[Sarma, Kant and Rajkumar, 2018] . . . 25
3.2.5 EEG-Based Eye Movement Recognition Using Brain–Computer Interface and Random Forests.[Antoniou, Bozios, Christou, Tzimourta, Kalafatakis, Tsipouras, Giannakeas and Tzallas, 2021] . . . 26
3.2.6 A One-Dimensional CNN-LSTM Model for Epileptic Seizure Recognition Using EEG Signal Analysis.[Xu, Ren, Chen and Che, 2020] 29
3.2.7 Predicting Human Intention-Behavior through EEG signal analysis using Multi-Scale CNN.[Huang, Xiao and Xu, 2020] . . . 31
3.2.8 Epileptic Seizure Detection on an Ultra-Low-Power Embedded RISC-V Processor Using a Convolutional Neural Network.[Bahr, Schneider, Francis, Lehmann, Barg, Buschhoff, Wulff, Strunskus and Faupel, 2021] . . . 32

4 Metodología desarrollada **35**
4.1 Fase 1 Extracción de datos . . . 35
4.1.1 Elección hardware-software . . . 35
4.1.2 Conexión cyton con entorno OpenBCI-GUI . . . 36
4.1.3 Conexión Cyton con Python . . . 36
4.1.4 Software EEG . . . 37

4.1.5 Artefactos Fisiológicos . . . 37
4.1.6 Gestos . . . 38
4.1.7 Preprocesamiento . . . 38
4.1.8 Base De Datos . . . 39
4.2 Fase 2 Generación de red . . . 40
4.2.1 Ventajas de convolucional . . . 40
4.2.2 Optimizador Adam . . . 41
4.2.3 Función de pérdida entropía cruzada . . . 42
4.2.4 Sobre entrenamiento . . . 42
4.2.5 Desarrollo de la red, código bucles . . . 44
4.3 Fase 3 Implementación . . . 49
4.3.1 Arduino nano 33 BLE . . . 49
4.3.2 Convertir modelo a binario con tensor flow lite . . . 51
4.3.3 Crear archivo .model . . . 52
4.3.4 Código de arduino para inferencia . . . 53
4.3.5 Proceso de Ejecución . . . 57

5 Conclusiones 60

6 Recomendaciones 62

Índice de figuras

1 Regiones de la corteza cerebral asociadas con las funciones cerebrales. Tomada de [Bitbrain, 2023] . . . 12
2 Colocación electrodos EEG estándar. Tomada de [Bitbrain, 2023] . . . 13
3 Relación entre las funciones cerebrales y la serie de electrodos estandarizados del EEG del IFCN. Tomada de [Bitbrain, 2023] . . . 14
4 Versatile EEG 32ch de Bitbrain. Tomada de [Bitbrain, 2023] . . . 15
5 . Diseño propuesto para la implementación del módulo de adquisición de señales EEG. Tomada de [Perez and Tarrillo, 2020]. . . . 20
6 . Configuración utilizada para la implementación del módulo de adquisición de señales EEG. Tomada de [Perez and Tarrillo, 2020]. . . . 20
7 . Configuración preliminar de hardware utilizada. Tomada de [Cruz and Sánchez, 2021] . . . 22
8 Sistema de adquisición, análisis y clasificación de datos. Tomada de [Pawuś and Paszkiel, 2022] . . . 24
9 Arquitectura del sistema de interfaz cerebro-computadora de bosques aleatorios (RF-BCI). Tomada de [Antoniou, Bozios, Christou, Tzimourta, Kalafatakis, Tsipouras, Giannakeas and Tzallas, 2021] . . . 28
10 Precisión obtenida por cada uno de los clasificadores en la Arquitectura del sistema de interfaz cerebro-computadora. Tomada de [Antoniou, Bozios, Christou, Tzimourta, Kalafatakis, Tsipouras, Giannakeas and Tzallas, 2021] . . . 29
11 Rendimiento de DNN, CNN y el modelo 1D CNN-LSTM propuesto en la tarea de clasificación de cinco clases. Tomada de [Xu, Ren, Chen and Che, 2020] . . . 30
12 Procedimiento predicción de comportamiento humano mediante CNN multiescala. Tomada de [Huang, Xiao and Xu, 2020] . . . 32
13 Representación esquemática de la arquitectura CNN que muestra las capas convolucionales con sus respectivas matrices de entrada (rectángulos azules) y núcleos (rectángulos rojos). Tomada de [Bahr, Schneider, Francis, Lehmann, Barg, Buschhoff, Wulff, Strunskus and Faupel, 2021] . . . 34
14 Código para convertir la información en dataframe . . . 39
15 Código para seleccionar los 8 canales relevantes . . . 40
16 Código donde se normaliza de 1 a -1 . . . 40
17 Gráfica de rendimiento modelo entrenado con 500 Epochs . . . 44
18 Parámetros que se varían para encontrar el mejor modelo. . . . 46
19 Bucles donde se ejecutan todos los posibles modelos. . . . 46
20 Representacion grafica del Mejor modelo . . . 48
21 Resultados del merjor modelo . . . 48
22 Matriz confusión mejor modelo . . . 49
23 Código python binarizar modelo de keras . . . 51
24 Contenido del archivo `model.h` que declara la matriz del modelo y su longitud. 52
25 El archivo `model.cpp` que inicializa la matriz del modelo y su longitud. . . 53
26 Codigo inicial del programa llamado a librerias e inicializacion de varibales. 54
27 Código de inicialización del puerto serial, carga del modelo, creación del intérprete y llamada a la función `printModelDetails`. . . . 55

28 Ejecución de la función `printModelDetails` que muestra las dimensiones de los tensores de entrada y salida y verifica la validez del modelo. 56
29 Proceso de carga de los datos, ejecución de la inferencia y presentación de resultados a través del puerto serial. 57
30 Notificación tras la compilación del código y su carga en el Arduino, mostrando el uso de memoria. 58
31 Información inicial al ejecutar el código, exhibiendo los detalles del modelo. 58
32 Predicción para una entrada clasificada en la categoría 0, correspondiente al parpadeo del ojo izquierdo. 59
33 Predicción para una entrada clasificada en la categoría 3, que representa la ausencia de gestos, o señal de control. 59

1 Introducción

En los últimos años, los avances tecnológicos han transformado de manera impresionante la calidad de vida de las personas con discapacidad. Estos logros han generado un amplio espectro de oportunidades, permitiendo a aquellos que enfrentan limitaciones superar desafíos y abordar obstáculos de manera más efectiva que nunca.

Un ejemplo destacado del impacto de la tecnología en este campo es el desarrollo de prótesis avanzadas. Antes, las personas con pérdida de extremidades experimentaban restricciones significativas en su movilidad y funcionalidad. Sin embargo, gracias a la innovación tecnológica, ahora existen prótesis robóticas que pueden replicar con asombrosa precisión el movimiento natural del cuerpo humano. Estas prótesis inteligentes han devuelto la independencia y la capacidad para llevar a cabo diversas actividades cotidianas, transformando radicalmente la vida de muchas personas.

Otra área prometedora es el uso de exoesqueletos, dispositivos mecánicos externos que mejoran la fuerza y movilidad de personas con discapacidades físicas. Estos exoesqueletos permiten caminar y realizar actividades antes inimaginables. Además, los avances en interfaces cerebro-computadora han dado lugar a sistemas que permiten a personas con discapacidades graves controlar dispositivos electrónicos, como computadoras y sillas de ruedas, simplemente mediante el poder del pensamiento.

En el contexto de Colombia, según el Censo Nacional de Población y Vivienda-CNPV 2018, se identificó que 3.134.036 personas presentan dificultades para realizar actividades básicas diarias, de las cuales 1.784.372 reportaron tener dificultades en los niveles de mayor severidad (niveles 1 y 2) de la escala del Grupo de Washington (WG por sus siglas en inglés)[Dane, 2020]. El nivel 1 se refiere a personas con discapacidad total que requieren asistencia y apoyo de terceros, mientras que el nivel 2 indica una grave disminución en la capacidad para llevar a cabo actividades, también con un alto grado de dependencia [Dane, 2020].

Aunque los avances tecnológicos han mejorado significativamente la calidad de vida de las personas con discapacidad, en Colombia se enfrenta una brecha digital que plantea desafíos en la implementación equitativa de soluciones. Según datos de 2021, solo el 37.9 porciento de los hogares en el país tenían acceso a computadoras o tabletas, y esta cifra es aún menor en áreas rurales [Quinchía, 2023]. Asimismo, Colombia presenta uno de los niveles más bajos de acceso a internet entre los países miembros de la Ocde, con solo el 60.5 porciento de la población teniendo acceso al servicio [Quinchía, 2023].

En este contexto, surge una cuestión relevante: ¿Es posible mejorar la vida de las personas con discapacidad mediante una mayor accesibilidad a la tecnología, a través de la implementación de una red neuronal en un sistema embebido para la identificación de gestos mediante el procesamiento de señales electroencefalográficas?

En la actualidad, el desarrollo de sistemas de identificación de gestos basados en el procesamiento de señales electroencefalográficas (EEG) ha experimentado un notable crecimiento. Estos sistemas tienen aplicaciones potenciales en diversas áreas, desde asistencia

médica hasta interfaces cerebro-computadora (BCI), realidad virtual y rehabilitación de pacientes con discapacidades motoras.

Uno de los enfoques más prometedores para abordar la identificación de gestos a partir de señales EEG es el uso de redes neuronales convolucionales (CNN). Estas arquitecturas han demostrado una alta eficacia en tareas de clasificación y reconocimiento de patrones, especialmente en el ámbito del procesamiento de imágenes y visión por computadora. Al implementar esta tecnología en sistemas embebidos con microcontroladores, se logra una mayor eficiencia energética y un menor consumo de recursos, lo que los hace adecuados para aplicaciones en tiempo real y con restricciones de energía.

En este contexto, se presenta este trabajo con el objetivo de desarrollar un modelo que implemente redes neuronales convolucionales para la clasificación de gestos a partir de señales EEG en un sistema embebido con microcontrolador. La elección de esta plataforma busca ofrecer una solución de accesibilidad mejorada para personas con discapacidad, permitiéndoles interactuar con dispositivos y sistemas de manera más natural y efectiva, sin depender de interfaces tradicionales.

La integración de redes neuronales convolucionales en sistemas embebidos con microcontroladores representa un avance significativo en el desarrollo de tecnologías más accesibles y eficientes para personas con discapacidad. Además, esta línea de investigación abre la puerta a futuros desarrollos y estudios que podrían ampliar las aplicaciones de la identificación de gestos basada en señales EEG en diversas áreas, incluyendo la medicina, la rehabilitación y la interacción humano-computadora, todo ello con un enfoque en la eficiencia energética y el menor consumo de recursos.

2 Objetivos

2.1 Objetivo General

Sintetizar en un microcontrolador una red neuronal convolucional que permita la identificación de 4 gestos usando señales electroencefalográficas.

2.2 Objetivos Específicos

- Establecer el equipo hardware-software mediante el cual se realizará la medición y adquisición de los potenciales bioeléctricos generados por las señales electroencefalográficas.
- Obtener y limpiar el conjunto de datos por medio del equipo hardware-software establecido, para la implementación de la red neuronal convolucional.
- Verificar el funcionamiento del modelo realizado mediante métricas de desempeño para un clasificador.
- Sintetizar la red neuronal en el sistema embebido seleccionado.

3 Marco de referencia

3.1 Marco Teórico

3.1.1 Electroencefalografía

Técnica para registrar e interpretar la actividad eléctrica del cerebro. Las células nerviosas del cerebro generan impulsos eléctricos que fluctúan rítmicamente en distintos patrones. En 1929, el científico alemán Hans Berger publicó los resultados del primer estudio que empleó un electroencefalógrafo, un instrumento que mide y registra estos patrones de ondas cerebrales. El registro producido por dicho instrumento se denomina electroencefalograma, comúnmente abreviado EEG.

Para registrar la actividad eléctrica del cerebro, se colocan de 8 a 16 pares de electrodos en el cuero cabelludo. Cada par de electrodos transmite una señal a uno de varios canales de registro del electroencefalógrafo. Esta señal consiste en la diferencia de voltaje entre el par. La fluctuación rítmica de esta diferencia de potencial se muestra como picos y valles en un gráfico de líneas por el canal de registro. El EEG de un adulto normal en un estado totalmente consciente pero relajado se compone de ondas oscilantes que se repiten regularmente conocidas como ondas alfa. Cuando una persona está excitada o asustada, las ondas alfa se reemplazan por ondas irregulares rápidas de bajo voltaje. Durante el sueño, las ondas cerebrales se vuelven extremadamente lentas. Tal es también el caso cuando una persona está en coma profundo. Otras condiciones anormales están asociadas con patrones de EEG particulares. Por ejemplo, las ondas lentas irregulares conocidas como ondas delta surgen de la vecindad de un área localizada de daño cerebral.

La electroencefalografía proporciona un medio para estudiar cómo funciona el cerebro y para rastrear las conexiones entre una parte del sistema nervioso central y otra. Sin embargo, su efectividad como herramienta de investigación es limitada, porque registra solo una pequeña muestra de la actividad eléctrica de la superficie del cerebro. Muchas de las funciones más complejas del cerebro, como las que subyacen a las emociones y el pensamiento, no pueden relacionarse estrechamente con los patrones de EEG. Además, el EEG no sirve para diagnosticar enfermedades psiquiátricas.[Britannica, 2023]

3.1.2 Distribución de electrodos

La capa exterior del cerebro se llama corteza cerebral. En esta zona se ejecutan muchas de las funciones clave del sistema nervioso. El córtex cerebral se divide en cuatro partes llamadas lóbulos. Estas son el lóbulo frontal, el lóbulo parietal, el lóbulo temporal y el lóbulo occipital. Cada región ha sido subdividida y está asociada a funciones cerebrales específicas.[Bitbrain, 2023] Las diferentes áreas o lóbulos del cerebro se pueden ver en la Figura[1].

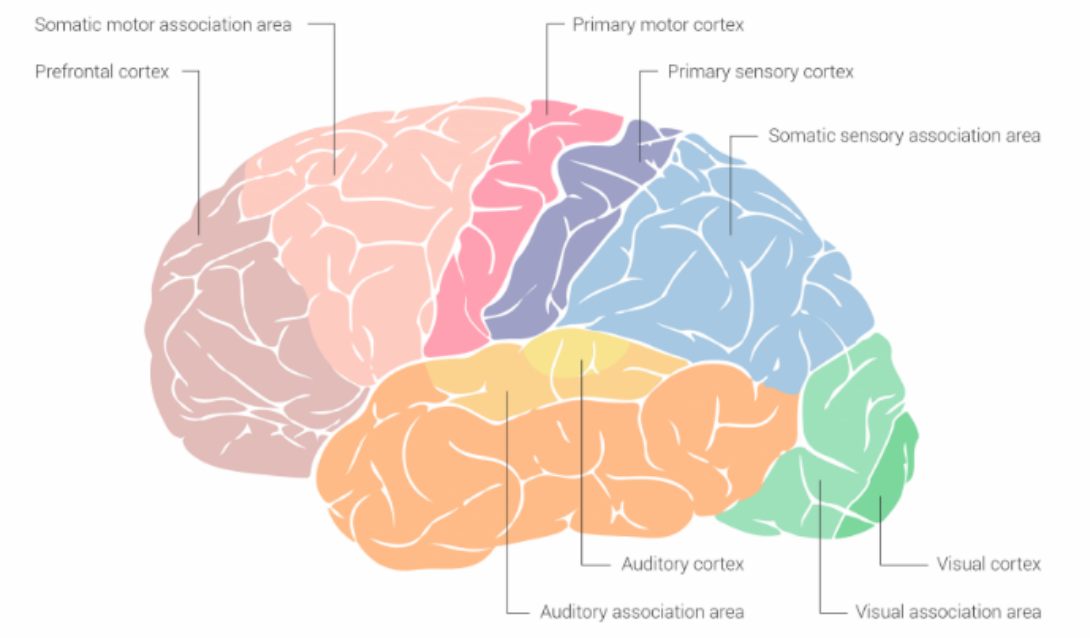


Figura 1: Regiones de la corteza cerebral asociadas con las funciones cerebrales. Tomada de [Bitbrain, 2023]

Estas son las diferentes áreas o lóbulos del cerebro y sus respectivas principales funciones.

Frontal: Razonamiento, control del habla y del movimiento, emociones y resolución de problemas.
Central: Sensoriomotor.
Parietal: Atención, percepción y procesamiento de los estímulos relacionados con los sentidos (temperatura, tacto, presión, dolor).
Occipital: Visión.
Temporal: La memoria, el significado y la interpretación, y procesamiento de los estímulos auditivos.

Los intercambios bioquímicos entre las células producen una pequeña actividad eléctrica cuando las neuronas se comunican entre sí. Una sola señal eléctrica de neurona a neurona no es registrable, pero cuando millones de neuronas se sincronizan, el campo eléctrico generado puede ser medido desde el cuero cabelludo. Esta actividad eléctrica del cerebro (o señales electroencefalográficas, EEG) se transmite a través de los tejidos, los huesos y el cabello antes de que se registre, y, para entonces, su amplitud está muy atenuada.
El sistema internacional de colocación de EEG toma cuatro puntos de referencia craneales universales (nasión, inión y ambos puntos pre-auricular), y distribuye proporcionalmente los electrodos del EEG sobre la superficie de la cabeza como se ve en la Figura[2].[Bitbrain, 2023]

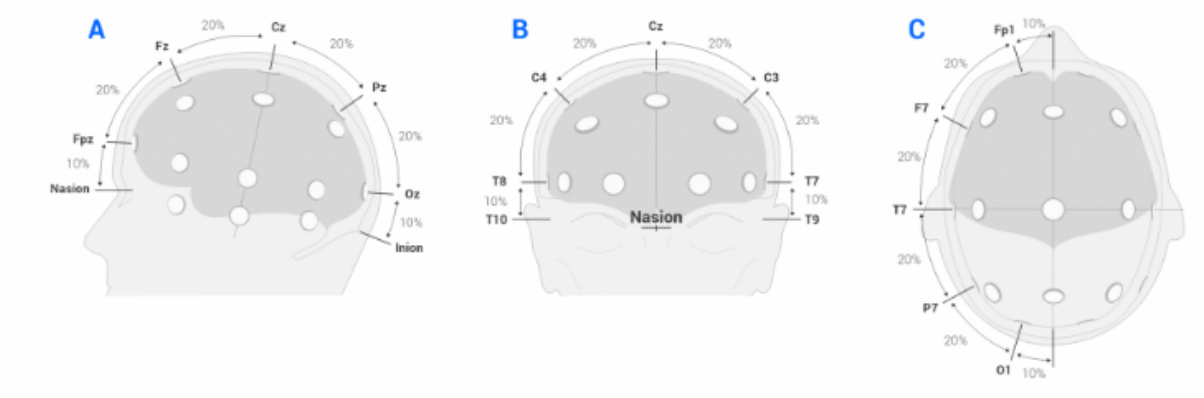


Figura 2: Colocación electrodos EEG estándar. Tomada de [Bitbrain, 2023]

La colocación de los electrodos para el EEG sigue el sistema internacional que los etiqueta según las áreas de la corteza cerebral mencionadas más abajo. Las etiquetas se refieren al lóbulo o área del cerebro que está siendo registrada por cada sensor.
Las principales áreas son frontopolar (Fp), frontal (F), central (C), temporal (T), parietal (P) y occipital (O).
En cuanto a su ubicación lateralizada, los números impares (1, 3, 5, 7) se refieren a los electrodos colocados en el hemisferio izquierdo, mientras que los números pares (2, 4, 6, 8) se refieren a los del hemisferio derecho.
Los electrodos que están sobre la línea media (línea cero) están etiquetados con la letra "z".

