

**UNIVERSIDAD PRIVADA DE TACNA
FACULTAD DE INGENIERÍA
ESCUELA PROFESIONAL DE INGENIERÍA
ELECTRÓNICA**



TESIS

**“DISEÑO E IMPLEMENTACIÓN DE PROTOTIPO DE UN
SISTEMA DE ADQUISICIÓN DE BIO POTENCIAL MUSCULAR
USANDO REDES NEURONALES PARA LA CLASIFICACIÓN DE
MOVIMIENTOS DE UNA MANO, 2023”**

PARA OPTAR:

TÍTULO PROFESIONAL DE INGENIERO ELECTRÓNICO

PRESENTADO POR:

Bach. GERSON MOISES NIZAMA SILVA

TACNA - PERÚ

2024

**UNIVERSIDAD PRIVADA DE TACNA
FACULTAD DE INGENIERÍA
ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA**

TESIS

**“DISEÑO E IMPLEMENTACIÓN DE PROTOTIPO DE UN
SISTEMA DE ADQUISICIÓN DE BIO POTENCIAL MUSCULAR
USANDO REDES NEURONALES PARA LA CLASIFICACIÓN DE
MOVIMIENTOS DE UNA MANO, 2023”**

Tesis sustentada y aprobada el 22 de octubre de 2024; estando el jurado calificador integrado por:

PRESIDENTE : Dr. ANIBAL JUAN ESPINOZA ARANCIAGA

SECRETARIO : Mtro. HUGO JAVIER RIVERA HERRERA

VOCAL : Mag. CARLOS ARMANDO RODRIGUEZ SILVA

ASESOR : Mag. TITO LEONCIO CÓRDOVA MIRANDA

DECLARACIÓN JURADA DE ORIGINALIDAD

Yo, Gerson Nizama Silva, egresado, de la Escuela Profesional de Ingeniería electrónica de la Facultad de Ingeniería de la Universidad Privada de Tacna, identificado con DNI 71728270, así como Tito Leoncio Córdova Miranda con DNI 04643495; declaramos en calidad de autor y asesor que:

1. Somos los autores de la tesis titulado: *Diseño e implementación de prototipo de un sistema de adquisición de bio potencial muscular usando redes neuronales para la clasificación de movimiento de una mano 2023*, la cual presentamos para optar el Título Profesional de Ingeniero Electrónico.
2. La tesis es completamente original y no ha sido objeto de plagio, total ni parcialmente, habiéndose respetado rigurosamente las normas de citación y referencias para todas las fuentes consultadas.
3. Los datos presentados en los resultados son auténticos y no han sido objeto de manipulación, duplicación ni copia.

En virtud de lo expuesto, asumimos frente a *La Universidad* toda responsabilidad que pudiera derivarse de la autoría, originalidad y veracidad del contenido de la tesis, así como por los derechos asociados a la obra.

En consecuencia, nos comprometemos ante a *La Universidad* y terceros a asumir cualquier perjuicio que pueda surgir como resultado del incumplimiento de lo aquí declarado, o que pudiera ser atribuido al contenido de la tesis, incluyendo cualquier obligación económica que debiera ser satisfecha a favor de terceros debido a acciones legales, reclamos o disputas resultantes del incumplimiento de esta declaración.

En caso de descubrirse fraude, piratería, plagio, falsificación o la existencia de una publicación previa de la obra, aceptamos todas las consecuencias y sanciones que puedan derivarse de nuestras acciones, acatando plenamente la normatividad vigente.

Tacna, 22 de octubre de 2024

Gerson Nizama Silva

DNI:71728270

Tito Leoncio Córdova Miranda

DNI:04643495

DEDICATORIA

Dedico esto a mis padres Moises y Liliana, por su incansable apoyo, sacrificio y enseñanzas que han guiado cada uno de mis pasos. A mi hermana Yumaira, por su inspiración constante y su ejemplo de dedicación. Sin ellos este logro no sería posible

Gerson Nizama Silva

AGRADECIMIENTO

A mi familia, por su amor incondicional y apoyo constante que me han dado la fuerza necesaria para alcanzar este objetivo. A mis docentes, por compartir su conocimiento guiándome en cada etapa de mi formación. A mis amigos, por su compañerismo y ánimo que me han acompañado a lo largo de este camino.

Gerson Nizama Silva

ÍNDICE GENERAL

PÁGINA DE JURADOS	ii
DECLARACIÓN JURADA DE ORIGINALIDAD.....	iii
DEDICATORIA.....	iv
AGRADECIMIENTO	v
ÍNDICE GENERAL.....	vi
ÍNDICE DE TABLAS	viii
ÍNDICE DE FIGURAS.....	ix
ÍNDICE DE ANEXOS	xi
RESUMEN.....	xii
ABSTRACT.....	xiii
INTRODUCCIÓN.....	1
CAPÍTULO I: EL PROBLEMA DE INVESTIGACIÓN	3
1.1. Descripción del problema	3
1.2. Formulación del problema	4
1.2.1. Problema general	4
1.2.2. Problemas específicos.....	4
1.3. Justificación e importancia.....	5
1.4. Objetivos.....	6
1.4.1. Objetivo general	6
1.4.2. Objetivos específicos.....	6
CAPÍTULO II: MARCO TEÓRICO.....	7
2.1. Antecedentes de la investigación.....	7
2.2. Bases Teóricas	11
2.3. Definición de términos	28
CAPÍTULO III: MARCO METODOLÓGICO	31
3.1. Diseño de la investigación	31
3.2. Acciones y actividades.....	31
3.3. Materiales e instrumentos.....	38
3.4. Operacionalización de variables	38
CAPÍTULO IV: RESULTADOS	40
4.1. Elección de tarjeta de adquisición de señales EMG	40
4.2. Elección de electrodos.....	40
4.3. Preparación del sujeto de prueba	40

4.4.	Generación de señales mioeléctricas	41
4.5.	Preprocesamiento de Datos.....	42
4.6.	Implementación de Filtros Digitales	43
4.7.	Análisis de correlación	45
4.8.	Implementación de red neuronal secuencial	50
4.9.	Implementación de red neuronal recurrente	53
4.10.	Implementación de red neuronal convolucional	58
4.11.	Validación cruzada.....	61
4.12.	Elección del modelo.....	68
4.13.	Prototipo final.....	69
CAPÍTULO V: DISCUSIÓN.....		70
CONCLUSIONES		71
RECOMENDACIONES.....		72
REFERENCIAS BIBLIOGRÁFICAS.....		73
ANEXOS.....		84

ÍNDICE DE TABLAS

Tabla 1. Operacionalización de variable de la variable dependiente	39
Tabla 2. Operacionalización de variable de la variable independiente	39
Tabla 3. Resultados de promedio de las métricas de las 3 redes neuronales	68

ÍNDICE DE FIGURAS

Figura 1. Ejemplos de sistemas top-down y sistemas bottom-up de inteligencia artificial	11
Figura 2. Dos señales EMG producido por la contracción del bíceps. La escala de Tiempo es 500ms por recuadro	19
Figura 3. Unidad motora	19
Figura 4. Diagrama de bloques de un sistema EMG.....	21
Figura 5. Ubicación de electrodos en el registro EMG de antebrazo	22
Figura 6. Ciclo de vida de cascada	32
Figura 7. Metodologías de codiseño	32
Figura 8. Metodología Top-Down	33
Figura 9. Diagrama de caja negra.....	34
Figura 10. Diagrama funcional.....	35
Figura 11. Diagrama de bloques	35
Figura 12. Diagrama de flujo.....	36
Figura 13. Diagrama de hardware	36
Figura 14. Diagrama de bloques de sensores	37
Figura 15. Diagrama de bloques de control	37
Figura 16. Preprocesamiento de datos con mano extendida	44
Figura 17. Preprocesamiento de datos con mano relajada	45
Figura 18. Preprocesamiento de datos con mano cerrada.....	45
Figura 19. Matriz de Correlación.....	46
Figura 20. Gráfica de RMS del sensor 1	47
Figura 21. Gráfica de RMS del sensor 2	47
Figura 22. Gráfica de MAV del sensor 1	47
Figura 23. Gráfica de MAV del sensor 2	48
Figura 24. Gráfica de MNF del sensor 1	48
Figura 25. Gráfica de MNF del sensor 2	48
Figura 26. Gráfica de ZC del sensor 1	48
Figura 27. Gráfica de ZC de sensor 2	49
Figura 28. Gráfica de out1	49
Figura 29. Gráfica de out2	49
Figura 30. Gráfica de out3	49
Figura 31. Gráfica de la arquitectura de la red neuronal secuencial	50

Figura 32. Gráfico de Curva de Aprendizaje de Red Neuronal Secuencial	51
Figura 33. Graficas de Perdidas durante el Entrenamiento de Red Neuronal Secuencial	52
Figura 34. Matriz de Confusión Normalizada de Red Neuronal Secuencial	53
Figura 35. Gráfica de la arquitectura de la red neuronal recurrente	54
Figura 36. Gráfico de Curva de Aprendizaje de Red Neuronal Recurrente	55
Figura 37. Graficas de Perdidas durante el Entrenamiento de Red Neuronal Recurrente	56
Figura 38. Matriz de Confusión Normalizada de Red Neuronal Recurrente.....	57
Figura 39. Gráfica de la arquitectura de la red neuronal convolucional.....	58
Figura 40. Gráfico de Curva de Aprendizaje de Red Neuronal Convolucional	59
Figura 41. Graficas de Perdidas durante el Entrenamiento de Red Neuronal Convolucional	60
Figura 42. Matriz de Confusión Normalizada de Red Neuronal Convolucional.....	61
Figura 43. Gráfica de MAV del sensor 1 para la validación cruzada	62
Figura 44. Gráfica de MAV del sensor 2 para la validación cruzada	62
Figura 45. Gráfica de MNF del sensor 1 para la validación cruzada	63
Figura 46. Gráfica de MNF del sensor 2 para la validación cruzada	63
Figura 47. Gráfica de out1 para la validación cruzada	64
Figura 48. Gráfica de out2 para la validación cruzada	64
Figura 49. Gráfica de out3 para la validación cruzada	65
Figura 50. Matriz de Confusión Normalizada de la Validación de Red Neuronal Secuencial	66
Figura 51. Matriz de Confusión Normalizada de la Validación de Red Neuronal Recurrente	67
Figura 52. Matriz de Confusión Normalizada de la Validación de Red Neuronal Convolucional	68
Figura 53. Diagrama Final de prototipo de clasificación de estados de la mano	69

ÍNDICE DE ANEXOS

Anexo 1. Matriz de consistencia	85
Anexo 2. Datos de tarjeta de adquisición de señales EMG SicheyRay 2Ch	86
Anexo 3. Manual de software EMG SicheyRay 2Ch	95
Anexo 4. Hoja técnica y de seguridad de electrodos BioProtech	103
Anexo 5. Código de programación en Python para el desarrollo del sistema de clasificación de movimientos de una mano	107
Anexo 6. Código de programación en Python para la validación y evaluación de rendimiento de los modelos desarrollados.	110

RESUMEN

El presente trabajo de tesis tuvo como objetivo diseñar e implementar un sistema de adquisición de señales no invasivo capaz de reconocer patrones de biopotencial muscular y clasificarlos en diferentes estados de movimiento de una mano. Se abordó la problemática de la pérdida de extremidades superiores y su impacto en la calidad de vida, así como la demanda creciente de sistemas de asistencia y rehabilitación basados en la clasificación de movimientos. Se revisaron antecedentes internacionales y nacionales, destacando la importancia de desarrollar prótesis más accesibles y sistemas de rehabilitación efectivos. Se establecieron bases teóricas sobre inteligencia artificial, redes neuronales, señales mioeléctricas y técnicas de caracterización y clasificación de señales. Metodológicamente, se optó por un diseño experimental, manipulando los movimientos de la mano para evaluar el prototipo en un entorno controlado. Se definieron requerimientos y se planificó el desarrollo utilizando un modelo de proceso iterativo e incremental, siguiendo el ciclo de vida de cascada y metodologías de codiseño. Se implementó el hardware y software, utilizando sensores, amplificadores, digitalizadores y algoritmos de clasificación. Siendo sus resultados un prototipo con una precisión global del 92,69 % clasificando los estados de la mano del sujeto de pruebas, esperándose que el prototipo contribuya al avance de las prótesis mioeléctricas y los sistemas de rehabilitación, mejorando la calidad de vida de las personas con discapacidad motora.

Palabras clave: Biopotencial muscular; señales mioeléctricas; redes neuronales; prototipo.

ABSTRACT

The objective of this thesis work was to design and implement a non-invasive signal acquisition system capable of recognizing muscle biopotential patterns and classifying them in different states of movement of a hand. The problem of the loss of upper extremities and its impact on quality of life was addressed, as well as the growing demand for assistance and rehabilitation systems based on the classification of movements. International and national antecedents were reviewed, highlighting the importance of developing more accessible prostheses and effective rehabilitation systems. Theoretical bases were established on artificial intelligence, neural networks, myoelectric signals and signal characterization and classification techniques. Methodologically, an experimental design was chosen, manipulating hand movements to evaluate the prototype in a controlled environment. Requirements were defined and development was planned using an iterative and incremental process model, following the waterfall life cycle and co-design methodologies. The hardware and software were implemented, using sensors, amplifiers, digitizers and classification algorithms. The results being a prototype with an accuracy of 92,69 % in classifying the hand states of the subject, it is expected that the prototype will contribute to the advancement of myoelectric prostheses and rehabilitation systems, improving the quality of life of people with motor disabilities.

Keywords: Muscle biopotential; myoelectric signals; neural networks; prototype.

INTRODUCCIÓN

La capacidad motora del ser humano siempre ha sido una función vital importante, que permite garantizar su desenvolvimiento e interacción con el entorno. A lo largo de la historia, el hombre ha buscado adaptarse y superar las limitaciones físicas, ya sea por causas congénitas o eventos traumáticos. En este contexto, el desarrollo de prótesis ha sido fundamental para facilitar las actividades cotidianas de quienes han experimentado la pérdida de extremidades superiores.

Sin embargo, la mayoría de las prótesis disponibles en el mercado son pasivas o mecánicas, ofreciendo una funcionalidad limitada. Aunque existen modelos más avanzados, como las prótesis mioeléctricas, que interpretan señales musculares para permitir movimientos más naturales, su alto costo y complejidad de adaptación restringen su acceso.

El objetivo de esta tesis es diseñar e implementar un sistema de adquisición de señales no invasivo capaz de reconocer patrones de biopotencial muscular y clasificarlos en diferentes estados de movimiento de una mano. Se busca superar las limitaciones de las prótesis existentes y contribuir al desarrollo de sistemas de asistencia y rehabilitación más accesibles y efectivos.

Para lograr este objetivo, se investigaron los antecedentes internacionales y nacionales en el campo de las prótesis mioeléctricas y la clasificación de movimientos de mano. Se establecieron bases teóricas sobre inteligencia artificial, redes neuronales, señales mioeléctricas y técnicas de caracterización y clasificación de señales.

La metodología empleada fue de tipo experimental, manipulando los movimientos de la mano para evaluar el prototipo en un entorno controlado. Se definieron los requerimientos del sistema y se planificó su desarrollo utilizando un modelo de proceso iterativo e incremental, siguiendo el ciclo de vida de cascada y metodologías de codiseño.

El prototipo se implementó utilizando sensores para adquirir las señales bioeléctricas, amplificadores y digitalizadores para acondicionarlas, y algoritmos de clasificación basados en redes neuronales para interpretar los patrones musculares y determinar los estados de movimiento de la mano.

La presente tesis se compone de cinco capítulos: Capítulo I, El problema de investigación, se refiere a la descripción del problema, formulación del problema,

justificación y objetivos; Capítulo II, denominado Marco Teórico, se presentan los antecedentes del estudio, la base teórica y definición de términos; Capítulo III, denominado Marco metodológico, se detalla el diseño de la investigación, acciones y actividades, materiales e instrumentos, operacionalización de variables y el procesamiento y análisis de datos; Capítulo IV, denominado Resultados, se describen los resultados obtenidos en el diseño e implementación de un sistema de adquisición de señales no invasivo para reconocer los patrones de biopotencial muscular y clasificar los movimientos de una mano; y el Capítulo V denominado Discusiones, Conclusiones y Recomendaciones, se analizan, interpretan y discuten los resultados en relación a la implementación de un sistema de adquisición de señales no invasivo, utilizando redes neuronales.

CAPÍTULO I: EL PROBLEMA DE INVESTIGACIÓN

1.1. Descripción del problema

Manz et al. (2022) indican que la pérdida de extremidades superiores ya sea por causas congénitas o eventos traumáticos como accidentes, plantea un desafío considerable en la vida diaria de quienes la experimentan. En consecuencia, se ven obligados a adaptarse para realizar actividades cotidianas, a pesar de que se han desarrollado prótesis para facilitar estas tareas. Sin embargo, la disponibilidad de estas prótesis es limitada, dado que la mayoría de estos dispositivos son pasivos o mecánicos, ofreciendo solo una funcionalidad básica. Si bien los modelos más avanzados, como las prótesis mioeléctricas, pueden interpretar señales musculares para permitir movimientos más naturales, su alto costo de producción y la complejidad de adaptación restringen su acceso en el mercado. Además, requieren un proceso complejo para adaptarse a las necesidades específicas de cada usuario, considerando su condición física y nivel de amputación.

Rapczyński et al. (2021) manifiestan que, a nivel mundial, las enfermedades musculoesqueléticas son la segunda causa de discapacidad más común, afectando a aproximadamente 1 300 millones de personas. Dentro de este panorama, las lesiones en la mano son una de las principales causas de discapacidad física, afectando a 31 millones de personas, y se estima que el 50 % de ellas experimentarán dolor y disfunción a largo plazo. Asimismo, la prevalencia de enfermedades neuromusculares, que afectan la capacidad de movimiento, alcanza a 50 millones de personas en todo el mundo, y se espera que esta cifra aumente en un 17 % para el año 2030. Además, Safiri et al. (2021) plantean que la parálisis cerebral, afecta a 570 000 niños cada año, es la segunda causa más común de discapacidad en la infancia. El impacto económico de estas discapacidades es considerable, generando un costo anual de 1 billón de dólares a nivel mundial, de los cuales el 75 % corresponde a pérdidas de productividad. Por lo tanto, la demanda de sistemas de asistencia y rehabilitación basados en la clasificación de movimientos de mano está en constante crecimiento, y se estima que el mercado global de dispositivos de rehabilitación alcanzará un valor de 22 400 millones de dólares para el año 2028.

En mención a Sabastizagal-Vela et al. (2020) plantean que en Latinoamérica, las lesiones por movimientos repetitivos (LMR) constituyen un problema de salud pública relevante, afectando a millones de trabajadores en la región. Se estima que entre el 20

% y el 30 % de las enfermedades profesionales en la región son LMR, lo que evidencia su impacto significativo en la productividad, la economía y la calidad de vida de los trabajadores. Para ilustrar esta problemática, la Organización Panamericana de la Salud (OPS) estima que estas lesiones cuestan a la región más de 200 mil millones de dólares al año en pérdida de productividad y costos de atención médica. En países como Brasil, las LMR son responsables de más de 1 millón de reclamaciones de trabajadores lesionados cada año, con un costo promedio de \$20 000 por reclamo, mientras que en México representan el 15 % de todas las enfermedades profesionales.

A nivel nacional, en adaptación de Macri et al. (2023), en el Perú, la incidencia de amputaciones de mano, ya sea por accidentes laborales o malformaciones congénitas, representa un problema significativo para la calidad de vida y la productividad de las personas afectadas. Según datos del Instituto Nacional de Salud (INS), las lesiones por movimientos repetitivos (LMR) constituyen el 3 % de todas las enfermedades profesionales registradas en el país. En el 2021, se reportaron más de 100 casos de LMR, cada uno con un costo promedio de S/. 1 000, lo que resalta el impacto económico de estas lesiones a nivel nacional, debido a que estos datos estadísticos no solo reflejan el costo financiero de las LMR, sino también el impacto en la calidad de vida de los trabajadores afectados, dado a que las LMR pueden causar dolor crónico, discapacidad y pérdida de la capacidad laboral, lo que a su vez puede llevar a la pobreza y la exclusión social.

1.2. Formulación del problema

Según lo expuesto en la descripción del problema se plantea el problema general y los problemas específicos.

1.2.1. Problema general

¿Cómo el diseño e implementación de un sistema de adquisición de señales no invasivo reconocerá los patrones de biopotencial muscular y determinar los estados mínimos necesarios para clasificar los movimientos de una mano?

1.2.2. Problemas específicos

- a. ¿Cuáles son los estados mínimos necesarios para clasificar los movimientos de una mano?

- b. ¿Qué parámetros que se deben tomar en cuenta para reconocer los patrones de biopotencial muscular?
- c. ¿Qué elementos de hardware y software son necesario para el diseño del prototipo?

1.3. Justificación e importancia

Según el diario gestión, diariamente más de 30 trabajadores tienen un accidente con lesión de extremidad superior y uno de ellos sufre la amputación total o parcial de las manos o dedos cada dos días, esto publicado en 2017. Esto se suma al hecho que una de las discapacidades que más afecta a las personas en el Perú es la incapacidad motora para mover o caminar brazos o piernas alcanzando a 932 mil personas que representan un 59,2 % de la población que presenta por lo menos una incapacidad en nuestro país según un estudio realizado por el INEI en 2013.

La aseguradora Rímac Seguros estima que el costo directo para la asistencia de una amputación total está en el orden de lo S/ 1 800 que agregando lo factores que se asocian como dolor o perdida de sensibilidad que puede incrementar desde 5 a 12 veces más y algo mucho más importante, el alto costo social y las pérdidas de desarrollo para una persona

A lo largo de la evolución en el desarrollo de prótesis, los controles de estas han variado desde prótesis estáticas, hasta el manejo por medio de dispositivos electromecánicos. Recientemente se ha incursionado en el desarrollo de prótesis que se controle de una forma más natural para la persona, utilizando señales de control provenientes del propio cuerpo humano, por ejemplo, las señales mioeléctrica de los músculos.

Las prótesis mioeléctrica que utilizan algoritmos de clasificación presentan ventajas sobre la mecánica relacionadas al manejo sencillo y fuerza de prensión; sin embargo, el elevado costo, mantenimiento y desarrollo complejo de estas hace y casi imposible a la población media contar con dicho presupuesto para adquirir una.

En nuestro país no se desarrollan las prótesis mioeléctrica, ya que para ello primero se debe contar con determinada investigación para un cumplimiento de las normativas requeridas. En Latinoamérica hay pocos países que trabajan en investigación y desarrollo de prótesis tales como Brasil y México.

1.4. Objetivos

1.4.1. Objetivo general

Diseñar e implementar un sistema de adquisición de señales no invasivo para reconocer los patrones de biopotencial muscular y determinar los estados mínimos necesarios para clasificar los movimientos de una mano.

1.4.2. Objetivos específicos

- a. Determinar los estados mínimos necesarios para clasificar los movimientos de una mano.
- b. Determinar los elementos de hardware y software necesarios para el diseño del prototipo
- c. Determinar los parámetros que se deben tomar en cuenta para reconocer los patrones de biopotencial muscular

CAPÍTULO II: MARCO TEÓRICO

2.1. Antecedentes de la investigación

2.1.1. Antecedentes internacionales

Romo (2022) en sus tesis de maestría titulada “clasificación de señales mioeléctrica por medio de algoritmos genéticos y máquinas de soporte de vectores” realizó un análisis de las señales mioeléctrica provenientes de los músculos para determinar la manera más optima de caracterizar estas señales de manera global para reducir la dimensionalidad de estas para reducir el número de predictores necesarios para su clasificación y disminuir su complejidad.

Aayesha et al. (2021) en la investigación “Modelo de clasificación de señales EEG basado en aprendizaje automático para la detección de ataques epilépticos”, tuvieron como objetivo identificar las características más discriminatorias y distintivas de los registros EEG de las convulsiones para desarrollar un enfoque que emplee algoritmos de aprendizaje automático tradicionales y de base difusa para la detección de crisis epilépticas, siendo su población y muestra 120 pacientes con epilepsia, siendo de metodología cuantitativa, experimental, debido a que sus resultados fueron en que se evaluó un modelo de clasificación de redes neuronales convolucionales (CNN) para la detección de convulsiones epilépticas en registros de EEG de pacientes con epilepsia, debido a que el modelo logró una precisión del 99,1 %, una sensibilidad del 98,8 % y una especificidad del 99,4 %,m concluyendo en que las crisis epilépticas causan la disfunción de la salud física y mental normal de los pacientes epilépticos, además, la detección de ataques epilépticos mediante enfoques basados en aprendizaje automático para la clasificación de señales EEG se ha empleado con frecuencia en la literatura.

Gadekallu et al. (2021) en la investigación “Clasificación de gestos con las manos mediante un novedoso algoritmo de búsqueda de cuervos de CNN”, tuvo como objetivo implementar un modelo de redes neuronales convolucionales basado en búsqueda de cuervos en el reconocimiento de gestos musculares con la mano, debido a que su población y muestra fueron 384 ejemplos de potencia, siendo de metodología cuantitativa, experimental, debido a que sus resultados alcanzaron una precisión del 100 % en el entrenamiento y las pruebas del modelo, lo que demuestra su superioridad frente a los modelos tradicionales de última generación, en cuanto al rendimiento en diferentes conjuntos de datos, el modelo se probó con diferentes conjuntos de datos de

imágenes de gestos de mano estáticas y logró una tasa de reconocimiento del 99 % para apertura completa y del 95,32 % para una apertura de uno a ocho, además, en el caso de CNN con 8 capas, se obtuvo una precisión del 89,6 % y 96,9 % en conjuntos de datos escalados y no escalados. Y concluyéndose en que la clasificación de gestos con las manos también propuso un nuevo modelo CNN, que logró una tasa de reconocimiento del 99 % en el conjunto de datos de imágenes de gestos estáticos y del 95,32 % en el conjunto de datos de gestos de video.

Avilés y Gaibor (2021) en su tesis “Diseño e implementación de una prótesis robótica con señales EMG usando técnicas de inteligencia artificial” desarrolló una prótesis mecánica controlada con un microcontrolador Esp32, el cual se encargaba del movimiento de la prótesis y la recolección de datos en tiempo real utilizando tensorflow. Finalmente consigue clasificar en 3 estado el movimiento de la prótesis con una precisión del 78,67 %.

Fontana et al. (2021) en su artículo “Clasificación de señales mioeléctrica para el control de una mano robótica” realizaron una comparación entre 3 algoritmos de clasificación para 5 movimiento de una mano donde concluye que las máquinas de vectores de soporte y combinación de clasificadores obtuvieron un mejor desempeño con una fiabilidad global mayor al 90 %.

Sanchez et al. (2021) en la investigación “Uso de señales cerebrales para controlar drones y tratar el trastorno por déficit de atención e hiperactividad” tuvieron como objetivo desarrollar una aplicación de un sistema aéreo no tripulado (RPAS) que ayude en el tratamiento del Trastorno por Déficit de Atención e Hiperactividad (TDAH) haciendo uso de una interfaz cerebro-ordenador, que se base en las medidas detectadas por un sensor EEG, siendo su población y muestra 30 participantes, debido a que su metodología fue experimental, obteniéndose de resultados en cuanto a la tasa de éxito de la clasificación con ojos abiertos fue de 92 % (27 de 30 participantes), de ojos cerrados con un 88 % (26 de 30 participantes); el tiempo de respuesta (en segundos) ante el cambio de ojos abiertos a cerrados: Media = 2,5 s, Desviación estándar = 0,8 s, como también el cambio de ojos cerrados a abiertos: Media = 1,8 s, Desviación estándar = 0,6 s, del mismo modo, la precisión del control del dron arrojo un error medio en la altitud objetivo: 5 cm, el tiempo medio para alcanzar la altitud objetivo: 1,2 s, además, la mejora en la atención (medida por una escala estandarizada de TDAH) en cuanto al antes del entrenamiento: Media = 65, Desviación estándar = 10 y después del entrenamiento: Media = 58, Desviación estándar = 8. Y concluyendo en que la técnica de Neurofeedback puede llevarse a cabo proporcionando la retroalimentación

con un dron, debido a la señal alfa asociada a la acción de abrir o cerrar los ojos ha sido una elección perfecta para demostrar la viabilidad de la técnica.

2.1.2. Antecedentes nacionales

Reátegui (2024) en su investigación “Diseño e Implementación de un Sistema Embebido Portátil para la Adquisición y Procesamiento de Señales Electromiográficas del Antebrazo” tuvo como objetivo crear una herramienta útil para aplicaciones de control y rehabilitación, superando las limitaciones de dispositivos comerciales como el MYO en términos de ancho de banda y resolución, siendo de población personas con amputaciones de miembro superior y de muestra un participante, tuvo de metodología en diseño instrumental, siendo de resultados en que se realizaron pruebas de reconocimiento de gestos con un voluntario con amputación transradial, logrando una precisión de alrededor del 94,80 % en la estimación de cinco gestos de mano utilizando un algoritmo SVM multiclase , además, se demostró la capacidad del sistema para controlar un brazo robótico en tiempo real a partir de las señales EMG adquiridas. Y concluyendo en que el sistema embebido desarrollado es una herramienta útil para pruebas de adquisición de datos y proyectos que requieren el reconocimiento de gestos, tanto para usuarios sin discapacidad como para usuarios con discapacidad, como en el desarrollo de prótesis de mano.

Sulla (2023), en su tesis “Diseño de una Prótesis de Brazo Electromiográfico Intuitivo Asistido con Inteligencia Artificial”, tuvo como objetivo mejorar la calidad de vida de personas con amputaciones de miembros superiores, permitiéndoles recuperar funcionalidad de manera más natural e intuitiva, siendo de población y muestra señales EMG de 10 participantes, abarcando movimientos de mano, antebrazo y codo. Utilizó sensores Myoware 2.0, Arduino UNO R3 y MATLAB R2023b para la adquisición, procesamiento y análisis de datos, respectivamente, siendo de resultados en que el sistema diseñado pudo reconocer y clasificar los movimientos de apertura y cierre de mano, pronación y supinación del antebrazo, y flexión y extensión del codo, se entrenó una red neuronal con 21 muestras, se validó con 4 y se probó con otras 4, logrando una precisión de 89,7 % en la clasificación de 3 movimientos con 2 salidas booleanas, en otra prueba con 2 salidas codificadas para 3 movimientos, se alcanzó una precisión del 100 %, finalmente, al reducir la cantidad de movimientos a 3 y codificar las salidas de manera diferente, se obtuvo nuevamente una precisión del 100 %, concluyendo en que se demostró la viabilidad de utilizar señales EMG e inteligencia artificial para el control intuitivo de prótesis de brazo, lo que podría mejorar significativamente la calidad de vida de las personas con amputaciones.

Ayala y Quispe (2022) en la investigación “Diseño e Implementación de un Exoesqueleto Adaptable para Rehabilitación de Miembros Inferiores en Pacientes con Lesión Medular Controlado Mediante Lógica Difusa y Monitoreo Usando IOT”, tuvieron como objetivo mejorar la calidad de vida de estos pacientes, siendo de población el exoesqueleto diseñado para pacientes y de muestra un participante, dado a que la metodología fue de enfoque de diseño de ingeniería, que abarcó áreas de mecánica, eléctrica, electrónica y de programación, obteniéndose de resultados en cuanto al exoesqueleto estuvo diseñado para pacientes con lesión lumbar T11 hasta el sacro, con límites físicos en cuanto a altura, longitud de piernas y peso del paciente, además en las pruebas, el exoesqueleto alcanzó un rango de movimiento de 0° a 60° en flexión de cadera y rodilla, similar a la marcha humana normal, dado a que los motores proporcionaron una fuerza de asistencia de hasta 50 N en cada articulación, reduciendo el esfuerzo del paciente durante la terapia, la batería de litio permitió una autonomía de 2 horas de funcionamiento continuo, se realizaron pruebas con un voluntario que permitieron validar el correcto funcionamiento del sistema en cuanto a la adquisición y procesamiento de señales EMG, así como el control de los actuadores mediante lógica difusa; en conclusión, se demuestró la viabilidad de diseñar e implementar un exoesqueleto adaptable para la rehabilitación de miembros inferiores en pacientes con lesión medular.