Según el porcentaje de la distancia entre los sensores, tenemos el layout 10-20 con un total de 21 sensores, si partimos de una distribución de distancias del 10 % y el 20 % de las curvas de referencia central sagital y coronal. Si estas líneas centrales se dividen en un 10 %, entonces tenemos el layout 10-10 con 81 sensores. Finalmente, si añadimos la resolución con distancias del 5 %, entonces, tenemos el sistema 10-5 para EEG con 320 electrodos.
Con este estándar de ubicación de los electrodos para EEG del cuero cabelludo, uno puede asociar fácilmente el EEG de un sensor dado con diferentes funciones cerebrales dependiendo de su ubicación en el layout del sensor. En la Figura[3] se representan las funciones del cerebro por área y su equivalente con el sistema de electrodos 10-20.[Santos, 2015]

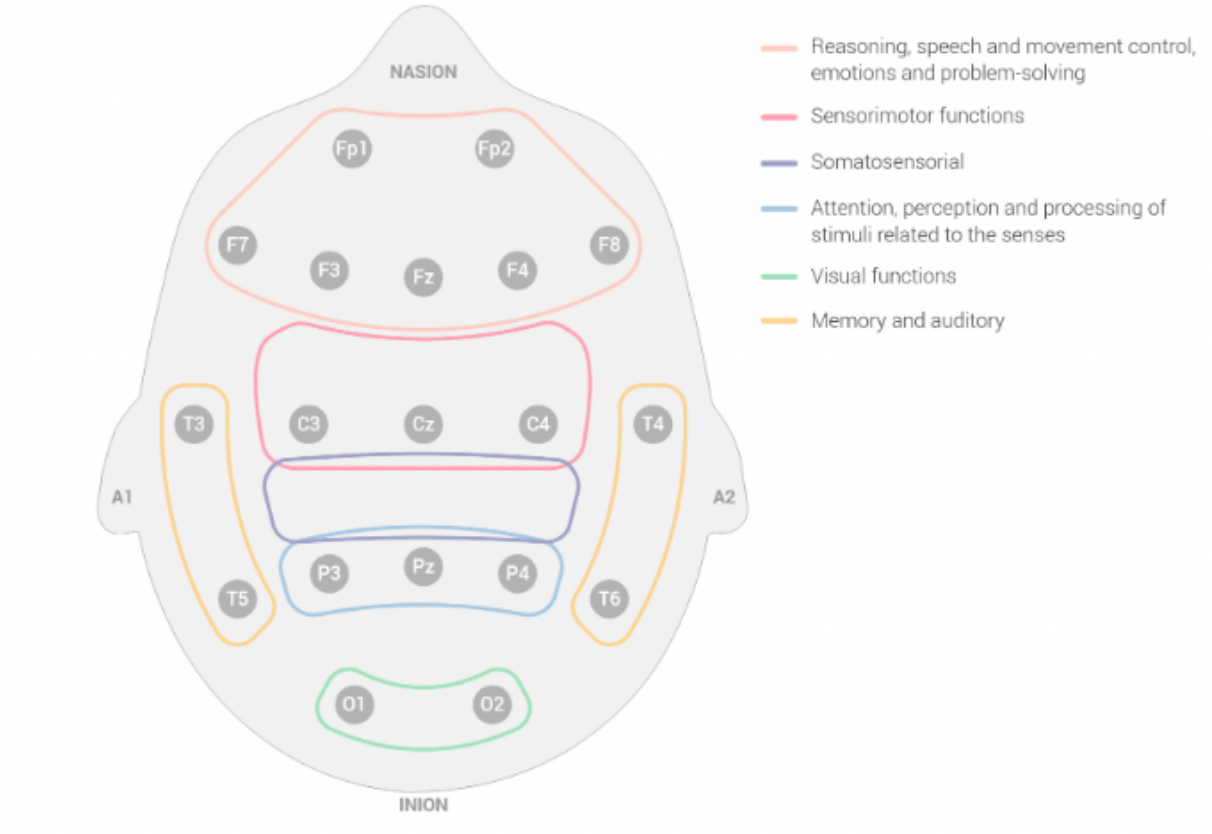


Figura 3: Relación entre las funciones cerebrales y la serie de electrodos estandarizados del EEG del IFCN. Tomada de [Bitbrain, 2023]

Estos sistemas permiten mover sus posiciones intercambiables para acomodar diferentes experimentos. Esto suele ocurrir en entornos de laboratorio, durante las fases de investigación exploratoria, en donde se prioriza la cobertura de los sensores de la cabeza y la flexibilidad por encima de otros aspectos como la comodidad o la ergonomía.
Estos sistemas de EEG suelen estar compuestos por un gorro de tela para EEG que cubre toda la cabeza con una cinta colocada sobre la barbilla para asegurar la tecnología, una serie de sensores y un amplificador. El gorro conserva el sistema 10-10 o 10-20 para EEG con carcasas etiquetadas junto con su patrón donde se pueden insertar los electrodos como se ve en la Figura[4]. Por lo general, hay varios tamaños para adaptarse mejor a todos.[Bitbrain, 2023]



Figura 4: Versatile EEG 32ch de Bitbrain. Tomada de [Bitbrain, 2023]

3.1.3 Tipos de señales

Las señales del EEG son divididas en grupos basadas en su contenido de frecuencia, la manera más común es dividir dichas frecuencias en bandas, estas bandas se han definido mediante estudios de fenómenos psicológicos y fisiológicos asociados a la actividad cerebral, muchos estudios en los últimos años han propuesto un aproximado de qué ritmos cerebrales deben ser utilizados.[Gómez, 2016]

Ritmo Delta: también es conocido como la onda del sueño, ya que este ritmo suele aparecer generalmente en estados de sueño profundo. Su amplitud varía de los 20 a 200 uV, y su rango de frecuencia oscila entre 0,5 y 3,5 Hz.

Ritmo Theta: se presenta generalmente con la activación del lóbulo temporal, su banda de frecuencia está entre los 4 y 8 Hz y su amplitud varía entre los 20 y 100 uV, y a pesar de tener otras explicaciones este ritmo se suele presentar frecuentemente en niños o en adultos que se encuentran sometidos a altos niveles de estrés.

Ritmo Alfa : este ritmo corresponde al estado de relajación, en donde la persona tiene poca actividad mental o se encuentra con los ojos cerrados, este ritmo se atenúa cuando la persona inicia alguna clase de actividad mental que requiera concentración. Las señales asociadas a este ritmo presentan una frecuencia comprendida entre los 8 y 14 Hz, una amplitud de entre 20 y 60 uV; la fuente principal de estas señales se encuentra en la región occipital del cerebro.

Ritmo Beta: su amplitud se encuentra en el rango entre los 2 y 20 uV y su frecuencia entre los 14 y 26 Hz, está vinculado principalmente con los movimientos de las extremidades y actividades musculares, pero también puede asociarse a un estado mental de concentración por parte de la persona y se puede encontrar principalmente en la región frontal y central del cerebro.

Ritmo Gamma: Son las frecuencias superiores a 30 Hz, suelen tener la frecuencia más alta y la amplitud más baja, y se relacionan con actividades cerebrales intensas. Existe

también un sexto ritmo cerebral llamado Ritmo µ, el cual puede observarse en el rango de frecuencias de 8 a 12 Hz. Aunque es básicamente el mismo rango del ritmo alpha, el Ritmo µ se observa en la corteza sensorio motora, mientras que el alpha solo se observa en la corteza visual. [Gómez, 2016]

3.1.4 Open BCI Cyton

El Ultracortex Mark IV es un casco para imprimir en 3D que se puede comprar completamente ensamblado o como un kit de bricolaje para auto ensamblaje. El diseño cuenta con electrodos secos que permiten una configuración rápida (en menos de 60 segundos) y elimina la necesidad de una pasta pegajosa para electrodos. El casco viene en tres tamaños diferentes y están diseñados para funcionar tanto con la placa Open BCI de 32 bits como con la nueva placa Ganglion, Cyton o Cyton+Daisy. Los desarrolladores afirman que decidieron centrarse en un diseño impreso en 3D porque permite una mayor personalización y la capacidad de seleccionar ubicaciones preferidas para los electrodos. Además, el diseño impreso en 3D significa que a medida que se disponga de nuevos y mejores electrodos y dispositivos electrónicos, el equipo no tendrá que reemplazar todo el auricular.[Santos, 2015]

La placa OpenBCI Cyton es una interfaz neuronal de 8 canales compatible con Arduino con un procesador de 32 bits. En esencia, la placa OpenBCI Cyton implementa el microcontrolador PIC32MX250F128B, lo que le proporciona mucha memoria local y velocidades de procesamiento rápidas. La placa viene preinstalada con el gestor de arranque chipKIT™ y el firmware OpenBCI más reciente. Los datos se muestrean a 250 Hz en cada uno de los ocho canales.

La placa OpenBCI Cyton se puede utilizar para tomar muestras de la actividad cerebral (EEG), la actividad muscular (EMG) y la actividad cardíaca (ECG). La placa se comunica de forma inalámbrica con una computadora a través del dongle USB OpenBCI utilizando módulos de radio RFDuino. También puede comunicarse de forma inalámbrica con cualquier dispositivo móvil o tableta compatible con Bluetooth Low Energy (BLE).[Gómez, 2016]

3.1.5 Red neuronal convolucional

Las redes neuronales convolucionales (CNNs) son análogas a las redes neuronales (RNN) con unas pequeñas variaciones, pero en su núcleo parten de un mismo paradigma donde se encuentra una entrada una operación y una salida que modifica los parámetros de la operación acorde a un error que permite generar una auto-optimización del resultado. Estas redes han sido grandemente utilizadas para el procesamiento de imágenes, por su gran desempeño en lo que se ha llamado la extracción de característica principales.

Existen comúnmente 3 tipos de capas en las redes convolucionales, estas son, capas convolucionales, capas de pooling y capas totalmente conectadas. La capa convolucional determinará la salida de las neuronas de las cuales son conectadas a las regiones locales de la entrada a través del cálculo del producto escalar entre sus pesos y la región conectada, La capa de agrupación simplemente realizará una reducción de muestreo a lo largo de la dimensionalidad espacial de la entrada dada, reduciendo aún más la cantidad de parámetros dentro de esa activación, La capa totalmente conectada se describe como una RNN convencional.

- Capa convolucional:

  La capa convolucional es el bloque de creación principal de una CNN, y es donde se realizan la mayoría de los cálculos. Requiere algunos componentes, como datos de entrada, un filtro y un mapa de características. Supongamos que la entrada es una imagen en color compuesta por una matriz de píxeles en 3D. Esto significa que la entrada tendrá tres dimensiones: altura, anchura y profundidad, que corresponden a la composición RGB en una imagen. También hay un detector de características, conocido como kernel o filtro, que se mueve por los campos receptivos de la imagen para comprobar si la característica está presente. Este proceso se denomina convolución [Ibm, s.f].
  El detector de características es una matriz bidimensional (2D) de pesos que representa una parte de la imagen. Aunque su tamaño puede variar, el tamaño del filtro suele ser una matriz de 3x3; esto también determina el tamaño del campo receptivo. A continuación, el filtro se aplica a un área de la imagen y se calcula un producto escalar entre los píxeles de entrada y el filtro. Este producto escalar se introduce en una matriz de salida. Después, el filtro se desplaza un poco y repite el proceso hasta que el kernel haya recorrido toda la imagen. El resultado final de la serie de productos escalares de la entrada y el filtro se conoce como mapa de características, mapa de activación o característica convolucionada [Ibm, s.f].

  Después de cada operación de convolución, la CNN aplica una transformación de unidad lineal rectificada (relu) al mapa de características, introduciendo la no linealidad en el modelo.

  Como se ha mencionado anteriormente, la capa de convolución inicial puede ir seguida de otra capa de convolución. Cuando esto sucede, la estructura de la CNN puede volverse jerárquica, ya que las capas siguientes pueden ver los píxeles en los campos receptivos de las capas anteriores. Por ejemplo, supongamos que queremos determinar si una imagen contiene una bicicleta. Podemos considerar la bicicleta una suma de varias partes. Se compone de un cuadro, un manillar, ruedas, pedales, etc. Cada parte de la bicicleta forma un patrón de nivel inferior en la red neuronal, y la combinación de todas las partes representa un patrón de nivel superior, lo que crea una jerarquía de características dentro de la CNN [Ibm, s.f].

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau \tag{1}$$

- Capa de agrupación:
  La agrupación de capas, también conocida como submuestreo, permite reducir la dimensión mediante la reducción del número de parámetros de la entrada. De manera similar a la capa convolucional, la operación de agrupación barre toda la entrada con un filtro, pero la diferencia es que este filtro no tiene ningún peso. En su lugar, el kernel aplica una función de agregación a los valores dentro del campo receptivo y llena así la matriz de salida. Hay dos tipos principales de agrupación:

Agrupación máxima: a medida que el filtro recorre la entrada, selecciona el píxel con el valor más alto para enviarlo a la matriz de salida. Este enfoque suele utilizarse más que la agrupación media. Agrupación media: conforme el filtro avanza por la entrada, calcula el valor promedio dentro del campo receptivo para enviarlo a la matriz de salida. Aunque se pierde mucha información en la capa de agrupación, esta tiene una serie de beneficios para la CNN. Ayuda a reducir la complejidad, mejora la eficiencia y limita el riesgo de sobre ajuste [Ibm, s.f].

$$O_{ij} = \max_{k,l \in [0,n-1]} I_{i+k,j+l} \tag{2}$$

Donde:
O_ijes el valor en el mapa de características resultante en la posición (i, j).
I es el mapa de características de entrada.
max denota la operación de tomar el valor máximo dentro de la ventana especificada.

- Capa totalmente conectada:
  El nombre de la capa totalmente conectada describe con precisión la capa en sí. Como se ha mencionado anteriormente, los valores de píxel de la imagen de entrada no están conectados directamente con la capa de salida en las capas parcialmente conectadas. Sin embargo, en la capa totalmente conectada, cada nodo de la capa de salida sí está conectado directamente a un nodo de la capa anterior.

  Esta capa realiza la tarea de clasificación basándose en las características extraídas de las capas anteriores y sus diferentes filtros. Las capas convolucionales y de agrupación suelen utilizar funciones relu, mientras que las capas totalmente conectadas generalmente usan una función de activación softmax para clasificar adecuadamente las entradas y generar una probabilidad entre 0 y 1 [Ibm, s.f].

3.1.6 TinyML

TinyML es un campo de estudio en aprendizaje automático y sistemas integrados que explora los tipos de modelos que puede ejecutar en dispositivos pequeños y de bajo consumo, como los microcontroladores. Permite la inferencia de modelos de baja latencia, baja potencia y bajo ancho de banda en dispositivos perimetrales. Mientras que una CPU de consumo estándar consume entre 65 y 85 vatios y una GPU de consumo estándar consume entre 200 y 500 vatios, un microcontrolador típico consume energía del orden de milivatios o micro vatios. Eso es alrededor de mil veces menos consumo de energía. Este bajo consumo de energía permite que los Dispositivos TinyML funcionen desenchufados con baterías durante semanas, meses y, en algunos casos, incluso años, mientras ejecutan aplicaciones ML al límite.[Arun, 2023]

3.1.7 Ventajas de TinyML

- Baja latencia: dado que el modelo se ejecuta en el borde, los datos no tienen que enviarse a un servidor para ejecutar la inferencia. Esto reduce la latencia de la salida.
- Bajo consumo de energía: como comentamos antes, los microcontroladores consumen muy poca energía. Esto les permite funcionar sin ser cargados durante mucho tiempo.

- Bajo ancho de banda: como los datos no tienen que enviarse al servidor constantemente, se usa menos ancho de banda de Internet.
- Privacidad: dado que el modelo se ejecuta en el borde, sus datos no se almacenan en ningún servidor.[Arun, 2023]

3.1.8 Aplicaciones de TinyML

Al resumir y analizar datos en el perímetro en dispositivos de bajo consumo, TinyML ofrece muchas soluciones únicas. Aunque TinyML es un campo emergente, se ha utilizado en producción durante años. Las palabras de activación "OK Google","Alexa", "Hey Siri"son un ejemplo de TinyML. Aquí, los dispositivos están siempre encendidos y analizan tu voz para detectar la palabra de activación.[Santos, 2015]

- Mantenimiento predictivo industrial: las máquinas son propensas a fallar. Usando TinyML en dispositivos de baja potencia, es posible monitorear la máquina y predecir fallas con anticipación constantemente. Este mantenimiento predictivo puede generar importantes ahorros de costos. Ping Services, una startup australiana, presentó un dispositivo IoT que monitorea de forma autónoma las turbinas eólicas al unirse magnéticamente al exterior de la turbina y analizar datos detallados en el borde. Este dispositivo puede alertar a las autoridades sobre posibles problemas incluso antes de que ocurran.
- Atención médica: el proyecto Solar Scare Mosquito utiliza TinyML para frenar la propagación de enfermedades transmitidas por mosquitos como el dengue, la malaria, el virus Zika, el chikungunya, etc. Funciona detectando las condiciones de reproducción de los mosquitos y agita el agua para evitar que se reproduzcan. Funciona con energía solar y, por lo tanto, puede funcionar indefinidamente.
- Agricultura: la aplicación Nuru ayuda a los agricultores a detectar enfermedades en las plantas simplemente tomándoles una foto ejecutando modelos de Machine Learning en el dispositivo usando TensorFlow Lite. Dado que funciona en el dispositivo, no hay necesidad de una conexión a Internet. Este es un requisito crucial para los agricultores remotos, ya que es posible que no tengan una conexión a Internet adecuada en su lugar.
- Conservación de la vida oceánica: los dispositivos inteligentes con tecnología ML se utilizan para monitorear ballenas en tiempo real en vías fluviales alrededor de Seattle y Vancouver para evitar colisiones con ballenas en rutas de navegación concurridas.[Arun, 2023]

3.2 Estado del arte

3.2.1 Implementation of an active-filtering circuit for electroencephalographic signal acquisition using an 8-bit microcontroller. [Perez and Tarrillo, 2020]

Año: 2021
Autor: Jose Perez, Jimmy Tarrillo.
Resumen: Debido a que las señales electroencefalográficas se tratan de señales de bajo

nivel de tensión, su digitalización representa un gran reto y actualmente solo existen varias propuestas y BCI, muchas de ellas basadas en ADC y DSP complicados y costosos para recopilar y procesar datos digitalmente. En este artículo los autores presentan el diseño e implementación de un circuito de filtrado activo para la adquisición de señales electroencefalográficas en un rango de frecuencia entre 0,15 Hz y 50 Hz utilizando un microcontrolador basado en AVR RISC de 8 bits.
Desarrollo: La Figura[5] muestra el diagrama de bloques. El diseño está compuesto por cuatro bloques: bloque de recepción, circuito de filtrado, circuito de adaptación ADC y comunicación digital. Cada señal analógica seleccionada por el bloque de recepción pasa por el circuito de filtrado, luego por el circuito de adaptación del ADC y finalmente, la señal digitalizada se envía a un dispositivo externo a través del circuito de comunicación digital como se muestra en la Figura[5].

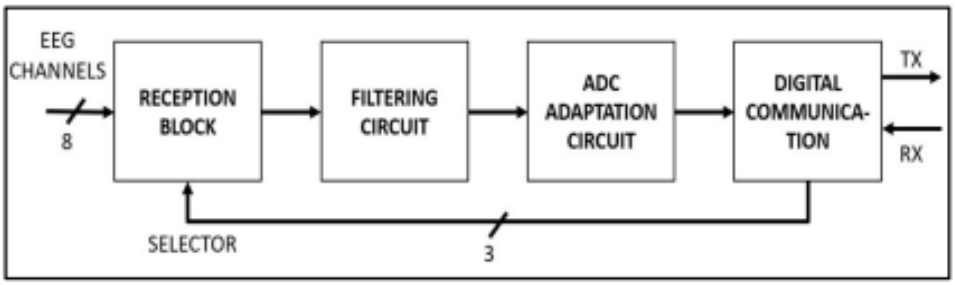


Figura 5: . Diseño propuesto para la implementación del módulo de adquisición de señales EEG. Tomada de [Perez and Tarrillo, 2020].

Metodología:
Los pasos a seguir para realizar la implementación del filtro activo para señales EEG son los mostrados en la Figura[6].

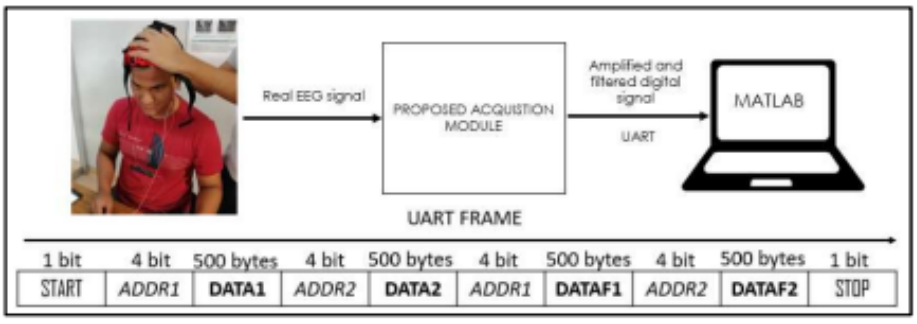


Figura 6: . Configuración utilizada para la implementación del módulo de adquisición de señales EEG. Tomada de [Perez and Tarrillo, 2020].

- Diseño de la placa.
- Validación de funcionamiento, emulando la salida del electrodo.
- Fase de implementación, utilizando la configuración mostrada en la Figura[6].

Aplicación: Las señales EEG reales adquiridas con el sistema presentado validan que la amplificación y el filtrado activo implementado estaban cumpliendo con su objetivo de adquisición de señales electroencefalográficas en un rango de frecuencia entre 0,15 Hz y 50 Hz.

Tecnología Involucrada:

- Electrodo DRL para obtener la señal e instrumentación diferencial Op-Amp INA114 para amplificar la señal y atenuar el ruido.
- Circuito de filtrado compuesto por un filtro de paso alto, un filtro de paso bajo, un amplificador diferencial y un filtro de muesca de 60 Hz. Utilizando la topología de Sallen-Key para diseñar todos los filtros.
- Microcontrolador ATMega328P de 8 bits para convertir y enviar los datos a través de sus periféricos ADC y UART de 10 bits.
- Matlab para visualización de señales.

3.2.2 Prótesis de mano controlada con señales EEG.[Cruz and Sánchez, 2021]

Año: 2021

Autor: Andrés M. Cruz, Hernan Sanchez.

Resumen: Este articulo describe como las ondas cerebrales producidas en el lóbulo lateral izquierdo del cerebro humano y detectadas con un auricular NeuroSky MindFlex fueron usadas para controlar una prótesis de mano. Las señales EEG son detectadas por una BCI (Interfaz de control cerebral), dichas señales son analizadas con la ayuda de un microcontrolador (ARDUINO) y su espectro de frecuencias es dividido en rangos Alfa, Beta, Theta y Gamma. Como resultado del análisis de las señales EEG, se encuentra que el parpadeo voluntario afecta principalmente los valores de potencia relacionados con la banda alfa y finalmente se utiliza esta banda de frecuencias para dirigir los movimientos de abducción y aducción de los dedos de la prótesis.

Desarrollo: Para el desarrollo del trabajo experimental se dividió el proceso en diferentes etapas, primero se realizo un análisis de una unidad MINDFLEX Duel, identificando el hardware y las señales eléctricas generadas.

Estas señales, se transmiten por conductores a un micro controlador Arduino Uno, (emulando el micro controlador interno del Auricular), para luego ser enviadas vía USB a un computador, en el cual a través de diseño de código de programación y con ayuda de diferentes librerías que ofrece la plataforma Arduino, se registran los datos obtenidos, en formato ASCII o CVS. Esta configuración se muestra en la Figura[7].

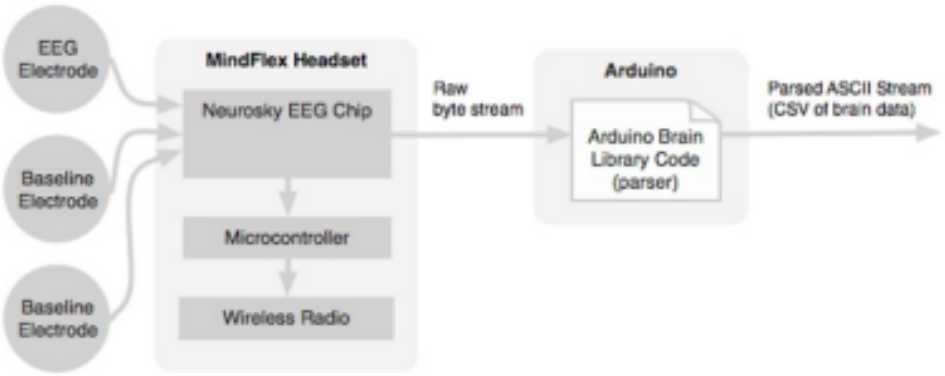


Figura 7: . Configuración preliminar de hardware utilizada. Tomada de [Cruz and Sánchez, 2021]

Finalmente se establece un protocolo en Arduino llamado "eyes blink", que detecta señales no aleatorias de baja intensidad imbuidas en el ruido y que permitan accionar la prótesis a través de los impulsos de voltaje generados por estas. Posteriormente se enlaza el sistema a través de un código que transforma las señales provenientes del parpadeo en señales eléctricas de potencia media las cuales accionan un actuador (en este caso un motor-reductor), y a través de este y acoples mecánicos, se acciona la prótesis

Metodología:

- Análisis unidad MINDFLEX Duel y señales eléctricas producidas.
- Transmisión por conductores a un micro controlador Arduino Uno (emulando el micro controlador interno del Auricular).
- Instalación de la librería Brian Serial Out para visualizar las señales transmitidas desde el auricular como datos CVS y graficarlas.
- Establecer protocolo en Arduino llamado "eyes blink", para detectar señales no aleatorias de baja intensidad imbuidas en el ruido.
- Enlazar el sistema través de un código que transforma las señales provenientes del parpadeo en señales eléctricas de potencia media las cuales accionan un actuador (en este caso un motor-reductor), y a través de este y acoples mecánicos accionar la prótesis

Aplicación: La presencia de potenciales en las diferentes regiones cerebrales, especialmente en la región dorso lateral izquierda donde se realizaron las mediciones gráficas se identificaron como correspondientes a los niveles llamados de atención, meditación y relajación. La intensidad de las señales alfa, beta, gama, entre otras, que un individuo puede generar son controlables y reproducibles a través del entrenamiento y facilitan la personalización de la rutinas para un manejo controlado de la prótesis.

Tecnología Involucrada:

- Auricular NeuroSky MindFlex para detectar las señales EEG.

- ARDUINO PRO-Mini para analizarlas señales detectadas, al cual se le incorpora un sistema Bluetooth HC-06 compatible con Arduino que permite transmitir los datos en forma inalámbrica.
- Módulo de conexión USB FTDI 232 que permite no solo ahorrar espacio, sino que representa una mejora en la velocidad de transmisión de datos.
- Protocolo en Arduino llamado "eyes blink", que detecta señales no aleatorias de baja intensidad.
- Moto-reductor y prótesis.

3.2.3 BCI Wheelchair Control Using Expert System Classifying EEG Signals Based on Power Spectrum Estimation and Nervous Tics Detection.[Pawuś and Paszkiel, 2022]

Año: 2022.

Autor: Dawid Pawuś, Szczepan Paszkiel.

Resumen: Los autores proponen una solución avanzada utilizando el análisis de señales, utilizando un enfoque de diseño de sistema BCI que utiliza inteligencia artificial para el análisis avanzado de señales que contienen expresiones faciales como comandos de control. Las señales estaban cargadas con numerosos artefactos causados por tics nerviosos simulados. El sistema experto propuesto constaba de dos redes neuronales. La primera red permitió el análisis de muestras de un segundo de señales EEG de electrodos seleccionados sobre la base de formas de onda de estimación de espectro de potencia. Por lo tanto, fue posible generar una señal de control adecuada como resultado de los comandos de expresión facial apropiados. La segunda de las redes neuronales detectó la aparición y tipo de tics nerviosos en la señal. Además, los participantes se vieron afectados por interferencias como el sonido de la calle y la televisión o la radio, Wi-Fi y ondas de radio. El sistema así diseñado se adapta a las exigencias del día a día de las personas con discapacidad, en particular de aquellas en silla de ruedas, cuyo control se basa en la tecnología BCI.

Desarrollo: La señal de EEG sin procesar se registró con un auricular Emotiv EPOC Flex Gel. Las medidas necesarias se recopilaron utilizando el software dedicado EmotivPRO. La señal de EEG sometida a un filtrado multietapa apropiado se evaluó en un análisis de dos etapas por un sistema experto. El sistema consistió en dos redes neuronales implementadas en el software Matlab. En este caso se utilizó la versión 2020b. El primero de ellos evaluó la señal en cuanto a la detección de expresiones faciales, con la ayuda de la cual el sistema BCI debía emitir comandos de control. La clasificación se llevó a cabo mediante la preparación consistente de la señal en el proceso de clasificación, comenzando con un filtrado adecuado y finalizando con un análisis espectral. La segunda red neuronal aprendió a analizar la señal en busca de tics neuronales.

Metodología:

- Toma de 15 medidas a ocho personas de señales repetitivas que consistían en varias expresiones faciales y tics nerviosos simulados.

- Preclasificación. Filtrado de señales sin procesar para la detección de comandos de movimiento, así como el filtrado realizado para aislar los tics nerviosos, como se evidencia en la Figura[8].
- Análisis espectral asignando el gesto de fruncir el ceño (orden de avanzar), entrecerrar el ojo derecho (comando para mover a la derecha) y entrecerrar el ojo izquierda (comando para mover a la izquierda).
- Entrenamiento de la red neuronal.
- Clasificación mediante una red neuronal artificial.

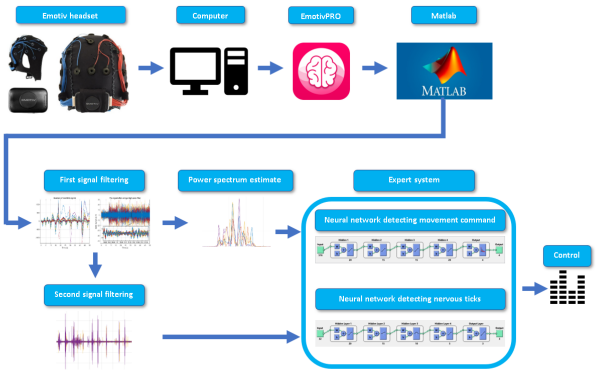


Figura 8: Sistema de adquisición, análisis y clasificación de datos. Tomada de [Pawuś and Paszkiel, 2022]

Aplicación: Fue posible lograr una eficiencia aceptable para el algoritmo y, sobre todo, confirmar su aplicabilidad a las condiciones de la vida cotidiana, que era el objetivo principal. También fue importante tener un enfoque práctico del problema que realmente puede afectar la vida de las personas con discapacidad. Además, se diseñó un sistema para permitir el análisis de señales filtradas sujetas al análisis espectral. Lo que es más importante, estas señales estaban muy distorsionadas por artefactos, pero un sistema de clasificación experto solucionó el problema.

Tecnología involucrada:

- Auricular Emotiv EPOC Flex Gel
- Software dedicado EmotivPRO para recopilación de señales sin procesar.
- Computadora y software Matlab para implementación de filtrado y red neuronal.
- Tecnología Bluetooth para conexión entre la computadora y el dispositivo auricular..
- La red neuronal utilizada para reconocer los patrones de formas de onda espectrales fue un diagrama con 516 entradas que constaba de 4 capas ocultas, compuestas sucesivamente por capas de 20, 15, 15 y 20 neuronas tangentes hiperbólicas, así como una capa como salida. La última capa tenía cuatro salidas, pero solo tres eran comandos de movimiento.

- La red neuronal para el reconocimiento de tics nerviosos utilizó una red que consta de 4 capas ocultas. La primera capa consistía en una función tangente hiperbólica con 20 neuronas. La segunda capa tenía una función de activación log-sigmoidea, compuesto por 15 neuronas. La tercera capa, a su vez, era la función de base radial, solo se utilizaron 10 neuronas. En la última capa oculta se volvió a utilizar la función tangente hiperbólica, pero esta vez con 5 neuronas.

3.2.4 Multi-functional system for persons with disabilities using electroencephalography signals of eye blink.[Sarma, Kant and Rajkumar, 2018]

Año: 2018.

Autor: Subhra Sankha Sarma, Piyush Kant and Rajkumar.

Resumen: Se presenta un sistema que puede operarse utilizando señales de electroencefalografía (EEG) generadas durante el parpadeo y, por lo tanto, puede ser útil para personas con discapacidades locomotoras y de otro tipo para realizar sus actividades diarias. Las señales de EEG son procesadas por un microcontrolador y, basándose en la programación, el microcontrolador toma la decisión de realizar la tarea deseada accionando un dispositivo correspondiente entre varios dispositivos conectados al sistema. Una característica importante del sistema es que se puede adaptar a las necesidades particulares del usuario y se puede conectar/desconectar para accionar diferentes aparatos según la condición y los requisitos del usuario.

Desarrollo: Se utilizan señales EEG generadas durante un parpadeo intencional para el desarrollo de un sistema que puede controlar múltiples dispositivos. Se utiliza un sistema BIOPAC MP150 para adquirir las señales EEG. Los electrodos se colocan de acuerdo con el sistema internacional 10/20 en las posiciones Fp1, Fp2 y C3. Las señales de EEG adquiridas se procesan digitalmente utilizando un filtro de paso de banda con una frecuencia de corte más baja Wl = 8 Hz y una frecuencia de corte más alta Wh = 22 Hz. Se utiliza un filtro Butterworth de orden N = 8. Se utiliza un filtro de muesca de frecuencia central de 50 Hz para eliminar el ruido de interferencia introducido por las líneas eléctricas.
Es probable que dos parpadeos intencionales consecutivos tengan la misma amplitud. Para activar un dispositivo, si el sujeto parpadea dos veces intencionadamente, seguido de otro parpadeo no intencionado como efecto posterior, la diferencia de amplitud entre los parpadeos intencionados y no intencionados será significativa. Si la diferencia de amplitud entre las dos señales supera un tercio del parpadeo anterior, se consideraría una señal falsa y el sistema la rechazaría. Esto elimina casi todas las detecciones que ocurren debido a un parpadeo involuntario. La señal procesada digitalmente se envía al temporizador T1 del microcontrolador AVR Atmega16. Para cualquier parpadeo entrante, el contador se incrementa y comprueba si hay otro parpadeo consecutivo. Si se cuenta otro parpadeo consecutivo, se repite la misma función; de lo contrario, se realiza la tarea preprogramada de detección de un parpadeo y se activa el dispositivo correspondiente. Se otorga un retraso de 400 ms para verificar parpadeos consecutivos. El accionamiento de los dispositivos mediante parpadeos se demuestra mediante la activación de la pantalla LCD y el funcionamiento de un motor de CC.

Metodología:

- Se utiliza un sistema BIOPAC MP150 para adquirir las señales EEG.
- Los electrodos se colocan de acuerdo con el sistema internacional 10/20 en las posiciones Fp1, Fp2 y C3.
- Las señales de EEG adquiridas se procesan digitalmente utilizando un filtro de paso de banda con una frecuencia de corte más baja Wl = 8 Hz y una frecuencia de corte más alta Wh = 22 Hz. Se utiliza un filtro Butterworth de orden N = 8.
- La señal procesada digitalmente se envía al temporizador T1 del microcontrolador AVR Atmega16. Para cualquier parpadeo entrante, el contador se incrementa y comprueba si hay otro parpadeo consecutivo. Si se cuenta otro parpadeo consecutivo, se repite la misma función; de lo contrario, se realiza la tarea preprogramada de detección de un parpadeo y se activa el dispositivo correspondiente. Se otorga un retraso de 400 ms para verificar parpadeos consecutivos.
- El accionamiento de los dispositivos mediante parpadeos se demuestra mediante la activación de la pantalla LCD y el funcionamiento de un motor de CC.

Aplicación: Estos experimentos demuestran claramente que el sistema es adecuado para la interfaz hombre-máquina y puede usarse para el desarrollo de dispositivos que sean útiles para personas con discapacidad. El sistema permite a una persona tener libertad para controlar dispositivos utilizando sus señales cerebrales sin ninguna sesión de entrenamiento extensa. El mismo sistema se puede utilizar para múltiples aplicaciones, como mostrar mensajes, movilidad, encender/apagar dispositivos como alarmas, bombillas, ventiladores y controlar el motor de la cama de un paciente, etc.

Tecnología involucrada:

- sistema BIOPAC MP150.
- Filtros digitales.
- microcontrolador AVR Atmega16.
- LCD display.

3.2.5 EEG-Based Eye Movement Recognition Using Brain–Computer Interface and Random Forests.[Antoniou, Bozios, Christou, Tzimourta, Kalafatakis, Tsipouras, Giannakeas and Tzallas, 2021]

Año: 2021.

Autor: Evangelos Antoniou, Pavlos Bozios, Vasileios Christou, Katerina D. Tzimourta, Konstantinos Kalafatakis, Markos G. Tsipouras, Nikolaos Giannakeas and Alexandros T. Tzallas.

Resumen: Este artículo presenta un sistema novedoso que utiliza una interfaz cerebro-computadora (BCI) para capturar señales electroencefalográficas (EEG) de sujetos humanos durante el movimiento ocular y posteriormente clasificarlas en seis categorías aplicando un algoritmo de clasificación de bosques aleatorios (RF). Las categorías de la interfaz cerebro-computadora de bosques aleatorios propuesta (RF-BCI) se definen de acuerdo con la posición de los ojos del sujeto: abierto, cerrado, izquierda, derecha, arriba y abajo. El propósito de RF-BCI es utilizarse como sistema de control basado en EEG para conducir una silla de ruedas electromecánica (dispositivo de rehabilitación). El enfoque propuesto se probó utilizando un conjunto de datos que contiene 219 registros tomados de 10 pacientes diferentes. Cada sensor captó cuatro ondas cerebrales diferentes (delta, theta, alfa y beta) por segundo. Luego, estas señales se dividieron en ventanas de 4 segundos, lo que dio como resultado 512 muestras por registro y se extrajo la energía de la banda para cada ritmo de EEG. El sistema propuesto se comparó con los algoritmos Naïve Bayes, Bayes Network, k-vecinos más cercanos (K-NN), perceptrón multicapa (MLP), máquina de vectores de soporte (SVM), árbol de decisión J48-C4.5 y clasificación de ensacado. Los resultados experimentales mostraron que el algoritmo RF superó a los otros enfoques y se obtienen altos niveles de precisión (85,39 %) para una clasificación de 6 clases. Este método explota la alta información espacial adquirida del dispositivo portátil de registro de EEG Emotiv EPOC Flex y examina con éxito el potencial de este dispositivo para ser utilizado en la tecnología de sillas de ruedas BCI.