Además, Zubieta (2020) realizó el trabajo “Diseño y desarrollo del prototipo GR19 para el monitoreo en terapias para rehabilitación de la mano en Jireh Medical Import SAC, SJL 2019”, cuyo objetivo fue analizar, diseñar y desarrollar un prototipo que se convierta en apoyo para la rehabilitación de la mano además de un interfaz y que comprende también implementar un sistema electrónico. Para su elaboración se identificaron los parámetros vitales a ser evaluados para su monitoreo mediante un enfoque de metodología cualitativa. Entre sus resultados, está la evaluación del prototipo de acuerdo a cada etapa por la que se pasó, lográndose cumplir los objetivos planteados. Como conclusión, el trabajo cumplió con los rangos, especificaciones y características planteadas para elaborar este prototipo.

Finalmente, Llantoy (2020) realizó el estudio “Diseño e implementación del sistema electrónico para una prótesis transradial mioeléctrica” que tuvo de propósito diseñar e implementar un sistema de bajo costo para una prótesis que permita una mejor ejecución de gestos en las prótesis de mano. En su metodología se describió el sistema electrónico a ser usado con enfoque cualitativo y tipo descriptivo. Entre los resultados podemos decir que hay que evaluar el valor inicial y final de los sensores de fuerza y posición, y en general estos fueron satisfactorios obtenidos, Como conclusión, se logró

tener un diseño e implementación funcional del sistema electrónico para una prótesis transradial mioeléctrica con un costo de S/ 2475,22.

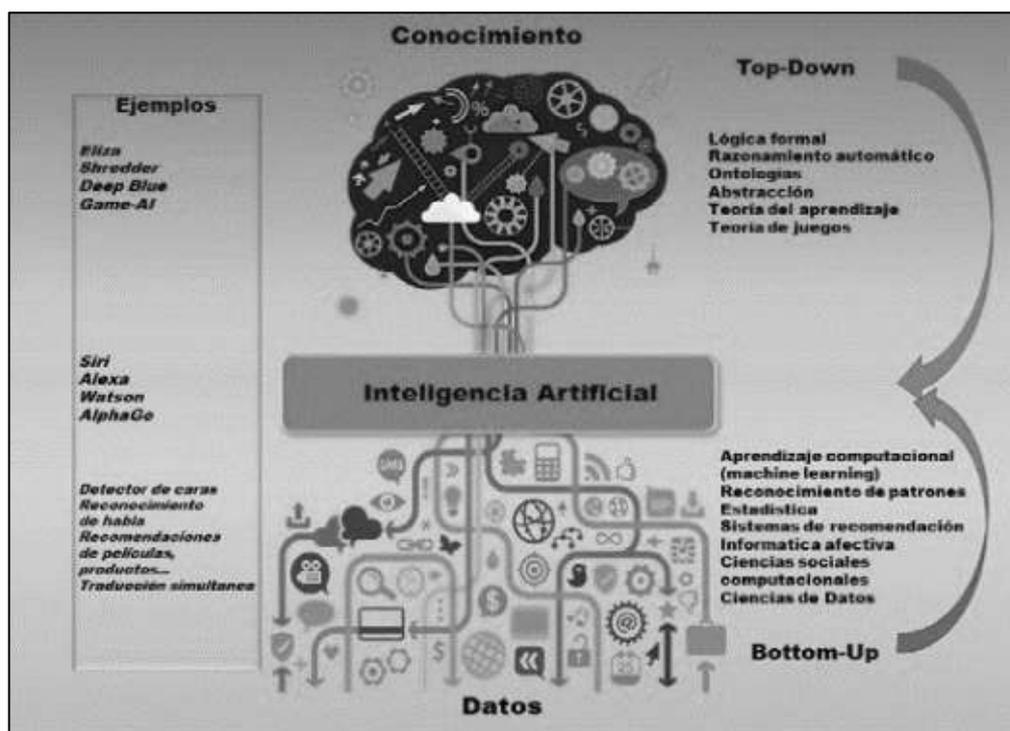
2.2. Bases Teóricas

2.2.1. Inteligencia artificial

En adaptación de Dubova (2022) indica que la inteligencia artificial (IA) se define como la capacidad de una máquina para imitar comportamientos inteligentes similares a los de los humanos, como el razonamiento, el aprendizaje y la adaptación. Según Teigens et al, (2020), la IA involucra la creación de sistemas que pueden igualar o superar el desempeño humano en tareas específicas. Estos sistemas son diseñados para interpretar datos, aprender de ellos y tomar decisiones informadas, lo cual es esencial en aplicaciones como el reconocimiento de voz, la visión por computadora y los sistemas de recomendación, tal como se puede observar en la figura 1 ejemplos de sistemas de inteligencia artificial.

Figura 1

Ejemplos de sistemas top-down y sistemas bottom-up de inteligencia artificial



Nota. Obtenido de Iniciativa Ciudadana para el Control del Sistema de Inteligencia ICCSI Teigens et al, (2020), teorías de la inteligencia artificial.

2.2.2. Redes neuronales

Manifiesta Kumar y Thakur (2012) que las redes neuronales son un subcampo crucial de la IA, inspirado en la estructura y el funcionamiento del cerebro humano, estas redes consisten en capas de nodos (o neuronas artificiales) conectadas entre sí, donde cada conexión tiene un peso ajustable que se aprende durante el entrenamiento, las redes neuronales son especialmente útiles para reconocer patrones complejos y realizar tareas de clasificación y regresión, siendo las más usuales las siguientes:

2.2.2.1. Redes neuronales secuenciales

Kenji (2011) indica que las redes neuronales secuenciales, también conocidas como redes feedforward, son aquellas en las que las conexiones entre neuronas no forman ciclos, estas redes son adecuadas para problemas donde los datos pueden ser procesados en una secuencia fija, como la clasificación de imágenes o la predicción de series temporales. Según Tortajada (2023), una red secuencial típica incluye una capa de entrada, una o más capas ocultas y una capa de salida, dado a que cada capa oculta procesa la información recibida y la pasa a la siguiente capa, permitiendo que la red aprenda representaciones más abstractas de los datos de entrada.

2.2.2.2. Redes neuronales recurrentes

Además, Lipton et al. (2015) manifiestan que las redes neuronales recurrentes (RNN, por sus siglas en inglés) son especialmente efectivas para procesar datos secuenciales y temporales, ya que poseen conexiones que permiten la retroalimentación. Esto significa que las salidas de ciertas neuronas se pueden reintroducir como entradas, lo cual es útil para tareas donde el contexto previo es crucial, como el procesamiento del lenguaje natural. Las unidades de memoria especializadas, como las Long Short-Term Memory (LSTM) y las Gated Recurrent Units (GRU), permiten a las RNN retener información relevante a lo largo del tiempo y actualizarla según sea necesario Huang Ling-fang (2010).

2.2.2.3. Redes neuronales convolucionales

Manifiestan Yadav et al. (2021) que las Redes Neuronales Convolucionales (CNN) son una categoría especializada de redes neuronales que han sido muy efectivas en tareas de reconocimiento de imágenes y voz. Según Keijsers (2010), las CNN son una variante de las redes neuronales multicapa perceptrón que requieren un mínimo de

preprocesamiento. Estas redes utilizan una variación de la neurona estándar de múltiples capas perceptrón llamada neurona de convolución.

Además, Li et al. (2021) plantean que las CNN se inspiran en la organización del sistema visual animal, y son especialmente efectivas para tareas de procesamiento de imágenes. Según Suah (2017), las CNN son capaces de capturar patrones espaciales y temporales en los datos de entrada al aplicar filtros relevantes. La arquitectura de una CNN está diseñada para aprovechar la estructura 2D de una imagen (o de otras señales 2D). Esto se logra con la operación de convolución.

En tanto, Hossain y Alam Sajib (2019) manifiestan que las CNN tienen tres tipos de capas: capas de convolución, capas de pooling o submuestreo, y capas completamente conectadas. Las capas de convolución aplican un conjunto de filtros a la entrada. Cada filtro detecta características a nivel bajo, medio y alto como bordes, texturas, partes de objetos, etc. Las Capas de Pooling o Submuestreo reducen la dimensión espacial (ancho x alto) de la entrada, lo que ayuda a disminuir la cantidad de parámetros, controlando así el sobreajuste. Las Capas Completamente Conectadas se utilizan al final de la red, donde se realiza la clasificación final.

2.2.3. Correlación de variables

Además, Senthilnathan (2019) indica que la correlación de variables es un concepto estadístico que describe la relación o asociación entre dos o más variables. Cuando dos variables están correlacionadas, los cambios en una variable tienden a coincidir con cambios en la otra variable.

Ho y Kuvaas (2020) explican que la correlación puede ser positiva, lo que significa que ambas variables aumentan o disminuyen juntas, o negativa, lo que significa que a medida que una variable aumenta, la otra disminuye. Schober et al. (2018) agregan que la fuerza de esta relación se mide con un coeficiente de correlación, que varía de -1 (correlación negativa perfecta) a +1 (correlación positiva perfecta), donde 0 indica que no hay relación lineal entre las variables.

2.2.4. Métricas utilizadas para el análisis y evaluación de modelo de redes neuronales

Abiodun et al. (2019) indican que las métricas utilizadas para el análisis y evaluación de modelos de redes neuronales son herramientas esenciales para comprender el

rendimiento y eficacia de estos modelos en tareas de clasificación, predicción y reconocimiento de patrones.

2.2.4.1. Matriz de confusión

Es una tabla que resume el desempeño de un modelo de clasificación, mostrando las predicciones correctas e incorrectas por clase. Como explican Gu et al. (2009), los elementos de la diagonal principal representan aciertos (verdaderos positivos y verdaderos negativos), y los elementos fuera de la diagonal, errores (falsos positivos y falsos negativos).

2.2.4.2. Precisión

Es la proporción de predicciones correctas sobre el total de predicciones. Aunque es una métrica intuitiva, Fontes et al. (2022) advierten que puede ser engañosa cuando las clases están desbalanceadas.

2.2.4.3. Sensibilidad

Es la proporción de verdaderos positivos sobre el total de muestras positivas reales. Kallstrom et al. (2020) indican que mide la capacidad del modelo para identificar correctamente las muestras de una clase específica.

2.2.4.4. Perdidas

Es una función que cuantifica la diferencia entre las predicciones del modelo y los valores reales. Shanthamallu y Spanias (2022) señalan que el objetivo del entrenamiento de una red neuronal es minimizar esta función de pérdida.

2.2.4.5. Curva de aprendizaje

Es un gráfico que muestra la evolución del rendimiento del modelo (pérdida o precisión) a lo largo del tiempo o la cantidad de datos de entrenamiento. Permite evaluar cómo el modelo mejora a medida que aprende y si está sobreajustando o subajustando, según explican Tian Seng (2016).

2.2.5. Underfitting

Zhang et al. (2019) manifiestan que el underfitting, o subajuste, ocurre cuando un modelo de aprendizaje automático no es lo suficientemente complejo para capturar la relación subyacente entre los datos de entrada y salida, además en el contexto de las redes neuronales, esto puede manifestarse como un modelo que no logra aprender los patrones relevantes en los datos de entrenamiento, lo que resulta en un rendimiento deficiente tanto en el conjunto de entrenamiento como en el de prueba. Este problema puede surgir debido a una arquitectura de red neuronal demasiado simple, con pocas capas ocultas o un número insuficiente de neuronas en cada capa.

2.2.6. Overfitting

En adaptación de Lai et al. (2018), el overfitting, o sobreajuste, es un problema común en el aprendizaje automático, especialmente en redes neuronales profundas, debido a que ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento, aprendiendo no solo los patrones relevantes sino también el ruido y las fluctuaciones aleatorias presentes en los datos. C. Zhang et al. (2021) indican como resultado, el modelo pierde su capacidad de generalizar a nuevos datos, mostrando un buen rendimiento en el conjunto de entrenamiento, pero un rendimiento deficiente en el conjunto de prueba. El overfitting puede ser causado por un modelo excesivamente complejo, con demasiados parámetros ajustables en relación con la cantidad de datos de entrenamiento disponibles Salman y Liu (2019).

2.2.7. Método de gradiente descendiente

Hikmat Haji y Mohsin Abdulazeez (2021) indican que el método de gradiente descendiente es un algoritmo de optimización ampliamente utilizado en el entrenamiento de redes neuronales dado a que su objetivo es minimizar la función de pérdida del modelo, que mide la discrepancia entre las predicciones del modelo y los valores reales. El algoritmo funciona calculando el gradiente de la función de pérdida con respecto a los parámetros del modelo y ajustando iterativamente estos parámetros en la dirección opuesta al gradiente dado a que este proceso se repite hasta que la función de pérdida converge a un mínimo, lo que indica que el modelo ha aprendido a realizar predicciones precisas (Barron, 2019).

2.2.8. Métodos de regularización

Nusrat y Jang (2018) plantean que existen diversas técnicas para abordar los problemas de overfitting y underfitting en el desarrollo de modelos de redes neuronales, dado a que algunas de las estrategias más comunes incluyen:

2.2.8.1. Regularización

La regularización implica agregar un término de penalización a la función de pérdida del modelo, lo que desalienta la complejidad excesiva y promueve la generalización. Ejemplos de técnicas de regularización incluyen la regularización L1 y L2.

2.2.8.2. Dropout

El dropout es una técnica que consiste en desactivar aleatoriamente un porcentaje de neuronas durante el entrenamiento. Esto evita que el modelo se vuelva demasiado dependiente de un conjunto específico de neuronas y fomenta la redundancia, lo que mejora la capacidad de generalización.

2.2.8.3. Aumento de Datos

El aumento de datos implica generar nuevas muestras de entrenamiento a partir de los datos existentes mediante transformaciones como rotaciones, traslaciones y cambios de escala. Esto aumenta la diversidad del conjunto de entrenamiento y ayuda a prevenir el overfitting.

2.2.8.4. Parada temprana

La parada temprana consiste en detener el entrenamiento del modelo antes de que alcance la convergencia completa. Esto puede evitar que el modelo se ajuste demasiado a los datos de entrenamiento y mejorar su capacidad de generalización.

2.2.9. Filtros digitales tipo butter pasa altos

Widmann et al. (2015) plantean que los filtros digitales tipo Butterworth pasa altos son herramientas de procesamiento de señales que permiten eliminar o atenuar las componentes de baja frecuencia de una señal, conservando las componentes de alta frecuencia, dado a que estos filtros se caracterizan por tener una respuesta en

frecuencia plana en la banda de paso y una atenuación gradual en la banda de rechazo. Mello et al. (2007) indican que en el contexto del procesamiento de señales mioeléctricas (EMG), los filtros Butterworth pasa altos se utilizan para eliminar el ruido de baja frecuencia, como el ruido de movimiento y el ruido de línea eléctrica, que pueden interferir con la detección y análisis de las señales musculares.

2.2.10. Normalización de datos

Lou (2007) indica que la normalización de datos es un paso crucial en el preprocesamiento de datos para redes neuronales, ya que asegura que todas las características tengan una escala similar, evitando que algunas dominen el aprendizaje debido a magnitudes mayores. En esencia, este proceso transforma los datos originales a un rango común, a menudo entre 0 y 1. La ecuación 1 es una fórmula común para la normalización Min-Max.

$$x_{norm} = \frac{(x - x_{min})}{(x_{max} - x_{min})} \quad (1)$$

Donde x es el valor original, x_{min} y x_{max} son los valores mínimo y máximo de la característica, respectivamente. Según Singh (2020), esta técnica es especialmente útil cuando los datos tienen distribuciones variadas y no siguen una distribución normal. No obstante, es importante considerar que la normalización puede ser sensible a valores atípicos, por lo que se recomienda una inspección previa de los datos.

2.2.11. Correlación de variables en aplicaciones de redes neuronales

Jovanović et al. (2015) indican que la correlación de variables desempeña un papel fundamental en el diseño y entrenamiento de redes neuronales. En esencia, mide la relación lineal entre dos variables, indicando si tienden a aumentar o disminuir juntas. Esta información es crucial para seleccionar las características más relevantes y evitar la redundancia en los datos de entrada. En la ecuación 2 se muestra la fórmula matemática para calcular el coeficiente de correlación de Pearson es:

$$\rho(X, Y) = \frac{\text{Cov}(X, Y)}{(\sigma_X * \sigma_Y)} \quad (2)$$

Donde $\text{Cov}(X, Y)$ es la covarianza entre X e Y , y σ_X y σ_Y son las desviaciones estándar de X e Y , respectivamente. Según Aduña et al. (2024), las variables altamente correlacionadas pueden introducir ruido y dificultar el aprendizaje de la red, mientras que las variables poco correlacionadas pueden ser irrelevantes. Por lo tanto, analizar la

correlación de variables permite optimizar la arquitectura de la red y mejorar su rendimiento.

2.2.12. Señales mioeléctricas

El cuerpo humano genera diferentes tipos de señales eléctricas, dependiendo de la parte que la genere, estas pueden clasificarse en oculoográficas, electroencefalográficas y electromiografías. Estas últimas generadas por la contracción de los músculos del cuerpo, en brazos, piernas, abdomen, etc., y son producidas por el intercambio de iones a través de las membranas musculares. A la detección de estas señales se le conoce como electromiografía. Al contraer los músculos, se genera una señal eléctrica de unos cuantos microvoltios, por lo que se hace necesario amplificar esta señal para poder utilizarla (Ahsan et al., 2009).

2.2.13. Electromiografía (EMG)

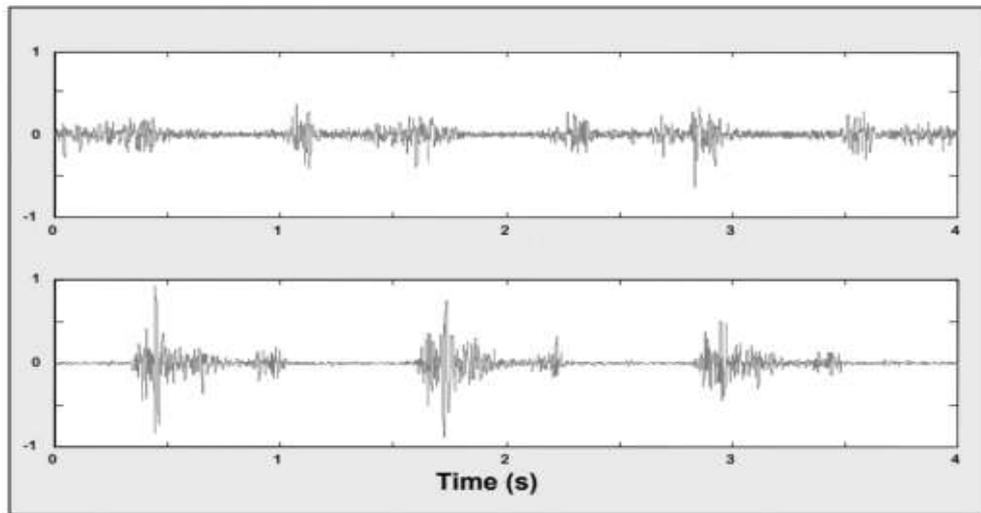
La electromiografía detecta, analiza y procesa las señales eléctricas emitidas por la contracción de los músculos.

En la actualidad, estas señales se obtienen por equipos electrónicos digitales y por tal motivo el término señal Mioeléctrica se adecua en el presente. Sin embargo, hoy EMG sigue siendo el término dominante en ambientes clínicos Chowdhury et al. (2013).

En el interior de los músculos se generan tensiones de alrededor 100mV al contraerse. La amplitud de estas señales es atenuada por el tejido interno y la piel, pero es posible ser medidas en la superficie de la piel. Los valores más comunes de EMG para los músculos grandes, como el bíceps, están dentro de 2,1mV en amplitud además de que estas señales oscilan a frecuencias de 2 Hz a 500 Hz Ludvig et al. (2011). En la figura 2 se puede observar dos señales EMG realizada por el bíceps.

Figura 2

Dos señales EMG producido por la contracción del bíceps. La escala de Tiempo es 500ms por recuadro



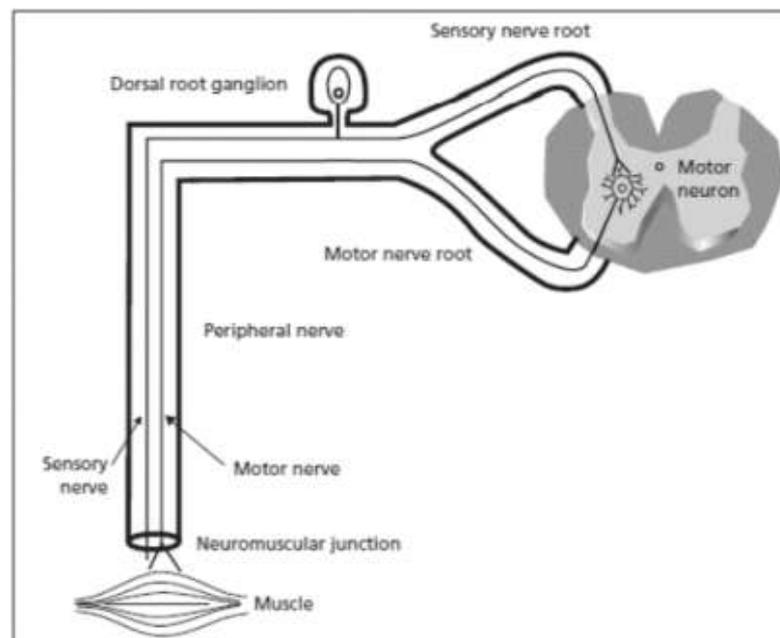
Nota. Tomado de "Imaging the Behavior of Motor Units by Decomposition of the EMG signal". De Luca. Delsys Inc. Boston. 2008, p. 6. Generación de señales EMG.

2.2.13.1. Unidad motora

Se encuentran ubicadas en la corteza cerebral motora y se conectan con otras neuronas del tronco cerebral para pasar por la medula espinal y finalmente llegar a los músculos receptores tal y como se muestra en la figura 3 (Heckman y Enoka, 2004).

Figura 3

Unidad motora



Nota. "Why do Electrodiagnostic Studies?" L. Weiss, J. Silver y J. Weiss. Easy EMG. 1ª Ed. 2004. Cap 2, pp. 5-8.

2.2.13.2. Fibras musculares

Son aquellas que se encuentran ramificadas por el cuerpo humano y se manifiestan con una fuerza de contracción cuando reciben una excitación de manera consecutiva Srivastava y Chosdol (1997).

2.2.13.3. Axón de la unidad motora

Permite la ramificación de las motoneuronas y son las que soportan la sinapsis para el envío de las señales a los músculos Myers y Lovelace (1994).

2.2.14. Composición de una señal electromiografía

En adaptación de Del Vecchio et al. (2017) la composición de una señal electromiográfica (EMG) se basa en la actividad eléctrica generada por las unidades motoras (UM) presentes en los músculos, además una UM está compuesta por una motoneurona y las fibras musculares que inerva, cuando una motoneurona se activa, genera un potencial de acción que se propaga a lo largo de su axón y llega a las fibras musculares, provocando su contracción, debido a que este potencial de acción individual se denomina potencial de unidad motora (PUM).

2.2.14.1. Unidad motora

En la unidad motora se realizan una serie de disparos conocida como la tasa de excitación de unidades motoras, estas producen la fuerza de contracción en las fibras musculares además de que es directamente proporcional a la fuerza de salida Duchateau y Enoka (2011).

2.2.14.2. Potencial de unidad motora

Este término se le conoce como PUM y hace referencia al potencial eléctrico que produce una unidad motora hacia las fibras musculares Rodríguez y Place (2014).

2.2.14.3. Tren de potencial de unidad motora

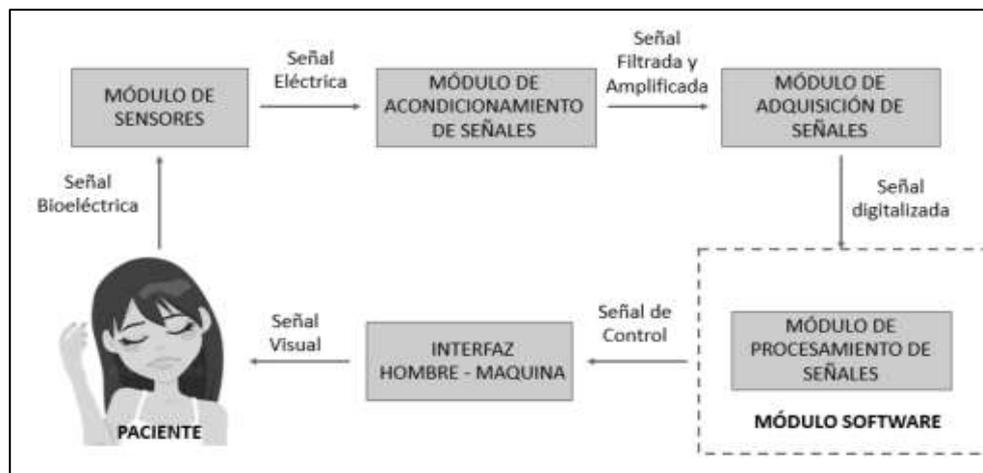
Este término se le conoce como TPUM y hace referencia a la serie producida por las PUM. Los TPUM son lo que producen la señal EMG Petit y Loccufier (2009).

2.2.15. Características técnicas de un dispositivo EMG

Este dispositivo cuenta con un módulo de sensores el cual se encarga de detectar las señales mioeléctrica generadas por las contracciones musculares del paciente y convertirlas en señales eléctricas con una muy baja amplitud. Estas señales eléctricas se dirigen al módulo de acondicionamiento de señales el cual se encarga de amplificarlas y filtrarlas para eliminar ruido proveniente de la red eléctrica y el ambiente que pueden generar lecturas falsas de la información; en el bloque de adquisición de señales se realiza la digitalización de los datos para entrar al procesador. Para el procesado de las señales se debe caracterizar y clasificar Al-Ayyad et al. (2023). La figura 4, muestra el diagrama de bloques de un sistema EMG.

Figura 4

Diagrama de bloques de un sistema EMG



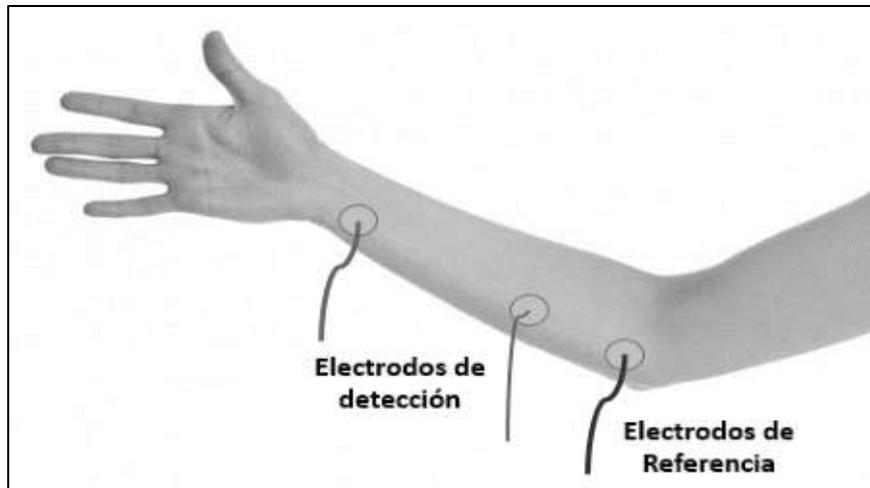
Nota. Aplicaciones de las señales mioeléctrica para el control de interfaz hombre-máquina. Edgar Muñoz Burbano universidad del Cauca. 2015, pp. 2-3.

2.2.15.1. Sensores no invasivos

El electrodo es el sensor que entra en contacto con la piel y tiene un mejor desempeño cuando este cuenta con una baja impedancia. Para lograr esto se usa un gel conductor comúnmente compuesto por Ag/AgCl. además, se debe contar con un circuito de seguridad para proteger al paciente de riesgo eléctrico Mao et al. (2023) la ubicación de los electrodos se puede observar en la figura 5.

Figura 5

Ubicación de electrodos en el registro EMG de antebrazo



Nota. Aplicaciones de las señales mioeléctrica para el control de interfaz hombre-máquina. Edgar Muñoz Burbano, Universidad del Cauca. 2015, pp. 2-3.

2.2.16. Amplificadores

En adaptación de Sharma (2019), los amplificadores su función es amplificar los pequeños niveles de voltajes adquiridos en los músculos de manera que puedan estar a una escala que pueda reconocer el dispositivo de procesamiento o visualización, su ganancia puede ser superior al millón de veces (60 dB), lo cual permite visualizar una señal de 5 microvoltios. Además, debe responder con fidelidad a señales entre los 40 y los 10KHz.

Según Barea (2016), las principales características de los amplificadores utilizados en EMG son:

- Número de canales: 2 (lo más habitual).
- Sensibilidad: 1 pV/div. a 10 mV/div.
- Impedancia de entrada: 100 MW//47 pF.
- CMRR a 50 Hz > 100 dB.
- Filtro de paso alto: entre 0,5 Hz y 3 kHz (6 dB/octava).
- Filtro de paso bajo: entre 0,1 y 15 kHz (12 dB/octava).
- Ruido: (1 pV eficaz entre 2 Hz y 10 kHz con la entrada cortocircuitada).

2.2.17. Inteligencia artificial

La inteligencia artificial puede definirse de diversas maneras según el enfoque de la aplicación. La inteligencia artificial involucra crear sistemas que sean capaces de imitar comportamientos inteligentes como los procesos de pensamiento y razonamiento de los humanos. Además, este comportamiento puede ser medido para determinar el desempeño de la inteligencia artificial en una tarea específica la cual puede igual o superar en rendimiento a un ser humano (Shabbir y Anwer, 2018).

2.2.18. Redes neuronales

Por adaptación Journal et al. (2013) mencionan que las redes neuronales es una forma de computación inspirada en la biología para procesar datos de un entorno los cuales deben pasar por un número determinado de capas para obtener un resultado en función del aprendizaje que esta ha recibido. Estas redes neuronales se organizan en capas y cada unidad en una capa está conectada a las unidades de la capa anterior. Estas redes pueden tener una función de activación, esto permite a la red representar una función no lineal Sharma et al. (2020).

De acuerdo con Katte (2018), las redes pueden ser de dos tipos: redes neuronales secuenciales y redes neuronales recurrentes.

2.2.18.1. Redes neuronales secuenciales

Usadas generalmente en problemas relacionadas con problemas de soluciones lineales o temporales. las conexiones de las neuronas fluyen en una sola dirección y están diseñadas para trabajar con datos de series de tiempo (Weerakody et al., 2021).

Las redes se pueden componer de capas las cuales se describen a continuación:

Capa de entrada:

Es la encargada de recibir la secuencia de datos de entrada

Capas ocultas:

Son las capas intermedias que contienen neuronas que procesan la información y logran identificar patrones abstractos de la serie de datos. Además, una red neuronal se puede componer de una o más capas ocultas.

Capa de salida:

Es la encargada de generar la predicción esperada. Dependiendo de la aplicación pueden existir una o más neuronas en la capa de salida.

2.2.18.2. Redes neuronales recurrentes

Pascanu et al. (2014) mencionan que este tipo de redes se utilizan para trabajar con datos secuenciales. Estas cuentan con una arquitectura que les permiten tener una conexión de retroalimentación para procesar los datos. Esto es muy útil para problemas que deben tener en cuenta la información anterior de manera relevante.

Estas redes cuentan con conexiones retroalimentadas que unen la salida de un paso anterior de tiempo como entrada en el siguiente paso y como consecuencia permite a la red mantener una pseudo-memoria para comprender el contexto de la secuencia hasta ese paso de tiempo (Kim et al., 2020).