Desarrollo: Las señales de EEG se registraron a 128 Hz. Luego, con la ayuda de la herramienta de procesamiento de señales (sptool, MATLAB 2018a), se redujo el ruido no deseado y se diseñaron y aplicaron cuatro filtros digitales FIR (Equiripple) de paso de banda de 16 órdenes para descomponer la grabación de EEG en subbandas de interés específicas, correspondiente a 4 ritmos EEG (delta, theta, alfa y beta). También se aplicó un filtro FIR de paso alto con un corte de frecuencia de 0,5 Hz para eliminar bajas frecuencias alrededor de 0. Luego, se extrajo la energía en cada subbanda de interés. Los canales se procesaron aún más y la energía de la subbanda se asoció con los movimientos oculares horizontales y verticales y los diferentes estados visuales (ojos abiertos o cerrados). El conjunto de datos creado se dividió en entrenamiento y prueba mediante validación cruzada de k veces, donde el 90 % de las muestras se utilizaron como datos de entrenamiento para el clasificador de RF y el 10 % se retuvo para probar su capacidad de generalización en datos desconocidos.

Metodología:
La metodología descrita se observa en la Figura[9].

- La señal de EEG se captura con los auriculares y se somete a un procedimiento de filtrado de preprocesamiento para eliminar el ruido no deseado y separar las bandas.
- Extracción de características y asociación de cada entrada del conjunto de datos con un movimiento ocular.
- El conjunto de datos se utiliza para entrenar el clasificador de RF.
- Entrenamiento del clasificador.
- Clasificación mediante Random Forest.

- Comparación con otros clasificadores, utilizando los siguientes siete algoritmos de clasificación populares: naïve Bayes, Bayes Network, K-NN, MLP, SVM.

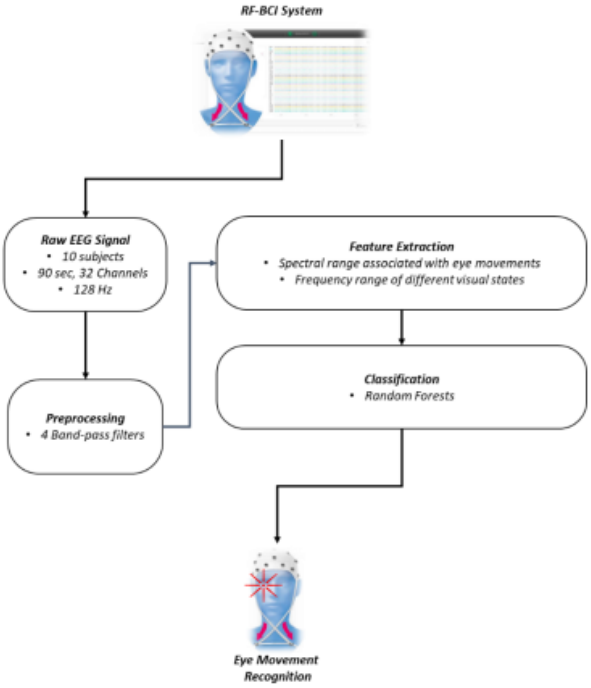


Figura 9: Arquitectura del sistema de interfaz cerebro-computadora de bosques aleatorios (RF-BCI). Tomada de [Antoniou, Bozios, Christou, Tzimourta, Kalafatakis, Tsipouras, Giannakeas and Tzallas, 2021]

Aplicación:

Se presentó un método BCI para capturar señales de EEG de sujetos humanos utilizando el sistema de desgaste cerebral EEG inalámbrico EPOC Flex. Después de una etapa de preprocesamiento y extracción de características, este método pudo clasificar los registros de EEG en seis categorías utilizando un algoritmo de RF. Se eligió el proceso RF porque tiene un tiempo de entrenamiento relativamente rápido, una pequeña cantidad de parámetros definidos por el usuario y un rendimiento superior en comparación con los otros cinco algoritmos de clasificación ampliamente utilizados como se observa en la Figura[10].

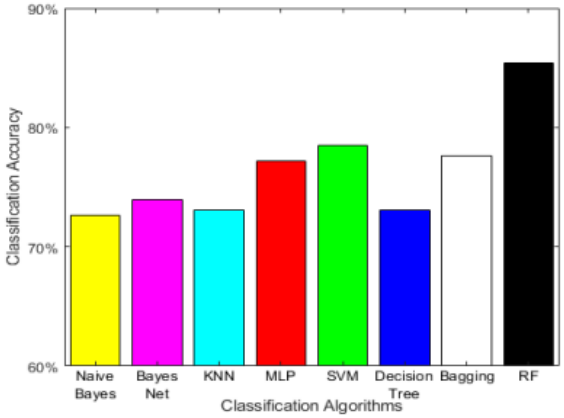


Figura 10: Precisión obtenida por cada uno de los clasificadores en la Arquitectura del sistema de interfaz cerebro-computadora. Tomada de [Antoniou, Bozios, Christou, Tzimourta, Kalafatakis, Tsipouras, Giannakeas and Tzallas, 2021]

Tecnología involucrada:

- El sistema RF-BCI propuesto utiliza el sistema de desgaste cerebral EEG inalámbrico EPOC Flex equipado con sensores de gel Ag/AgCl de Emotiv para registrar los datos del movimiento ocular.
- Herramientas de procesamiento de señales (sptool, MATLAB 2018a).
- Algortimos de clasificación: RF, naïve Bayes, Bayes Network, K-NN, MLP, SVM

3.2.6 A One-Dimensional CNN-LSTM Model for Epileptic Seizure Recognition Using EEG Signal Analysis.[Xu, Ren, Chen and Che, 2020]

Año: 2020.

Autor: Gaowei Xu, Tianhe Ren, Yu Chen and Wenliang Che.

Resumen: Este articulo expone que es importante detectar las crisis epilépticas y brindar tratamiento médico de manera oportuna. Actualmente, los expertos médicos reconocen la actividad de los ataques epilépticos mediante la inspección visual de registros de señales electroencefalográficas (EEG) de los pacientes basándose en su experiencia, lo que requiere mucho tiempo y esfuerzo. En vista de esto, este artículo propone un modelo de memoria a corto plazo de red neuronal convolucional unidimensional (1D CNN-LSTM) para el reconocimiento automático de ataques epilépticos mediante el análisis de señales.

Desarrollo: Debido a que un funcionario de la UCI procesó y reestructuró previamente el conjunto de datos original, en el proceso de preprocesamiento de datos, solo se normalizan los datos de la señal EEG proporcionados por el conjunto de datos oficiales de la UCI antes de introducirlos en la red neuronal. Posteriormente, la CNN 1D puede extraer las características efectivas y representativas de los datos de secuencia de series temporales 1D mediante la realización de operaciones de convolución 1D utilizando múltiples filtros. Durante todo el proceso del experimento, se selecciona el 90 % de los datos como conjunto

de entrenamiento y el 10 % de los datos como conjunto de prueba. En cuanto a los modelos de red neuronal profunda (DNN), CNN y LSTM convolucional 1D, el número de épocas de entrenamiento se establece en 100. Para mejorar el rendimiento de la generalización y evitar el problema de sobre ajuste, se utiliza la técnica de abandono en el método propuesto.

Metodología:

- Los datos de la señal EEG sin procesar se preprocesan y normalizan.
- El Diseño de una red neuronal convolucional (CNN) 1D para extraer de manera efectiva las características de los datos de la secuencia EEG normalizada.
- Las características extraídas luego son procesadas por las capas de LSTM para extraer aún más las características temporales. Después de eso, las características de salida se introducen en varias capas completamente conectadas para el reconocimiento final del ataque epiléptico.
- Entrenamiento y validación del modelo.
- Comparación de los resultados con los métodos tradicionales de aprendizaje automático, incluidos los k vecinos más cercanos, las máquinas de vectores de soporte y los árboles de decisión y otros métodos de aprendizaje profundo, incluida la red neuronal profunda estándar y CNN.

Aplicación:
Como se observa en la Figura[11] el modelo prouesto obtiene los mejores resultados, el rendimiento del modelo 1D CNN-LSTM propuesto se verifica en el conjunto de datos públicos de reconocimiento de ataques epilépticos de la UCI. Los resultados de los experimentos muestran que el método propuesto logra altas precisiones de reconocimiento del 99,39 % y 82,00 % en las tareas de reconocimiento de ataques epilépticos binarios y de cinco clases, respectivamente.

Métodos	Exactitud	Precisión	Recordar	Puntuación F1
DNN	68,78%	67,63%	67,67%	0.6691
cnn	67,30%	67,73%	66,76%	0.6705
Modelo propuesto	82,00%	81,78%	81,70%	0.8156

Figura 11: Rendimiento de DNN, CNN y el modelo 1D CNN-LSTM propuesto en la tarea de clasificación de cinco clases. Tomada de [Xu, Ren, Chen and Che, 2020]

Tecnología involucrada:

- Conjunto de datos públicos de reconocimiento de ataques epilépticos de la UCI.
- Herramientas de procesamiento de señales.
- Modelo 1D CNN-LSTM.

3.2.7 Predicting Human Intention-Behavior through EEG signal analysis using Multi-Scale CNN.[Huang, Xiao and Xu, 2020]

Año: 2020.

Autor: Chenxi Huang,Yutian Xiao, Gaowei xu.

Resumen: Este articulo expone que actualmente los métodos de clasificación de señales EEG basados en CNN existentes no tienen una precisión muy satisfactoria. Esto debido a que la mayoría de los métodos existentes solo utilizan los mapas de características en la última capa de CNN para la clasificación de señales EEG, lo que podría omitir información local y detallada para una clasificación precisa. Para abordar este desafío, este artículo propone un método de clasificación de señales EEG basado en modelos CNN de múltiples escalas. Los resultados experimentales muestran que la precisión promedio del método propuesto es del 73,9 %, lo que mejora la precisión de la clasificación en un 10,4 %, 5,5 % y 16,2 % en comparación con los métodos tradicionales que incluyen redes neuronales artificiales, máquinas de vectores de soporte y codificadores automáticos apilados.

Desarrollo: Se utilizaron datos de señales de EEG recopilados de nueve voluntarios. La frecuencia de adquisición de datos de señales de EEG en este conjunto de datos experimentales fue de 250 Hz, y se utilizan tres electrodos. Posteriormente para entrenar el modelo multiescala con más datos de entrenamiento en el método propuesto, se aumenta el conjunto de datos original obteniendo un conjunto de datos mucho más grande, esto se debe a que más datos de entrenamiento contribuirán al entrenamiento del modelo y evitarán el sobreajuste del modelo. Estas señales EEG sin procesar se convierten en imágenes de tiempo-frecuencia mediante el preprocesamiento de señales STFT, ya que se ha demostrado que cuando una persona realiza tareas de imágenes motoras, se produce una desincronización relacionada con eventos (ERD) y una sincronización relacionada con eventos (ERS). Para diferenciar dichas tareas se elige el rango de frecuencia de 4-14 Hz para la banda mu y el rango de frecuencia de 16-32 Hz para la banda beta. El modelo CNN multiescala aplicado en este artículo tiene siete capas, que son: una capa de entrada, dos capas convolucionales, una capa de agrupación máxima, una capa multiescala, una capa de conexión completa y una capa de salida softmax. La capa de múltiples escalas se conecta completamente a la capa de salida de softmax, donde la función softmax se utiliza para calcular la probabilidad de que la imagen de entrada pertenezca a diferentes clases de imágenes de motor.

Metodología: como se describe en la Figura[12] la metodología implementada es:

- Toma de datos de señales EEG a nueve voluntarios, en donde deben imaginar movimientos de la mano izquierda y derecha.
- Preparación y aumento de datos, basado en el método de expansión de datos para señales de series temporales.
- Preprocesamiento de datos, en donde se aplica el método STFT a segmentos de señales EEG.
- Construcción y entrenamiento de modelos CNN multiescala.

- Predicción de la intención-comportamiento humano.

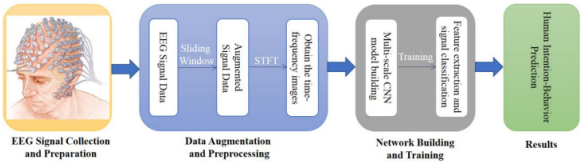


Figura 12: Procedimiento predicción de comportamiento humano mediante CNN multi-escala. Tomada de [Huang, Xiao and Xu, 2020]

Aplicación: Después del entrenamiento del modelo CNN de múltiples escalas, los espectrogramas STFT de las señales EEG sin procesar se pueden ingresar al modelo entrenado y se puede analizar la intención-comportamiento humano predicha. Por lo tanto, el método propuesto puede ayudar a las personas, especialmente discapacitados y personas mayores, a controlar dispositivos utilizando únicamente señales cerebrales, sin ningún tipo de movimiento fisico.

Tecnología involucrada:

- Electrodos para recopilación de señales EEG.
- Filtro pasa banda (0.5 Hz - 100 Hz) para eliminación del ruido de la señal EEG.
- Técnica de Transformada de Fourier de tiempo corto (STFT).
- Modelo CNN multiescala de siete capas, una capa de entrada, dos capas convolucionales, una capa de agrupación máxima, una capa multiescala, una capa de conexión completa y una capa de salida softmax.

3.2.8 Epileptic Seizure Detection on an Ultra-Low-Power Embedded RISC-V Processor Using a Convolutional Neural Network.[Bahr, Schneider, Francis, Lehmann, Barg, Buschhoff, Wulff, Strunskus and Faupel, 2021]

Año: 2021.

Autor: Andreas Bahr, Matthias Schneider, Maria Avitha Francis, Hendrik M. Lehmann, Igor Barg, Anna-Sophia Buschhoff, Peer Wulff, Thomas Strunskus y Franz Faupel.

Resumen: Este trabajo presenta una CNN para la detección de ataques epilépticos capaz de ejecutarse en un microprocesador de potencia ultrabaja. La CNN está implementada y optimizada en MATLAB. Además, la CNN también está implementada sobre un microprocesador GAP8 con arquitectura RISC-V. La capacitación, optimización y evaluación de la CNN propuesta se basan en el conjunto de datos CHB-MIT. La CNN alcanza una sensibilidad media del 90 % y una especificidad muy alta superior al 99 %, lo que corresponde a una tasa media de falsos positivos de 6,8 s por hora. Tras la implementación de la CNN en el microcontrolador se alcanza una sensibilidad del 85 %.

Desarrollo: Se utilizaron datos CHB-MIT de código abierto, que contiene registros continuos de EEG del cuero cabelludo de 24 niños con convulsiones intratables, etiquetados

por profesionales médicos. Para la . estructuración de datos, la CNN del detector de ataques epilépticos procesa señales de tiempo con una longitud fija y dedicada. Por lo cual el conjunto de datos se divide en partes con una longitud de 1 s. Estos bloques de datos son la entrada para la red neuronal. El entrenamiento, verificación y optimización de la CNN desarrollada y presentada en nuestro trabajo se realiza en MATLAB utilizando la caja de herramientas de aprendizaje profundo de MATLAB. La primera capa es una capa convolucional que utiliza un núcleo con dimensiones 23×17. En la primera capa, se elige una convolución sobre todos los electrodos, las siguientes tres capas convolucionales extraen características clave y reducen la dimensionalidad de la red. Como la tarea de clasificación consiste en decidir entre registros ictales e interictales, cada neurona de salida representa una clase. Con una función softmax justo al final de la red neuronal, se calcula la probabilidad para las dos clases de salida. Finalmente la red entrenada se implementa en un microprocesador de potencia ultrabaja GAP8 basado en RISC-V.

Metodología:

- Preparación del conjunto de datos, en donde la detección de convulsiones se puede modelar como un problema de clasificación de series temporales que clasifica los datos de entrada en partes ictales e interictales.
- Se aplica técnica de ventana deslizante, en donde una ventana con una longitud de 256 muestras/1 s se desliza sobre los datos de las convulsiones con un tamaño de paso de $S = 1$ muestra para generar convulsiones adicionales para un conjunto de entrenamiento equilibrado.
- Entrenamiento, verificación y optimización de la CNN en Matlab, donde se evalúa el tiempo de ejecución y los requisitos de memoria para un clasificador de detección de convulsiones energéticamente eficiente.
- Implementación de la red entrenada en un microprocesador de potencia ultrabaja GAP8 basado en RISC-V .
- Clasificación binaria entre registros ictales e interictales .

Aplicación: El detector propuesto reduce el consumo de energía en un factor de 6 en comparación con enfoques relacionados. Esto se logra mediante la adopción de CNN y explotando el motor de convolución de hardware del microprocesador GAP8, que permite un cálculo energéticamente eficiente del operador de convolución. La CNN presentada aquí está entrenada individualmente para cada paciente. Por consiguiente, este enfoque no se limita a un tipo específico de epilepsia.

Tecnología involucrada:

- Conjunto de datos CHB-MIT de código abierto, recopilados en el Children's Hospital Boston, contiene registros continuos de EEG del cuero cabelludo de 24 niños con convulsiones intratables, que han sido etiquetados por profesionales médicos.
- Placa GAPuino desarrollada por GreenWaves Technology. La unidad de procesamiento principal es un GAP8, que es un procesador RISC-V multinúcleo derivado de la plataforma PULP.

- Arquitectura CNN basaDA en la CNN de SeizureNet, que se muestra en la Figura[13].
- Para implementar la CNN entrenada en MATLAB en la placa GAPuino se utilizó el SDK de TensorFlow.

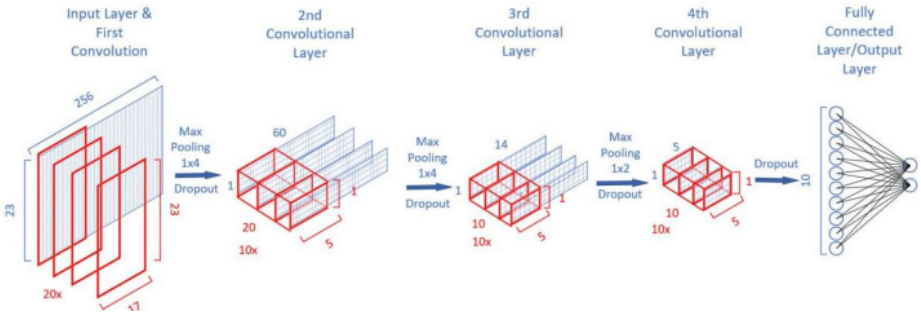


Figura 13: Representación esquemática de la arquitectura CNN que muestra las capas convolucionales con sus respectivas matrices de entrada (rectángulos azules) y núcleos (rectángulos rojos). Tomada de [Bahr, Schneider, Francis, Lehmann, Barg, Buschhoff, Wulff, Strunskus and Faupel, 2021]

4 Metodología desarrollada

4.1 Fase 1 Extracción de datos

La obtención de datos neuronales es un campo de investigación en constante evolución que ha revolucionado la forma en que se comprende y estudia el cerebro humano. En este contexto, OpenBCI Cyton, un dispositivo de interfaz cerebro-computadora (BCI), ha emergido como una herramienta esencial para la adquisición de datos neuronales. Utilizando Python, un lenguaje de programación ampliamente utilizado en la comunidad de ciencias de datos y neurociencias, se ha vuelto más accesible que nunca acceder y analizar datos provenientes de este dispositivo. Esto implica una serie de pasos que van desde la configuración inicial del hardware y el software hasta la adquisición y el procesamiento de señales cerebrales.

En este caso se tienen varias etapas para lograr la obtención de datos, OpenBCI Cyton se conecta a través de los electrodos al cuero cabelludo para capturar las señales EEG. Se pueden usar electrodos secos o electrodos con gel conductivo, estos datos serán visualizados en el Software OpenBCI GUI propio de OpenBCI que permite visualizar y grabar los datos capturados por el dispositivo. Esta interacción se hará mediante la biblioteca de python PyOpenBCI, es necesario tener instalado python con todos los complementos en el sistema donde se tomaran los datos.

4.1.1 Elección hardware-software

La elección de las herramientas y tecnologías adecuadas se ha convertido en un factor crítico para el éxito de este proyecto. Dado que nos encontramos en el terreno de las tecnologías emergentes, la documentación sólida y el soporte de las herramientas de software y hardware desempeñan un papel esencial en la consecución de los objetivos de investigación.

Se ha optado por un enfoque que considera cuidadosamente la documentación, la compatibilidad y la facilidad de acceso como criterios principales para la selección de herramientas clave:

- El uso de Cyton como herramienta de toma de datos se basa en su documentación detallada, su compatibilidad con otros componentes y su accesibilidad. La colaboración con el director de la monografía que proporcionó esta herramienta es un activo valioso.
- Arduino Nano 33 BLE se ha seleccionado como el microcontrolador principal debido a su documentación sólida, su compatibilidad con Cyton y TensorFlow Lite, y su amplia aceptación en la comunidad de Arduino.
- Python, como lenguaje principal, se ha elegido por su prominencia en la comunidad de software libre y su capacidad para integrarse con las herramientas de aprendizaje automático.
- TensorFlow se destaca como el framework de desarrollo para las redes neuronales debido a su popularidad, su documentación exhaustiva y su capacidad para convertir modelos a formatos binarios. La usabilidad y la capacidad de personalización de

TensorFlow facilitan la adaptación de la red neuronal a las necesidades específicas del proyecto.

En resumen, la elección de estas herramientas y tecnologías ha sido guiada por la premisa de que una base sólida de documentación y soporte es esencial para abordar con éxito un proyecto de investigación en el campo de las señales EEG y el procesamiento de gestos. La combinación de estas herramientas proporciona un enfoque robusto y prometedor para alcanzar los objetivos de la tesis, a pesar de los desafíos inherentes a las tecnologías emergentes y las limitaciones de hardware.

4.1.2 Conexión cyton con entorno OpenBCI-GUI

Conectar OpenBCI Cyton con el entorno OpenBCI-GUI es un proceso esencial para la adquisición y visualización de datos EEG en tiempo real. Enseguida se describe el proceso de conexión.

Para comenzar, es fundamental que el hardware esté correctamente conFigurado y alimentado. OpenBCI Cyton debe estar conectado a los electrodos colocados en el cuero cabelludo del sujeto de estudio, listo para capturar señales cerebrales, como el electroencefalograma (EEG). La siguiente etapa implica la instalación de OpenBCI-GUI en el sistema. Se debe hacer descargando la aplicación desde el sitio web oficial de OpenBCI y siguiendo las instrucciones de instalación proporcionadas. Una vez que el software esté en su lugar, se conecta físicamente OpenBCI Cyton a la computadora mediante el cable USB suministrado. Asegurándose de que el dispositivo esté encendido y listo para establecer una conexión.

Con el OpenBCIGUI ejecutado, lo siguiente es probar la conexión con la tarjeta Cyton, para ello se debe dar clic en la opción LIVE(from Cyton) → SERIAL FROM DONGLE, cabe aclarar que el dongle de Cyton debe estar conectado en el ordenador, dar clic en AUTO y por último START SESSION. En algunas ocasiones, el sistema no conecta directamente, por lo que es necesario dar clic en la opción MANUAL, seleccionar el puerto en donde esta conectado el dongle, AUTOSCAN, y por último START SESSION. Al hacerlo, habrá quedado preparado para establecer la conexión entre el dispositivo y la aplicación.

Con la conexión exitosa, se podrán visualizar los datos EEG en tiempo real a través de OpenBCI-GUI, lo que permitirá monitorear la actividad cerebral de manera efectiva.
Es importante conectar el Dongle antes de encender la placa Cyton. En ocasiones, si el flujo de datos parece interrumpido, es posible que deba desconectar el dongle USB y apagar la placa Cyton. Asegúrese de conectar primero el Dongle USB, luego encender la placa.

4.1.3 Conexión Cyton con Python

Como se hablo anteriormente es necesario tener la librería de python PyOpenBCI, esto se logra mediante la instalación de la biblioteca a través de pip, lo que permite interactuar con el dispositivo OpenBCI Cyton desde Python.

Esta librería permitirá lo siguiente:

- Adquirir los datos de cada dispositivo Open BCI: Ganglion, Cyton, Cyton con Daisy.
- Se puede ejecutar en cualquier sistema operativo que tenga instalado Python
- Visualizar por medio de gráficas 8 o 16 canales (dependiendo de el dispositivo, en este caso 8 canales) los datos adquiridos en tiempo real.
- Visualizar la gráfica de la transformada de Fourier de las señales adquiridas.
- Visualizar las bandas de frecuencias (Delta, Theta, Alfa, Beta).

4.1.4 Software EEG

Debido a que la librería PyOpenBCI carece de algunas de las funcionalidades requeridas para la realización del proyecto, se utiliza la aplicación diseñada en uno de los trabajos de referencia [Rodríguez and Sánchez, 2021], el cual brinda la facilidad de algunas funciones adicionales a las encontradas en la librería base. El Software EEG and EMG Detection usado permite a través de su interfaz de usuario realizar las siguientes funciones:

- Verifica que exista comunicación con el dongle y la tarjeta Cyton + Daisy.
- Visualización en tiempo real de los datos adquiriros.
- Crear bases de datos de cada usuario en donde se almacena información básica del mismo, consentimiento informado, carpetas individuales con los datos en archivo .CSV de cada gesto.
- Selección del gesto a realizar.
- Contador de sesión realizadas en cada gesto.
- Temporizador modificable por el usuario para indicar el tiempo de duración de la toma de datos de un gesto.
- Bocina indicativa de la ejecución del gesto al segundo 2 iniciado el temporizador.
- Gráficas de las señales capturadas.

4.1.5 Artefactos Fisiológicos

Los artefactos fisiológicos son distorsiones o señales no deseadas presentes en los datos recogidos durante procedimientos médicos o experimentos de investigación que son causadas por procesos naturales del cuerpo. Estos artefactos pueden provenir de varias fuentes internas del cuerpo y son cruciales de identificar y manejar para asegurar la precisión de las interpretaciones de los EEG.
Los artefactos fisiológicos que mas influyen en las señales EEG son:

- Artefactos Oculares: Provienen de los movimientos oculares y el parpadeo. Dado que los músculos oculares están cerca del cerebro, sus actividades eléctricas pueden ser recogidas por los electrodos del EEG, causando artefactos que pueden parecerse a la actividad cerebral real.

- Artefactos Musculares (Electromiografía, EMG): Los músculos del cuero cabelludo, cuello y cara, al contraerse, generan señales eléctricas que pueden ser detectadas por el EEG. Estos artefactos suelen ser de mayor frecuencia y amplitud que las señales cerebrales y pueden enmascarar o distorsionar la verdadera actividad cerebral.
- Artefactos Cardíacos (Electrocardiografía, ECG): Las señales eléctricas del corazón también pueden ser captadas por los electrodos del EEG. Aunque no son tan comunes como los artefactos oculares o musculares, pueden influir en los datos, especialmente cuando los electrodos están colocados cerca del cuello o la cabeza.

4.1.6 Gestos

Se tienen tres gestos para el desarrollo del proyecto, los cuales son parpadeo con cada ojo y el levantar las cejas, adicionalmente se tiene un gesto de control que es apretar la mandíbula, para cada uno de estos gestos se encuentran los canales que traen la información mas significativa con fin de minimizar la complejidad de la red a implementar:

- Parpadeo ojo izquierdo: En este gesto se encuentra que los canales que aportan información con amplitudes mas elevadas son los siguientes: CH1,CH7,CH9,CH13.
- Parpadeo ojo derecho: En este gesto se encuentra que los canales que aportan información con amplitudes mas elevadas son los siguientes: CH2,CH10,CH15.
- Levantar cejas: En este gesto se encuentra que los canales que aportan información con amplitudes mas elevadas son los siguientes: CH1,CH2,CH3,CH9,CH10.

Se tiene entonces que los canales que presentan información significativa en los gestos son 8: CH1,CH2,CH3,CH7,CH9,CH10,CH13,CH15. Canales con los cuales se trabajara en el proyecto.

4.1.7 Preprocesamiento

La toma de datos EEG con OpenBCI Cyton y Python es un paso fundamental en la exploración de la actividad cerebral y la interfaz cerebro-computadora. Sin embargo, los datos capturados por el dispositivo pueden estar sujetos a una serie de ruidos y artefactos que requieren un procesamiento cuidadoso para su interpretación precisa. Una de las prácticas comunes para mejorar la calidad de los datos es la aplicación de filtros, que permiten enfocarse en las señales de interés y eliminar interferencias no deseadas.

En este caso se usan los siguientes, filtro Butterworth pasa bandas de orden 5, se utiliza para seleccionar un rango específico de frecuencias de las señales EEG y atenuar todas las demás. Esto es particularmente útil para capturar componentes de frecuencia específicos, como las ondas alfa y mu que se encuentran en el rango de frecuencias de 8-13Hz y que tienen significados particulares en la actividad cerebral, a demas el filtro se usa para a eliminar el nivel DC ya que Cyton + Daisy transmite cada una de las 8 señales con un nivel DC distinto.

Por otro lado, el Filtro Notch tiene una función crítica en la eliminación de la interferencia de la frecuencia de línea de CA, 60 Hz. Esta interferencia eléctrica puede introducir ruido

no deseado en los datos EEG, y el filtro notch es esencial para eliminar esta frecuencia no cerebral y obtener señales limpias y más precisas.

4.1.8 Base De Datos

El experimento se lleva a cabo con usuarios sanos, los cuales se someten a 20 sesiones por cada gesto, cada sesión tiene una duración de 5 segundo, se realiza un descanso entre 5-10 minutos entre gestos.
El usuario se debe sentar en una silla normal, reposando los brazos sobre las piernas y con el cuerpo recto. Se le pedirá al usuario llenar un formulario con algunos datos básicos y de autorización para el uso de los mismos.
Para cada usuario se realizan 4 tomas de gesto distintas lo que corresponde a un total de 40 sesiones. Todos los experimentos se realizarán en condiciones similares, manteniendo la homogeneidad a lo largo de todas las pruebas (lugar, hora del experimento, etc.).

El software al ejecutarse crear un usuario y genera una carpeta con el nombre asignado en el lugar donde se tiene guardado el Script de Python GUI_qt.py, dentro de esta carpeta se encuentra la carpeta para el dispositivo Ultracortex archivo de texto con los datos del usuario y un consentimiento informado, en la carpeta Ultracortex se genera una carpeta por gesto y dentro de esta se encuentra un archivo .csv por cada sesión generada.

Estas carpetas generadas son alojadas en un drive con un nombre especifico para que luego mediante un script y utilizando la librería pandas de python se transformen en un dataframe, el cual contiene todos los datos de las señales de los pacientes en los diferentes canales y su respectiva salida, esto se logra con el siguiente código de la Figura[14].

```
def getDataBySignal(gesto):
  vectorTitulos=[]
  listadoPacientes = ! ls -l drive/MyDrive/DriveAnterior/DatosPrestados/
  listadoPacientes = filterByPath(listadoPacientes,'Paciente_ID')
  for paciente in listadoPacientes:
    path="drive/MyDrive/DriveAnterior/DatosPrestados/"+paciente+"/"+gesto+"/Datos_CSV_ULTRACORTEX/"
    listOfDataByPath = ! ls -l $path
    listOfDataByPath = filterByPath(listOfDataByPath,'datos')
    for dato in listOfDataByPath:
      name = '/drive/My Drive/DriveAnterior/DatosPrestados/'+paciente+'/'+gesto+'/Datos_CSV_ULTRACORTEX/'+dato
      vectorTitulos.append(name)

  return vectorTitulos
```

Figura 14: Código para convertir la información en dataframe

Luego de tener la información de esta manera se procede a obtener la información relevante de los canales mencionados en la sección de Gestos, para esto se utiliza la función descrita en la Figura[15].

```python
def convertirOchoCanales(lista_original):
  indices_canales_deseados = [0, 1, 2, 6, 8, 9, 12, 14]
  # Crear una lista para almacenar las matrices convertidas
  lista_convertida = []

  # Recorrer cada elemento de la lista original
  for elemento in lista_original:
      # Seleccionar los canales deseados utilizando los índices
      matriz_convertida = elemento[:, indices_canales_deseados]

      # Agregar la matriz convertida a la lista convertida
      lista_convertida.append(matriz_convertida)


  return lista_convertida
```

Figura 15: Código para seleccionar los 8 canales relevantes

Una vez se tiene la información de los 8 canales mas relevantes se realiza la normalización de estos datos donde el pico mayor sera equivalente a 1 y el pico menor sera equivalente a -1, esto se logra con la función descrita en la Figura[16].

```python
def convertDatosMaxMin(datos):
  scaler = preprocessing.MinMaxScaler(feature_range=(-1, 1))
  datosConvert = []
  for i in range(len(datos)):
    datosConvert.append(scaler.fit_transform(datos[i].reshape((625, 8))))
  return datosConvert
```

Figura 16: Código donde se normaliza de 1 a -1

4.2 Fase 2 Generación de red

4.2.1 Ventajas de convolucional

La elección de la CNN para este proyecto no es casualidad. Diversos estudios y literatura especializada [Rodríguez and Sánchez, 2021] han mostrado que las CNNs tienen una capacidad excepcional para clasificar señales EEG. Estas señales, que provienen de la actividad eléctrica del cerebro, presentan patrones y características que pueden ser sutiles y complejas. Las CNNs, con su habilidad para detectar y aprender características locales, son especialmente aptas para manejar este tipo de datos.

Además, las capas convolucionales de una CNN trabajan de manera similar a cómo lo hacen las neuronas en el cerebro humano cuando procesan información visual. Estas capas aplican filtros a pequeñas regiones de los datos de entrada, permitiendo que la red detecte características locales, como bordes, texturas o patrones específicos. A medida que los datos avanzan a través de la red, las capas subsiguientes combinan y reconfiguran estas características locales para detectar patrones más complejos y abstractos.

Otra ventaja significativa de las CNNs es su capacidad para manejar datos de alta dimensionalidad. Las señales EEG, aunque son series temporales, pueden ser tratadas como

imágenes bidimensionales donde el eje x representa el tiempo y el eje y muestra las lecturas de los diferentes electrodos. Esta representación permite que las CNNs procesen las señales EEG de manera similar a cómo procesarían imágenes, aprovechando su capacidad para manejar grandes volúmenes de datos y extraer características relevantes.

Finalmente, las CNNs son invariantes a la posición. Esto significa que pueden detectar una característica o patrón en cualquier lugar de la señal, independientemente de su ubicación. Esta propiedad es esencial para las señales EEG, donde un patrón importante puede aparecer en diferentes momentos o en diferentes electrodos.

4.2.2 Optimizador Adam

El optimizador Adam es uno de los algoritmos de optimización más utilizados en el entrenamiento de redes neuronales debido a su eficacia y facilidad de uso. La razón detrás de su popularidad y eficacia se puede entender mejor al observar sus fundamentos matemáticos y cómo combina las ideas de otros dos optimizadores: AdaGrad y RMSProp.

AdaGrad adapta la tasa de aprendizaje para cada parámetro de forma individual, dando pasos más pequeños para parámetros con gradientes frecuentes y pasos más grandes para parámetros con gradientes infrecuentes. La fórmula básica para AdaGrad es:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \tag{3}$$

donde w_t son los pesos en el tiempo t, η es la tasa de aprendizaje, g_t es el gradiente del error con respecto a los pesos en el tiempo t, G_t es la suma de los cuadrados de los gradientes pasados hasta el tiempo t, y ϵ es un término de suavizado para evitar la división por cero.

RMSProp modifica el denominador al utilizar una media móvil exponencialmente decreciente del cuadrado de los gradientes en lugar de una suma acumulativa, lo cual ayuda a evitar el problema de la rápida disminución de la tasa de aprendizaje en AdaGrad:

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2 \tag{4}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t \tag{5}$$

donde v_t es el promedio móvil exponencial de los cuadrados de los gradientes y β es un hiper parámetro que controla la tasa de decaimiento.

Adam combina estas ideas adaptando la tasa de aprendizaje para cada peso individualmente con estimaciones del primer y segundo momento del gradiente:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{6}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{7}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{8}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{9}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t \tag{10}$$

Aquí, m_t y v_t son estimaciones del primer y segundo momento respectivamente, y $\hat{m}_t$ y $\hat{v}_t$ son correcciones para el sesgo inicial hacia cero de estas estimaciones.

4.2.3 Función de pérdida entropía cruzada

La función de pérdida de entropía cruzada categórica es la más comúnmente utilizada en tareas de clasificación multiclase donde las etiquetas se presentan en un formato de codificación "one-hot". Esta función de pérdida compara la distribución de probabilidad predicha por el modelo (a través de la activación softmax en la capa de salida) con la distribución de probabilidad real de las etiquetas (la representación "one-hot").

Una codificación "one-hot" es un vector que tiene tantos componentes como clases posibles, y en el cual todos los componentes son 0, excepto para la clase a la que pertenece la muestra, que es 1. Por ejemplo, si tenemos tres posibles clases para clasificar y una muestra pertenece a la segunda clase, la codificación "one-hot" sería $[0, 1, 0]$.

La entropía cruzada categórica mide el rendimiento de un modelo de clasificación cuyas salidas son valores de probabilidad entre 0 y 1. La entropía cruzada aumenta a medida que la predicción probabilística se desvía de la etiqueta real. La fórmula de la entropía cruzada para una distribución de probabilidad verdadera y y una predicción $\hat{y}$ es la siguiente:

$$L(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y_i}) \tag{11}$$

donde:

- C es el número total de clases.
- y es el vector de la codificación "one-hot" de la etiqueta verdadera.
- $\hat{y}$ es el vector de probabilidades predicho por el modelo.
- y_i es el valor binario (0 o 1) del i-ésimo componente en el vector de etiqueta "one-hot".
- $\hat{y_i}$ es la i-ésima componente de la probabilidad predicha para la clase i.

En la práctica, se utiliza el término $y_i \log(\hat{y_i})$ porque y_i será 1 para la clase correcta y 0 para las demás, por lo tanto solo el término correspondiente a la clase verdadera contribuirá a la suma. La pérdida es simplemente el negativo del logaritmo de la probabilidad predicha para la clase verdadera, proporcionando una medida directa del rendimiento del modelo.

4.2.4 Sobre entrenamiento

El sobreentrenamiento, o .ºverfitting", en redes neuronales convolucionales (CNN) ocurre cuando un modelo aprende no solo las características subyacentes y útiles de los datos de entrenamiento, sino también el ruido y las peculiaridades específicas de ese conjunto de datos. Esto resulta en un modelo que tiene un rendimiento excepcionalmente bueno en los datos de entrenamiento pero falla al generalizar y realizar predicciones precisas en datos nuevos no vistos previamente, es decir, los datos de validación y prueba.

Para entender cómo ocurre esto, consideremos cómo se entrena una CNN. Durante el entrenamiento, la red ajusta sus pesos y sesgos para minimizar una función de pérdida, que mide la diferencia entre las predicciones del modelo y las etiquetas verdaderas. A medida que las épocas de entrenamiento (epochs) aumentan, el modelo continúa mejorando su

precisión en el conjunto de entrenamiento. Sin embargo, más allá de un cierto punto, estas mejoras en el conjunto de entrenamiento pueden no corresponder a mejoras en el conjunto de validación; en lugar de aprender representaciones generales, el modelo comienza a memorizar los datos.