Las unidades de memoria utilizadas en las redes neuronales recurrentes se denominan “Long short-term memory” (LSTM) y “Gated recurrent unit” (GRU) los cuales le dan la capacidad de memoria y actualizar la información en ella a lo largo del tiempo, elemento que es fundamental para captura de dependencia de datos secuenciales (Ashraf, 2021).

2.2.19. Caracterización de señales

Rampichini et al. (2020) postulan que la caracterización de señales mioeléctricas (sEMG) es fundamental para extraer información relevante sobre la actividad muscular y su relación con el movimiento. Este proceso implica analizar los patrones y tendencias presentes en estas señales, aparentemente caóticas, utilizando métodos estadísticos tanto en el dominio del tiempo como en el de la frecuencia.

Bhattacharya et al. (2017) plantean que, en el dominio del tiempo, características como el valor medio absoluto (MAV), la raíz cuadrática media (RMS) y el cruce por cero (ZC) han demostrado ser eficaces para cuantificar la amplitud y frecuencia de la señal sEMG. El MAV, por ejemplo, se calcula como se muestra en la ecuación 3:

$$MAV = \left(\frac{1}{N}\right) * \sum^{\circ} |x(n)| \quad (3)$$

donde N es el número de muestras y x(n) es la amplitud de la muestra n. Según Elamvazuthi et al. (2013) y Shahzaib y Shakil (2019), el MAV y el RMS son especialmente útiles para la detección de la activación muscular y la estimación de la fuerza de contracción.

En el dominio de la frecuencia, la frecuencia media (MNF) y la varianza (VAR) son características relevantes para analizar el contenido espectral de la señal sEMG. No obstante, Elamvazuthi et al. (2013) sugieren que las características en el dominio

del tiempo ofrecen un mejor rendimiento en el preprocesamiento de la señal, mientras que Shahzaib y Shakil (2019) incluyen la MNF en su análisis para complementar la información obtenida en el dominio del tiempo.

2.2.19.1. Raíz media cuadrática (RMS)

La raíz cuadrática media (RMS) es una medida estadística que cuantifica la magnitud de una señal variable en el tiempo (Ecuación 4). Se calcula como la raíz cuadrada del promedio de los cuadrados de los valores de la señal. En el contexto de las señales mioeléctricas (sEMG), el RMS se utiliza para estimar la amplitud y la intensidad de la actividad muscular. Según Berthouze y Farmer (2012), el RMS es una característica robusta y ampliamente utilizada en aplicaciones de reconocimiento de patrones y control de prótesis mioeléctricas.

Además, es un valor estadístico para realizar mediciones de magnitudes variables. Este método es muy utilizado para realizar mediciones de ondas que oscilan entre valores positivos y negativos (Hurtado et al., 2004).

$$RMS = \sqrt{\frac{1}{N} \sum_{n=1}^N x_n^2} \quad (4)$$

2.2.19.2. Valor medio absoluto (MAV)

La ecuación 5 muestra el valor absoluto se puede calcular tomado el promedio del valor absoluto del valor de la señal EMG y se puede definir de la siguiente manera.

$$MAV = \frac{1}{N} \sum_{n=1}^N |x_n| \quad (5)$$

El valor medio absoluto (MAV) es otra medida estadística que se utiliza para caracterizar la amplitud de una señal. Se calcula como el promedio de los valores absolutos de la señal. En el caso de las señales sEMG, el MAV se utiliza para detectar la activación muscular y estimar el nivel de contracción. De acuerdo con Resalat y Saba (2016), el MAV es una característica simple pero efectiva para aplicaciones de interfaz cerebro-computadora y rehabilitación.

2.2.19.3. Frecuencia media (MNF)

La frecuencia media se calcula utilizando el cuadrado de la media de la magnitud de la Transformada Rápida de Fourier (DFT) de la señal EMG en una ventana de tamaño N. Este cálculo se puede definir de la siguiente manera en la ecuación 6 y 7:

$$MNF = \frac{1}{N} \sum_{k=1}^N F(x_k) \quad (6)$$

donde:

$$F(x_k) = DFT(|x_k|^2) \quad (7)$$

De igual modo, la frecuencia media (MNF) es una medida espectral que indica la frecuencia promedio de una señal. Se calcula como el promedio ponderado de las frecuencias presentes en el espectro de la señal, donde los pesos son las magnitudes de las componentes de frecuencia. En el contexto de las señales sEMG, la MNF se utiliza para analizar el contenido de frecuencia de la señal y detectar cambios en la fatiga muscular. Según Merletti (2004), la MNF es una característica útil para evaluar la calidad de la señal y optimizar el procesamiento de la misma.

2.2.19.4. Cruce por cero (ZC)

El cruce por cero (ZC) es una medida temporal que cuenta el número de veces que una señal cruza el nivel cero en un intervalo de tiempo determinado. En el caso de las señales sEMG, el ZC se utiliza para estimar la frecuencia de la señal y detectar cambios en la actividad muscular. De acuerdo con Menon et al. (2017), el ZC es una característica simple pero sensible para aplicaciones de control de prótesis y rehabilitación.

2.2.19.5. Transformada rápida de Fourier

La ecuación 8 muestra la transformada de Fourier que es una herramienta matemática que permite llevar señales en el dominio del tiempo hacia el dominio de la frecuencia para observar las características de una señal que pueden usarse en sistemas DSP.

La transformada discreta de Fourier (DFT) trabaja con señales finitas y discretizadas en el tiempo dado como respuesta a una frecuencia discretizada (Menon et al., 2017).

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\left(\frac{2\pi}{N}\right)kn}, \quad 0 \leq k \leq N - 1 \quad (8)$$

La transformada de Fourier es una herramienta matemática fundamental que, según Willsky y Oppenheim (1998), "permite descomponer señales complejas en componentes de frecuencia individuales, facilitando el análisis y procesamiento de

dichas señales". Esta transformación posibilita trasladar una señal del dominio del tiempo al dominio de la frecuencia. Además, como señalan Willsky y Oppenheim, (1998), "la transformada discreta de Fourier (DFT) es la versión discretizada de la transformada de Fourier, específicamente diseñada para trabajar con secuencias finitas de muestras digitales". Por lo tanto, la DFT es particularmente útil en el procesamiento digital de señales, donde las señales se representan como secuencias discretas en el tiempo.

Como explica Mitra (2001), "la transformada de Fourier constituye una poderosa herramienta para el análisis de señales y sistemas debido a su capacidad de representar cualquier señal periódica como una sumatoria de componentes de frecuencia armónicas". Adicionalmente, Strauss (2000) destaca que "la transformada discreta de Fourier es un caso especial de la transformada de Fourier, adaptada para su aplicación eficiente en sistemas digitales que procesan secuencias finitas de datos discretos". En resumen, como establece Ingle y Proakis (2008), "la transformada de Fourier y su contraparte discreta son herramientas fundamentales en el análisis y procesamiento digital de señales, indispensables en áreas como las telecomunicaciones, el procesamiento de audio y video, entre otras aplicaciones".

2.2.19.6. Lenguaje de programación Python

Python es un lenguaje de programación de alto nivel, interpretado, multipropósito y de código abierto (Van Rossum y Drake, 2012). Debido a su sintaxis clara y legible, se ha convertido en uno de los lenguajes más populares en diversas áreas, incluyendo el procesamiento de datos y el aprendizaje automático (Ramalho, 2015).

En el campo del procesamiento de datos, Python ofrece varias bibliotecas y herramientas que facilitan el manejo, análisis y visualización de conjuntos de datos. Una de las bibliotecas más importantes es NumPy, que proporciona estructuras de datos y funciones para cálculos numéricos eficientes (McKinney, 2012). Otra biblioteca esencial es Pandas, que permite trabajar con estructuras de datos tabulares y series de tiempo (Chollet, 2021). Además, librerías como Matplotlib, Seaborn y Plotly ofrecen capacidades de visualización de datos.

2.2.19.7. Librería Keras

En el ámbito del aprendizaje automático y el desarrollo de redes neuronales, Python cuenta con bibliotecas altamente eficientes y populares. Una de las más destacadas es

TensorFlow, desarrollada por Google, que proporciona un marco de trabajo para el aprendizaje automático y la computación numérica (Stancin y Jovic, 2019). Otra biblioteca ampliamente utilizada es PyTorch, creada por Facebook, que ofrece una interfaz de programación intuitiva y flexible para el desarrollo de modelos de aprendizaje profundo Paszke et al. (2019).

Además de estas bibliotecas principales, existen otras herramientas complementarias para el desarrollo de modelos de redes neuronales en Python. Una de ellas es Keras, una biblioteca de alto nivel que actúa como una interfaz intuitiva sobre TensorFlow o Theano, lo que facilita la construcción y experimentación con modelos de aprendizaje profundo (Chollet, 2021). Keras proporciona una API fácil de usar y modular, lo que permite a los desarrolladores construir y entrenar redes neuronales de manera eficiente, sin tener que lidiar con los detalles de bajo nivel.

2.2.19.8. Google Colab

El entorno de desarrollo Google Colaboratory, comúnmente conocido como Google Colab, se ha consolidado como una herramienta fundamental en el campo de la inteligencia artificial y el aprendizaje automático (Bisong y Ekaba, 2019). Este entorno, basado en la nube, ofrece un acceso gratuito a recursos computacionales potentes, incluyendo unidades de procesamiento gráfico (GPU) y unidades de procesamiento tensorial (TPU), lo que facilita la ejecución de modelos complejos y el procesamiento de grandes volúmenes de datos (Gulli y Pal, 2017).

Google Colab se distingue por su interfaz intuitiva y su integración con otras herramientas de Google, como Google Drive, lo que permite una gestión eficiente de los proyectos y la colaboración en tiempo real (Bisong y Ekaba, 2019). Además, Colab admite la ejecución de código en Python, el lenguaje de programación predominante en el ámbito de la IA, y ofrece una amplia gama de bibliotecas preinstaladas, como TensorFlow y PyTorch, que simplifican el desarrollo y la experimentación con modelos de aprendizaje automático (Gulli y Pal, 2017).

2.3. Definición de términos

2.3.1. Señal mioeléctrica

La señal mioeléctrica es la actividad eléctrica generada por la contracción de las fibras musculares durante el intercambio iónico en las células musculares (Pinto et al., 2020).

2.3.2. Electromiografía de superficie (EMG)

Técnica no invasiva que registra la actividad eléctrica muscular mediante electrodos colocados en la piel sobre los músculos de interés (Solis, 2020).

2.3.3. Caracterización de señales

Técnica de análisis de las señales para extraer características en distintos intervalos de tiempo y espectro mediante métodos estadísticos (Oppenheim y Schafer, 1999).

2.3.4. Clasificación EMG

Método computarizado que clasifica a partir de patrones característicos de las señales EMG para el control de prótesis Englehart y Hudgins, (2003).

2.3.5. Biopotencial

Los biopotenciales son los potenciales de acción transmitidos a través del tejido muscular debido al intercambio iónico, generando una diferencia de potencial a lo largo del tejido (Kandel et al., 2017).

2.3.6. Redes neuronales artificiales

Las redes neuronales artificiales son algoritmos computacionales inspirados en la biología que procesan información a través de capas para producir una salida según su entrenamiento (Goodfellow y Bengio, 2017).

2.3.7. Google Colaboratory (Colab)

Entorno de desarrollo en la nube que brinda acceso gratuito a recursos computacionales potentes, como GPU y TPU, para la ejecución de modelos complejos y el procesamiento de grandes volúmenes de datos (Bisong y Ekaba, 2019).

2.3.8. Python

Lenguaje de programación de alto nivel ampliamente utilizado en el campo de la inteligencia artificial y el aprendizaje automático, conocido por su sintaxis clara y su amplia gama de bibliotecas especializadas (Van Rossum y Drake, 2012).

2.3.9. IA

La inteligencia artificial (IA) se basa en el desarrollo de algoritmos que emplean métodos estadísticos sofisticados para analizar datos y descubrir patrones, con el objetivo de resolver problemas (Mohanty et al., 2016).

2.3.10. RMS

El valor RMS es una medida estadística utilizada para caracterizar y cuantificar la magnitud de una señal oscilante, como las señales electromiográficas (EMG). Se calcula como la raíz cuadrada del promedio de los cuadrados de los valores de la señal en un determinado intervalo de tiempo (Phinyomark et al., 2018).

2.3.11. MAV

El MAV es otra característica temporal utilizada para analizar señales EMG. Se calcula como el promedio de los valores absolutos de la señal en una ventana de tiempo determinada (Phinyomark et al., 2018).

2.3.12. MNF

El MNF es una característica espectral que representa la frecuencia media de una señal EMG en un intervalo de tiempo determinado. Se calcula a partir de la transformada rápida de Fourier (FFT) de la señal (Phinyomark et al., 2018).

CAPÍTULO III: MARCO METODOLÓGICO

3.1. Diseño de la investigación

El diseño de esta investigación fue de tipo experimental, ya que se manipuló deliberadamente la variable independiente (movimientos de la mano) para observar su efecto en la variable dependiente (diseño del prototipo de sistema de adquisición de señales) en un entorno controlado (Hernandez, et al. 2014). Este enfoque permitió establecer relaciones causales entre las variables y evaluar la eficacia del prototipo en la clasificación de los movimientos de la mano.

3.2. Acciones y actividades

3.2.1. Determinación de requerimientos y planeación

3.2.1.1. Planeación

Se desarrolló el diseño de un prototipo de circuito electrónico basado en sistemas embebidos el cual utilizará sensores, un proceso de filtrado, amplificación y de procesado para clasificar el estado de la mano.

Para el funcionamiento del prototipo se requirió contar con sensores a nivel de hardware y software para desarrollar los algoritmos lógicos que van a controlar el procesado de las señales.

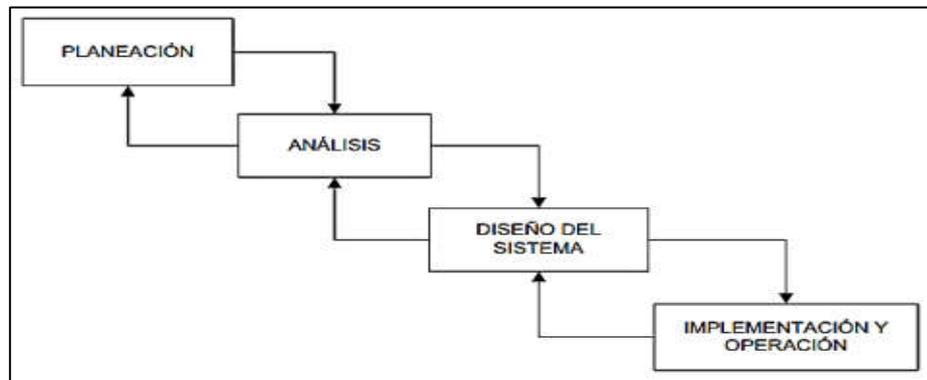
Para el desarrollo del proyecto se utilizó un modelo de proceso iterativo e incremental para lo cual se utilizará el ciclo de vida de cascada ya que una de sus ventajas es que permite realizar correcciones tempranas de errores, sin embargo es muy necesario conocer todos los requerimientos del prototipo desde un inicio.

3.2.1.2. Ciclo de vida de cascada

En la figura 6 se observa el esquema del ciclo de vida de cascada.

Figura 6

Ciclo de vida de cascada

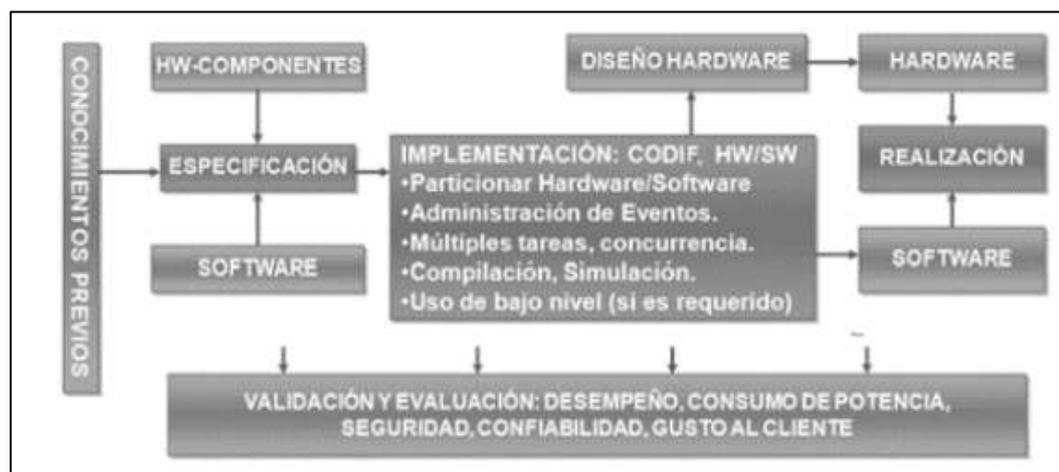


Nota. “Desarrollo de un prototipo funcional de bajo costo para la Medición de la severidad vibratoria en máquinas rotativas según las normas ISO 2372 y 10816-3”, Serna.2015, p. 51.

Como metodología del diseño de hardware y software se utilizaron metodologías de codiseño con el fin de obtener un el software y hardware más óptimo en función de los requisitos de tal manera que se pueda realizar un desarrollo en paralelo tanto en software como en hardware, tal y como se muestra en la figura 7.

Figura 7

Metodologías de codiseño

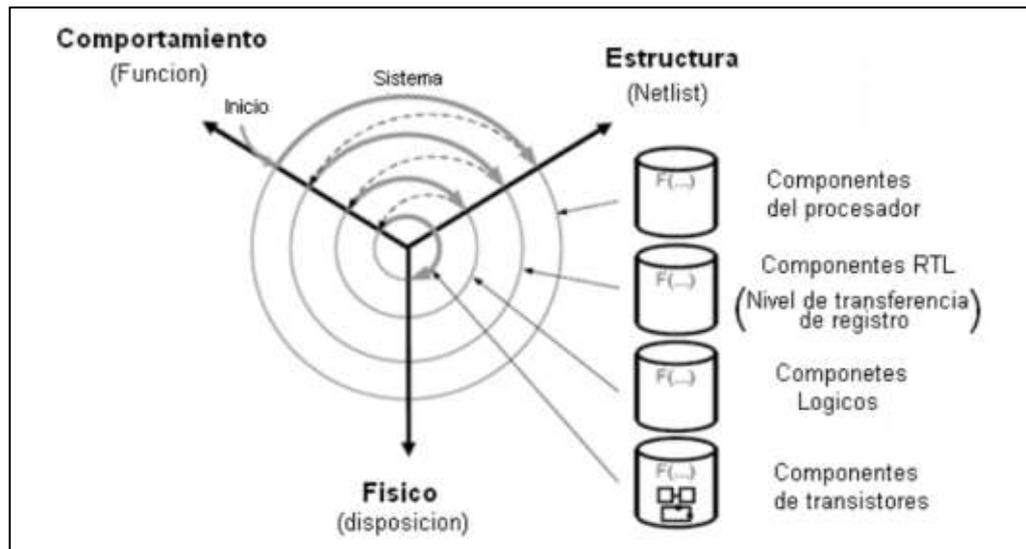


Nota. “Embedded Systems Design”. Marwedel.2006, p. 250.

Además, se tomó en cuenta la metodología Top Down (Figura 8), la cual realizó una perspectiva desde los niveles altos del hardware los cuales definen los requerimientos del prototipo hacia los inferiores donde se diseñan los componentes funcionales.

Figura 8

Metodología Top-Down



Nota. "Embedded System Design Modeling". Gajski .2009, p. 366.

3.2.1.3. Requerimientos

3.2.1.3.1. Funcionalidad

El prototipo recolectó las señales musculares a través de sensores especializados. Estas señales, una vez captadas, serán transformadas en señales eléctricas que necesitarán ser amplificadas y filtradas para asegurar su claridad y precisión. Posteriormente, estas señales deberán ser digitalizadas, lo que permitirá su ingreso al módulo encargado del procesamiento de las mismas. En este módulo, un algoritmo avanzado clasificará el estado del movimiento de la mano mediante el reconocimiento de patrones en las señales recibidas.

3.2.1.3.2. Composición de hardware

El hardware debe consistir en una etapa, obtención de las señales bioeléctricas del organismo, una etapa de pre-amplificación, filtrado y amplificación para ingresar al ADC para su digitalización de la señal para que el CPU se encargue de procesar las señales para obtener una salida.

3.2.1.3.3. Especificaciones técnicas

Se determinó el nivel de voltaje y frecuencia que necesita para su funcionamiento; además de las especificaciones técnicas que se requieren en cada etapa.

3.2.1.3.4. Parámetros de sensores

Se va a determinar el nivel de voltaje que debe proveer los sensores para poder ingresar al apartado de pre-amplificación.

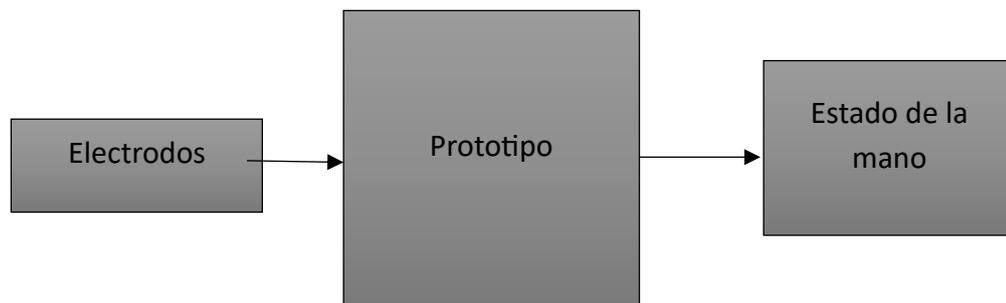
3.2.1.4. Concepción del sistema

3.2.1.4.1. Caja negra

En la figura 9 se muestra una caja negra la cual obtiene sus entradas a partir de los electrodos y da como respuesta el estado de la mano.

Figura 9

Diagrama de caja negra

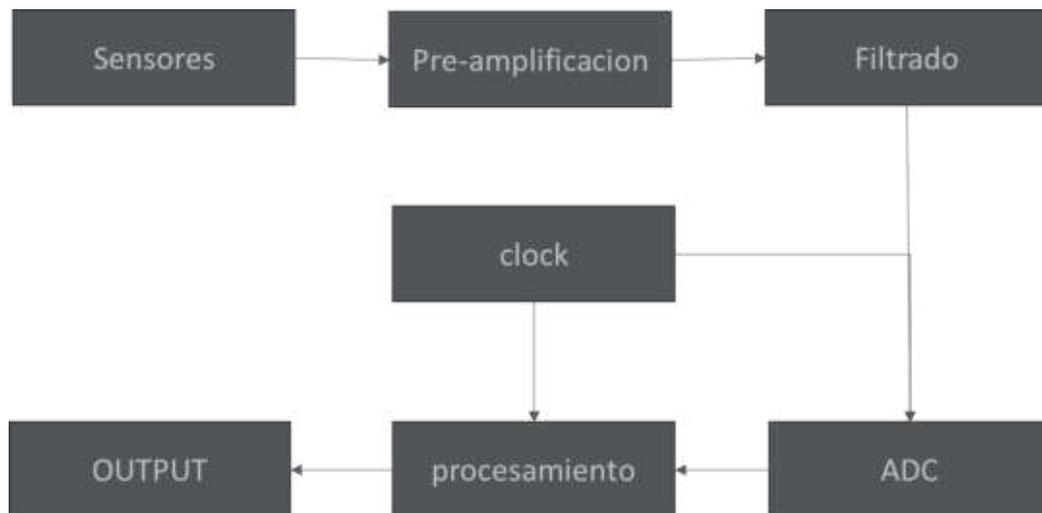


3.2.1.4.2. Diagrama funcional

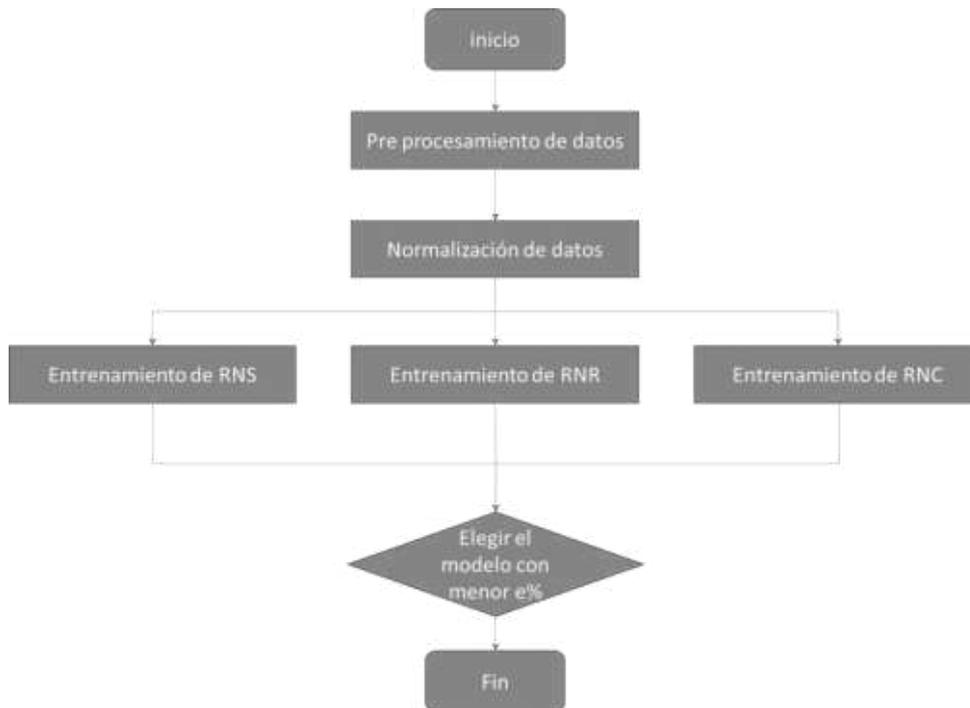
En la figura 10 se muestran los bloques funcionales para las etapas que se consideran en los requerimientos del prototipo.

Figura 10*Diagrama funcional***3.2.1.5. Diagrama de bloques**

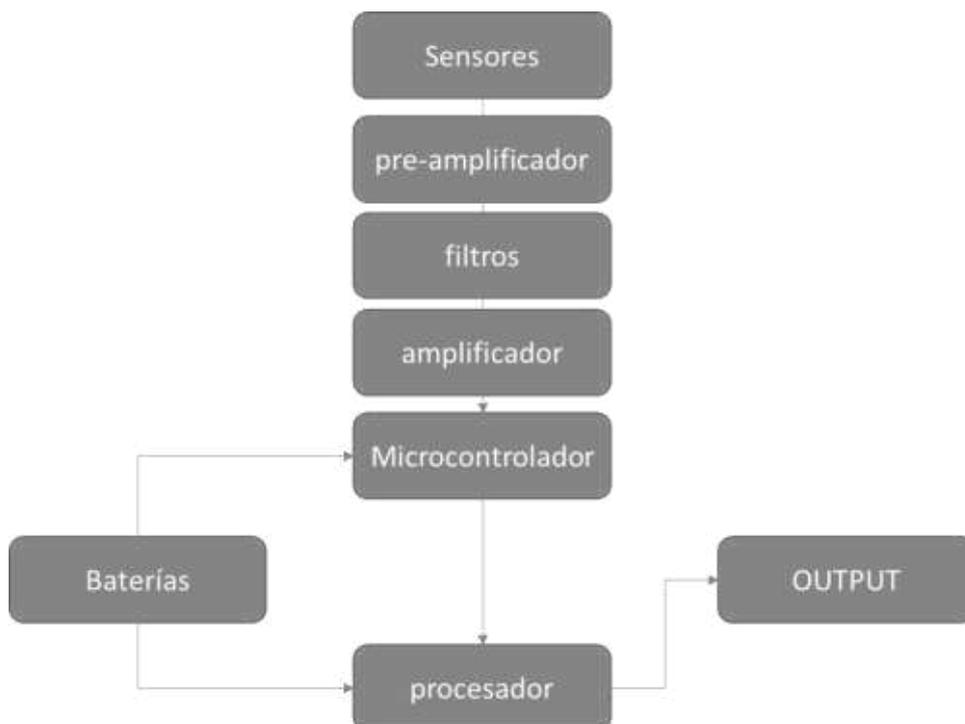
En la figura 11 se muestra el diagrama de bloques del prototipo en base a los bloques de funcionalidad del sistema.

Figura 11*Diagrama de bloques***3.2.1.6. Diagrama de flujo**

En la Figura 12 se muestra el diagrama de flujo de la información el cual permite al sistema tomar una decisión en función de los parámetros de entrada para indicar el modelo de red neuronal con mejor desempeño.

Figura 12*Diagrama de flujo***3.2.1.7. Diagrama de hardware del sistema**

En la Figura 13 se muestran los componentes de hardware para el desarrollo del prototipo.

Figura 13*Diagrama de hardware*

3.2.1.8. Análisis y diseño

3.2.1.8.1. Diseño lógico

3.2.1.8.1.1. Diagrama de bloques de sensores

En la figura 14 se muestra el bloque de sensores se encarga de transformar los bioseñales en señales de corriente los cuales tienen niveles de voltaje y frecuencia los cuales envía al sistema de control.

Figura 14

Diagrama de bloques de sensores



3.2.1.8.1.2. Diagrama de bloques de procesado

Este bloque se encarga de analizar las señales amplificadas y filtradas, por lo que será el encargado de dar la respuesta del output tal y como se puede observar en la figura 15.

Figura 15

Diagrama de bloques de control



3.2.1.8.1.3. Diseño físico

La estructura de la prótesis activa debe poder ser portátil y obtener energías recargables mediante las baterías además de que este será probado en un ambiente de laboratorio.

3.2.1.8.2. Desarrollo e implementación

3.2.1.8.2.1. Implementación del hardware

Su desarrollo e implementación del este prototipo se realizará en un laboratorio para poder crear condiciones apropiadas para el experimento y ya que su implementación será en protoboard se utilizarán modelos de desarrollo para su funcionamiento.

3.2.1.8.2.2. Programación del sistema

La programación se llevará a cabo en un ordenador portátil el cual contará con el software necesario para la programación y análisis del prototipo. De tal manera se obtendrá una comodidad para las condiciones de trabajo que se llevaran a cabo.

3.2.1.8.2.3. Pruebas

Funcionamiento de la obtención de las señales.

Correcta interpretación de los datos.

Correcta asignación de patrones a los estados correspondientes.

3.3. Materiales e instrumentos

El estudio utiliza electrodos de superficie para captar señales electromiográficas en los músculos de la mano, digitalizándolas mediante un dispositivo de adquisición de datos (DAQ). Para la integración del sistema se utilizará un microcontrolador o una placa de desarrollo. Para el análisis y procesamiento de las señales se utilizará software especializado en aprendizaje automático, librerías como TensorFlow o Scikit-learn. Se utilizará un ordenador con suficiente capacidad de procesamiento para las simulaciones y clasificaciones de los movimientos de la mano. Se utilizarán herramientas de creación de prototipos y montaje para construir y probar el sistema prototipo.

3.4. Operacionalización de variables

Independiente: movimiento de la mano de una persona.

Dependiente: el diseño del prototipo (sistema de adquisición de señales).

Identificación y/o caracterización de las variables.

En la tabla 1 y 2 se muestra la operacionalización de la variable dependiente e independiente donde se indican los parámetros y los indicadores de los mismos.