Uno de los enfoques más efectivos para detectar y prevenir el sobre entrenamiento es monitorear el rendimiento del modelo no solo en los datos de entrenamiento, sino también en un conjunto de datos de validación durante el entrenamiento. Esto implica evaluar la red en el conjunto de validación después de cada época y registrar la precisión o la pérdida. Normalmente, en las primeras épocas, tanto la pérdida de entrenamiento como la de validación disminuyen, lo que indica que el modelo está aprendiendo. Sin embargo, en el momento en que la pérdida de validación deja de disminuir (o incluso comienza a aumentar) mientras que la pérdida de entrenamiento sigue disminuyendo, es una señal de que el modelo está comenzando a sobre entrenar.

Para contrarrestar el sobre entrenamiento una vez que se ha identificado, podemos tomar el punto en el que la pérdida de validación comienza a empeorar y utilizar esa cantidad de épocas o incluso un poco menos para el entrenamiento final. Este punto se conoce como el punto de detención temprana o early stopping. La lógica aquí es entrenar el modelo solo hasta que esté aprendiendo información generalizable y detener el entrenamiento antes de que comience a aprender el ruido específico del conjunto de datos de entrenamiento.

Para abordar el desafío específico, se llevaron a cabo pruebas preliminares utilizando los parámetros al límite máximo permitido, culminando en un modelo de 400,000 bytes. Después de una serie inicial de experimentos que incluyeron 50, 100, 300, 500 y 1000 épocas, se evidenció sobre entrenamiento para valores superiores a 100. Como se ilustra en la Figura[17], la exactitud (Accuracy) deja de incrementarse para los datos de validación (representados por la línea naranja) tras superar las 100 épocas, mientras que la exactitud en el conjunto de entrenamiento se aproxima a 1. En la gráfica de Pérdida (Loss), se observa que el error en los datos de validación no solo deja de disminuir, sino que empieza a incrementar, a diferencia del error en el conjunto de entrenamiento, que se mantiene cercano a cero, indicando que la red está memorizando los datos en lugar de generalizar. En consecuencia, se decidió ajustar el número de épocas a un rango más reducido de 25, 30, 40, 80, 100, y 150 para buscar el modelo óptimo variando estos parámetros.

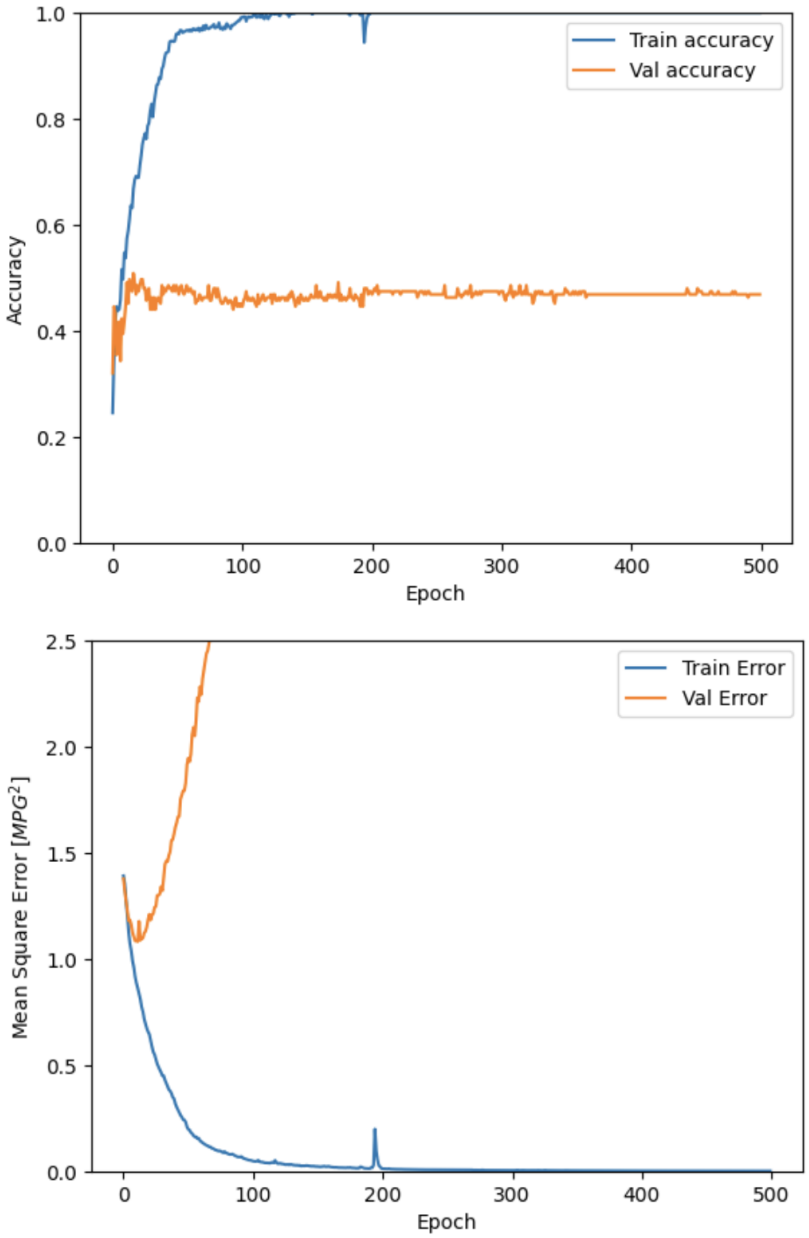


Figura 17: Gráfica de rendimiento modelo entrenado con 500 Epochs

4.2.5 Desarrollo de la red, código bucles

Inicialmente, el proyecto comenzó con una matriz de datos 16x625, que corresponde a 16 canales y 625 muestras temporales por canal. Tras un meticuloso análisis, se concluyó que solo 8 de estos canales son fundamentales para la identificación precisa de gestos, lo que llevó a una optimización de la matriz a 8x625. Esta matriz ajustada se maneja como una imagen bidimensional, lo que es esencial para la aplicación de técnicas de convolución, características de las redes neuronales convolucionales (CNN).

Debido a las limitaciones del microcontrolador, se variaron los parámetros Cantidad de filtros, tamaño del filtro, numero de neuronas de la red totalmente conectada, funciones

de activación, de la red convolucional y de la red totalmente conectada, de la siguiente manera.

Arquitectura del Modelo

Capa Convolucional:

- Número de filtros (neuronas): 1-10.
- Tamaño del filtro (kernel): 3×3, 5×5, 7×7.
- Función de activación: Se experimentó con relu, sigmoid y tanh.

Capa de Max Pooling:

- Tamaño del pool: 2×2.

Capa de Aplanado.

Capa Densa (Totalmente Conectada):

- Número de neuronas: [1-10].
- Función de activación: Se experimentó con relu, sigmoid y tanh.

Capa de Salida:

- Número de neuronas: 4.
- Función de activación: softmax.

En la Figura[18], se ilustran los parámetros que se ajustarán, dispuestos en vectores para su fácil invocación en el código. La cantidad de filtros en la capa convolucional oscila entre 1 y 10, y se exploran tamaños de kernel de 3x3, 5x5 y 7x7. Aunque la función de activación relu es la más utilizada en las CNN, se decidió experimentar también con sigmoid y tanh, que son comúnmente empleadas. Para la capa de Max Pooling, se seleccionó un tamaño de filtro de 2x2, que efectivamente reduce la dimensionalidad a la mitad. Seguidamente, se implementa una capa densa totalmente conectada con una cantidad variable de neuronas, de 1 a 10, y se evalúa la influencia de las funciones de activación sigmoid, relu y tanh. Respecto al número de épocas de entrenamiento, y para evitar el sobre entrenamiento como se mencionó en la sección anterior, se considerará una gama que incluye 25, 30, 40, 80, 100 y 150. Finalmente, se emplea una capa de salida con la función de activación softmax para asignar una ponderación equitativa a cada categoría de la clasificación y facilitar la aplicación efectiva de la función de pérdida de entropía cruzada.

```
# Parametros a variar
Conv= [1,2,3,4,5,6,7,8,9,10]
FlattenCap = [1,2,3,4,5,6,7,8,9,10]
functionsOne = ["relu","tanh","sigmoid"]
functionsTwo = ["relu","tanh","sigmoid"]
StepsTrain = [25,30,40,80,100,150]
filterConv = [3,5,7]
```

Figura 18: Parámetros que se varían para encontrar el mejor modelo.

En la Figura[19], se observa, el uso de varios bucles for donde se varían los parámetros acorde a los valores establecidos en la Figura[18], se tiene en cuenta que no se variaron con valores mas altos debido a las limitaciones del microcontrolador, el tamaño máximo aproximado del tamaño de la red que se lograba implementar era de 400.000 bytes, con este código luego de ejecutarse se tiene que son 16.200 modelos entrenados, de los que se obtiene el mejor modelo.

```
# For anidados acorde a los parametros a variar
modelsComplete = []
contador = 0
for i in Conv:
  for a in FlattenCap:
    for functionO in functionsOne:
      for functionT in functionsTwo:
        for filterConvValue in filterConv:
          for steps in StepsTrain:
            contador = contador + 1
            print("contador ", contador)
            model = Sequential()
            model.add(Conv2D(i, (filterConvValue, filterConvValue), activation=functionO, input_shape=(625, 8, 1)))
            model.add(MaxPooling2D((2, 2)))
            model.add(Flatten())
            model.add(Dense(a, activation=functionT))
            model.add(Dense(num_classes, activation='softmax'))

            # Compilar el modelo
            model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

            # Entrenar el modelo
            model.fit(X_train, y_train, epochs=steps, validation_data=(X_test, y_test), verbose=False)

            # Evaluar el modelo en el conjunto de prueba
            loss, accuracy = model.evaluate(X_test, y_test)
            print("Loss: %.2f, Accuracy: %.2f" % (loss, accuracy))
            # Convertir el modelo a TensorFlow Lite
            converter = tf.lite.TFLiteConverter.from_keras_model(model)
            tflite_model = converter.convert()

            # Guardar el modelo TensorFlow Lite en disco
            tflite_file_path = f"model_not_quantized_{contador}.tflite"
            open(tflite_file_path, "wb").write(tflite_model)

            # Obtener el tamaño del modelo TensorFlow Lite en bytes
            model_size = os.path.getsize(tflite_file_path)
            # Agregar la información a la lista de modelos completos
            modelsComplete.append({
                "Loss": loss,
                "Accuracy": accuracy,
                "Model": model,
                "Conv2D": i,
                "Flatten": a,
                "FunctionOne": functionO,
                "FunctionTwo": functionT,
                "Steps": steps,
                "ModelSizeBytes": model_size
            })
```

Figura 19: Bucles donde se ejecutan todos los posibles modelos.

Construcción y Evaluación del Modelo Desde el Código

Como se muestra el código en la Figura[19], para cada combinación única de parámetros, se construye un modelo secuencial de la siguiente manera:

- Se añade una capa convolucional `Conv2D` con i filtros y un kernel de tamaño filterConvValue, filterConvValue, utilizando `functionO` como la función de activación y especificando la forma de entrada si es la primera capa.
- Una capa de `Max Pooling` con tamaño $(2, 2)$ se utiliza para reducir la dimensionalidad.
- Una capa `Flatten` convierte los mapas de características 2D en un vector 1D para facilitar el procesamiento por capas densas.
- Se añade una capa densa con a neuronas y `functionT` como función de activación.
- La capa de salida es una capa densa con 4 neuronas (debido a `num_classes = 4`) y utiliza la función de activación softmax, que es adecuada para problemas de clasificación multiclase.

Compilación del Modelo:

El modelo se compila con el optimizador adam, la función de pérdida categorical crossentropy, que es estándar para la clasificación multiclase, y se monitorea la métrica accuracy.

Entrenamiento del Modelo:

Se entrena el modelo utilizando los conjuntos de datos de entrenamiento y validación con un número variable de épocas definido por `StepsTrain`.

Evaluación del Modelo:

Después de entrenar el modelo, se evalúa utilizando el conjunto de prueba para obtener métricas de pérdida y precisión.

Conversión a TensorFlow Lite:

El modelo entrenado se convierte al formato TensorFlow Lite, que es más adecuado para dispositivos con recursos limitados como teléfonos móviles y dispositivos IoT.

Guardado del Modelo y Recopilación de Información:

El modelo en formato TensorFlow Lite se guarda en el disco. Además, se recopila información sobre el rendimiento del modelo y los hiper parámetros utilizados para cada iteración, y se almacena en la lista `modelsComplete`. Esta información incluye la pérdida y precisión del modelo, el tamaño del archivo del modelo TensorFlow Lite, y los valores de los parámetros que dieron lugar a ese modelo (número de filtros, tamaño del filtro, funciones de activación, número de neuronas en la capa densa y número de épocas de entrenamiento).

Evaluando resultados

A partir de los datos recopilados en el vector `modelsComplete`, se procedió a identificar el modelo con la mayor precisión. El mejor modelo, como se muestra en la Figura[21], presenta una configuración de 8 filtros de 3×3, utiliza la función de activación relu para la convolución y cuenta con 9 neuronas en su capa densa la representacion grafica de la red se puede ver en la Figura[20]. Este modelo fue entrenado durante 80 épocas, alcanzando una precisión de 0.645714 y una pérdida de 1.773125, con un tamaño aproximado de 350,000 bytes. Considerando la simplicidad del modelo y los recursos limitados del microcontrolador, estos resultados son adecuados dentro del espectro de parámetros que se evaluaron.

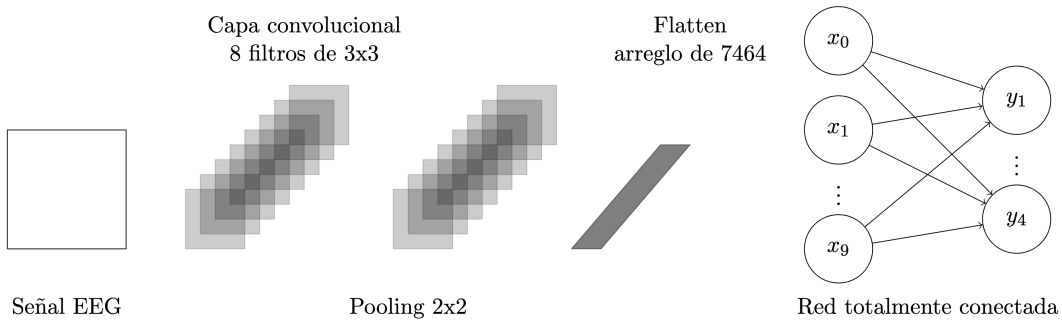


Figura 20: Representacion grafica del Mejor modelo

Loss	Accuracy	Model	Conv2D	Flatten	FunctionOne	FunctionTwo	Steps
1.773125	0.645714	<tensorflow.python.keras.engine.sequential.Seq...	8.0	9.0	relu	relu	80.0

Figura 21: Resultados del merjor modelo

Dado que se trata de un modelo de clasificación multiclase, la exactitud por sí sola no es suficiente para evaluar el desempeño del modelo, ya que no proporciona información sobre su comportamiento en cada clase individual. Por ello, se generó la matriz de confusión, presentada en la Figura[22]. Esta matriz revela un rendimiento equilibrado del modelo: la clase 0 corresponde al parpadeo del ojo izquierdo, la clase 1 al parpadeo del ojo derecho, la clase 2 al levantamiento de cejas, y la clase 3 a la señal de control, que representa la ausencia de señal. Se observa que el modelo confunde ligeramente entre las clases de parpadeo, mientras que distingue de manera efectiva el levantamiento de cejas y la señal de control. Estos resultados indican que el modelo posee un buen equilibrio y discriminación entre las clases bajo las condiciones dadas.

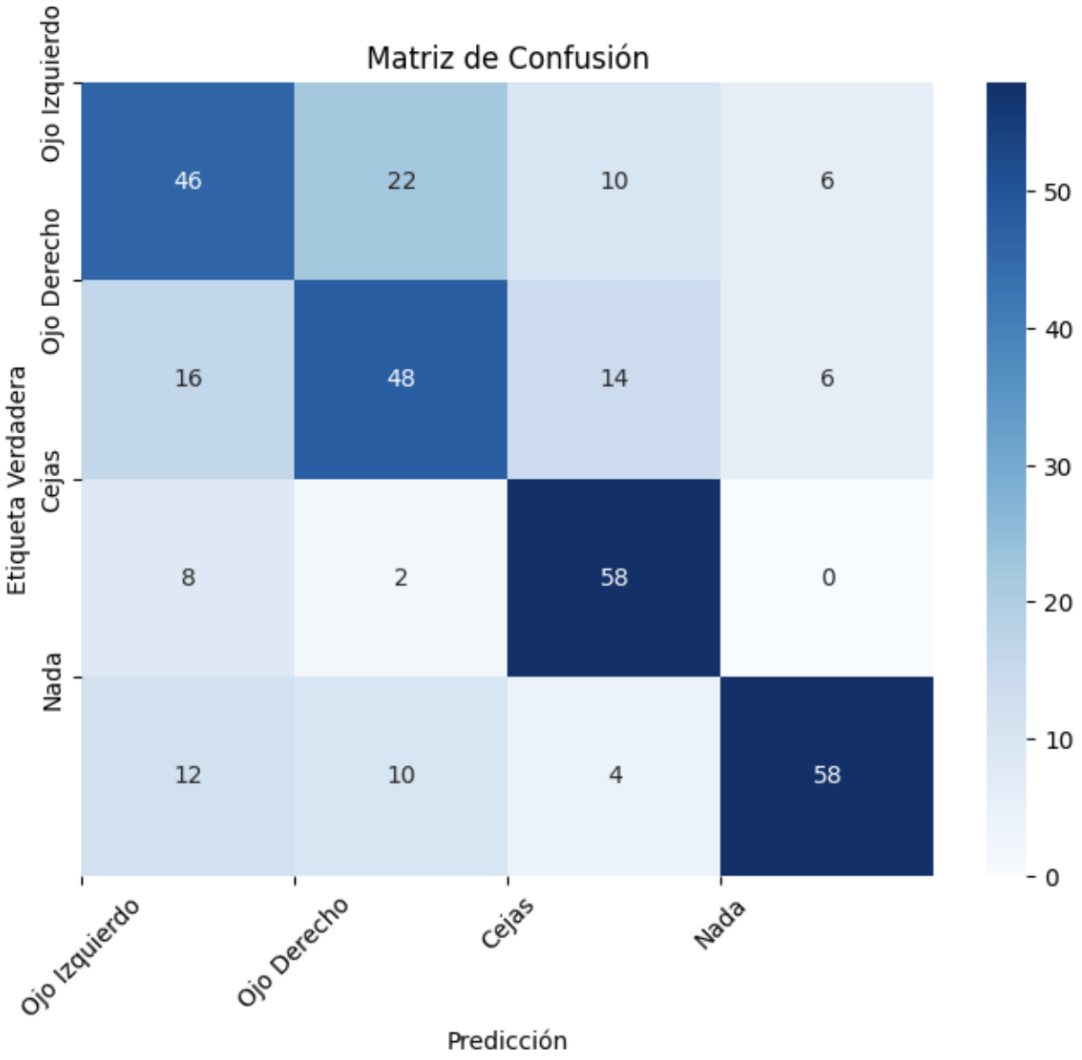


Figura 22: Matriz confusión mejor modelo

4.3 Fase 3 Implementación

4.3.1 Arduino nano 33 BLE

El Arduino Nano 33 BLE es una pequeña y potente placa de desarrollo basada en la nRF52840 de Nordic Semiconductor, una solución de 32 bits ARM Cortex-M4 CPU que opera a 64 MHz. Esta placa es parte de la familia Arduino Nano, conocida por su tamaño compacto, y está diseñada específicamente para proyectos de IoT (Internet de las Cosas) con un enfoque en la conectividad Bluetooth y las aplicaciones de baja energía.[Arduino, s.f]

Características Principales:

- Microcontrolador: nRF52840 de Nordic Semiconductor. CPU: ARM Cortex-M4 de 32 bits. Frecuencia de CPU: 64 MHz. Flash Memory: 1 MB. SRAM: 256 KB.