Tabla 1*Operacionalización de variable de la variable dependiente*

Variable dependiente	Definición operacional	Parámetros	Definición	Indicador
Clasificación de movimiento de la mano	Posición de la mano en función de valores promedios de voltaje y frecuencia	Algoritmo	Desarrollado en el ordenador para indicar el estado de la mano	Estado de la mano: abierto, cerrado, relajado

Tabla 2*Operacionalización de variable de la variable independiente*

Variable independiente	Definición operacional	Parámetros	Definición	Indicador
Sistema de adquisición de biopotencial muscular	Sistema que adquiere los datos digitalizados que provienen de los electrodos ubicados brazo del paciente	Voltaje	Diferencia de potencial que existe entre 2 puntos de ubicación de los electrodos en el músculo del brazo	mV
		Frecuencia	El periodo entre impulsos bioeléctricos	Hz

CAPÍTULO IV: RESULTADOS

4.1. Elección de tarjeta de adquisición de señales EMG

Se optó trabajar con una tarjeta existente en el mercado de la marca SichyRay que cumplía con los requerimientos planteados (anexo 2). Cuenta con la posibilidad de energizarse con una batería de 9V para mostrar 2 canales de salida ideal para la lectura de potencial de los músculos extensor y flexor, las señales son acondicionadas y son enviadas al ordenador mediante una interfaz bluetooth para ser almacenadas en raw haciendo uso del software del producto (anexo 3).

4.2. Elección de electrodos

Los electrodos utilizados con este proyecto fueron los electrodos con gel conductor de la marca 3M disponibles en las farmacias locales. El gel conductor ayuda a la conductividad de las señales mioeléctricas el cual incrementa el performance de la tarjeta de adquisición de señales EMG (anexo 4).

4.3. Preparación del sujeto de prueba

Para esta etapa de la preparación del sujeto de prueba, se realizó una limpieza minuciosa del área del antebrazo donde se colocarían los sensores, asegurando que la piel estuviera completamente libre de suciedad, grasa o cualquier otro residuo que pudiera interferir con la conductividad de los electrodos. Esto se hizo utilizando materiales y técnicas específicas, como el uso de alcohol o toallitas de limpieza especializadas, para garantizar una superficie cutánea óptima para la colocación de los sensores.

Una vez que la piel estaba adecuadamente preparada, se procedió a la identificación de las zonas del antebrazo donde se colocarían los sensores en los músculos, siguiendo un análisis anatómico detallado. Esta identificación se basó en localizar las áreas musculares específicas que serían más relevantes para la medición precisa de señales mioeléctricas. La correcta identificación de estas áreas es crucial, ya que una colocación incorrecta de los sensores podría resultar en una captura de datos errónea o en señales de baja calidad.

Posteriormente, los sensores fueron colocados en las posiciones exactas recomendadas por el fabricante del equipo siendo colocados los electrodos de

diferencia de potencial en los músculos flexor y extensor de los dedos de la mano ubicados en el antebrazo, siguiendo cuidadosamente tanto las indicaciones específicas para la medición de señales mioeléctricas como las recomendaciones para la disposición de los electrodos de medición de voltaje y el electrodo de referencia sin superar los 30mm de distancia. En este proceso, se tuvo en cuenta la correcta orientación y espaciado de los electrodos para minimizar la interferencia de señales no deseadas y maximizar la precisión de la lectura. Además, se verificó que los electrodos estuvieran bien adheridos a la piel para evitar desplazamientos durante el experimento y asegurar una conexión eléctrica constante y de calidad.

El canal 1 fue asignado al musculo extensor y el canal 2 fue asignado al musculo extensor de la mano ubicados en el antebrazo.

4.4. Generación de señales mioeléctricas

Se realizaron muestras en tres tipos de estados de las manos, estos se definieron y se denominaron de la siguiente manera:

- **Mano extendida:**

El sujeto debe extender todos los dedos de la mano sin flexión, activando principalmente músculos extensores como el extensor común de los dedos y el extensor largo del pulgar. Las señales mioeléctricas captadas durante esta posición reflejan la actividad muscular eléctrica relacionada con la extensión de los dedos y la muñeca, y mantener una tensión muscular constante es crucial para evitar variaciones.

- **Mano relajada:**

El sujeto relaja la mano sin tensión muscular, con los dedos en una postura neutra. Esto sirve como referencia de base, ya que las señales mioeléctricas registradas en este estado son mínimas o nulas. El análisis de estas señales permite comparar la activación muscular en otros estados y excluir interferencias.

- **Mano cerrada:**

El sujeto cierra la mano en un puño, lo que provoca contracciones importantes en los músculos flexores del antebrazo. Esta intensa activación produce señales mioeléctricas de mayor amplitud. La presión constante es crucial para asegurar

la consistencia de las señales obtenidas, requiriendo que el sujeto mantenga una presión constante durante el tiempo establecido.

Se entrenó al sujeto en la postura de la mano para tres estados, con pruebas preliminares para garantizar la comprensión y la estabilidad. Mantuvieron la postura durante 3 segundos para cada estado, lo que permitió a los sensores capturar datos para su análisis. Se repitieron los ciclos de captura en diferentes días y franjas horarias para mantener la coherencia y la fiabilidad, minimizando factores externos como la fatiga o las variaciones de temperatura muscular.

La ganancia de los sensores de electromiografía se ajustó mediante potenciómetros para garantizar una calidad óptima en la adquisición de señales mioeléctricas. La ganancia determina la amplificación de las señales eléctricas, garantizando una amplificación suficiente sin ruidos ni distorsiones. La calibración se realizó antes de cada sesión de medición para adaptarse a los cambios en la conductividad de la piel o en la posición de los electrodos.

Un estudio tomó 667 muestras de señales mioeléctricas de tres estados de la mano, registrando la actividad eléctrica muscular durante la posición del sujeto. Los datos se recogieron en diferentes momentos y días, lo que garantizó la representatividad de los resultados. Los datos se analizaron para evaluar la actividad muscular en cada estado e identificar patrones específicos en la activación de los músculos del antebrazo, asegurando que se recogía una amplia gama de datos.

4.5. Preprocesamiento de Datos

4.5.1. Obtención de datos en bruto

El software de la tarjeta de adquisición de señales del fabricante se utilizó para obtener datos brutos de las señales mioeléctricas durante las pruebas. Permite la comunicación directa y la configuración de parámetros. La tarjeta convierte las señales analógicas en digitales, que son interpretadas y procesadas por el software. A continuación, los datos se exportaron en formato CSV, compatible con diversas plataformas de análisis de datos, por lo que resulta ideal para el preprocesamiento y la manipulación de datos.

4.5.2. Procesamiento en Google Colab

Para el procesamiento de datos se utilizó Google Colab, una plataforma en la nube que permite ejecutar código Python sin instalación local. Ofrece potentes

recursos informáticos para el análisis de datos y el aprendizaje automático. Se añadió una etapa de previsualización de datos para proporcionar una vista panorámica de los datos e identificar anomalías como valores fuera de rango o incoherencias en los archivos. Se utilizó la biblioteca Python Pandas para la manipulación de estructuras de datos y el análisis preliminar (anexo 5).

4.5.3. Filtración de datos

Se realizó una fase de filtrado de datos para eliminar las celdas vacías o incompletas y garantizar así la precisión. Esto se hizo para evitar sesgos o errores en los resultados finales. Para el análisis de las señales mioeléctricas se seleccionaron los datos de potencia de la tarjeta de adquisición de señales, ya que se trata de un indicador crucial de la actividad muscular.

4.5.4. Trazabilidad

Para mantener la integridad de los datos originales, los datos filtrados y la traducción del título se almacenaron en un nuevo marco de datos, una estructura tabular utilizada en bibliotecas de análisis de datos como Pandas. Esto garantiza que ninguna operación o transformación afecte a los datos brutos originales, lo que permite la trazabilidad y la posibilidad de volver a los datos originales en caso necesario. Esto también facilita la comparación entre los datos originales y los procesados.

4.6. Implementación de Filtros Digitales

El diseño de los filtros digitales se basó en el marco teórico de las señales mioeléctricas, que suelen tener un espectro de frecuencias de 10 a 500 Hz, con la mayor parte de la información relevante en el rango de 20 a 150 Hz, para optimizar la captura de la señal y minimizar el ruido.

4.6.1. Filtros pasa bajos

Se utilizó un filtro de paso bajo con una frecuencia de corte de 450 Hz para eliminar el ruido de alta frecuencia de las señales mioeléctricas, dejando pasar sólo los componentes de baja frecuencia que contienen información muscular relevante. Las frecuencias superiores a 450 Hz suelen estar dominadas por

ruido, como interferencias de dispositivos electrónicos o señales no relacionadas con la actividad muscular.

4.6.2. Filtro pasa altos

Se utilizó un filtro de paso alto con una frecuencia de corte de 5 Hz para eliminar el ruido de baja frecuencia, como las fluctuaciones lentas o el ruido de componente continuo, de las señales capturadas. Este filtro permite que las señales mioeléctricas más rápidas y relevantes dominen el análisis, evitando artefactos como los movimientos de la piel o las interferencias de CC.

Se aplicaron los filtros digitales y se realizaron comparaciones visuales entre las señales antes y después del proceso de filtrado, mostrando las señales captadas en tres estados de la mano antes, durante y después del proceso de filtrado (Figura 16, 17 y 18).

Figura 16

Preprocesamiento de datos con mano extendida

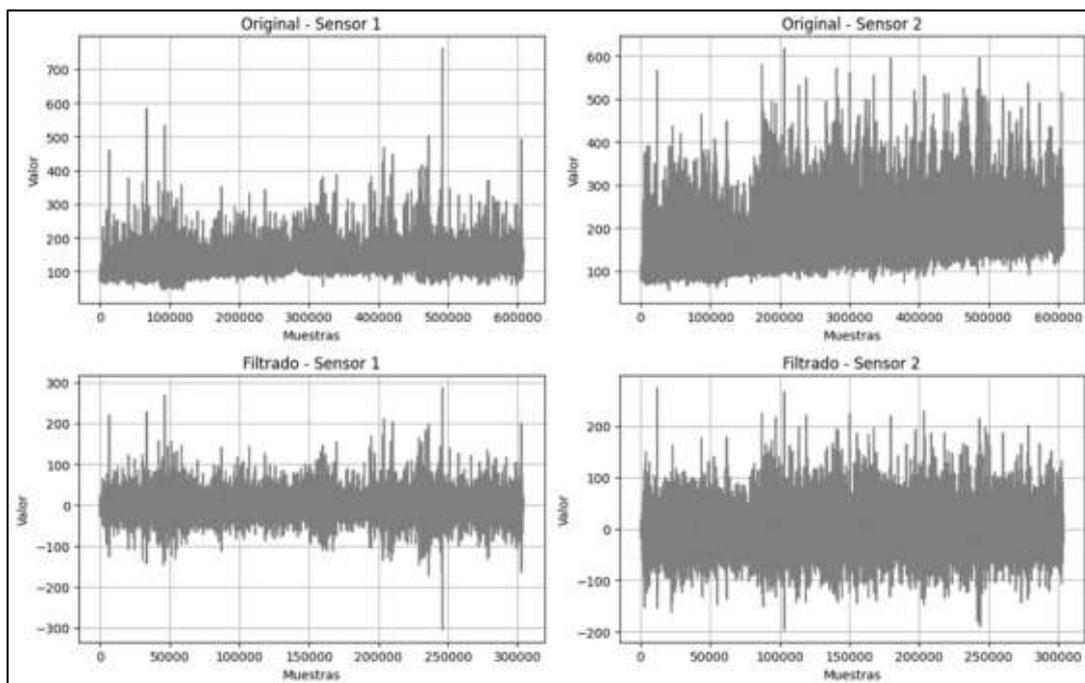
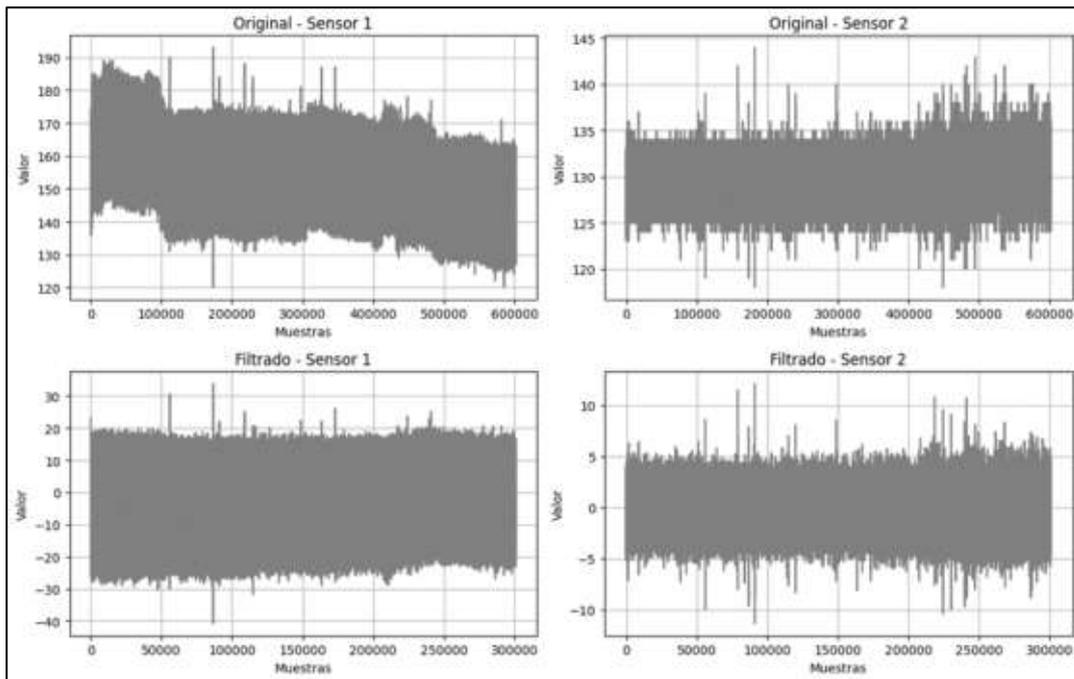
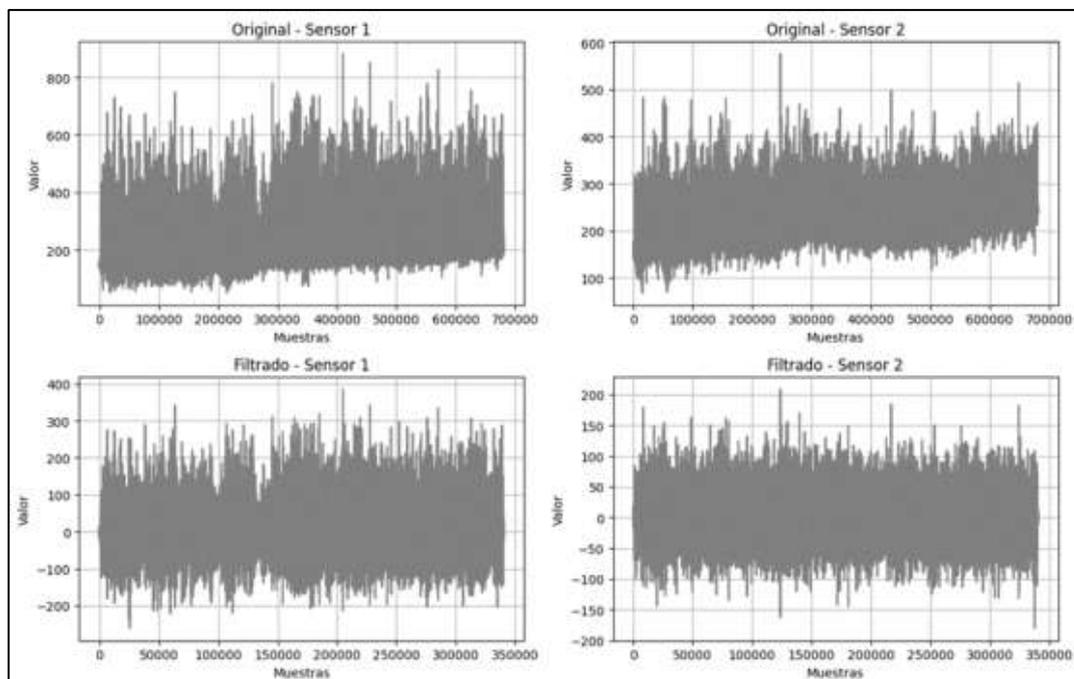


Figura 17*Preprocesamiento de datos con mano relajada***Figura 18***Preprocesamiento de datos con mano cerrada*

4.7. Análisis de correlación

Como parte del análisis de los datos obtenidos, se revisó la correlación entre las variables de entrada y salida para seleccionar los mejores datos de entrada en función

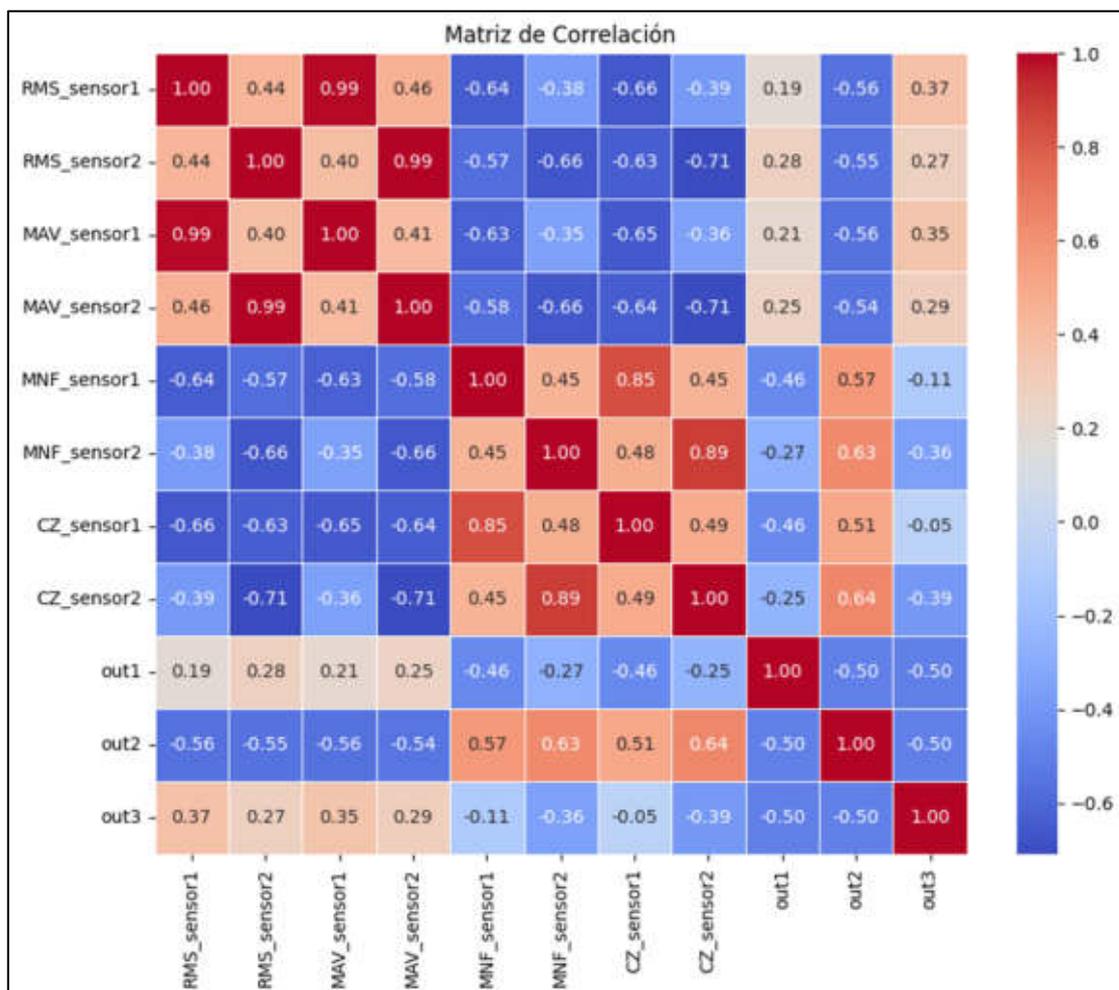
de su valor de correlación con las variables de salida. Se priorizó la selección de variables que mostraran una alta correlación con las salidas, y al mismo tiempo, se evitó una alta correlación entre las variables de entrada para prevenir el sobreajuste durante el entrenamiento del modelo.

Para visualizar las relaciones entre las variables, se generó un mapa de calor de la matriz de correlación. Este mapa permitió identificar de manera clara cuáles eran las variables de entrada más relevantes. De acuerdo con el análisis realizado, los datos de entrada que mostraron una mayor correlación con las variables de salida fueron MAV_sensor1, MAV_sensor2, MNF_sensor1 y MNF_sensor2. Estas variables se consideraron como las más significativas para el proceso de modelado.

En la figura 19 se presenta el mapa de calor de la matriz de correlación, donde se destacan las relaciones mencionadas entre las variables de entrada y salida.

Figura 19

Matriz de Correlación



Con la matriz realizada, se presenta la gráfica de cada columna del dataframe de caracterización de las señales de los sensores obtenidos en las figuras del 20 al 30.

Figura 20

Gráfica de RMS del sensor 1

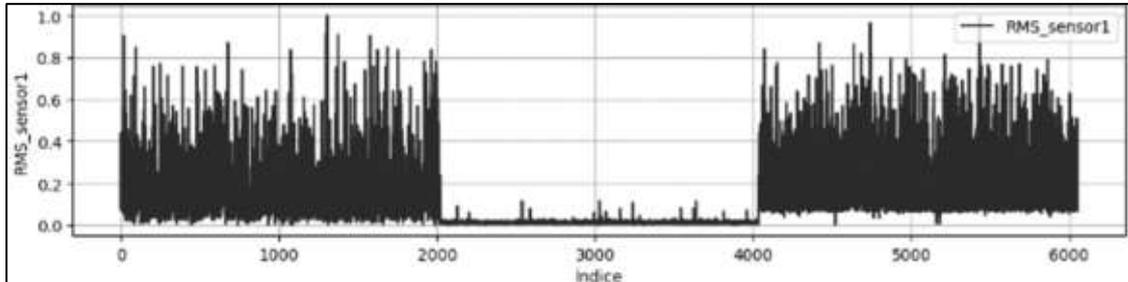


Figura 21

Gráfica de RMS del sensor 2

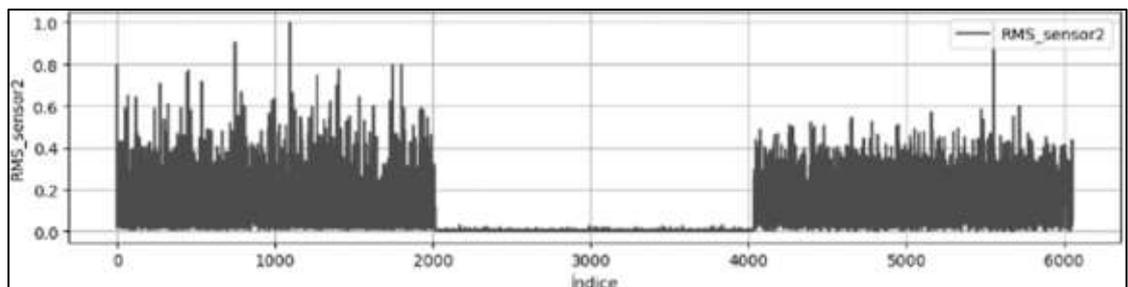


Figura 22

Gráfica de MAV del sensor 1

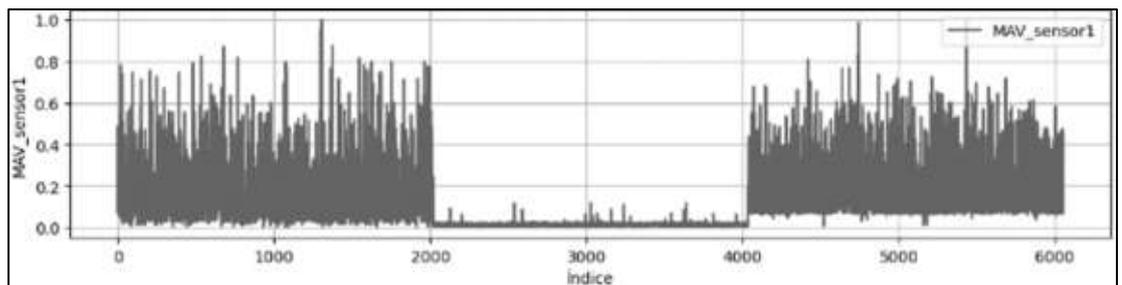


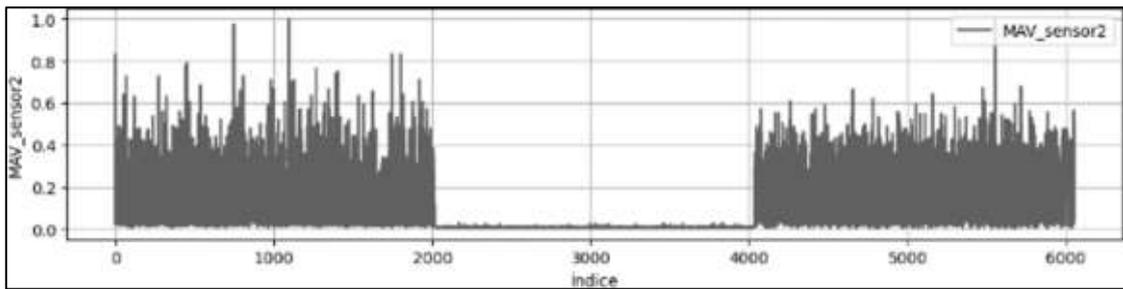
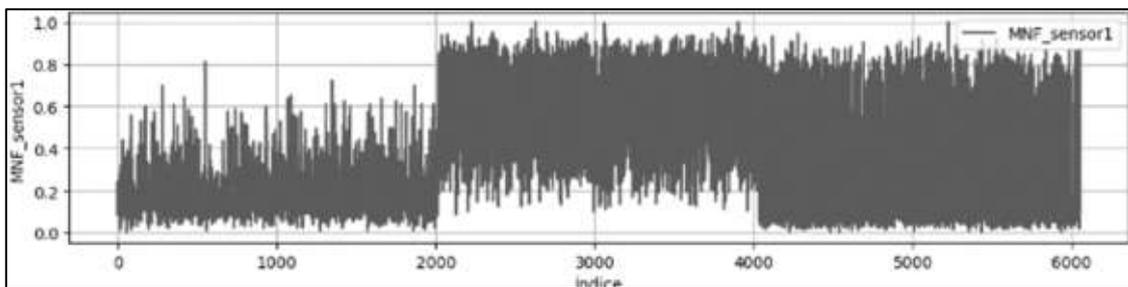
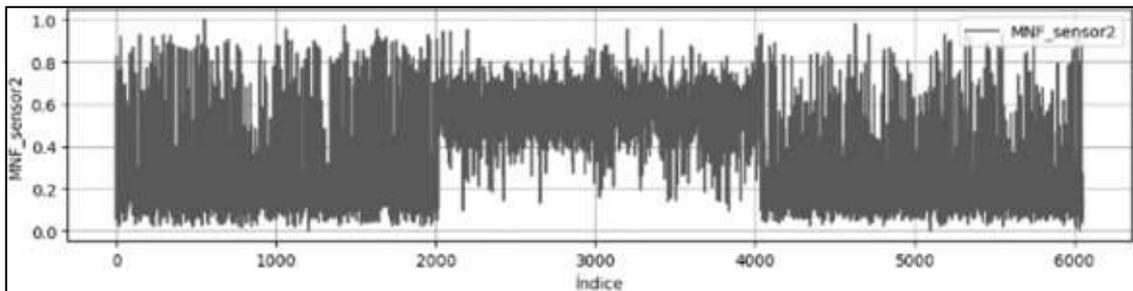
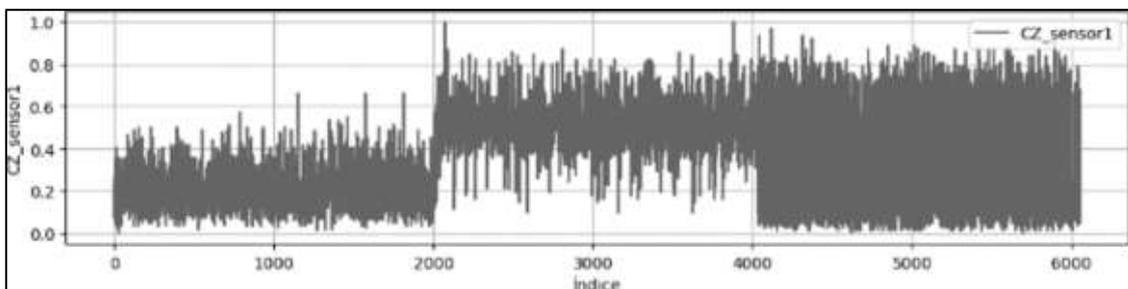
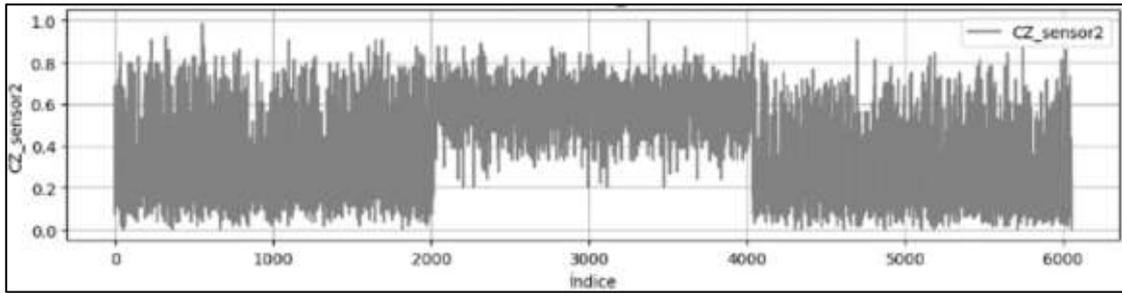
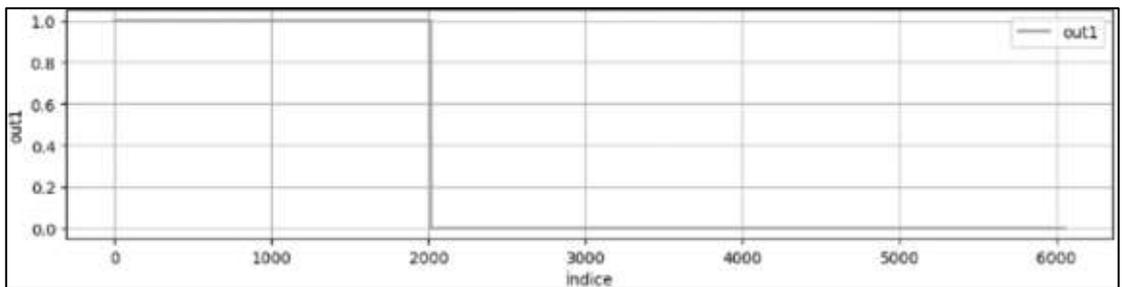
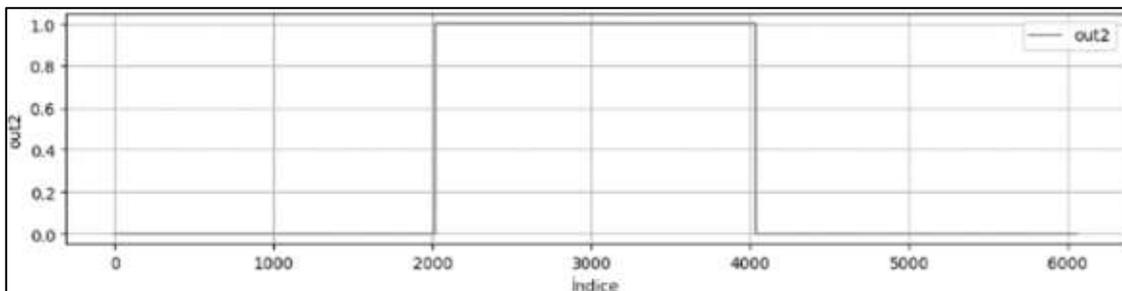
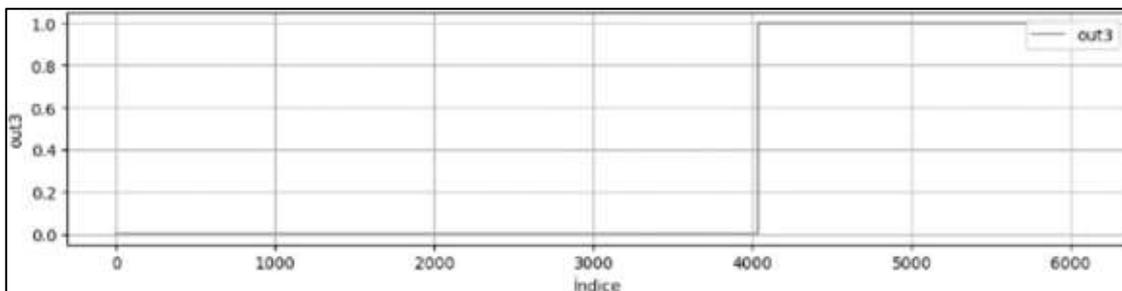
Figura 23*Gráfica de MAV del sensor 2***Figura 24***Gráfica de MNF del sensor 1***Figura 25***Gráfica de MNF del sensor 2***Figura 26***Gráfica de ZC del sensor 1*

Figura 27*Gráfica de ZC de sensor 2***Figura 28***Gráfica de out1***Figura 29***Gráfica de out2***Figura 30***Gráfica de out3*

4.8. Implementación de red neuronal secuencial

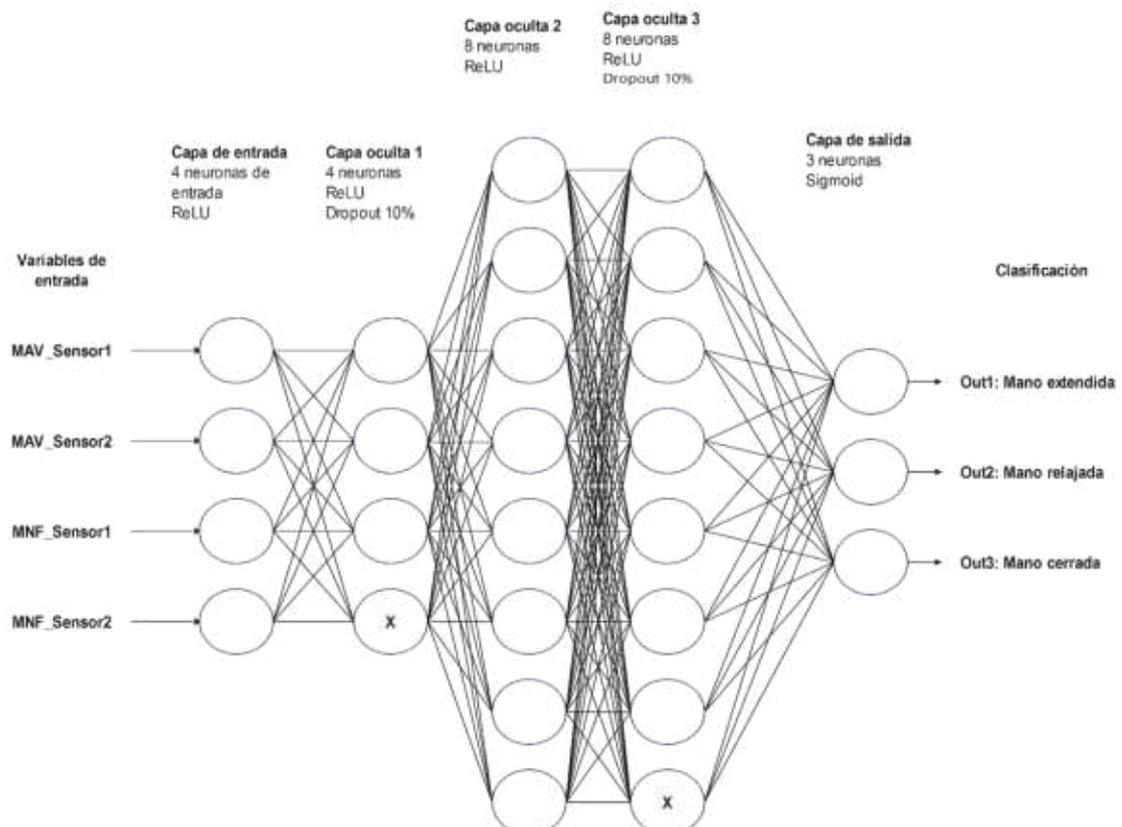
Para la clasificación de los estados de la mano, se implementó un modelo secuencial. Este modelo comienza con una capa de entrada que recibe las 4 características seleccionadas previamente: MAV_sensor1, MAV_sensor2, MNF_sensor1, y MNF_sensor2. Estas características fueron elegidas por su alta correlación con las variables de salida, lo que asegura que el modelo reciba la información más relevante desde el inicio.

Después de ello se incorporaron dos capas ocultas, cada una con 8 neuronas. Estas capas permiten al modelo aprender patrones complejos a partir de las características de entrada. Para prevenir el sobreajuste durante el entrenamiento, se añadió una capa de dropout del 10 % después de cada capa oculta, lo que desactiva aleatoriamente algunas neuronas y mejora la capacidad de generalización del modelo.

Finalmente, se incluyó una capa de salida con 3 neuronas, cada una correspondiente a uno de los tres estados de la mano: extendida, relajada o cerrada. Se utilizó una función de activación sigmoide en esta capa para realizar la clasificación, asignando una probabilidad a cada estado. La clase con la mayor probabilidad se toma como la predicción final del modelo (Figura 31).

Figura 31

Gráfica de la arquitectura de la red neuronal secuencial



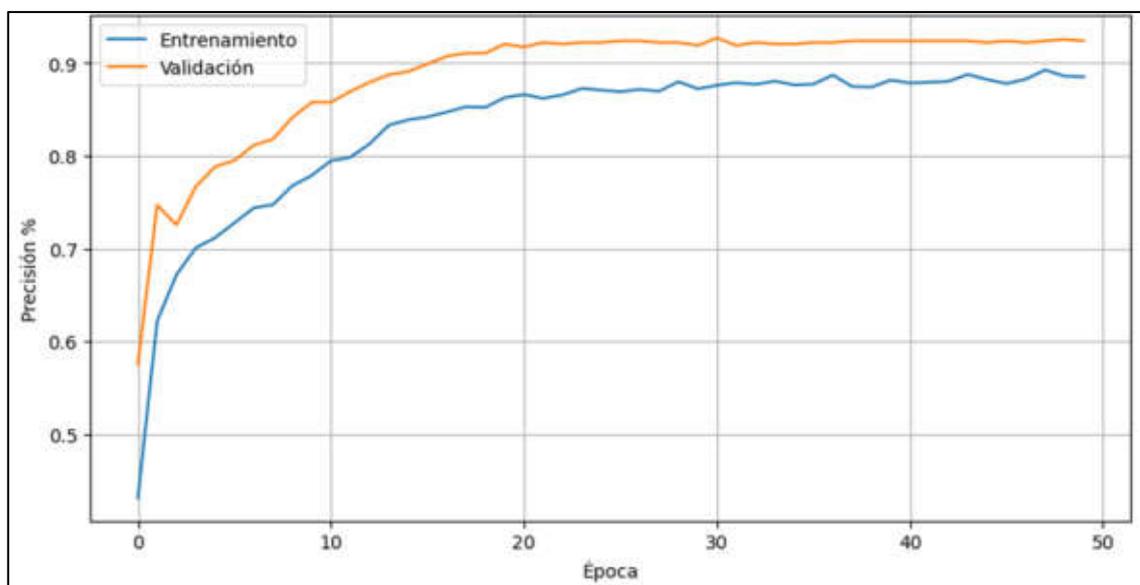
El entrenamiento de la red se realizó utilizando el 80 % de los datos disponibles en el dataframe, lo que permitió que el modelo aprendiera de una gran cantidad de muestras representativas. Este conjunto de datos se utilizó para ajustar los pesos y sesgos de la red durante el proceso de entrenamiento. El 20 % restante se destinó a la validación, lo que significa que estos datos no se utilizaron para entrenar la red, sino para evaluar su desempeño en cada época. Al validar el modelo durante el entrenamiento, se pudo monitorear su capacidad para generalizar a nuevos datos, evitando así el sobreajuste.

El entrenamiento del modelo se llevó a cabo durante 50 épocas, un número de iteraciones suficiente para permitir que la red aprendiera patrones significativos en los datos. Durante cada época, el modelo ajustó sus parámetros en función de la retroalimentación proporcionada por el error de predicción. La validación periódica en cada época permitió identificar si el modelo estaba mejorando su rendimiento o si empezaba a sobreajustarse, lo que ayudó a tomar decisiones sobre posibles ajustes en los hiperparámetros o la arquitectura de la red.

En la figura 32 se observa la curva de entrenamiento en azul la cual muestra un incremento gradual, esto indica que el modelo aprende y mejora su precisión a medida que se entrena hasta alcanza su estabilidad al 90 %. La curva naranja muestra la precisión del modelo al clasificar los datos del conjunto de validación, en la gráfica se puede observar que esta curva es ligeramente superior a la curva de entrenamiento por lo es un indicador de que el modelo ha logrado generalización de los datos.

Figura 32

Gráfico de Curva de Aprendizaje de Red Neuronal Secuencial



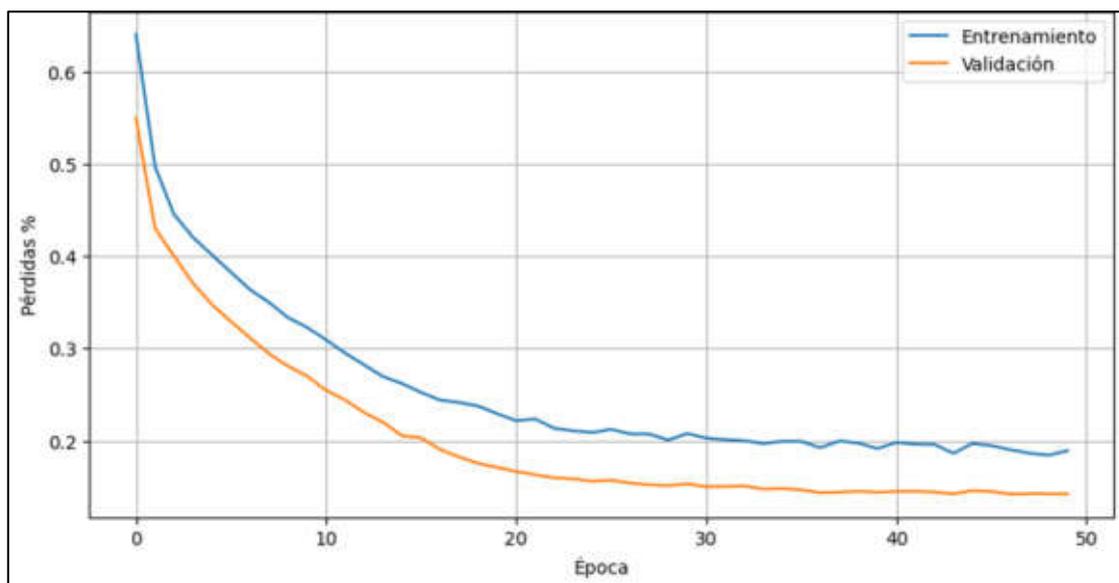
En la figura 33 se observa la curva azul la cual muestra las pérdidas del modelo durante el entrenamiento, que disminuyen a medida que avanza el proceso de entrenamiento. Esto indica una mejora en el ajuste de los datos de entrenamiento y la reducción de la diferencia entre las predicciones y los valores reales. La curva se estabiliza en torno a la época 30, lo que indica que el modelo ha alcanzado la máxima minimización de errores.

La curva de validación sigue un patrón descendente, mostrando una mejora continua en la generalización a datos no vistos. Ambas curvas descienden y se estabilizan, indicando un equilibrio óptimo.

No se observan signos de sobreajuste ni de subajustes, lo que indica que el modelo aprendió patrones importantes sin depender excesivamente de ejemplos concretos. Con ello, se puede observar la matriz de confusión a continuación.

Figura 33

Graficas de Perdidas durante el Entrenamiento de Red Neuronal Secuencial

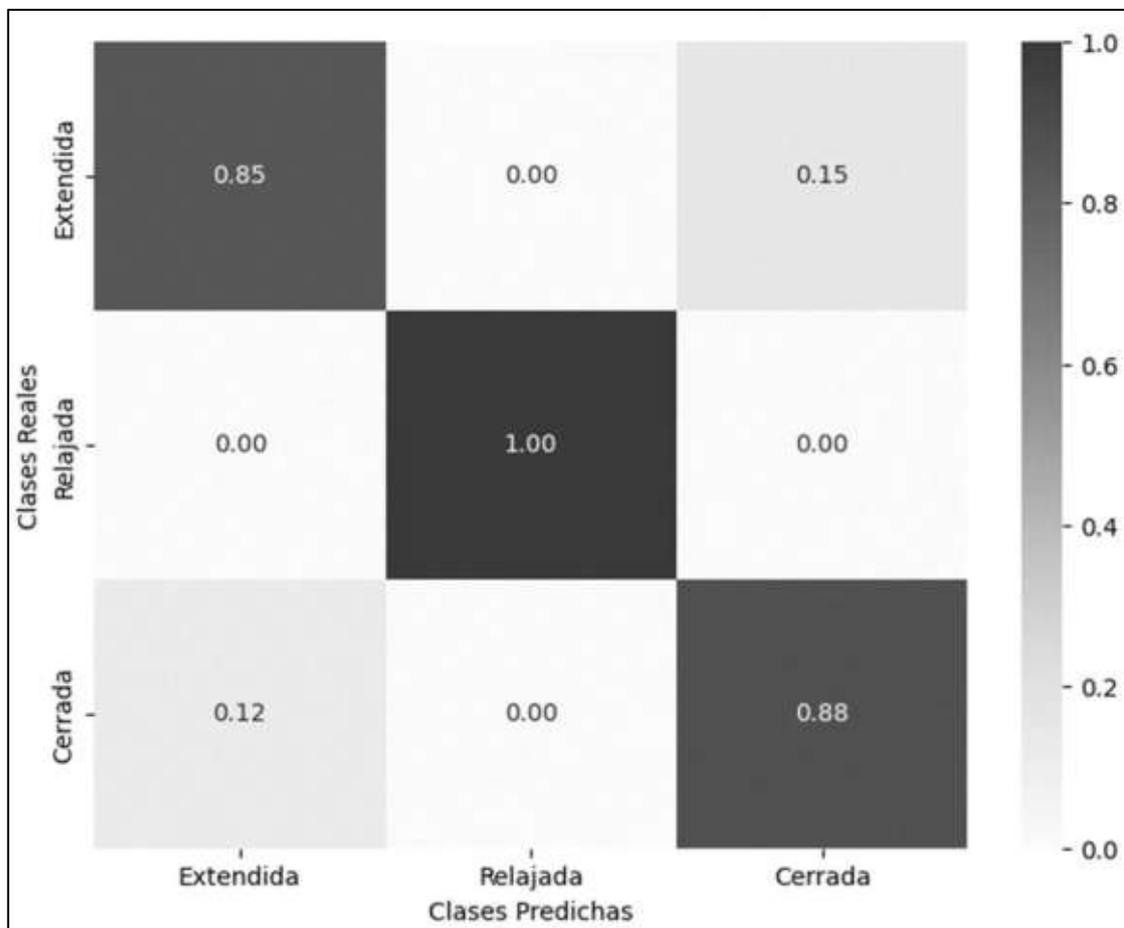


De la matriz de confusión (figura 34) se puede extraer que el modelo clasificó correctamente el 85 % de los estados de mano extendida, pero el 15 % se clasificó erróneamente como mano cerrada. El modelo también mostró un rendimiento excepcional al predecir el 100 % de los estados de mano relajada y el 88 % de los estados de mano cerrada. Sin embargo, el 12 % de los casos se clasificaron erróneamente como mano extendida.

El modelo tuvo una precisión del 90,92 % durante el entrenamiento y la validación, lo que indica su capacidad para distinguir entre diferentes estados de la mano. Las pérdidas calculadas del modelo fueron del 15,52 %.

Figura 34

Matriz de Confusión Normalizada de Red Neuronal Secuencial



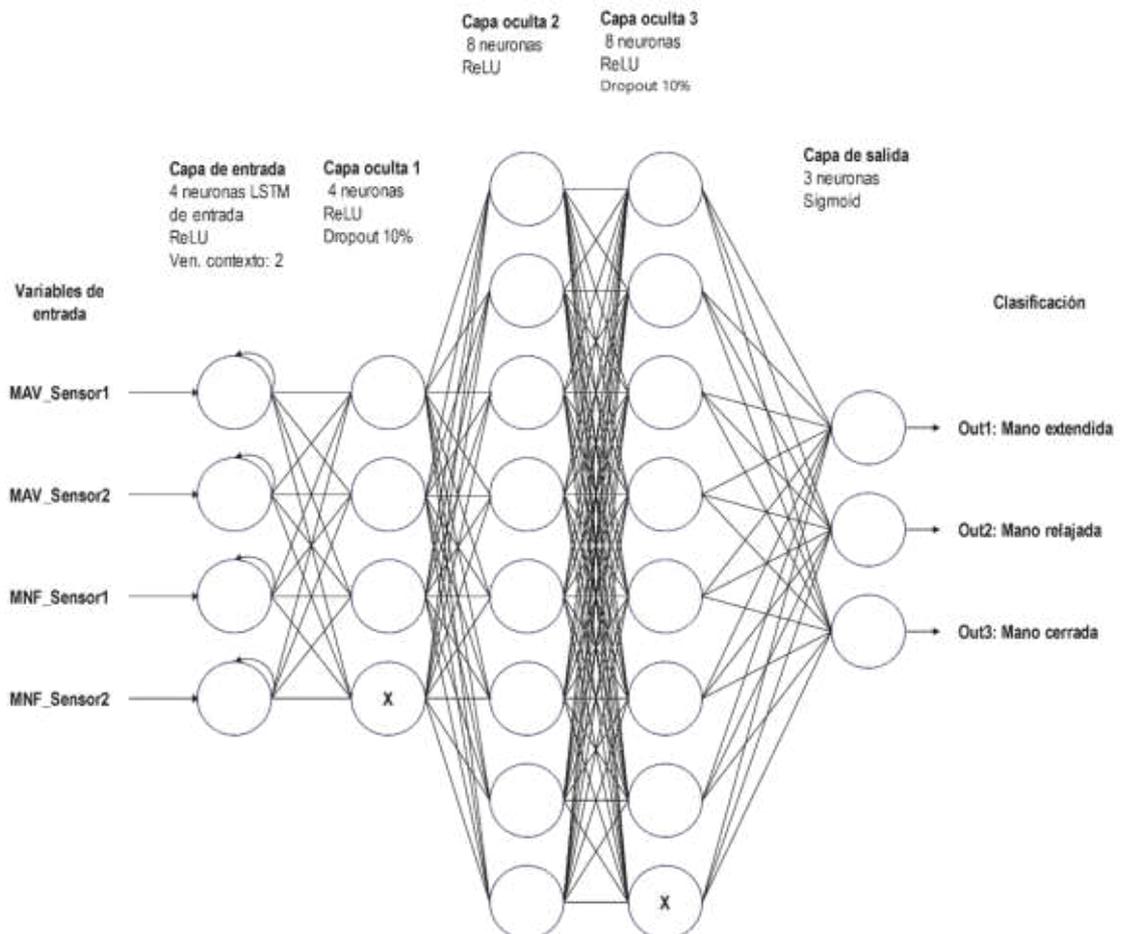
4.9. Implementación de red neuronal recurrente

Se desarrolló una red neuronal recurrente para capturar secuencias temporales en las características de entrada, mejorando su capacidad para clasificar los estados de la mano. La red comienza con una capa de entrada LSTM de 4 neuronas, que procesa las secuencias temporales y maneja las dependencias a lo largo del tiempo. Se añadieron

dos capas ocultas con 8 neuronas para profundizar en el análisis de las características extraídas por la capa LSTM. Se aplicó un dropout del 10 % a cada capa oculta para mantener la robustez y evitar el sobreajuste. La capa de salida, formada por 3 neuronas para cada estado de la mano, genera las predicciones finales y asigna probabilidades a cada estado posible (Figura 35).

Figura 35

Gráfica de la arquitectura de la red neuronal recurrente



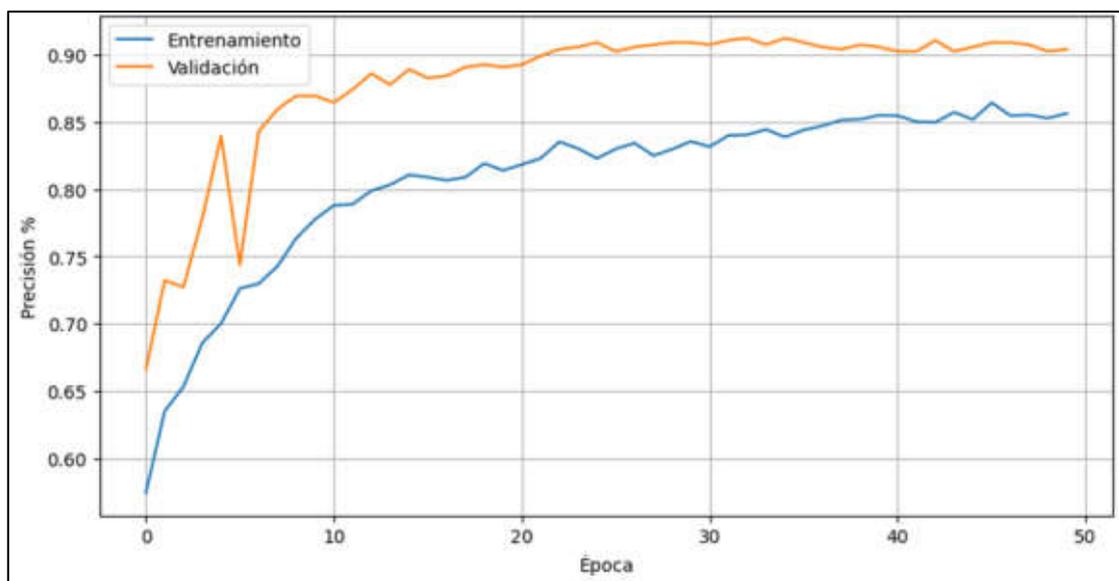
La red se entrenó utilizando el 80 % de los datos del marco de datos seleccionados aleatoriamente, ajustando los pesos y los sesgos para minimizar los errores de predicción. El 20 % restante se utilizó como conjunto de validación. La red se entrenó durante 50 épocas, cada una de las cuales representaba un ciclo de aprendizaje completo. El rendimiento se controló tanto en el conjunto de entrenamiento como en el de validación para identificar sobreajustes y ajustar el proceso de entrenamiento.

Se utilizaron técnicas como el dropout para reducir el sobreajuste y mejorar la capacidad de generalización del modelo. La supervisión constante de las métricas de rendimiento garantizó que la red aprendiera patrones esenciales sin sobreajuste.

En la figura 36 se muestra de cómo las curvas de rendimiento del modelo durante el entrenamiento y la validación mostraron una mejora constante en el aprendizaje a partir de los datos. Las curvas de precisión y validación del modelo aumentaron, lo que indica que estaba captando patrones relevantes y mejorando la clasificación. En la época 40, las curvas se estabilizaron, lo que indica un equilibrio entre aprendizaje y generalización.

Figura 36

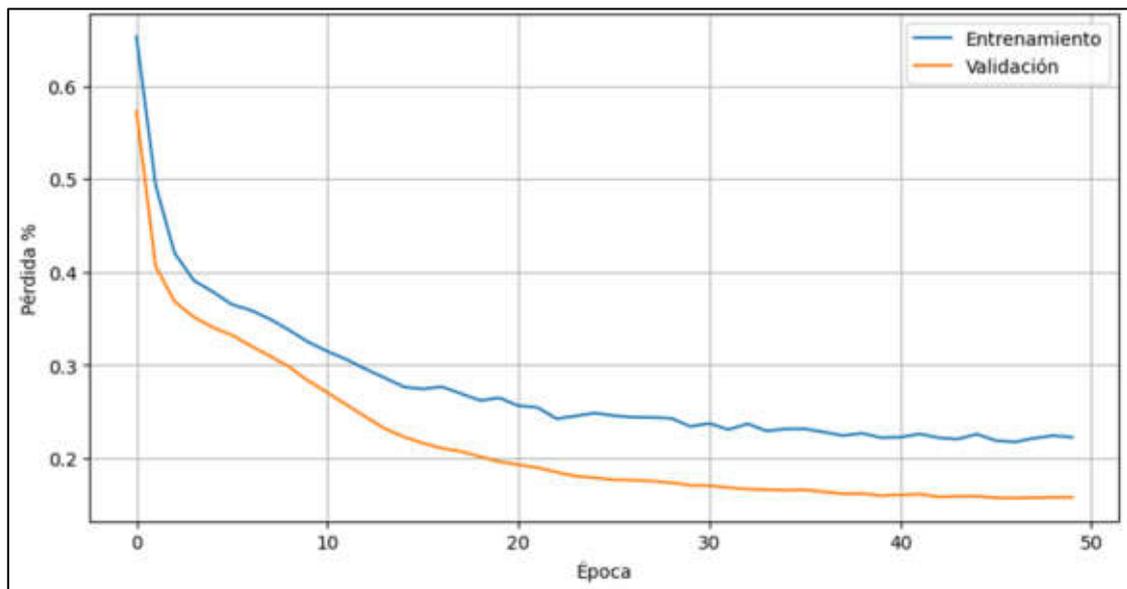
Gráfico de Curva de Aprendizaje de Red Neuronal Recurrente



Las curvas de pérdidas del modelo durante el entrenamiento y la validación (Figura 37) mostraron una disminución gradual en cada época, lo que indica su capacidad para ajustarse a los datos con eficacia. Esto indica que el modelo está aprendiendo, reduce la diferencia entre las predicciones y los valores reales.

Figura 37

Graficas de Perdidas durante el Entrenamiento de Red Neuronal Recurrente



La disminución continua de las curvas de pérdidas en los conjuntos de entrenamiento y validación indica que el modelo capta patrones subyacentes en los datos sin memorizar ejemplos específicos. Esto sugiere que el modelo puede generalizar bien, funcionando bien tanto con los datos entrenados como con los nuevos.

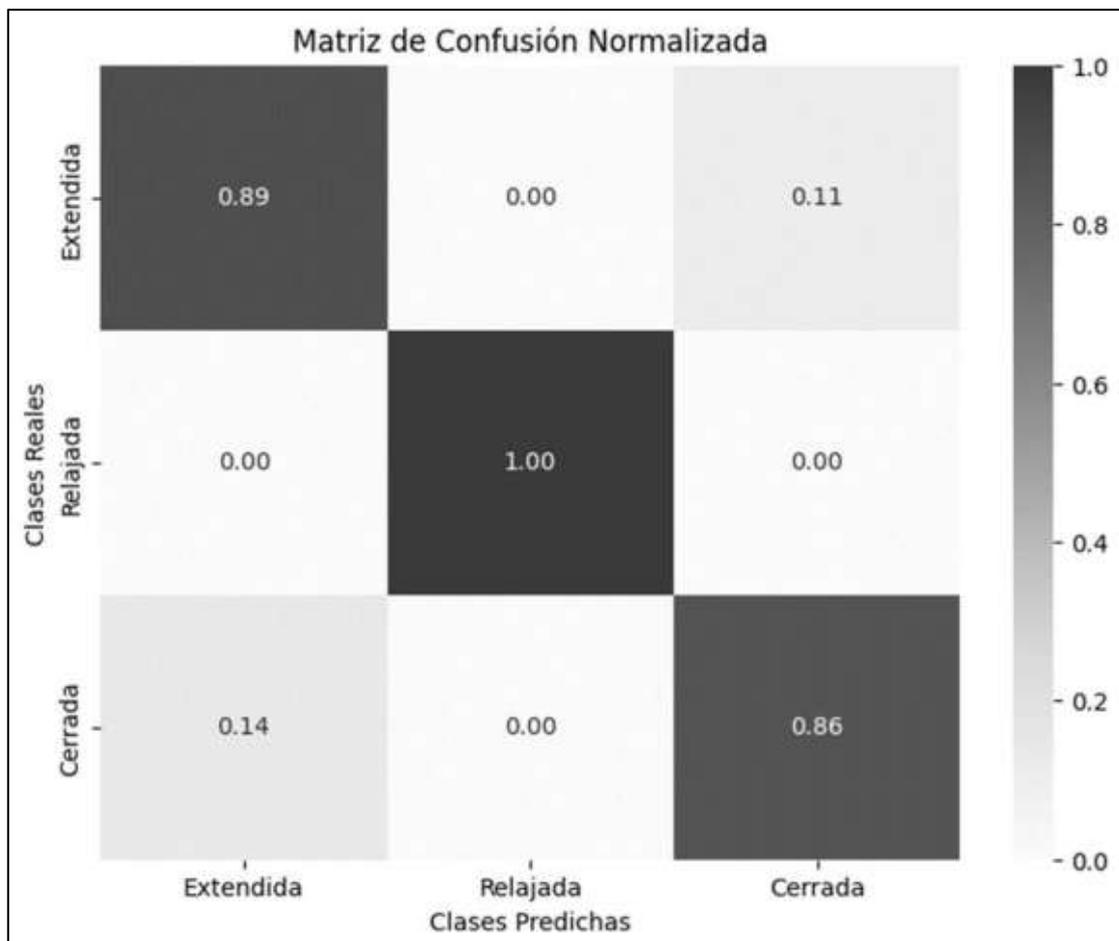
La estabilidad de las curvas a medida que se acercan a las últimas épocas refuerza esta idea, ya que, si el modelo hubiera empezado a sobreajustarse, se habría observado una divergencia entre el conjunto de entrenamiento y las curvas de validación. Sin embargo, ambas curvas permanecen próximas y disminuyen de forma similar, lo que indica un buen equilibrio entre ajuste y generalización. La convergencia del modelo hacia una disminución estable en épocas posteriores indica su punto de entrenamiento óptimo.

De la matriz de confusión (Figura 38) se puede extraer que el modelo clasificó con precisión el 89 % de los casos de manos extendidas, con un 11 % de casos clasificados incorrectamente como manos cerradas. El modelo mostró un rendimiento excepcional en la clasificación de estados de manos relajadas, alcanzando una precisión del 100 %. En los casos de manos cerradas, el modelo acertó en el 86 % de los casos, con un 14 % de clasificaciones incorrectas como manos extendidas.

En conjunto, el modelo alcanzó una precisión del 91,74 % durante el entrenamiento y la validación, lo que indica una gran eficacia en la clasificación de los estados de la mano. Sin embargo, el modelo tuvo pérdidas calculadas del 14,26 %, lo que indica un margen de error que podría optimizarse con ajustes adicionales.