- Conectividad Inalámbrica: Bluetooth® 5: La placa incluye soporte para Bluetooth 5, lo que permite una mayor velocidad de transferencia de datos y un mayor rango en comparación con versiones anteriores de Bluetooth.
- Entradas y Salidas: Pines Digitales: 14 I/O (de los cuales 8 pueden usarse como salidas PWM). Pines Analógicos: 8 entradas. Salidas PWM: 8 (usando pines digitales). Voltaje de Operación de los Pines I/O: 3.3V.
- Interfaz de Programación: Se programa a través del IDE de Arduino, utilizando un cable USB. La placa es compatible con una amplia gama de bibliotecas Arduino y ejemplos de código.
- Sensores Integrados: Incluye un giroscopio, un acelerómetro y un magnetómetro, lo que la hace ideal para proyectos que involucran seguimiento de movimiento o navegación.
- Alimentación: La placa puede ser alimentada a través de la conexión USB o con una batería externa, lo que es conveniente para proyectos portátiles o embebidos.
- Compatibilidad con Voltaje: La placa opera a 3.3V, lo que es importante considerar al conectar sensores o módulos que requieren un voltaje específico.[Arduino, s.f]

El Arduino Nano 33 BLE es una opción atractiva para tareas de sintetización de redes neuronales por varias razones, especialmente si se compara con otros microcontroladores que no están orientados a aplicaciones de IoT o que carecen de capacidades de procesamiento de bajo consumo. Aquí están algunos de los factores clave:

Potente Microcontrolador: Equipado con el nRF52840 de Nordic Semiconductor, el Arduino Nano 33 BLE cuenta con un procesador ARM Cortex-M4 de 32 bits con FPU (Floating Point Unit), lo que le permite manejar cálculos complejos que son típicos en algoritmos de redes neuronales.

Bluetooth Low Energy (BLE): La conectividad BLE integrada permite que el dispositivo se comunique de manera eficiente con otros dispositivos, como smartphones, tablets o incluso otros Arduinos, lo que es esencial para aplicaciones IoT.

Capacidades de Bajo Consumo: Su eficiencia energética lo hace ideal para aplicaciones portátiles y de campo, donde la gestión de la energía es crucial.

Soporte para TensorFlow Lite: El Arduino Nano 33 BLE es compatible con TensorFlow Lite, una versión ligera de TensorFlow diseñada para dispositivos móviles y embebidos. Esto permite implementar modelos de aprendizaje automático que han sido entrenados y optimizados para ejecutarse en hardware con recursos limitados.

Sensores Integrados: Viene con sensores incorporados como acelerómetro, giroscopio y magnetómetro, que pueden ser útiles para proyectos que requieren reconocimiento de patrones o procesamiento de señales en tiempo real.

Comunidad y Soporte: Arduino tiene una gran comunidad de usuarios y desarrolladores, lo que significa que hay una gran cantidad de bibliotecas y ejemplos disponibles. Esto

puede acelerar el desarrollo y la solución de problemas.

Portabilidad y Factor de Forma: Su tamaño compacto lo hace ideal para integrar en dispositivos donde el espacio es limitado.

Facilidad de Uso y Programación: La programación se realiza a través del conocido Arduino IDE, que es accesible para principiantes, mientras que también ofrece funcionalidades avanzadas para usuarios más experimentados.

4.3.2 Convertir modelo a binario con tensor flow lite

La sección previa documentó el desarrollo de un modelo con una precisión del 64 %, una configuración que no corresponde a la de mayor complejidad ni implica el uso de la máxima cantidad de épocas de entrenamiento, lo cual podría contradecir suposiciones iniciales. Dicho modelo fue inicialmente construido y afinado dentro del marco de Keras, que opera sobre la infraestructura de TensorFlow, una configuración típica para la implementación de redes neuronales en entornos de servidores web que facilitan el acceso a través de interfaces de programación de aplicaciones (APIs) remotas.

No obstante, el propósito de esta investigación es la implementación del modelo en un entorno de microcontrolador para ejecución de inferencias en el borde (edge computing), independiente de conectividad de red. Esto implica la transición del modelo de una arquitectura de servidor a un sistema embebido. Para ello, se ha recurrido a TensorFlow Lite, una biblioteca especializada contenida en el ecosistema de TensorFlow destinada al entorno Python, que proporciona herramientas para convertir modelos a un formato binario. Este formato es esencial para la compatibilidad con el lenguaje de programación C++, ampliamente adoptado en el desarrollo de firmware para microcontroladores.

El hardware seleccionado para la aplicación es el Arduino Nano 33 BLE, elegido por su compatibilidad con las características del modelo y su arquitectura ARM Cortex-M4, la cual ofrece capacidades de cómputo suficientes para la carga y ejecución de modelos de machine learning de tamaño moderado.

Para la conversión del modelo entrenado a un formato adecuado para el microcontrolador, se emplea un script específico en Python que facilita la binarización del modelo. La sintaxis de dicho script, detallada posteriormente, refleja el proceso de transformación del modelo y se encuentra ilustrada en la Figura[23].

```python
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Guardar el modelo TensorFlow Lite en disco
tflite_file_path = f"model_not_quantized_{contador}.tflite"
open(tflite_file_path, "wb").write(tflite_model)

# Obtener el tamaño del modelo TensorFlow Lite en bytes
model_size = os.path.getsize(tflite_file_path)
```

Figura 23: Código python binarizar modelo de keras

- `converter = tf.lite.TFLiteConverter.from_keras_model(model)`: Esta línea crea un convertidor para transformar un modelo de Keras (`model`) en su equivalente en

formato TensorFlow Lite. `tf.lite.TFLiteConverter` es una clase proporcionada por TensorFlow que tiene varios métodos de clase para la conversión de modelos. El método `from_keras_model` se usa para convertir modelos de Keras.

- `tflite_model = converter.convert()`: Una vez que se ha inicializado el convertidor con el modelo de Keras, esta línea lo utiliza para convertir efectivamente el modelo al formato TensorFlow Lite. El método `.convert()` realiza la conversión y `tflite_model` es una variable que ahora contiene el modelo convertido en un formato de datos binarios listo para ser guardado en un archivo.
- `tflite_file_path = f"model_not_quantized_{contador}.tflite"`: Esta línea crea una cadena de texto que será utilizada como el nombre del archivo para el modelo TensorFlow Lite. La cadena se forma utilizando la interpolación de cadenas de Python (f-string), donde el valor de la variable `contador` se inserta en el nombre del archivo, lo que ayuda a identificar el modelo específico y asegura que el nombre del archivo sea único.
- `open(tflite_file_path, "wb").write(tflite_model)`: Esta línea abre (o crea si no existe) un archivo en el modo de escritura binaria (`"wb"`) con la ruta y el nombre especificado por `tflite_file_path`. Luego escribe los datos del modelo TensorFlow Lite (`tflite_model`) en este archivo. Este paso guarda efectivamente el modelo en el disco duro para su uso posterior.

4.3.3 Crear archivo .model

Tras la binarización del modelo, resultando en el archivo `model_not_quantized_355.tflite`, se procede a la creación de los archivos `model.cpp` y `model.h`. Estos serán integrados en el entorno de desarrollo de Arduino. El archivo `model.h` incluye las declaraciones necesarias que, en este contexto, definen dos componentes principales: una matriz de bytes que encapsula el modelo de aprendizaje automático y un entero que especifica la longitud de dicha matriz. La estructura de `model.h` se ilustra en la Figura[24].

```
#ifndef TENSORFLOW_LITE_MICRO_EXAMPLES_HELLO_WORLD_MODEL_H_
#define TENSORFLOW_LITE_MICRO_EXAMPLES_HELLO_WORLD_MODEL_H_

extern const unsigned char g_model[];
extern const int g_model_len;

#endif
```

Figura 24: Contenido del archivo `model.h` que declara la matriz del modelo y su longitud.

Por su parte, `model.cpp` proporciona las definiciones correspondientes a las declaraciones en `model.h`. En particular, este archivo contiene los datos del modelo asignados a la variable `g_model`, como se demuestra en la Figura[25], y el tamaño del modelo asignado a la variable `g_model_len`, que en este ejemplo es 271772. Con la inclusión de estos archivos, el modelo puede ser invocado por el script principal `convolutional_model_eeg.ino` de Arduino.

```
#include "model.h"

alignas(8) const unsigned char g_model[] = {
 0x1C, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4C, 0x33, 0x14, 0x00, 0x20, 0x00, 0x04, 0x00, 0x08, 0x00,
   0x0C, 0x00, 0x10, 0x00, 0x14, 0x00, 0x00, 0x00, 0x18, 0x00, 0x1C, 0x00, 0x14, 0x00, 0x00, 0x00,
   0x03, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x2C, 0x00, 0x00, 0x00, 0xC8, 0x00, 0x00, 0x00,
   0x2C, 0x00, 0x00, 0x00, 0x70, 0x00, 0x00, 0x00, 0x68, 0x00, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00,
   0x74, 0x05, 0x00, 0x00, 0x88, 0x04, 0x00, 0x00, 0xB8, 0x03, 0x00, 0x00, 0xDC, 0x02, 0x00, 0x00,
   0x84, 0x01, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0xC0, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00,
   0x38, 0x25, 0x04, 0x00, 0x34, 0x25, 0x04, 0x00, 0x20, 0x24, 0x04, 0x00, 0x80, 0x23, 0x04, 0x00,
   0x00, 0x23, 0x04, 0x00, 0x8C, 0x22, 0x04, 0x00, 0x94, 0x08, 0x00, 0x00, 0xAC, 0x07, 0x00, 0x00,
   0x28, 0x06, 0x00, 0x00, 0x14, 0x25, 0x04, 0x00, 0x10, 0x25, 0x04, 0x00, 0x0C, 0x25, 0x04, 0x00,
   0x08, 0x25, 0x04, 0x00, 0x04, 0x25, 0x04, 0x00, 0x00, 0x25, 0x04, 0x00, 0x3C, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0C, 0x00,
   0x04, 0x00, 0x08, 0x00, 0x08, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x0F, 0x00, 0x00, 0x00,
   0x13, 0x00, 0x00, 0x00, 0x6D, 0x69, 0x6E, 0x5F, 0x72, 0x75, 0x6E, 0x74, 0x69, 0x6D, 0x65, 0x5F,
   0x76, 0x65, 0x72, 0x73, 0x69, 0x6F, 0x6E, 0x00, 0x56, 0xDC, 0xFB, 0xFF, 0x04, 0x00, 0x00, 0x00,
   0x10, 0x00, 0x00, 0x00, 0x31, 0x2E, 0x35, 0x2E, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x0F, 0x00, 0x00, 0x00, 0x4D, 0x4C, 0x49, 0x52, 0x20, 0x43, 0x6F, 0x6E,
   0x76, 0x65, 0x72, 0x74, 0x65, 0x64, 0x2E, 0x00, 0x00, 0x00, 0x0E, 0x00, 0x18, 0x00, 0x04, 0x00,
   0x08, 0x00, 0x0C, 0x00, 0x10, 0x00, 0x14, 0x00, 0x0E, 0x00, 0x00, 0x00, 0x14, 0x00, 0x00, 0x00,
   0x4C, 0x00, 0x00, 0x00, 0x50, 0x00, 0x00, 0x00, 0x54, 0x00, 0x00, 0x00, 0x6C, 0x00, 0x00, 0x00,
   0x0E, 0x00, 0x00, 0x00, 0x04, 0x24, 0x04, 0x00, 0x8C, 0x23, 0x04, 0x00, 0xE0, 0x22, 0x04, 0x00,
   0x4C, 0x22, 0x04, 0x00, 0xE0, 0x21, 0x04, 0x00, 0x70, 0x21, 0x04, 0x00, 0x78, 0x07, 0x00, 0x00,
   0x84, 0x06, 0x00, 0x00, 0x70, 0x04, 0x00, 0x00, 0x84, 0x03, 0x00, 0x00, 0xB4, 0x02, 0x00, 0x00,
   0xD8, 0x01, 0x00, 0x00, 0x08, 0x01, 0x00, 0x00, 0x7C, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x0D, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00,
   0xE0, 0x03, 0x00, 0x00, 0xF8, 0x02, 0x00, 0x00, 0x54, 0x02, 0x00, 0x00, 0x5C, 0x01, 0x00, 0x00,
   0xA8, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x6D, 0x61, 0x69, 0x6E,
   0x00, 0x00, 0x00, 0x00, 0x7A, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x09, 0x04, 0x00, 0x00, 0x00,
   0x1C, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x3A, 0xDD, 0xFB, 0xFF,
   0x00, 0x00, 0x80, 0x3F, 0x01, 0x00, 0x00, 0x00, 0x0D, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00,
   0x0C, 0x00, 0x00, 0x00, 0x2C, 0xFC, 0xFF, 0xFF, 0x19, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x19,
   0x01, 0x00, 0x00, 0x00, 0xC0, 0xDC, 0xFB, 0xFF, 0x14, 0x00, 0x00, 0x00, 0x0E, 0x00, 0x00, 0x00,
   0x18, 0x00, 0x00, 0x00, 0x30, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
   0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x08, 0x00, 0x00, 0x00, 0x49, 0x64, 0x65, 0x6E,
   0x74, 0x69, 0x74, 0x79, 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
   0x04, 0x00, 0x00, 0x00, 0xAC, 0xDD, 0xFB, 0xFF, 0x00, 0x00, 0x0E, 0x00, 0x18, 0x00, 0x08, 0x00,
   0x0C, 0x00, 0x10, 0x00, 0x07, 0x00, 0x14, 0x00, 0x0E, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x08,
   0x03, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
   0xBC, 0xDC, 0xFB, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
   0x0B, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x48, 0xDD, 0xFB, 0xFF,
   0x14, 0x00, 0x00, 0x00, 0x0D, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x68, 0x00, 0x00, 0x00,
   0x58, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
   0x40, 0x00, 0x00, 0x00, 0x73, 0x65, 0x71, 0x75, 0x65, 0x6E, 0x74, 0x69, 0x61, 0x6C, 0x5F, 0x32,
   0x39, 0x35, 0x2F, 0x64, 0x65, 0x6E, 0x73, 0x65, 0x5F, 0x35, 0x39, 0x32, 0x2F, 0x4D, 0x61, 0x74,
   0x4D, 0x75, 0x6C, 0x3B, 0x73, 0x65, 0x71, 0x75, 0x65, 0x6E, 0x74, 0x69, 0x61, 0x6C, 0x5F, 0x32,
   0x39, 0x35, 0x2F, 0x64, 0x65, 0x6E, 0x73, 0x65, 0x5F, 0x35, 0x39, 0x32, 0x2F, 0x42, 0x69, 0x61,
   0x73, 0x41, 0x64, 0x64, 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0xFF, 0xFF, 0xFF, 0xFF,
   0x04, 0x00, 0x00, 0x00, 0x50, 0xDD, 0xFB, 0xFF, 0x7A, 0xFE, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x08,
   0x03, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00, 0x0C, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x06, 0x00, 0x08, 0x00, 0x07, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
   0x01, 0x00, 0x00, 0x00, 0x0B, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00, 0x0A, 0x00, 0x00, 0x00,
```

Figura 25: El archivo `model.cpp` que inicializa la matriz del modelo y su longitud.

4.3.4 Código de arduino para inferencia

Para satisfacer los requisitos del proyecto, el guión (script) fue desarrollado de la siguiente manera: en la fase inicial, se procede a invocar una serie de bibliotecas indispensables para la operatividad de TensorFlow Lite en ambientes de microcontroladores, tal como se detalla en la Figura[26]. Dichas bibliotecas comprenden una variedad de archivos de cabecera que proporcionan acceso a las clases y funciones vitales para la interpretación y ejecución de modelos de aprendizaje automático en la plataforma Arduino.

Se integra también el archivo model.h, que alberga la forma binarizada del modelo de aprendizaje automático junto con su longitud calculada. Además, se lleva a cabo la inicialización de variables clave destinadas a registrar errores y gestionar el "tensor arena", que es un bloque de memoria asignado específicamente para el procesamiento de cálculos intermedios requeridos durante la ejecución del modelo. El tamaño de este espacio de memoria no se determina mediante un procedimiento estándar; más bien, se ajusta experimentalmente, multiplicando un valor base por 1024 y aumentando este coeficiente hasta hallar una dimensión que permita el funcionamiento óptimo del modelo. El tamaño final de la arena de tensores variará en función de la complejidad inherente al modelo y se incrementará progresivamente hasta resolver cualquier error que surja al cargar el programa en el Arduino.

```
#include <TensorFlowLite.h>
#include <tensorflow/lite/micro/all_ops_resolver.h>
#include <tensorflow/lite/micro/micro_error_reporter.h>
#include <tensorflow/lite/micro/micro_interpreter.h>
#include <tensorflow/lite/schema/schema_generated.h>
#include <tensorflow/lite/micro/micro_mutable_op_resolver.h>
#include <tensorflow/lite/version.h>

#include "model.h"

tflite::MicroErrorReporter tflErrorReporter;
tflite::ErrorReporter* error_reporter = nullptr;
const tflite::Model* model = nullptr;
const tflite::Model* tflModel = nullptr;
tflite::MicroInterpreter* tflInterpreter = nullptr;
TfLiteTensor* tflInputTensor = nullptr;
TfLiteTensor* tflOutputTensor = nullptr;
constexpr int tensorArenaSize = 150 * 1024;
uint8_t tensor_arena[tensorArenaSize];
char option= ' ';
int led = LED_BUILTIN;


TfLiteTensor* model_input = nullptr;
TfLiteTensor* model_output = nullptr;
```

Figura 26: Codigo inicial del programa llamado a librerias e inicializacion de varibales.

El desglose linea a linea se muestra a continuación.

- `TensorFlowLite.h`: La biblioteca central para la funcionalidad de TensorFlow Lite en microcontroladores.
- `all_ops_resolver.h`: Incluye un resolver para todas las operaciones disponibles, esencial para la interpretación del modelo.
- `micro_error_reporter.h`: Define un reportero de errores para el monitoreo y diagnóstico durante la ejecución del modelo.
- `micro_interpreter.h`: Contiene el intérprete de TensorFlow Lite, encargado de la ejecución del modelo.
- `schema_generated.h`: Aporta el esquema generado necesario para la validación de la versión del modelo.
- `micro_mutable_op_resolver.h`: Facilita la mutación y registro de operaciones requeridas por el modelo.
- `version.h`: Ofrece información sobre la versión de la biblioteca TensorFlow Lite.
- `tflErrorReporter` es una instancia de `MicroErrorReporter` que sirve como el sistema de reporte de errores.
- `error_reporter`, `model`, `tflModel`, `tflInterpreter`, `tflInputTensor`, y `tflOutputTensor` son punteros inicialmente asignados a `nullptr`, que luego se asociarán con estructuras específicas del modelo y los tensores de entrada y salida.
- `tensorArenaSize` establece el tamaño de la memoria reservada para los tensores y datos intermedios, denominada “arena de tensores”.