Figura 38

Matriz de Confusión Normalizada de Red Neuronal Recurrente



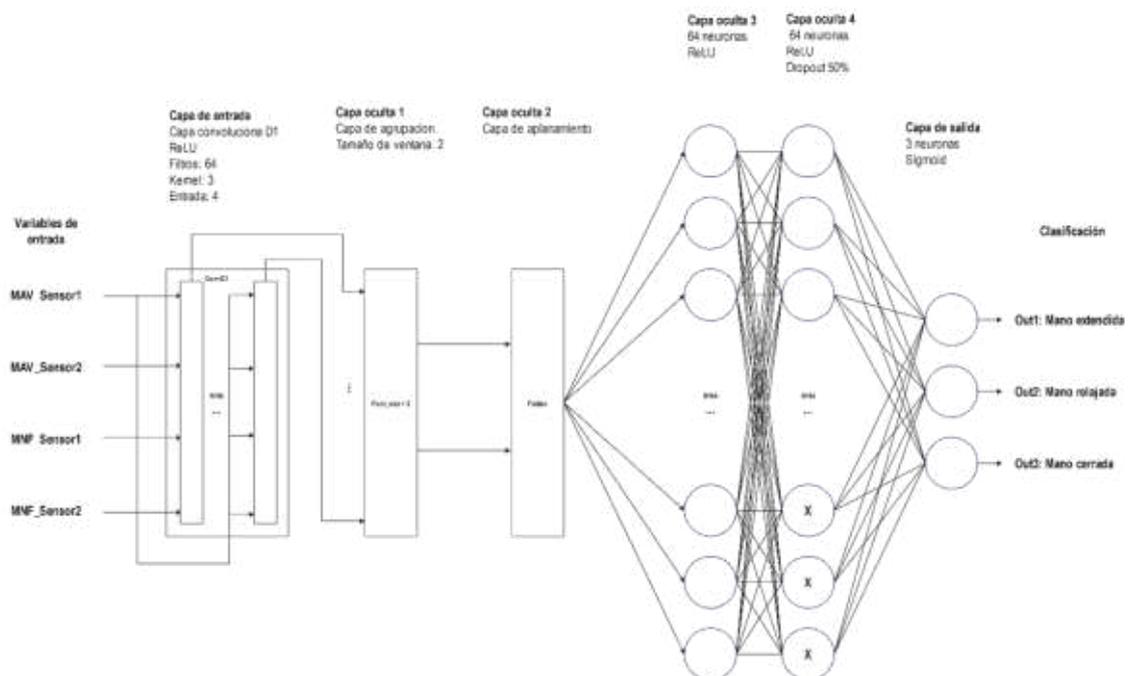
4.10. Implementación de red neuronal convolucional

Se creó una red neuronal convolucional para la clasificación del estado de la mano, que consta de una capa de entrada, tres capas ocultas y una capa de salida. La capa de entrada analiza las señales mioeléctricas en tres pasos temporales, mientras que las capas ocultas reducen la dimensionalidad, aplanan los datos y procesan características abstractas. La capa de salida predice los estados de mano extendida, relajada y cerrada, con un 50 % de dropout para mejorar la generalización.

A continuación, se presenta una gráfica (Figura 39) que ilustra la arquitectura completa de la red neuronal convolucional y cómo se organizan las capas para procesar las señales y generar las predicciones.

Figura 39

Gráfica de la arquitectura de la red neuronal convolucional



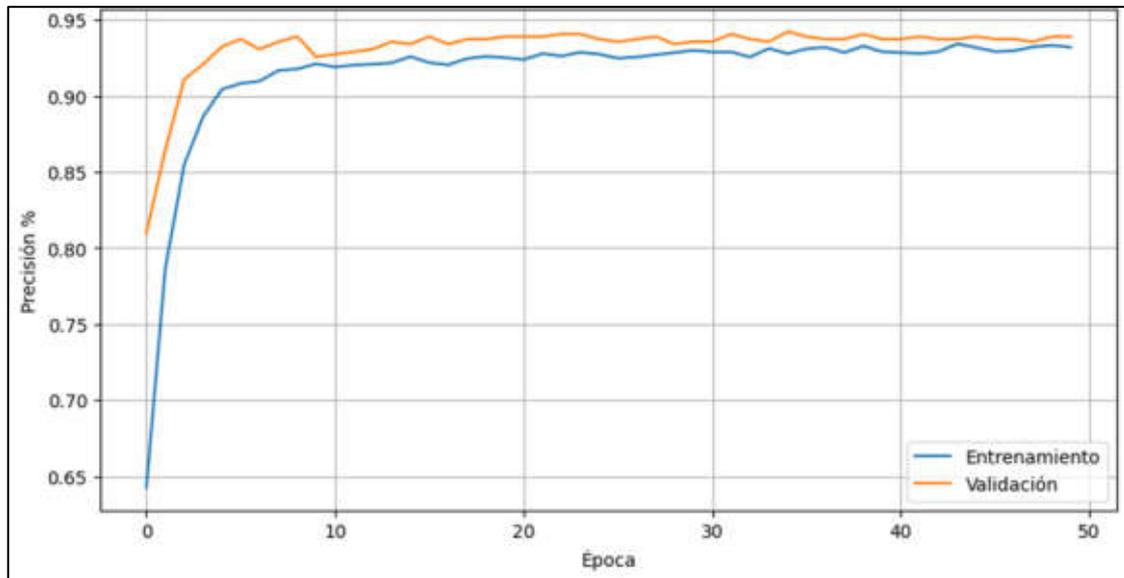
La red se entrenó utilizando el 80 % de los datos disponibles, lo que le permitió ajustar pesos y sesgos para minimizar el error de predicción. El 20 % restante se utilizó para validar el modelo y evaluar el rendimiento después de cada época. El entrenamiento se llevó a cabo durante 50 épocas, y el modelo ajustó los parámetros en función de los errores observados. Las métricas de rendimiento se registraron en cada epoch, lo que permitió un buen equilibrio entre aprendizaje y generalización.

El modelo alcanzó su máximo rendimiento al final del proceso, y se emplearon técnicas como el dropout para evitar el sobreajuste.

A continuación, en la figura 40 se muestra la curva de aprendizaje y pérdidas de entrenamiento y validación.

Figura 40

Gráfico de Curva de Aprendizaje de Red Neuronal Convolutiva



La curva de entrenamiento en azul muestra un aumento gradual de la precisión a lo largo de las épocas, lo que indica el aprendizaje y la mejora del modelo. La curva naranja, que representa la precisión del modelo en la clasificación de los datos del conjunto de validación, es ligeramente superior a la curva de entrenamiento en la mayoría de las épocas, lo que indica su capacidad para generalizar bien los nuevos datos.

La estabilidad de la curva de validación a lo largo de las épocas lo confirma, sin fluctuaciones ni caídas significativas en el rendimiento, lo que sugiere un buen equilibrio entre aprendizaje y generalización.

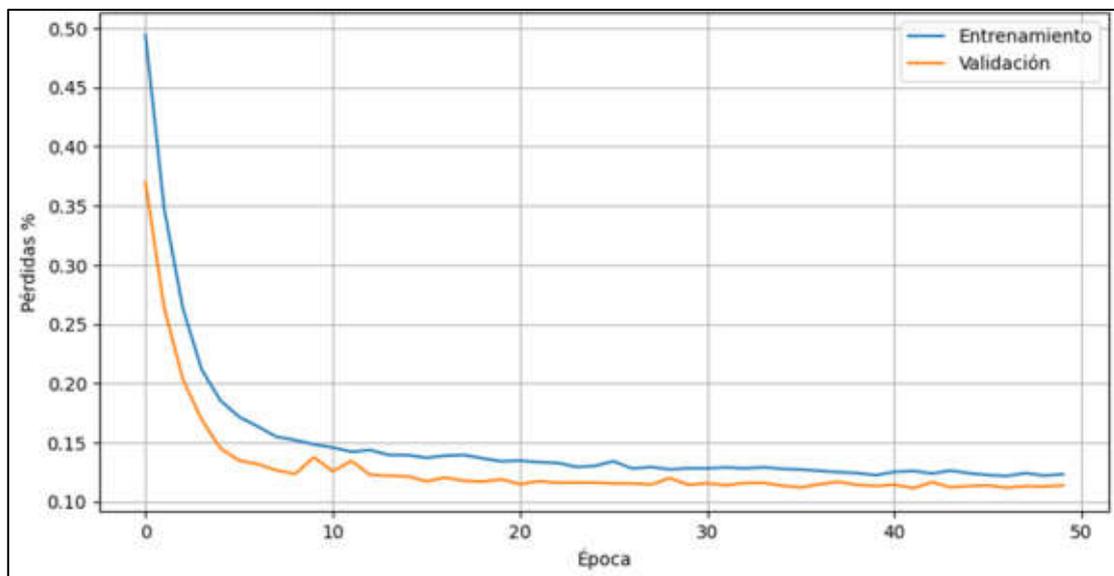
La curva naranja, que representa la precisión del modelo en la clasificación de los datos del conjunto de validación, es ligeramente superior a la curva de entrenamiento en la mayoría de las épocas, lo que indica su capacidad para generalizar bien los nuevos datos. La estabilidad de la curva de validación a lo largo de las épocas lo confirma, sin fluctuaciones ni caídas significativas del rendimiento, lo que sugiere un buen equilibrio entre aprendizaje y generalización.

En la figura 41 se observa la curva azul la cual muestra las pérdidas del modelo durante el entrenamiento y la validación, con una disminución gradual de las pérdidas desde las primeras épocas. Alrededor de la época 30, la curva se estabiliza, lo que indica mejoras mínimas en la reducción de pérdidas. La curva de validación también sigue un patrón descendente, lo que indica la capacidad del modelo para generalizar bien los nuevos datos.

La estabilización simultánea de ambas curvas en la época 30 indica que el modelo ha alcanzado su máximo rendimiento sin signos de sobreajuste ni de subajuste.

Figura 41

Graficas de Perdidas durante el Entrenamiento de Red Neuronal Convolutional

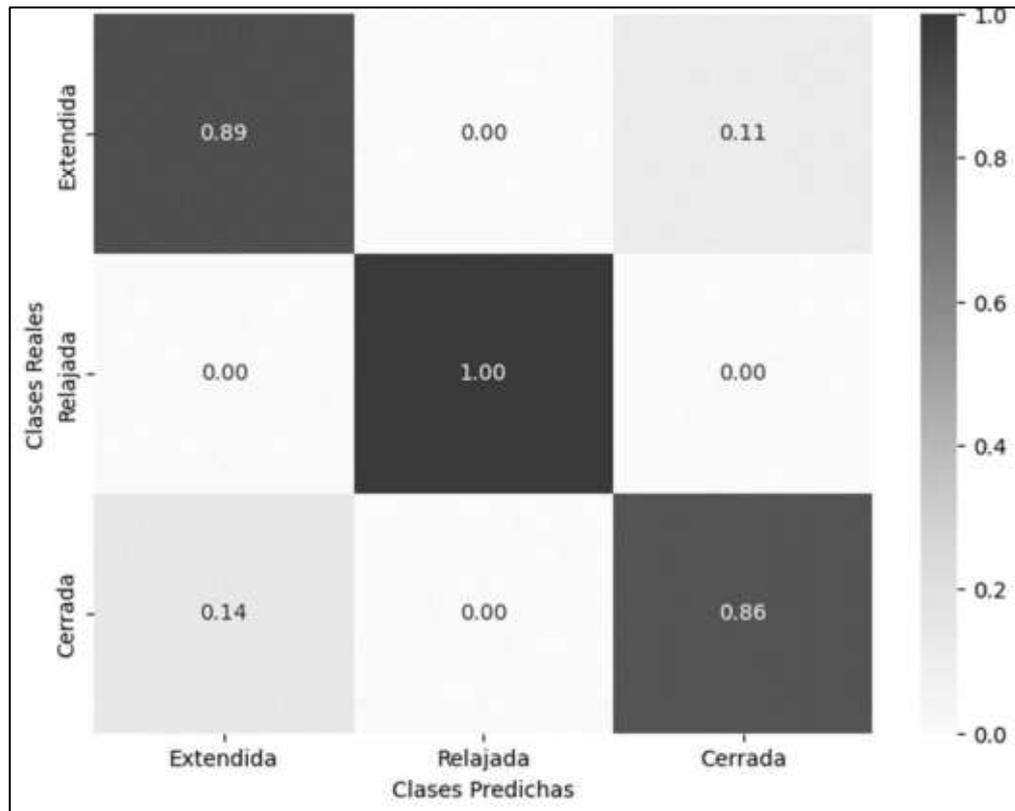


En la figura 42 se muestra que el modelo logró una clasificación correcta en el 89 % de los casos de manos extendidas, con un 11 % clasificado incorrectamente como manos cerradas. El modelo mostró una precisión del 100 % en el estado de mano relajada, lo que indica un buen aprendizaje. Sin embargo, clasificó los casos de manos cerradas con un 14 % mal clasificado como manos extendidas.

La precisión del modelo durante el entrenamiento y la validación fue del 91,91 %, con un margen de error del 12,53 %. El rendimiento del modelo en la práctica es bueno, pero la confusión entre las señales de manos extendidas y cerradas sugiere mejoras adicionales, como la arquitectura del modelo o los datos de entrenamiento.

Figura 42

Matriz de Confusión Normalizada de Red Neuronal Convolutiva



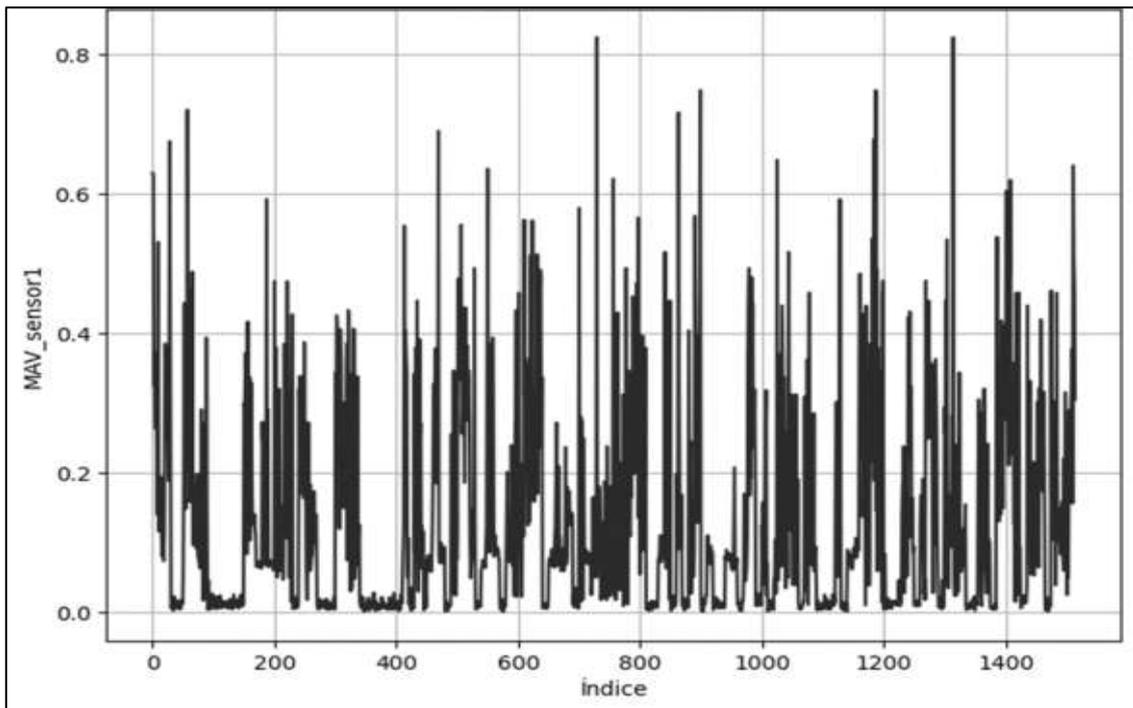
4.11. Validación cruzada

Para evaluar la capacidad de generalización de las redes neuronales entrenadas se utilizó un método de validación cruzada para evaluar la capacidad de generalización de las redes neuronales entrenadas (anexo 6). Se trata de utilizar un conjunto de datos distinto para evaluar el rendimiento del modelo con datos nuevos y de este modo se obtenía una estimación más precisa del rendimiento del modelo y se garantizaba su eficacia en situaciones reales.

El proceso de validación cruzada reforzó la confianza en la capacidad de generalización de los modelos. Los resultados confirmaron la solidez de los modelos y su capacidad para manejar variaciones en los datos de entrada. A continuación, se presentan las gráficas correspondientes a cada columna del dataframe utilizado en esta etapa del desarrollo. Estas gráficas permiten visualizar cómo se comportaron las diferentes características en el segundo conjunto de datos y proporcionan una representación visual del rendimiento del modelo en cada una de estas particiones tal y como se puede observar en las figuras del 43 al 49.

Figura 43

Gráfica de MAV del sensor 1 para la validación cruzada

**Figura 44**

Gráfica de MAV del sensor 2 para la validación cruzada

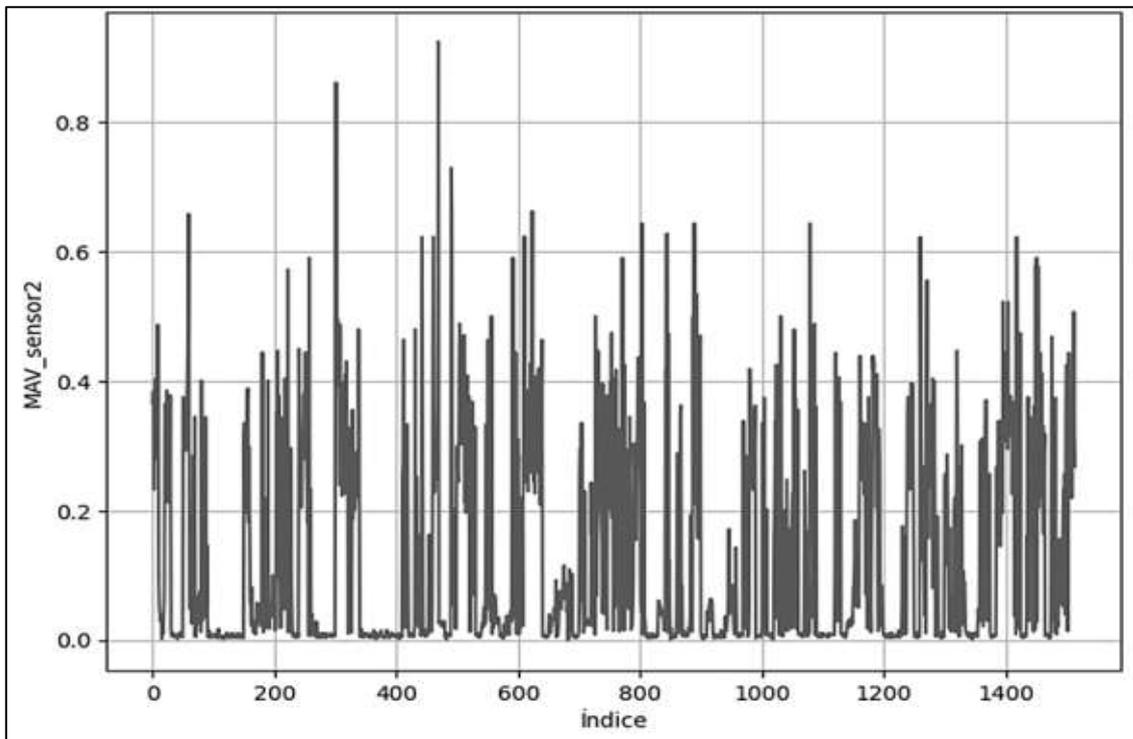
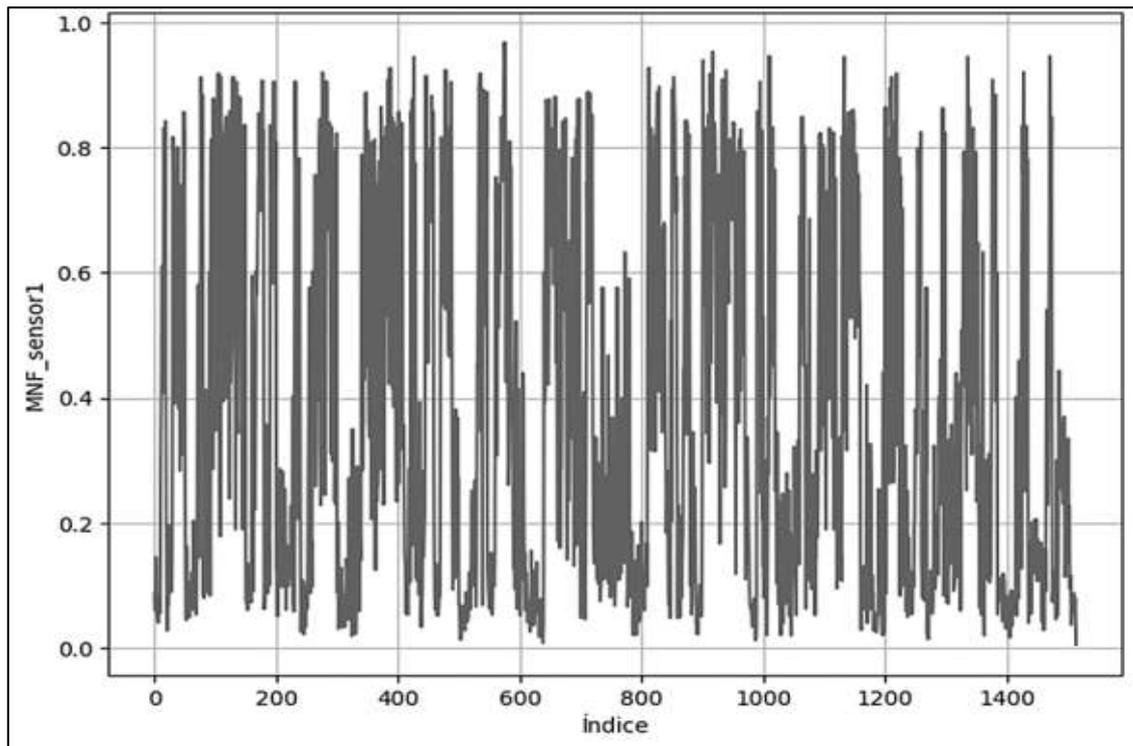


Figura 45

Gráfica de MNF del sensor 1 para la validación cruzada

**Figura 46**

Gráfica de MNF del sensor 2 para la validación cruzada

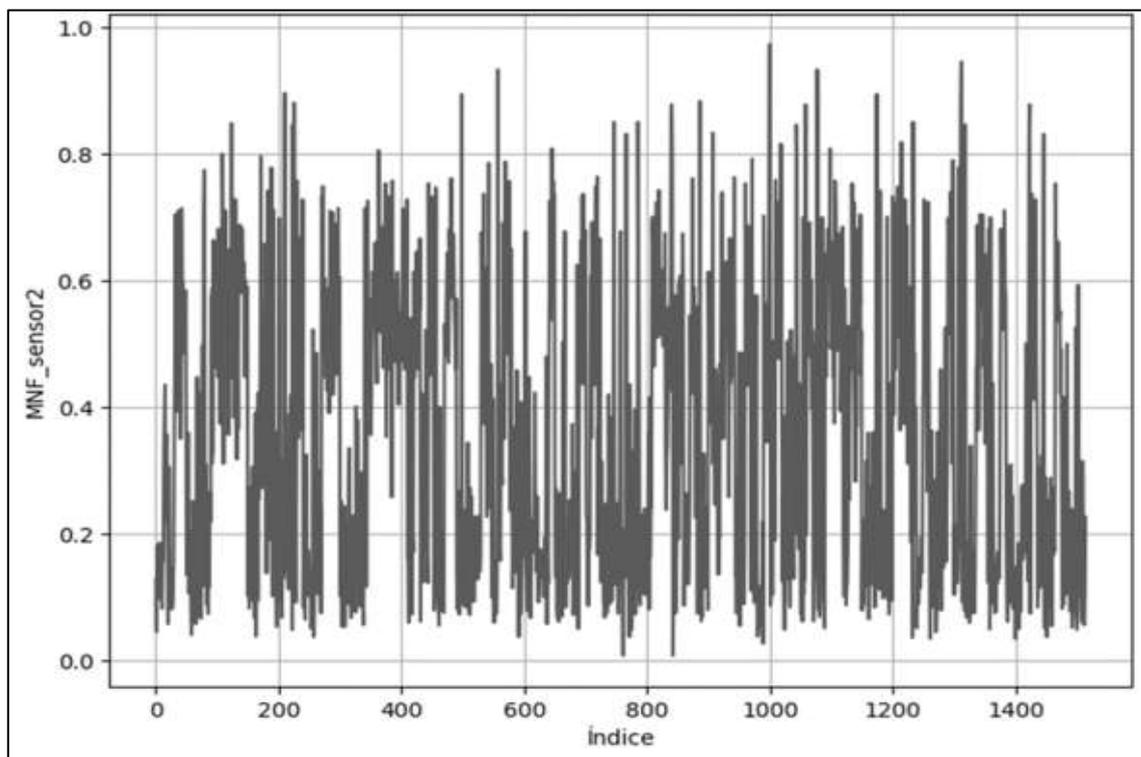
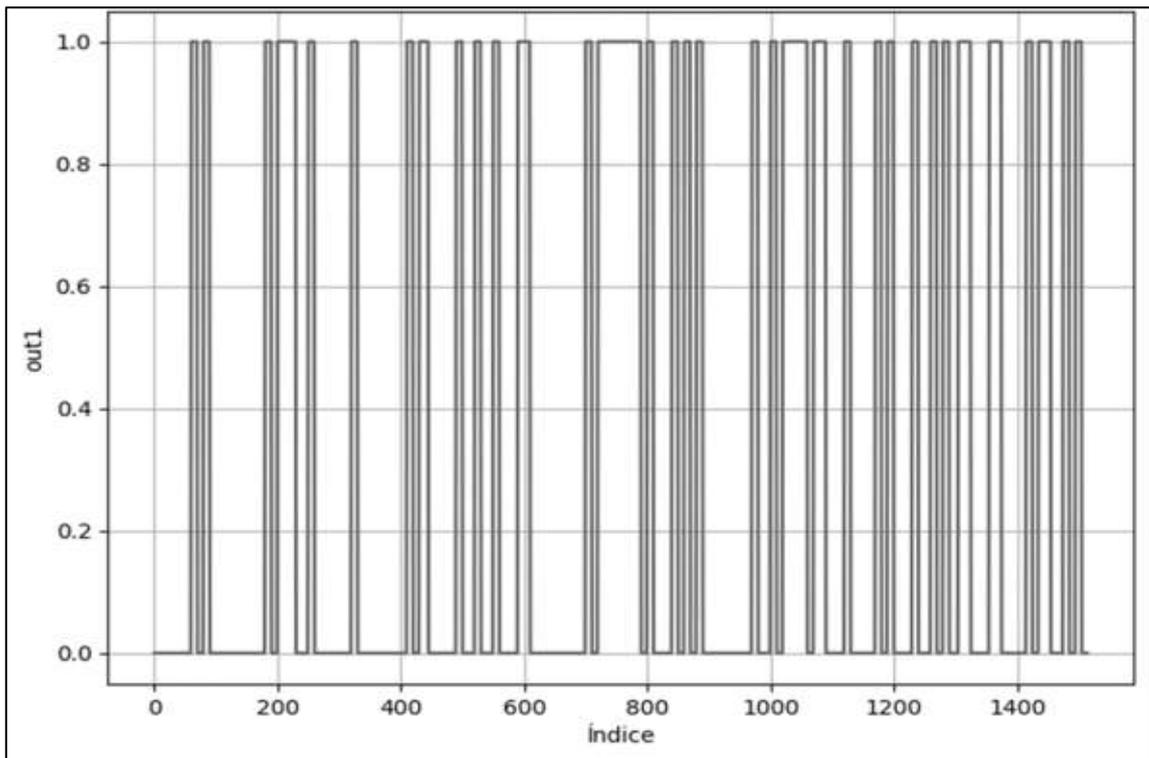


Figura 47

Gráfica de out1 para la validación cruzada

**Figura 48**

Gráfica de out2 para la validación cruzada

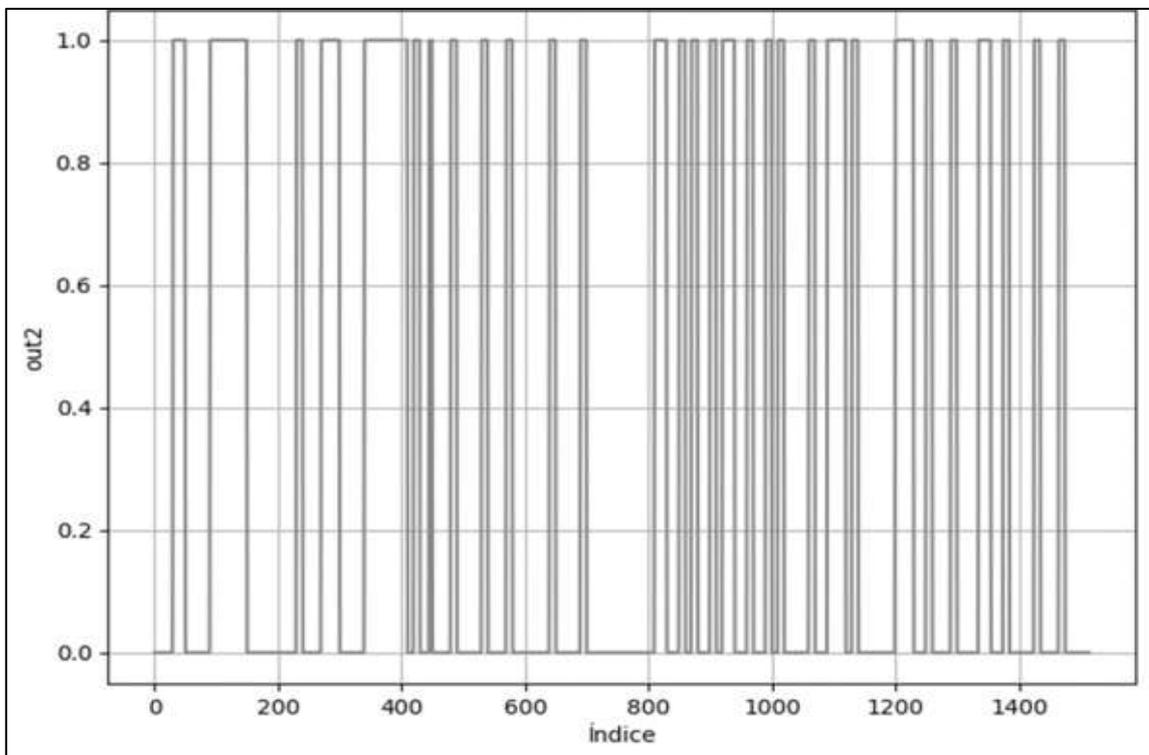
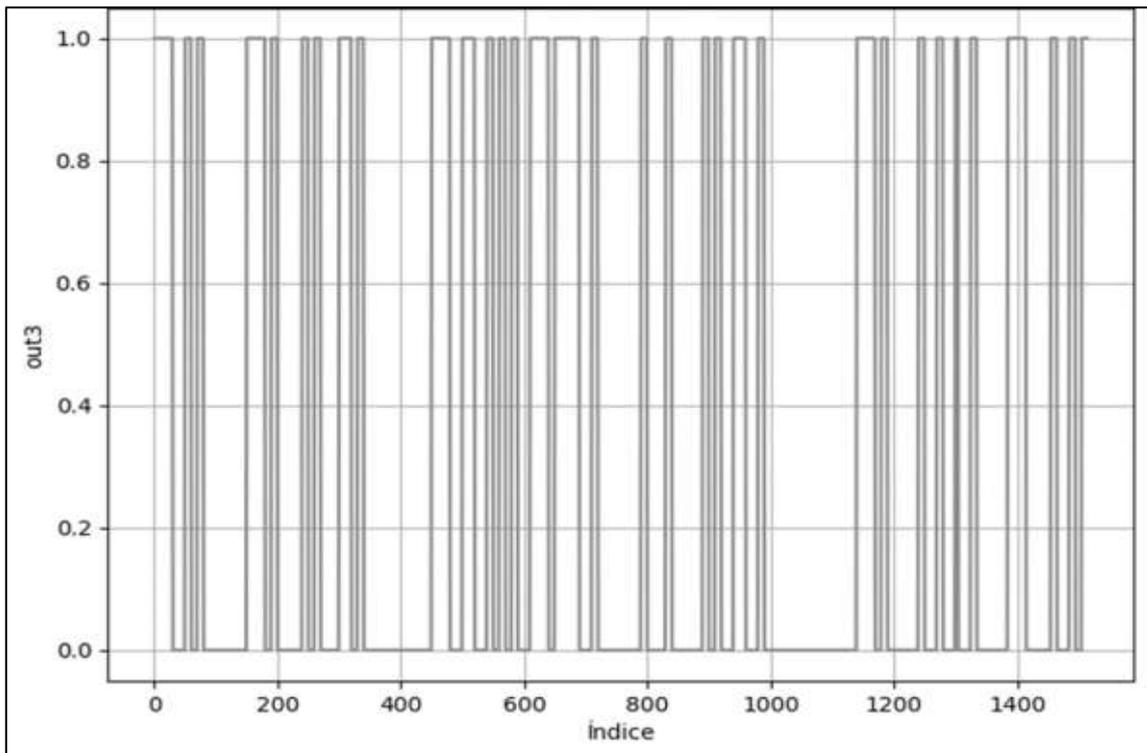


Figura 49

Gráfica de out3 para la validación cruzada



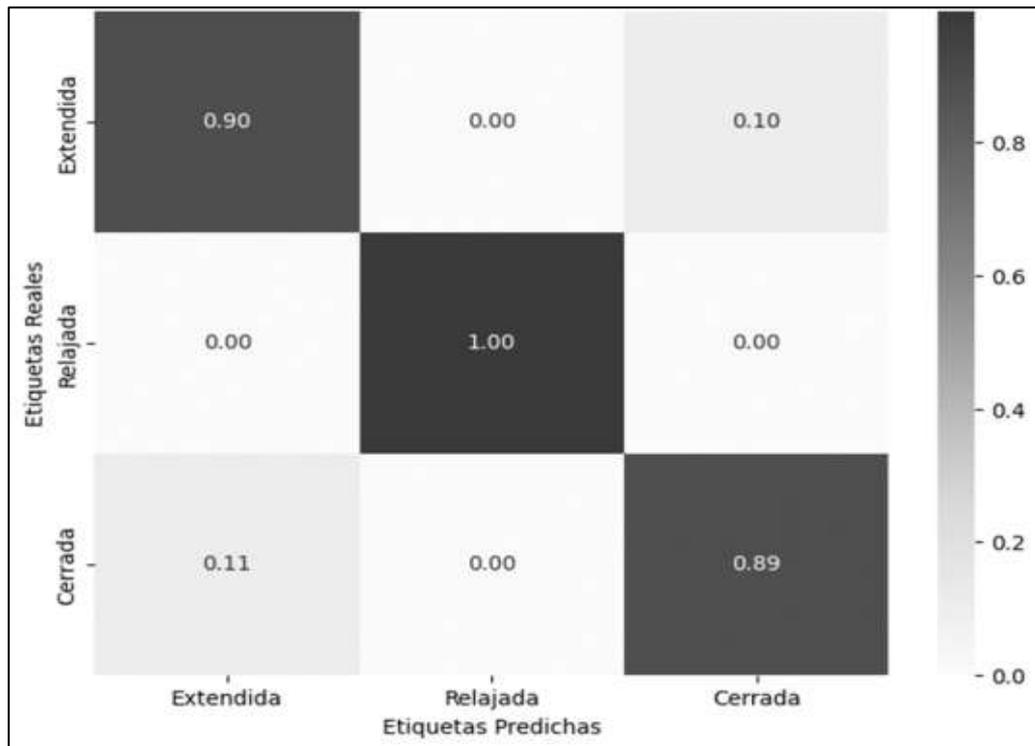
4.11.1. Validación de red neuronal secuencial

La matriz de confusión, generada utilizando un segundo conjunto de datos, evalúa el rendimiento de clasificación de un modelo mostrando el número de predicciones correctas e incorrectas para cada clase.