- `tensor_arena` es un arreglo que representa esta arena de tensores, dimensionado de acuerdo con `tensorArenaSize`.
- `option` y `led` son variables utilizadas para la gestión de la interfaz de usuario y la indicación visual de estados mediante un LED.
- `model_input` y `model_output` son punteros a `TfLiteTensor`, inicializados a `nullptr`, que representarán los tensores de entrada y salida del modelo tras la configuración del sistema.

En el fragmento de código presentado en la Figura[27], se inicializa la comunicación serial, esencial para el monitoreo del comportamiento del sistema. Se carga el modelo de aprendizaje automático y se verifica la ausencia de errores durante este proceso. Posteriormente, se instancia en memoria el intérprete de TensorFlow Lite, conFigurando los tensores de entrada y salida correspondientes. Por último, se efectúa la llamada a la función `printModelDetails(model)`, la cual exhibe los detalles configuracionales del modelo cargado.

```
void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Start");

  model = tflite::GetModel(g_model);
  if (model->version() != TFLITE_SCHEMA_VERSION) {
    TF_LITE_REPORT_ERROR(error_reporter,
                         "Model provided is schema version %d not equal "
                         "to supported version %d.",
                         model->version(), TFLITE_SCHEMA_VERSION);
    return;
  }

  tflModel = tflite::GetModel(g_model);
  if (tflModel->version() != TFLITE_SCHEMA_VERSION) {
    Serial.println("Model schema mismatch!");
    while (1);
  }

  static tflite::AllOpsResolver micro_op_resolver;


  // Create an interpreter to run the model
  tflInterpreter = new tflite::MicroInterpreter(tflModel, micro_op_resolver, tensor_arena, tensorArenaSize, &tflErrorReporter);

  // Allocate memory for the model's input and output tensors
  tflInterpreter->AllocateTensors();

  // Get pointers for the model's input and output tensors
  tflInputTensor = tflInterpreter->input(0);
  tflOutputTensor = tflInterpreter->output(0);

  printModelDetails(model);

}
```

Figura 27: Código de inicialización del puerto serial, carga del modelo, creación del intérprete y llamada a la función `printModelDetails`.

La función `printModelDetails(model)`, cuyo prototipo se ilustra en la Figura[28], está diseñada para inspeccionar y mostrar las dimensiones de los tensores de entrada y salida asociados con un modelo de aprendizaje automático proporcionado como argumento. Esta función desempeña un papel crítico en el proceso de depuración, ya que permite la detección y el diagnóstico inmediatos de posibles incongruencias dimensionales. En el caso de que la información del modelo no pueda ser recuperada o el modelo mismo resulte inválido, la función emitirá un mensaje de error claramente definido, señalando la condición de "Invalid Model". El propósito de esta función es ofrecer una herramienta ágil para la verificación de errores relacionados con la dimensionalidad de los tensores, facilitando así un más rápido proceso de resolución de problemas.

```
void printModelDetails(const tflite::Model* model) {
  Serial.println("Model Details:");
  if (model) {
    const auto* subgraph = model->subgraphs()->Get(0);

    // Obtener detalles del tensor de entrada
    const auto* input_tensor = subgraph->tensors()->Get(0);
    Serial.print("Input Tensor Shape: [");
    for (int i = 0; i < input_tensor->shape()->Length(); i++) {
      Serial.print(input_tensor->shape()->Get(i));
      if (i != input_tensor->shape()->Length() - 1) {
        Serial.print(", ");
      }
    }
    Serial.println("]");

    // Obtener detalles del tensor de salida
    const auto* output_tensor = subgraph->tensors()->Get(1);
    Serial.print("Output Tensor Shape: [");
    for (int i = 0; i < output_tensor->shape()->Length(); i++) {
      Serial.print(output_tensor->shape()->Get(i));
      if (i != output_tensor->shape()->Length() - 1) {
        Serial.print(", ");
      }
    }
    Serial.println("]");
    Serial.println();
  } else {
    Serial.println("Invalid Model!");
  }
}
```

Figura 28: Ejecución de la función `printModelDetails` que muestra las dimensiones de los tensores de entrada y salida y verifica la validez del modelo.

Con el modelo de aprendizaje automático ya inicializado, se cuenta con una variable `input_data` que almacena un conjunto de datos correspondientes a la señal de un gesto específico, con dimensiones 625*8. Como se evidencia en la Figura[29], se utiliza una serie de bucles `for` para poblar adecuadamente el tensor de entrada del modelo con estos datos. Acto seguido, se efectúa la inferencia a través del intérprete de TensorFlow Lite. En ausencia de errores durante el proceso inferencial, los resultados se despliegan en un formato one-hot, siendo un vector de cuatro elementos donde el de mayor valor se distingue con un uno, señalando la categoría pronosticada por el modelo. Se establece una pausa programada (`delay`) de un segundo antes de emprender la siguiente ronda inferencial.

```
void loop() {

float input_data[625][8] =
{

 for (int i = 0; i < 625; i++) {
  for (int j = 0; j < 8; j++) {
    tflInputTensor->data.f[i * 5 + j] = input_data[i][j];
  }
}
  // Realizar la inferencia
  TfLiteStatus invoke_status = tflInterpreter->Invoke();
  if (invoke_status != kTfLiteOk) {
    error_reporter->Report("Error al realizar la inferencia!");
    while (1);
  }

    // Imprimir el resultado
  Serial.print("Predicción: ");
    for (int i = 0; i < 4; i++) {
      Serial.print(i);
      Serial.print(": ");
      Serial.print(tflOutputTensor->data.f[i]);
      Serial.print(", ");
    }


  delay(1000);

}
```

Figura 29: Proceso de carga de los datos, ejecución de la inferencia y presentación de resultados a través del puerto serial.

4.3.5 Proceso de Ejecución

Con el objetivo de verificar la funcionalidad del programa en un entorno real, se seleccionaron aleatoriamente cuatro muestras de datos, cada una correspondiente a una categoría distinta del conjunto de datos de prueba. Estas muestras se utilizaron para compilar el programa en la plataforma Arduino y realizar predicciones específicas. Este proceso implicó modificar los valores dentro de la matriz `input_data` en el código fuente. Durante la carga del firmware en la unidad Arduino, el sistema operativo reportó una advertencia: "Poca memoria disponible, se pueden producir problemas de estabilidad". Esta alerta sugiere que el 54 % de la memoria de almacenamiento y el 77 % de la memoria dinámica estaban siendo utilizados, lo cual puede visualizarse en la Figura[30], y podría implicar un riesgo de comportamiento errático o inestabilidad durante la ejecución del programa.

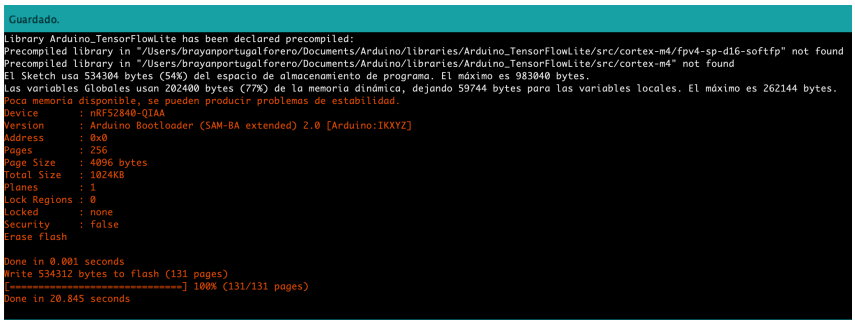


Figura 30: Notificación tras la compilación del código y su carga en el Arduino, mostrando el uso de memoria.

Tras la compilación e inicialización del monitor serie en el entorno de desarrollo integrado (IDE) de Arduino por parte del usuario, se exhibe un mensaje introductorio que detalla las características del modelo cargado. La Figura[31] ilustra esta información, desglosando la arquitectura tanto del tensor de entrada como del tensor de salida. En el tensor de entrada, se identifica una configuración de dimensiones $(1, 625, 8, 1)$. El primer valor '1' denota el tamaño del lote (batch size) para la inferencia, indicando el procesamiento de una sola instancia por cada ejecución. El número '625' representa la cantidad de características o puntos de datos de la entrada, mientras que el '8' refiere a la presencia de ocho canales en los datos de entrada. El último '1' establece que cada punto de dato es unidimensional, análogo a un píxel en imágenes monocromáticas.

En cuanto al tensor de salida, una estructura de $(1, 4)$ nuevamente pone de manifiesto que el '1' simboliza el tamaño del lote, promoviendo la salida por lotes de manera secuencial. El '4' especifica el número de clases o etiquetas de salida disponibles en el modelo, lo que implica su capacidad para clasificar las entradas en una de cuatro posibles categorías.

```
Start
Model Details:
Input Tensor Shape: [1, 625, 8, 1]
Output Tensor Shape: [1, 4]
```

Figura 31: Información inicial al ejecutar el código, exhibiendo los detalles del modelo.

La salida posterior muestra los resultados de la inferencia para cada clase, con las probabilidades asociadas a cada una, reflejando cómo el modelo clasifica los datos de entrada. Por ejemplo, como se revela en la Figura[32], una instancia fue clasificada correctamente como perteneciente a la clase 0 (parpadeo del ojo izquierdo) con una probabilidad del 68 %, mientras que la siguiente categoría más probable (ninguna acción o categoría 3) obtuvo un 32 %. Un fenómeno similar se observa en la Figura[33], donde el modelo determina con certeza que la entrada corresponde a la categoría 0 (ninguna acción), asignándole una probabilidad del 100 % y descartando las demás categorías.

```
Predicción: 0: 0.68 , 1: 0.00, 2: 0.00, 3: 0.32
```

Figura 32: Predicción para una entrada clasificada en la categoría 0, correspondiente al parpadeo del ojo izquierdo.

```
Predicción: 0: 0.00, 1: 0.00, 2: 0.00, 3: 1.00,
```

Figura 33: Predicción para una entrada clasificada en la categoría 3, que representa la ausencia de gestos, o señal de control.

5 Conclusiones

Se puede observar que si es posible mejorar la calidad de vida de las personas con discapacidad motora mediante la implementación de redes neuronales convolucionales aplicadas a señales EEG en microcontroladores de baja potencia que permite mayor accesibilidad y cobertura, tomando como referencia base este proyecto, ya que a pesar de que la exactitud del modelo es 64,5 %, es posible hacer mejoras para lograr subir el rendimiento de estos sistemas en los microcontroladores para soluciones masivas.

Al establecer el sistema hardware-software, se priorizó la elección de herramientas con documentación sólida, compatibilidad y accesibilidad, dado el contexto de tecnologías emergentes. Se seleccionó Cyton como herramienta de toma de datos, aprovechando su documentación y compatibilidad, herramienta que fue facilitada por el director de la monografía. Arduino Nano 33 BLE se eligió como microcontrolador debido a su documentación, compatibilidad con Cyton y TensorFlow Lite, y su disponibilidad en el mercado colombiano. Python se adoptó como lenguaje principal por su sólido apoyo en la comunidad de software libre. TensorFlow se destacó como el framework de desarrollo gracias a su documentación, curva de aprendizaje accesible y capacidad de personalización para modificar parámetros.

La recolección de datos se ha llevado a cabo de forma eficiente utilizando el software previamente establecido, integrar la placa cyton en un software para la adquisición de datos, se desarrollo basado en trabajos anteriores[Rodríguez and Sánchez, 2021], fue una tarea que no tomo mayor complejidad gracias al desarrollo ya generado. A los datos recopilados el proceso de limpieza fue bastante somero con el fin de hacer procesamiento en tiempo real en futuros trabajos, bastando con hacer normalización de -1 a 1 para reducir el tamaño de los números a procesar, y dado los resultados fue exitoso.

Tras la verificación del funcionamiento del modelo de clasificación implementado en un microcontrolador, se ha determinado que la precisión alcanzada es del 64.5 %. Esta cifra refleja las limitaciones inherentes a la capacidad de procesamiento del microcontrolador, que restringe la complejidad de la red neuronal que se puede ejecutar eficientemente en dicho entorno. Dado que el hardware condiciona el tamaño y la profundidad de la red neuronal, la precisión obtenida es moderada y está alineada con las expectativas para modelos simplificados adaptados a sistemas embebidos.

Se ha demostrado la viabilidad de implementar redes neuronales convolucionales en microcontroladores específicos, y la limitación principal encontrada fue la complejidad del modelo. Desde las primeras pruebas, se identificó que el tamaño máximo posible de implementación era de 400,000 bytes. El modelo con el mejor rendimiento tenía un tamaño de 350,000 bytes. Estos modelos son relativamente simples en términos de capas y cantidad de neuronas, y aunque podrían lograr un rendimiento superior al aumentar su complejidad, su implementación no sería factible en el microcontrolador seleccionado.

El conjunto de datos generado, compuesto por 1600 registros, posibilitó la creación y entrenamiento de los modelos, no obstante, incrementar la cantidad de datos con la inclusión de más personas podría enriquecer aún más las características de los modelos y potencialmente mejorar su rendimiento

6 Recomendaciones

Para futuros trabajos en la implementación y mejora de sistemas embebidos que manejan redes neuronales para el análisis de señales electroencefalográficas, se pueden considerar las siguientes recomendaciones.

- Optimización de Modelos de Red Neuronal: Dado que la capacidad de procesamiento del microcontrolador limita la complejidad de las redes neuronales, se recomienda la investigación en técnicas de optimización de modelos, como la poda de redes neuronales, la cuantificación y el diseño de arquitecturas de red más eficientes que puedan operar dentro de las restricciones de hardware sin sacrificar el rendimiento.

- Mejora de Algoritmos de Preprocesamiento: Aunque el preprocesamiento actual elimina eficazmente el ruido y los artefactos, se podrían explorar algoritmos avanzados de filtrado y técnicas de extracción de características para mejorar la calidad de los datos y, por ende, la precisión del modelo.

- Recopilación y Ampliación de Datasets: Ampliar el conjunto de datos con más variedad y volumen puede ayudar a mejorar la capacidad de generalización del modelo. Se recomienda utilizar técnicas de aumento de datos para enriquecer los datasets existentes.

- Enfoque en la Interfaz Hardware-Software: Una mayor integración y sinergia entre el hardware y el software podrían conducir a mejoras significativas en el rendimiento. La co-optimización del software de la red neuronal y el hardware del sistema embebido podría llevar a una mejor utilización de los recursos y a un aumento de la precisión.

- Evaluación de Métricas de Desempeño Adicionales: Además de la precisión, se sugiere la evaluación de otras métricas de rendimiento, como la sensibilidad, la especificidad y la curva ROC, para obtener una comprensión más completa del rendimiento del modelo en diferentes condiciones.

Referencias

[Dane, 2020] Dane. (2020). Personas con discapacidad, retos diferenciales en el marco del COVID-19. https://www.dane.gov.co/files/investigaciones/discapacidad/2020-Boletin-personas-con-discapacidad-marco-COVID-19.pdf

[Quinchía, 2023] Quinchía, A. Z. (2023). Colombia se raja en conectividad a internet: solo 60,5 % de la población accede al servicio — El Colombiano. www.elcolombiano.com. https://www.elcolombiano.com/negocios/colombia-se-raja-en-conectividad-a-internet-NE21529510

[Britannica, 2023] Britannica, T. Editors of Encyclopaedia (2023). electroencephalography. Encyclopedia Britannica. https://www.britannica.com/science/electroencephalography.

[Bitbrain, 2023] Bitbrain. (2023).Localización de electrodos del EEG: Layout Fijo vs. Variable. Bitbrain. https://www.bitbrain.com/es/blog/colocacion-electrodos-eeg.

[Ibm, s.f] Ibm. (s.f). ¿Qué son las redes neuronales convolucionales?. Ibm. https://www.ibm.com/es-es/topics/convolutional-neural-networks.

[Gómez, 2016] Gómez Figueroa, L. J. (2016). Análisis de señales EEG para detección de eventos oculares, musculares y cognitivos (Tesis de maestría, Automática y Robótica). E.T.S.I. Industriales, Universidad Politécnica de Madrid.

[Santos, 2015] Santos, M. (2015). OpenBCI just launched a kickstarter for the Ultracortex Mark IV. Futurism. https://futurism.com/ %20openbci-just-launched-a-kickstarter-for-the-ultracortex-mark-iv.

[OpenBCI, s.f] OpenBCI Online Store. (s.f). Cyton Biosensing Board (8-channels). openbci. https://shop.openbci.com/products/cyton-biosensing-board-8-channel.

[Arun, 2023] Arun. (2023). An Introduction to TinyML - towards Data Science. Medium. https://towardsdatascience.com/an-introduction-to-tinyml-4617f314aa79.

[Rodríguez and Sánchez, 2021] Rodríguez Daza, N., & Sánchez Bolaños, C. D. (2021). Desarrollo de un prototipo de habitación inteligente para pacientes discapacitados a través de señales EEG (Trabajo de grado, Ingeniería Electrónica). Universidad Distrital Francisco José de Caldas.

[Perez and Tarrillo, 2020] Perez, J., & Tarrillo, J. (2020). Implementation of an active-filtering circuit for electroencephalographic signal acquisition using an 8-bit microcontroller. In 2020 IEEE 14th Dallas Circuits and Systems Conference (DCAS) (pp. 1-5). IEEE.

[Cruz and Sánchez, 2021] Cruz-Rodríguez, A. M., & Sánchez-Machet, H. (2021). Prótesis de mano controlada con señales EEG. Momento, (63), 34-51.

[Pawuś and Paszkiel, 2022] Pawuś, D., & Paszkiel, S. (2022). BCI Wheelchair Control Using Expert System Classifying EEG Signals Based on Power Spectrum Estimation and Nervous Tics Detection. Applied Sciences, 12(10385). https://doi.org/10.3390/app122010385.

[Sarma, Kant and Rajkumar, 2018] Sarma, S. S., Kant, P., & Rajkumar. (2018). Multifunctional system for persons with disabilities using electroencephalography signals of eye blink. Current Science, 114(1), 193–195. http://www.jstor.org/stable/26493478.

[Antoniou,Bozios,Christou,Tzimourta,Kalafatakis,Tsipouras,Giannakeas and Tzallas,2021] Antoniou, E., Bozios, P., Christou, V., Tzimourta, K. D., Kalafatakis, K., Tsipouras, M. G., Giannakeas, N., & Tzallas, A. T. (2021). EEG-Based Eye Movement Recognition Using Brain–Computer Interface and Random Forests. Sensors, 21(2339). https://doi.org/10.3390/s21072339.

[Xu, Ren, Chen and Che, 2020] Xu, G., Ren, T., Chen, Y., & Che, W. (2020). A One-Dimensional CNN-LSTM Model for Epileptic Seizure Recognition Using EEG Signal Analysis. Frontiers in Neuroscience, 14:578126. https://doi.org/10.3389/fnins.2020.578126.

[Huang, Xiao and Xu, 2020] Huang, C., Xiao, Y., & Xu, G. (2020). Predicting Human Intention-Behavior Through EEG Signal Analysis Using Multi-Scale CNN. IEEE/ACM Transactions on Computational Biology and Bioinformatics. https://doi.org/10.1109/TCBB.2020.3039834.

[Bahr, Schneider, Francis, Lehmann, Barg, Buschhoff, Wulff, Strunskus and Faupel, 2021] Bahr, A., Schneider, M., Francis, M. A., Lehmann, H. M., Barg, I., Buschhoff, A.-S., Wulff, P., Strunskus, T., & Faupel, F. (2021). Epileptic Seizure Detection on an Ultra-Low-Power Embedded RISC-V Processor Using a Convolutional Neural Network. Biosensors, 11.

[Arduino, s.f] Arduino. (s.f). Arduino Nano 33 BLE. Arduino. https://store.arduino.cc/products/arduino-nano-33-ble.