En la figura 50 se muestra que el modelo logró una clasificación correcta en el estado de “mano extendida” en el 90 % de los casos, lo que indica un buen equilibrio entre el aprendizaje de patrones específicos y la generalización a nuevos datos. Sin embargo, el 10 % de los casos se etiquetaron incorrectamente como “mano cerrada”, lo que sugiere ambigüedad en las características de la señal. El modelo también clasificó correctamente el estado de “mano relajada” con una precisión del 100 %, lo que indica una señal distintiva. El estado de “mano cerrada” se clasificó correctamente con un 89 % de precisión, pero el 11 % se etiquetó incorrectamente como “mano extendida”. Las pérdidas calculadas del modelo fueron del 7,19 %, lo que indica bajos errores de predicción.

Figura 50

Matriz de Confusión Normalizada de la Validación de Red Neuronal Secuencial



4.11.2. Validación de red neuronal recurrente

Como se vio en la matriz de confusión de red neuronal secuencial, se generó una matriz de confusión utilizando un segundo conjunto de datos que evalúa el rendimiento de clasificación del modelo recurrente mostrando el número de predicciones correctas e incorrectas para cada clase.

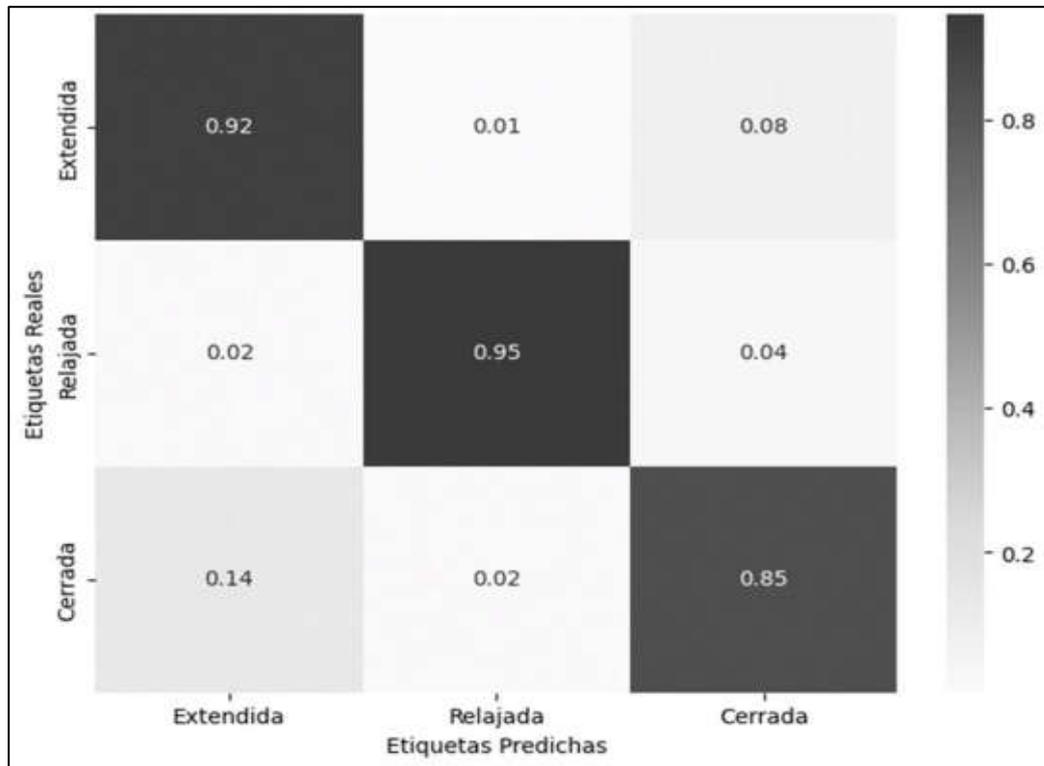
En la figura 51 se muestra que el modelo logró una clasificación correcta en el estado de “mano extendida” en el 92 % de los casos, captando características distintivas de la señal mioeléctrica. Sin embargo, el 1 % de los casos se etiquetaron incorrectamente como “mano relajada” y el 8 % se clasificaron erróneamente como “mano cerrada”. Estas clasificaciones erróneas sugieren que el modelo puede tener similitudes en las señales que pueden confundir una mano extendida con una mano cerrada, posiblemente debido a variaciones en la fuerza muscular. El modelo alcanzó una precisión del 95 % en la identificación del estado relajado de la mano, pero el 2 % de los casos se clasificaron erróneamente como “mano extendida” y el 4 % se confundieron con “mano cerrada”. El modelo también clasificó correctamente el 85 %

de los estados de “mano cerrada”, pero el 14 % se clasificó erróneamente como “mano extendida” y el 2 % como “mano relajada”.

La precisión del modelo es del 90,15 % con un error del 9,84 %.

Figura 51

Matriz de Confusión Normalizada de la Validación de Red Neuronal Recurrente



4.11.3. Validación de red neuronal convolucional

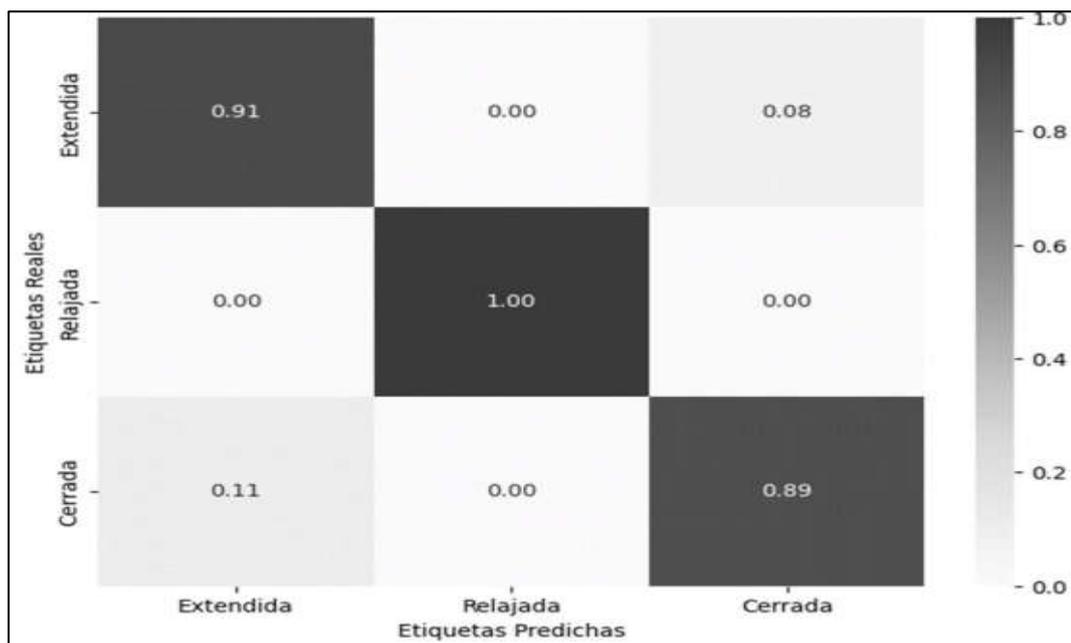
El mismo proceso se realizó en la matriz de confusión de red neuronal convolucional, se generó utilizando un segundo conjunto de datos.

En la figura 52 se muestra que el modelo etiquetó correctamente el estado de mano extendida en el 91 % de los casos, lo que indica su capacidad para identificar señales mioeléctricas. Sin embargo, el 8 % de los casos se etiquetaron incorrectamente como mano cerrada. El modelo también demostró una precisión del 100 % en la clasificación de los estados de mano relajada, lo que indica la solidez del modelo en la identificación de estos estados. El modelo clasificó correctamente el 89 % de los casos de mano cerrada, pero el 11 % se clasificó incorrectamente como mano extendida.

En conjunto, el modelo alcanzó una precisión del 93,46 %, lo que indica su capacidad para generalizar bien a partir de los datos de entrenamiento.

Figura 52

Matriz de Confusión Normalizada de la Validación de Red Neuronal Convolutiva



Elección del modelo

Para la elección del modelo se optó por obtener el promedio de las métricas de cada modelo y elegir aquel que obtiene mejores resultados. Al promediar estas métricas, se obtiene una visión global del rendimiento de cada modelo.

El modelo seleccionado fue aquel que tuvo el mejor promedio en estas métricas, lo que indica que tiene un rendimiento equilibrado y sólido en todas las áreas. Esto también ayuda a minimizar el sesgo hacia un modelo que pueda destacar en una métrica, pero fallar en otras, garantizando que el modelo sea el más robusto y confiable para el despliegue en situaciones reales. Los resultados obtenidos se muestran en la tabla 3:

Tabla 3

Resultados de promedio de las métricas de las 3 redes neuronales

Red neuronal	Precisión	Perdidas
Secuencial	91,86 %	8,14 %
Recurrente	90,95 %	9,05 %
Convolutiva	92,69 %	7,31 %

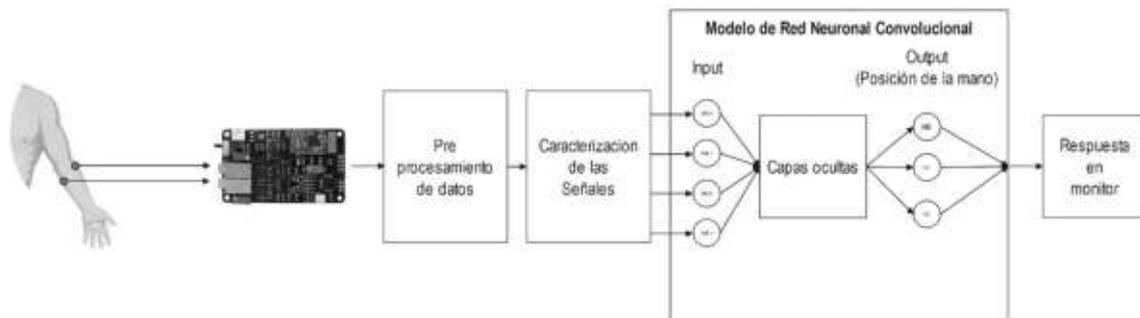
4.12. Prototipo final

Se ha desarrollado un prototipo de adquisición de señales mioeléctricas para clasificar los movimientos de la mano mediante una red neuronal convolucional (CNN) (Figura 53). El sistema utiliza sensores en los músculos del antebrazo para captar señales bioeléctricas, que se procesan y pasan como entradas a la CNN.

El modelo, entrenado para reconocer patrones, alcanza una precisión del 92,69 % en la clasificación de los movimientos de la mano. Esta elevada precisión hace que el prototipo sea adecuado para aplicaciones prácticas, como el control de prótesis, donde la clasificación precisa de los movimientos de la mano es crucial.

Figura 53

Diagrama Final de prototipo de clasificación de estados de la mano



CAPÍTULO V: DISCUSIÓN

El proceso de clasificación de los diferentes estados de la mano mediante redes neuronales ha mostrado resultados prometedores en cuanto a la precisión y la capacidad de generalización de los modelos implementados. Se utilizaron varias arquitecturas de redes neuronales, cada una con características específicas que buscaban maximizar el rendimiento en la clasificación de las señales mioeléctricas adquiridas.

En primer lugar, la red neuronal secuencial demostró un buen equilibrio entre el aprendizaje de patrones y la generalización de los datos, logrando una precisión del 91,86 %. Sin embargo, se observaron algunos casos de confusión entre los estados de "mano extendida" y "mano cerrada", lo que indica que, si bien el modelo captó la mayoría de los patrones relevantes, algunas señales presentan similitudes que pueden inducir a error en la clasificación.

Por otro lado, la red neuronal recurrente, diseñada para capturar dependencias temporales en las señales, logró una precisión del 90,95 %. Este modelo también mostró una capacidad de generalización adecuada, aunque se observó un comportamiento similar al de la red secuencial en términos de confusiones entre los estados de "mano extendida" y "mano cerrada". Esto sugiere que, aunque las dependencias temporales aportan información valiosa, es necesario un análisis más profundo para diferenciar con mayor precisión entre estos estados.

Finalmente, la red neuronal convolucional presentó el mejor rendimiento, con una precisión del 92,69 %. Este modelo aprovechó las características espaciales de las señales mioeléctricas, lo que le permitió identificar patrones con mayor precisión. A pesar de su alto rendimiento, se mantuvo una ligera confusión entre los estados mencionados, aunque en menor medida que en los modelos anteriores. Este resultado resalta la importancia de la extracción de características espaciales en la clasificación de señales mioeléctricas.

CONCLUSIONES

Se pudo diseñar e implementar un sistema de adquisición de señales no invasivos que reconocen los patrones de biopotencial muscular y determinar los estados los 3 estados mínimos para clasificar los movimientos de la mano en función de los niveles de voltaje y frecuencia de las señales mioeléctricas haciendo uso de tarjetas electrónicas diseñadas para la adquisición de biopotenciales, algoritmos y técnicas utilizadas en el campo de redes neuronales.

El modelo de red neuronal convolucional implementado logró una precisión del 92,69 %, lo que indica su alta capacidad para clasificar correctamente los estados de la mano a partir de señales mioeléctricas, superando a las otras arquitecturas evaluadas.

Todos los modelos demostraron una buena capacidad de generalización, evitando el sobreajuste y manteniendo un rendimiento estable en datos no vistos durante el entrenamiento, lo que valida la solidez de los enfoques implementados.

A pesar de la eficacia de los modelos, se observaron diferencias significativas en su rendimiento, siendo la red convolucional la más precisa, mientras que la recurrente presentó una ligera disminución en la precisión, aunque con un rendimiento aún competitivo.

Se identificaron confusiones entre los estados de "mano extendida" y "mano cerrada", lo que sugiere la necesidad de optimizaciones adicionales en la diferenciación de estas señales que presentan características similares.

El prototipo desarrollado muestra viabilidad para su uso en aplicaciones prácticas, como el control de prótesis, gracias a su alta precisión y robustez en la clasificación de movimientos de la mano.

RECOMENDACIONES

Se recomienda explorar técnicas de preprocesamiento más avanzadas, como la reducción de ruido y algoritmos de machine learning, para mejorar la diferenciación entre estados similares, como "mano extendida" y "mano cerrada".

Es recomendable ampliar el conjunto de datos de entrenamiento para incluir una mayor variedad de casos y movimientos, lo que podría ayudar a mejorar la generalización y precisión del modelo en situaciones más diversas.

Considerar la experimentación con otras arquitecturas de redes neuronales, como redes neuronales profundas o híbridas, que podrían ofrecer un mejor rendimiento al capturar patrones más complejos en las señales mioeléctricas.

Para reducir aún más el riesgo de sobreajuste y mejorar la robustez del modelo, se sugiere implementar técnicas de regularización adicionales, como la penalización L2 o el uso de dropout en capas adicionales.

Finalmente, se recomienda probar el prototipo en entornos reales, donde se puedan observar las variaciones en las señales mioeléctricas durante el uso cotidiano, para ajustar el modelo y asegurar su efectividad en aplicaciones del mundo real.

REFERENCIAS BIBLIOGRÁFICAS

- Aayesha, Qureshi, M. B., Afzaal, M., Qureshi, M. S., y Fayaz, M. (2021). Machine learning-based EEG signals classification model for epileptic seizure detection. *Multimedia Tools and Applications*, 80(12), 17849–17877. <https://doi.org/10.1007/s11042-021-10597-6>
- Abiodun, O. I., Kiru, M. U., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., y Gana, U. (2019). Comprehensive Review of Artificial Neural Network Applications to Pattern Recognition. *IEEE Access*, 7, 158820–158846. <https://doi.org/10.1109/ACCESS.2019.2945545>
- Adugna, T. D., Ramu, A., y Haldorai, A. (2024). A Review of Pattern Recognition and Machine Learning. In *Journal of Machine and Computing* 4(1). <https://doi.org/10.53759/7669/jmc202404020>
- Ahsan, M. R., Ibrahimy, M. I., y Khalifa, O. O. (2009). EMG signal classification for human computer interaction: A review. *European Journal of Scientific Research*, 33(3), 480–501. <https://n9.cl/xpcyhr>
- Al-Ayyad, M., Owida, H. A., De Fazio, R., Al-Naami, B., y Visconti, P. (2023). Electromyography Monitoring Systems in Rehabilitation: A Review of Clinical Applications, Wearable Devices and Signal Acquisition Methodologies. *Electronics*, 12(7), 1520. <https://doi.org/10.3390/electronics12071520>
- Ashraf Zargar, S. (2021). *Introduction to Sequence Learning Models: RNN, LSTM, GRU*. April. <https://doi.org/10.13140/RG.2.2.36370.99522>
- Avilés y Gaibor. (2021). *Diseño e implementación de una prótesis robótica con señales EMG usando técnicas de Inteligencia Artificial* [Tesis de Pregrado, Escuela superior politécnica litoral] Repositorio Institucional: <http://www.dspace.espol.edu.ec/handle/123456789/54698>
- Ayala Solsol, K. R., y Quispe Erasmo, J. R. (2022). *Diseño e Implementación de un Exoesqueleto Adaptable para rehabilitación de miembros inferiores en pacientes con Lesión Medular controlado mediante Lógica Difusa y Monitoreo usando IOT* [Tesis de Pregrado, Universidad Ricardo Palma] Repositorio Institucional: <https://repositorio.urp.edu.pe/handle/20.500.14138/5921>
- Barron, J. T. (2019). A General and Adaptive Robust Loss Function. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019-June,

4326–4334. <https://doi.org/10.1109/CVPR.2019.00446>

- Berthouze, L., y Farmer, S. F. (2012). Adaptive time-varying detrended fluctuation analysis. *Journal of Neuroscience Methods*, 209(1), 178–188. <https://doi.org/10.1016/j.jneumeth.2012.05.030>
- Bhattacharya, A., Sarkar, A., y Basak, P. (2017). Time domain multi-feature extraction and classification of human hand movements using surface EMG. *2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 1–5. <https://doi.org/10.1109/ICACCS.2017.8014594>
- Bisong y Ekaba. (2019). Building machine learning and deep learning models on Google Cloud Platform. In *Building Machine Learning and Deep Learning Models on Google Cloud Platform* (pp. 333–343). Apress. https://doi.org/10.1007/978-1-4842-4470-8_29
- Chollet. (2021). *Deep Learning with Python, Second Edition* (Simon and Schuster (ed.)). https://books.google.es/books?id=mjVKEAAAQBAJ&dq=There+are+other+complementary+tools+for+developing+neural+network+models+in+Python.+One+of+them+is+Keras,+a+high-level+library+that+acts+as+an+intuitive+interface+on+top+of+TensorFlow+or+Theano,+making+it+easy+to+build+and+experiment+with+learning+models&lr=&hl=es&source=gbs_navlinks_s
- Chowdhury, R., Reaz, M., Ali, M., Bakar, A., Chellappan, K., y Chang, T. (2013). Surface Electromyography Signal Processing and Classification Techniques. *Sensors*, 13(9), 12431–12466. <https://doi.org/10.3390/s130912431>
- Del Vecchio, A., Negro, F., Felici, F., y Farina, D. (2017). Associations between motor unit action potential parameters and surface EMG features. *Journal of Applied Physiology*, 123(4), 835–843. <https://doi.org/10.1152/jappphysiol.00482.2017>
- Dubova, M. (2022). Building human-like communicative intelligence: A grounded perspective. *Cognitive Systems Research*, 72(September 2021), 63–79. <https://doi.org/10.1016/j.cogsys.2021.12.002>
- Duchateau, J., y Enoka, R. M. (2011). Human motor unit recordings: Origins and insight into the integrated motor system. *Brain Research*, 1409, 42–61. <https://doi.org/10.1016/j.brainres.2011.06.011>
- Elamvazuthi, I., Ling, G. A., Nurhanim, K. A. R. K., Vasant, P., y Parasuraman, S. (2013). Surface electromyography (sEMG) feature extraction based on Daubechies wavelets. *Proceedings of the 2013 IEEE 8th Conference on Industrial Electronics*

- and Applications, ICIEA 2013, June, 1492–1495.*
<https://doi.org/10.1109/ICIEA.2013.6566603>
- Englehart, K., y Hudgins, B. (2003). A robust, real-time control scheme for multifunction myoelectric control. *IEEE Transactions on Biomedical Engineering*, 50(7), 848–854. <https://doi.org/10.1109/TBME.2003.813539>
- Fontana, Gonzalo, O'Brien, M. (2021). Clasificación de señales mioeléctricas para el control de una mano robótica. *Instituto Para El Desarrollo Agroindustrial y de La Salud*, I(1), 1–4.
<https://amcaonline.org.ar/maci/index.php/maci2021/maci2021/paper/viewFile/5356/721>
- Fontes, I. H. D., Melo, A. M. A., y Aquino, A. L. L. (2022). Monitoramento e classificação sonora em UTI Neonatal usando redes neurais. *Anais Do XIV Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP 2022)*, 101–110.
<https://doi.org/10.5753/sbcup.2022.223179>
- Gadekallu, T. R., Alazab, M., Kaluri, R., Maddikunta, P. K. R., Bhattacharya, S., Lakshmana, K., y M, P. (2021). Hand gesture classification using a novel CNN-crow search algorithm. *Complex & Intelligent Systems*, 7(4), 1855–1868.
<https://doi.org/10.1007/s40747-021-00324-x>
- Gu, Q., Zhu, L., y Cai, Z. (2009). Evaluation Measures of the Classification Performance of Imbalanced Data Sets. In *Communications in Computer and Information Science* 51, pp. 461–471. https://doi.org/10.1007/978-3-642-04962-0_53
- Gulli y Pal. (2017). *Deep learning with Keras. Packt Publishing.*
[https://github.com/sanjaysheel/Data-Science-Books-1/blob/master/Deep Learning with Keras by Antonio Gulli.pdf](https://github.com/sanjaysheel/Data-Science-Books-1/blob/master/Deep%20Learning%20with%20Keras%20by%20Antonio%20Gulli.pdf)
- Heckman, C. J., y Enoka, R. M. (2004). Physiology of the motor neuron and the motor unit. In *Handbook of Clinical Neurophysiology* 4C, pp. 119–147. Elsevier B.V.
[https://doi.org/10.1016/S1567-4231\(04\)04006-7](https://doi.org/10.1016/S1567-4231(04)04006-7)
- Hernandez Sampieri, Roberto-Fernandez Collado, Carlos-Baptista Lucio, P. (2014). *Metodología de la Investigación.* [https://www.esup.edu.pe/wp-content/uploads/2020/12/2. Hernandez, Fernandez y Baptista-Metodología Investigacion Cientifica 6ta ed.pdf](https://www.esup.edu.pe/wp-content/uploads/2020/12/2.Hernandez,%20Fernandez%20y%20Baptista-Metodolog%C3%ADa%20Investigacion%20Cientifica%206ta%20ed.pdf)
- Hikmat Haji, S., y Mohsin Abdulazeez, A. (2021). Comparison of optimization techniques based on gradient descent algorithm: a review. *Journal Of Archaeology Of Egypt/Egyptology*, 18(4), 2715–2743.

<https://archives.palarch.nl/index.php/jae/article/view/6705>

- Ho, H., y Kuvaas, B. (2020). Human resource management systems, employee well-being, and firm performance from the mutual gains and critical perspectives: The well-being paradox. *Human Resource Management*, 59(3), 235–253. <https://doi.org/10.1002/hrm.21990>
- Hossain, M. A., y Alam Sajib, M. S. (2019). Classification of Image using Convolutional Neural Network (CNN). *Global Journal of Computer Science and Technology*, 19(May), 13–18. <https://doi.org/10.34257/GJCSTDVOL19IS2PG13>
- Huang Ling-fang. (2010). Artificial Intelligence. *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*, 4, 575–578. <https://doi.org/10.1109/ICCAE.2010.5451578>
- Hurtado, J. M., Rubchinsky, L. L., y Sigvardt, K. A. (2004). Statistical Method for Detection of Phase-Locking Episodes in Neural Oscillations. *Journal of Neurophysiology*, 91(4), 1883–1898. <https://doi.org/10.1152/jn.00853.2003>
- Ian Goodfellow, Yoshua Bengio, A. C. (2014). Front Matter. In *Linear Algebra* (Vol. 521, Issue 7553, pp. i–ii). Elsevier. <https://doi.org/10.1016/B978-0-12-391420-0.09987-X>
- Ingle, V. K., y Proakis, J. G. (2008). Digital Signal Processing Using MATLAB (3rd Edition). In *Electronics & Communications Engineering Journal* 5(1).
- Journal, E., Eluyode, O. S., y Akomolafe, D. T. (2013). Scholars Research Library Comparative study of biological and artificial neural networks. *Of Applied Engineering and Scientific Research*, 2(1), 36–46. <https://n9.cl/6nwcni>
- Jovanović, R. Ž., Sretenović, A. A., y Živković, B. D. (2015). Ensemble of various neural networks for prediction of heating energy consumption. *Energy and Buildings*, 94, 189–199. <https://doi.org/10.1016/j.enbuild.2015.02.052>
- Källström, E., Olsson, T., Lindström, J., Hakansson, L., y Larsson, J. (2020). On-board Clutch Slippage Detection and Diagnosis in Heavy Duty Machine. *International Journal of Prognostics and Health Management*, 9(1), 1–14. <https://doi.org/10.36001/ijphm.2018.v9i1.2693>
- Kandel, Schwartz, Jessell, Siegelbaum, H. (2017). Principio de neurales en ciencia. In *Вестник Росздрава* 4(1). [https://ia801508.us.archive.org/34/items/PrinciplesOfNeuralScienceFifthKANDEL/Principles of Neural Science, Fifth - KANDEL.pdf](https://ia801508.us.archive.org/34/items/PrinciplesOfNeuralScienceFifthKANDEL/Principles%20of%20Neural%20Science,%20Fifth%20-%20KANDEL.pdf)

- Katte, T. (2018). Recurrent Neural Network and its Various Architecture Types. *International Journal of Research and Scientific Innovation*, V(III), 124–129. <https://n9.cl/234y2>
- Keijsers, N. L. W. (2010). Neural Networks. In *Encyclopedia of Movement Disorders* (pp. 257–259). Elsevier. <https://doi.org/10.1016/B978-0-12-374105-9.00493-7>
- Kenji, S. (2011). *Artificial Neural Networks: Methodological Advances and Biomedical Applications - Google Kitaplar* (Issue April 2011).
- Kim, D.-K., Omidshafiei, S., Papis, J., y How, J. P. (2020). Crossmodal attentive skill learner: learning in Atari and beyond with audio–video inputs. *Autonomous Agents and Multi-Agent Systems*, 34(1), 16. <https://doi.org/10.1007/s10458-019-09439-5>
- Kumar, K., y Thakur, G. S. M. (2012). Advanced Applications of Neural Networks and Artificial Intelligence: A Review. *International Journal of Information Technology and Computer Science*, 4(6), 57–68. <https://doi.org/10.5815/ijitcs.2012.06.08>
- Lai, G., Chang, W.-C., Yang, Y., y Liu, H. (2018). Modeling Long- and Short-Term Temporal Patterns with Deep Neural Networks. *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 95–104. <https://doi.org/10.1145/3209978.3210006>
- Li, G., Huang, Y., Chen, Z., Chesser, G. D., Purswell, J. L., Linhoss, J., y Zhao, Y. (2021). Practices and Applications of Convolutional Neural Network-Based Computer Vision Systems in Animal Farming: A Review. *Sensors*, 21(4), 1492. <https://doi.org/10.3390/s21041492>
- Lipton, Z. C., Berkowitz, J., y Elkan, C. (2015). *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 1–38. <http://arxiv.org/abs/1506.00019>
- Llantoy, A. (2020). *Diseño e implementación del sistema electrónico para una prótesis transradial mioeléctrica*. [Tesis de Pregrado, Pontificia Universidad Católica del Perú].
- Lou, A. (2007). *Data Preparation in Neural Network Data Analysis* (pp. 39–62). https://doi.org/10.1007/978-0-387-71720-3_3
- Ludvig, D., Visser, T. S., Giesbrecht, H., y Kearney, R. E. (2011). Identification of Time-Varying Intrinsic and Reflex Joint Stiffness. *IEEE Transactions on Biomedical Engineering*, 58(6), 1715–1723. <https://doi.org/10.1109/TBME.2011.2113184>
- Macrì, M., Flores, N. V. G., Stefanelli, R., Pegreffì, F., y Festa, F. (2023). Interpreting the

- prevalence of musculoskeletal pain impacting Italian and Peruvian dentists likewise: A cross-sectional study. *Frontiers in Public Health*, 11(1). <https://doi.org/10.3389/fpubh.2023.1090683>
- Manz, S., Valette, R., Damonte, F., Avanci Gaudio, L., Gonzalez-Vargas, J., Sartori, M., Dosen, S., y Rietman, J. (2022). A review of user needs to drive the development of lower limb prostheses. *Journal of NeuroEngineering and Rehabilitation*, 19(1), 119. <https://doi.org/10.1186/s12984-022-01097-1>
- Mao, P., Li, H., y Yu, Z. (2023). A Review of Skin-Wearable Sensors for Non-Invasive Health Monitoring Applications. *Sensors*, 23(7), 3673. <https://doi.org/10.3390/s23073673>
- McKinney. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (l. O'Reilly Media (ed.)). https://books.google.es/books?id=v3n4_AK8vu0C&dq=In+the+field+of+data+processing,+Python+offers+several+libraries+and+tools+that+make+it+easy+to+manage,+analyze,+and+visualize+data+sets.+One+of+the+most+important+libraries+is+NumPy&lr=&hl=es&source=gbs_navlinks_s
- Mello, R. G. T., Oliveira, L. F., y Nadal, J. (2007). Digital Butterworth filter for subtracting noise from low magnitude surface electromyogram. *Computer Methods and Programs in Biomedicine*, 87(1), 28–35. <https://doi.org/10.1016/j.cmpb.2007.04.004>
- Menon, R., Di Caterina, G., Lakany, H., Petropoulakis, L., Conway, B. A., y Soraghan, J. J. (2017). Study on Interaction Between Temporal and Spatial Information in Classification of EMG Signals for Myoelectric Prostheses. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(10), 1832–1842. <https://doi.org/10.1109/TNSRE.2017.2687761>
- Mitra, S. K. (2001). Digital Signal Processing - Computer Based Approach. In *Microelectronics Journal* 128(8), p. 879. [https://doi.org/10.1016/S0026-2692\(98\)00072-X](https://doi.org/10.1016/S0026-2692(98)00072-X)
- Mohanty, S. P., Hughes, D. P., y Salathé, M. (2016). Using Deep Learning for Image-Based Plant Disease Detection. *Frontiers in Plant Science*, 7(September), 1–10. <https://doi.org/10.3389/fpls.2016.01419>
- Myers, S. J., y Lovelace, R. E. (1994). The Motor Unit and Muscle Action Potentials. In *The Physiological Basis of Rehabilitation Medicine* (Issue January 2000, pp. 243–282). Elsevier. <https://doi.org/10.1016/B978-1-4831-7818-9.50017-4>

- Nusrat, I., y Jang, S.-B. (2018). A Comparison of Regularization Techniques in Deep Neural Networks. *Symmetry*, 10(11), 648. <https://doi.org/10.3390/sym10110648>
- Oppenheim, A., y Schafer, R. (1999). *Discrete-Time Processing- Second Edition*. https://research.iaun.ac.ir/pd/naghsh/pdfs/UploadFile_2230.pdf
- Pascanu, R., Gulcehre, C., Cho, K., y Bengio, Y. (2014, del 13 al 14 de abril). How to construct deep recurrent neural networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 1–13. <https://arxiv.org/pdf/1312.6026>
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury Google, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Xamla, A. K., Yang, E., Devito, Z., Raison Nabla, M., Tejani, A., Chilamkurthy, S., Ai, Q., Steiner, B., ... Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems. NeurIPS, NeurIPS, 8026–8037*. https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf
- Petit, F., y Loccufer, M. (2009). Torsional Vibrations on a Hopper Dredger Due to Transient Conditions. *Volume 1: 22nd Biennial Conference on Mechanical Vibration and Noise, Parts A and B, 1(Parts A And B)*, 1215–1224. <https://doi.org/10.1115/DETC2009-87374>
- Phinyomark, A., Khushaba, R. N., y Scheme, E. (2018). Feature extraction and selection for myoelectric control based on wearable EMG sensors. *Sensors (Switzerland)*, 18(5), 1–17. <https://doi.org/10.3390/s18051615>
- Pinto, R. A., Coronel, F. S., Bueno, F. L., y Galan, J. (2020). Reconocimiento de tres patrones básicos de movimiento de la mano usando electromiografía de superficie y algoritmos inteligentes. *Revista Cubana de Investigaciones Biomédicas*. 2020;39(2):E226, 39(2), 1–14.
- Rampichini, S., Vieira, T. M., Castiglioni, P., y Merati, G. (2020). Complexity Analysis of Surface Electromyography for Assessing the Myoelectric Manifestation of Muscle Fatigue: A Review. *Entropy*, 22(5), 529. <https://doi.org/10.3390/e22050529>
- Rapczyński, M., Werner, P., Handrich, S., y Al-Hamadi, A. (2021). A Baseline for Cross-Database 3D Human Pose Estimation. *Sensors*, 21(11), 3769. <https://doi.org/10.3390/s21113769>
- Reátegui Pinazo. (2024). *Diseño e implementación de un sistema embebido portátil para*

la adquisición y procesamiento de señales electromiográficas del antebrazo [Tesis de Pregrado, Pontificia universidad catolica del Perú]. <http://hdl.handle.net/20.500.12404/27800>

- Resalat, S. N., y Saba, V. (2016). A study of various feature extraction methods on a motor imagery based Brain Computer Interface system. *Basic and Clinical Neuroscience*, 7(1), 13–20.
- Rodriguez-Falces, J., y Place, N. (2014). Effects of muscle fibre shortening on the characteristics of surface motor unit potentials. *Medical & Biological Engineering & Computing*, 52(2), 95–107. <https://doi.org/10.1007/s11517-013-1112-z>
- Romo Avilés, M. (2022). *Clasificación de señales mioeléctricas por medio de algoritmos genéticos y máquinas de soporte de vectores* [Tesis de Maestría, Universidad autonoma de Queretaro]. <https://ri-ng.uaq.mx/bitstream/123456789/3769/1/RI006792.pdf>
- Sabastizagal-Vela, I., Astete-Cornejo, J., y Benavides, F. G. (2020). Condiciones de trabajo, seguridad y salud en la población económicamente activa y ocupada en áreas urbanas del Perú. *Revista Peruana de Medicina Experimental y Salud Pública*, 37(1), 32–41. <https://doi.org/10.17843/rpmesp.2020.371.4592>
- Safiri, S., Kolahi, A., Cross, M., Hill, C., Smith, E., Carson-Chahhoud, K., Mansournia, M. A., Almasi-Hashiani, A., Ashrafi-Asgarabad, A., Kaufman, J., Sepidarkish, M., Shakouri, S. K., Hoy, D., Woolf, A. D., March, L., Collins, G., y Buchbinder, R. (2021). Prevalence, Deaths, and Disability-Adjusted Life Years Due to Musculoskeletal Disorders for 195 Countries and Territories 1990–2017. *Arthritis & Rheumatology*, 73(4), 702–714. <https://doi.org/10.1002/art.41571>
- Salman, S., y Liu, X. (2019). *Overfitting Mechanism and Avoidance in Deep Neural Networks*. <http://arxiv.org/abs/1901.06566>
- Sanchez-Carmona, Villanueva-Cañizares, Gómez.Rodríguez, G.-H. (2021). Clasificación de señales cerebrales para el control de RPAS en el tratamiento del trastorno por déficit de atención e hiperactividad. *Ingeniería y Tecnología Aeronáutica*, 96(1), 1–6. <https://doi.org/10.6036/9496>
- Schober, P., Boer, C., y Schwarte, L. A. (2018). Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia & Analgesia*, 126(5), 1763–1768. <https://doi.org/10.1213/ANE.0000000000002864>
- Senthilnathan, S. (2019). Usefulness of Correlation Analysis. *SSRN Electronic Journal*. <https://doi.org/10.2139/ssrn.3416918>

- Shabbir, J., y Anwer, T. (2018). *Artificial Intelligence and its Role in Near Future*. 14(8), 1–11. <http://arxiv.org/abs/1804.01396>
- Shahzaib, M., y Shakil, S. (2019). Hand electromyography circuit and signals classification using artificial neural network. *2018 14th International Conference on Emerging Technologies, ICET 2018*, 1–6. <https://doi.org/10.1109/ICET.2018.8603587>
- Shanthamallu, U. S., y Spanias, A. (2022). Neural Networks and Deep Learning. In *Synthesis Lectures on Signal Processing*. https://doi.org/10.1007/978-3-031-03758-0_5
- Sharma, K., y Sharma, R. (2019). Design considerations for effective neural signal sensing and amplification: a review. *Biomedical Physics & Engineering Express*, 5(4), 042001. <https://doi.org/10.1088/2057-1976/ab1674>
- Sharma, S., Sharma, S., y Athaiya, A. (2020). Activation functions in neural networks. *International Journal of Engineering Applied Sciences and Technology*, 04(12), 310–316. <https://doi.org/10.33564/IJEAST.2020.v04i12.054>
- Singh, D., y Singh, B. (2020). Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97, 105524. <https://doi.org/10.1016/j.asoc.2019.105524>
- Solis. (2020). *Particularidades de la interacción humano-robot (HRI)*. December 2019. <https://doi.org/http://dx.doi.org/10.13140/RG.2.2.13375.94888>
- Srivastava, T., y Chosdol, K. (1997). Molecular Physiology. *The Journal of Physiology*, 499(suppl), 121–126. <https://doi.org/10.1113/jphysiol.1997.sp021989>
- Stancin, I., y Jovic, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 977–982. <https://doi.org/10.23919/MIPRO.2019.8757088>
- Strauss, W. (2000). Digital signal processing. In *IEEE Signal Processing Magazine* 17(2). <https://doi.org/10.1109/79.826412>
- Suah, F. B. M. (2017). Preparation and characterization of a novel Co(II) optode based on polymer inclusion membrane. *Analytical Chemistry Research*, 12, 40–46. <https://doi.org/10.1016/j.ancr.2017.02.001>
- Sulla Espinoza. (2023). Prótesis de brazo electromiográfico intuitivo asistido con inteligencia artificial [Tesis Doctoral, Universidad nacional de San Agustín de

- Arequipa]. In *Repositorio universidad nacional San Agustín de Arequipa*: 1(1). <https://repositorio.unsa.edu.pe/server/api/core/bitstreams/177bc796-9acb-4e2c-bb3b-962d28e42850/content>
- Teigens, V., Skalfist, P., Mikelsten, D. (2020). *Inteligencia artificial: la cuarta revolución industrial* (Cambridge Stanford Books (ed.)). https://books.google.es/books?id=sR3NDwAAQBAJ&dq=la+IA+involucra+la+creación+de+sistemas+que+pueden+igualar+o+superar+el+desempeño+humano+en+tareas+específicas&lr=&hl=es&source=gbs_navlinks_s
- Tian Seng. (2016). Machine Learning. In *Studies in Systems, Decision and Control* 65, pp. 121–151. https://doi.org/10.1007/978-981-10-1509-0_9
- Tortajada, P. E. (2023). Inteligencia Artificial. In *Revista de Derecho Civil* 10(2). <https://doi.org/10.29057/xikua.v3i5.1271>
- Van Rossum, G., y Drake, F. L. (2012). *The Python Language Reference Release 3.2.3*. 121. <http://marvin.cs.uidaho.edu/Teaching/CS515/pythonReference.pdf>
- Weerakody, P. B., Wong, K. W., Wang, G., y Ela, W. (2021). A review of irregular time series data handling with gated recurrent neural networks. *Neurocomputing*, 441, 161–178. <https://doi.org/10.1016/j.neucom.2021.02.046>
- Widmann, A., Schröger, E., y Maess, B. (2015). Digital filter design for electrophysiological data – a practical approach. *Journal of Neuroscience Methods*, 250, 34–46. <https://doi.org/10.1016/j.jneumeth.2014.08.002>
- Willsky, A. S., y Oppenheim, A. V. (1998). Señales y Sistemas. In *Señales Y Sistemas*. http://www2.fisica.unlp.edu.ar/~jarne/Clases-EET-N2/Sistemas_de_comunicaciones/Oppenheim_Senales_y_Sistemas.pdf
- Yadav, O., Bastola, L., y Sharma, J. (2021). Speech Emotion Recognition using Convolutional Recurrent Neural Network. *Proceedings of 10th IOE Graduate Conference, June*. <https://doi.org/10.37200/IJPR/V24I8/PR280260>
- Zhang, C., Bengio, S., Hardt, M., Recht, B., y Vinyals, O. (2021). Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3), 107–115. <https://doi.org/10.1145/3446776>
- Zhang, H., Zhang, L., y Jiang, Y. (2019). Overfitting and Underfitting Analysis for Deep Learning Based End-to-end Communication Systems. *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, 1–6. <https://doi.org/10.1109/WCSP.2019.8927876>

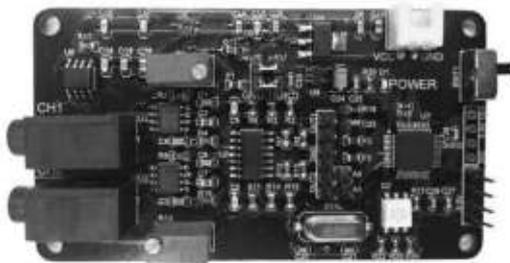
Zubieta, L. (2020). *Diseño y desarrollo del prototipo GR19 para el monitoreo en terapias para rehabilitación de la mano en Jireh Medical Import SAC , SJL 2019*. [Tesis de Pregrado, Universidad Tecnológica del Perú] Repositorio Institucional: <https://hdl.handle.net/20.500.12867/3509>.

ANEXOS

Anexo 1. Matriz de consistencia

Problema	Objetivos	Variables	Indicadores	Índice	Metodología
<p>Problema general</p> <p>¿Cómo el diseño e implementación de un sistema de adquisición de señales no invasivo reconocerá los patrones de biopotencial muscular y determinar los estados mínimos necesario para clasificar los movimientos de una mano?</p> <p>Problemas específicos</p> <p>a. ¿Cuáles son los estados mínimos necesarios para clasificar los movimientos de una mano?</p> <p>b. ¿Qué parámetros que se deben tomar en cuenta para reconocer los patrones de biopotencial muscular?</p> <p>c. ¿Qué elementos de hardware y software son necesario para el diseño del prototipo?</p>	<p>Objetivo general</p> <p>-Diseñar e implementar un sistema de adquisición de señales no invasivo para reconocer los patrones de biopotencial muscular y determinar los estados mínimos necesarios para clasificar los movimientos de una mano.</p> <p>Objetivos específicos</p> <p>-Determinar los estados mínimos necesarios para clasificar los movimientos de una mano.</p> <p>-Determinar los elementos de hardware y software necesarios para el diseño del prototipo.</p> <p>-Determinar los parámetros que se deben tomar en cuenta para reconocer los patrones de biopotencial muscular.</p>	<p>Dependiente:</p> <p>Clasificación de un movimiento de la mano.</p> <p>Independiente:</p> <p>Sistema de adquisición de biopotencial muscular.</p>	<p>- Estado de la mano: - Abierta - Relajada - Cerrado</p> <p>- Nivel de voltaje - Niveles de frecuencia</p>	<p>-mV -Hz</p>	<p>Tipo de investigación: -Aplicada</p> <p>Diseño de la investigación: -Investigación experimental</p> <p>Metodología del diseño: -Diseño basado en el ciclo de un producto: cascada</p>

Anexo 2. Datos de tarjeta de adquisición de señales EMG SichyRay 2Ch

SICHIRAY2018 年 11 月 28 日
Rev1.0

2 channel EMG Sensor Module

[Brief Intro]

The 2 channel EMG module includes an analog acquisition circuit and a digital signal filtering process .The front-end acquisition circuit collects the muscle electrical signals of the human arm or leg through CH1 and CH2. After the signal amplified and filtered, the analog acquisition data are output by OUT1 and OUT2. The waveform of the muscle electrical signal can be observed directly through the Wave Filter. We use a single-chip microcomputer for digital filtering processing. A0, A1 are connected with OUT1 and OUT2, and the values of muscle electric power are collected for processing to obtain power. Then, the power value of the muscle electricity is output through the serial port; the mean value of the muscle electricity; the collected value of the muscle electricity; the muscle strength value.

SÍCHIRAY

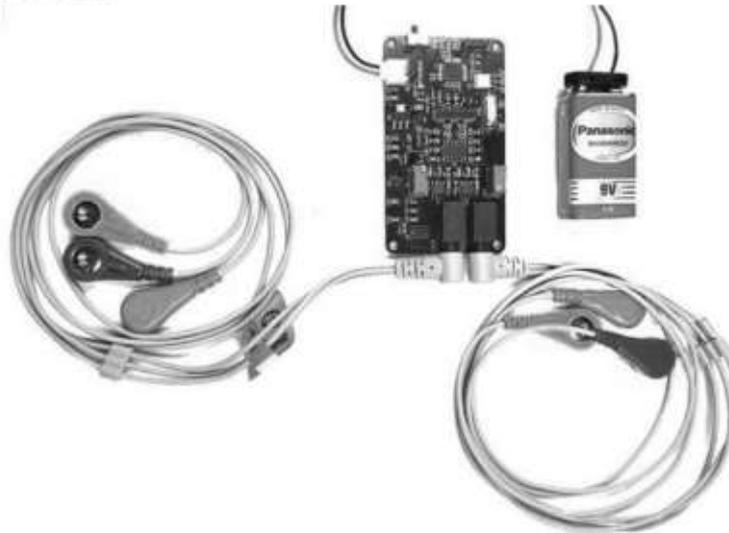
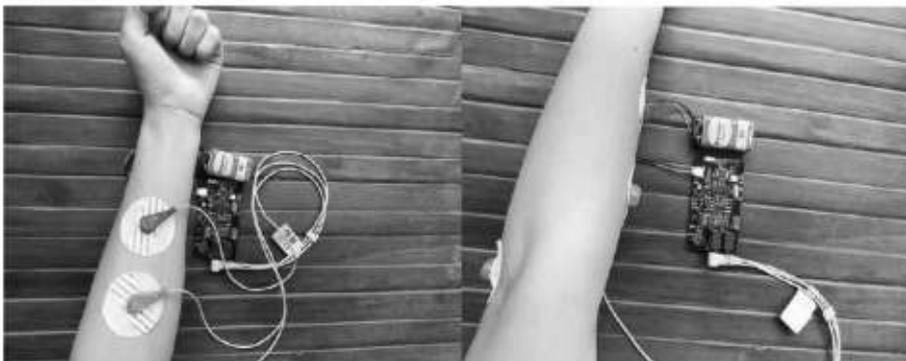
2018 年 11 月 28 日
Rev1.0

This product uses RGB lights for simple motion recognition. Gently swing your arm, only the green light is flashing; and waving your arm, the red light and green light flashes alternately. We will share the data format. Developers could read the data though the serial port.

【 Parameter 】

- Connector: USB、Pinhead
- Voltage: 9V battery
- Communication: Serial COM Port
- Indicator: Power LED

【 Hookup 】

SICHIRAY2018 年 11 月 28 日
Rev1.0**【Demo】**

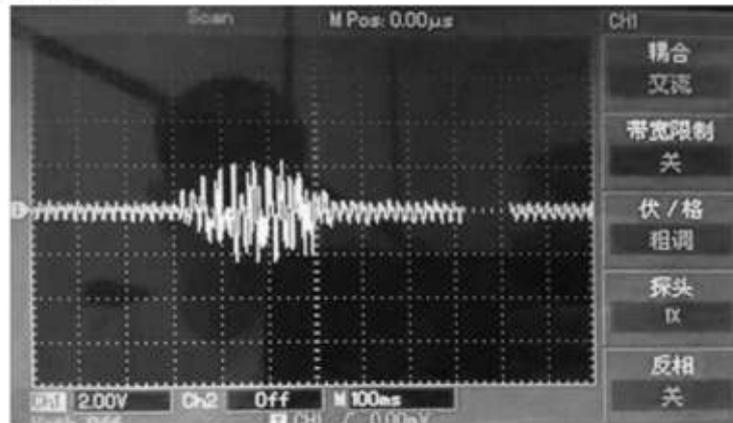
The red electrode is connected to the elbow (We need to select the area without muscle activity as the reference electrode)

Green and yellow electrodes are connected to the muscle to be tested

This product has 2 channels signal. For convenient demo, the image only user one.

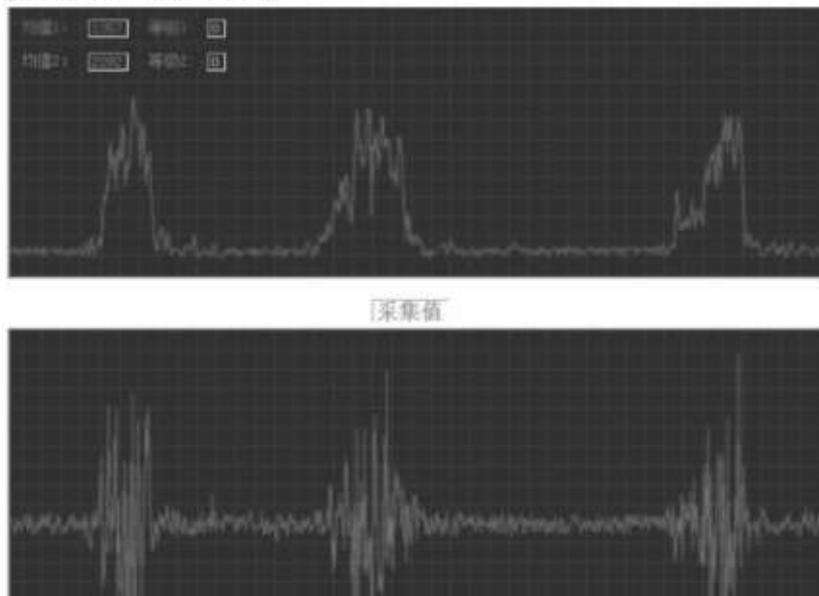
【Oscilloscope Waveform】

SICHIRAY

2018年11月28日
Rev1.0

Connect the oscilloscope to OUT1 or OUT2 and swing your arm. You could observe the waveform changes.

【Software waveform】



The waveform above is the power value, and the waveform below is the acquisition value. For more details, see the "2ch EMG PC Manual"

【Data Format】

- Baudrate : 115200, 1 byte stop, 8-byte data bit with parity
- The serial output package uses the two bytes of 0x0d 0x0a as the phrase header

header	package
0x0d 0x0a	byte 1 0x2c byte2 0x2c byte3 0x2c byte4 0x2c byte5 0x2c byte6 0x2c byte7 0x2c byte8

- Each data contains eight valid data, each with 0x2c as the interval, the overall data format:

3. Explanations:

byte 1: A0 Mean value of acquisition data;

byte 2: A0 Value after simple filtering;

byte 3: A0 Power value;

byte 4: A0 The muscle strength level.;

byte 5: A1Mean value of acquisition data;

byte 6: A1Value after simple filtering;

byte 7: A1Power value;

byte 8: A1 The muscle strength level.

4. The output format of each byte is in string form, such as A0 value of acquisition data is 1500, output format is 0x310x350x300x30, A0 value of acquisition data is 980 output format is 0x390x380x30;

5. For a instance, the total packaged data :

2107, 2112, 5, 0, 2111, 2103, 6, 0

Corresponding HEX format:

0D 0A 32 31 30 37 2C 32 31 31 32 2C 35 2C 32 31 31 31
2C 32 31 30 33 2C 36 2C 30

6、You could read the below data though the serial com assistance:



【 User Guide 】

1. Connect 9V battery (the ripple of battery is relatively small, it is not recommended to use other power supply method). Turn on the switch and observe that the power indicator light is on to indicate that the power supply is normal.

2. +VS and+5V, GND and DGND are connected by jumper cap (default status)
3. Connect A0 to OUT1, A1 to OUT2 with dupont wire
4. Turn off the power, properly wear the electrode according to the wearing diagram. Turn on the power, then the RGB red light will be on for 1.5s, ready to initialize the acquisition, keep the arm relax. the RGB blue light is on for 0.5s, indicating that the initial data is being collected. The acquisition is completed and the RGB indicator is off.

【 PC software 】

1. The serial pin of the module is connected with Bluetooth (the CH340), which is connected to another Bluetooth receiver. The receiver is plugged into the USB port of the computer. Interface connection definition: RX->TX, TX->RX, 3.3V->3.3V, DGND->GND.
2. Open the software, observe the waveform according to the "2ch EMG PC Manual". If the waveform is small, adjust the gain resistance.

【 Attention Notes 】

- Please follow the 4th step of **【 User Guide 】** strictly, keep the muscle relaxing and then turn on the power for the first Initialization
- When using oscilloscope or PC software to monitor the waveform, if the waveform is small, you could adjusting potentiometer to amplify the wave.

- Due to the DuPont wire connection, there might be the problem of bad contact and clutter during the data collection. Be sure to use the good quality wire to avoid the issues.
- Users can develop their own algorithms to use this module.

Technical Specifications

Name	2 Channel EMG Sensor Module
Connection mode	USB/Pin header
Working voltage	9V battery
Communication	Serial COM Port
Output amplitude	V (0-4.5V)
Input bias current	250pA, max 1nA
Input impedance	800 ohm
Common mode rejection ratio	150 dB
Gain Equation	$G=200$
Filters	High-pass Filter: Active 1st order, $f_c = 20.8$ Hz, 2-20dB Low-pass Filter: Active 1st order, $f_c = 498.4$ Hz 2-20dB

SÍCHIRAY2018 年 11 月 28 日
Rev1.0

Disclaim: This is not a medical device and cannot be used for any official diagnosis or medical purposes!

Anexo 3. Manual de software EMG SichyRay 2Ch

Instruction of PC software

Installation and Running

1. Find the double-conductive muscle electrical installation package in the folder.exe installation package file, double-click Run.
2. The software installation bit silent installation mode, double-click without any operation, the software will complete the installation itself.



3. When the software is installed, a shortcut is automatically created at the interface and double-click to run the software. The shortcut icons are as follows:



Usage

1. Connect the device to the computer before the software runs to ensure that the serial port (115200) gets the data normally.



2. Open the software, set the string slogan, and select the correct string slogan. The software will automatically record the string slogan set last time, and the next run will automatically match the same setting last time, as long as the adapter is inserted in the same USB port.



3. Sampling frequency settings can control the number of data collected per second by the software, by using the sampling rate, with a recommended value of between (900Hz~1000Hz), thus ensuring the accuracy of the number of samples. The sampling frequency will automatically stop the recording thread of the icon refresh and recording function to ensure the consistency of the data sampling frequency. Please click the image start button on the right side after the setting is completed.



4. The refresh interval time setting is designed to control the time when the chart data moves from left to right. This function can be set dynamically, which only requires the refresh time required to be input in the text box. The recommended time range for dual guide muscle



power is between 10 seconds and ~15 seconds.

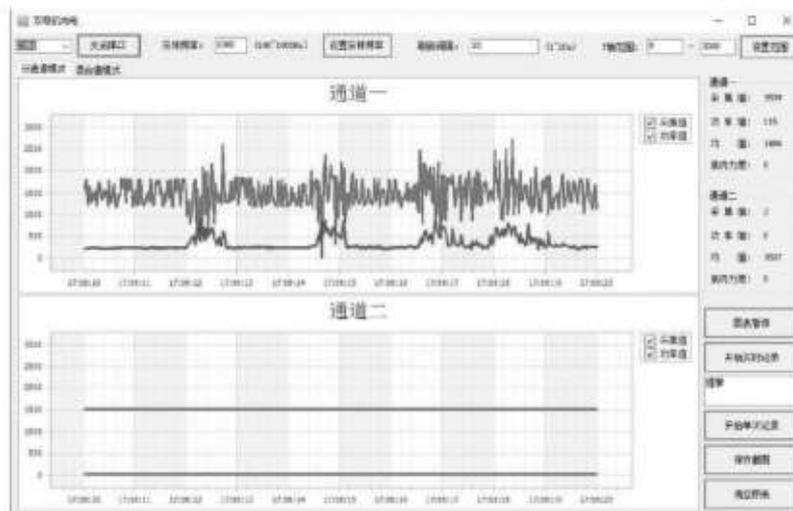
5. The Y-axis range setting can control the display range of the Y-axis by clicking the setting range. Because the muscle fiber strength and each muscle release strength are different, so the collection value range may also appear large fluctuations. You can set the Y-axis range to control the line chart display in the chart to prevent data from copying the table drawing range.



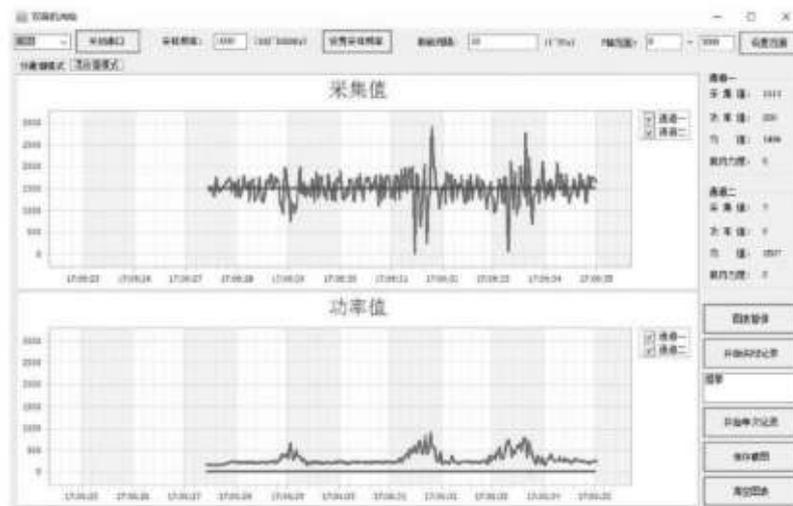
6. The split channel mode is to draw the data of two channels into two tables, each representing one channel, each containing two broken lines of acquisition value and power value. Also, of course, the check box to display the data broken line on the legend.

Collection value: the power value of the raw wave data after a certain filtering processing

Power value: The power value extracted by a certain algorithm can be used for simple eigenvalue identification.



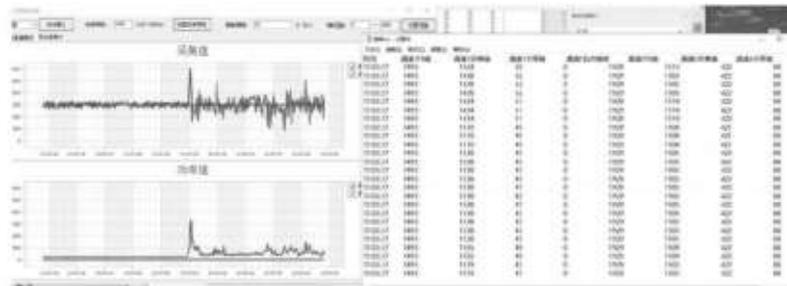
7. Mixed value mode, which compares the acquisition value and power values of the two channels, including the acquisition value and power values of channel one channel two, respectively.



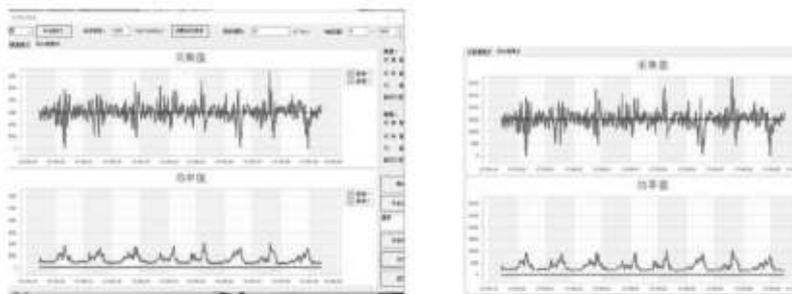
8. Chart start / pause button: You can control the chart start and stop refresh function; Start / Stop Real-time Record: After clicking the start record, the data of each channel will be recorded in real time until the button is clicked again (the button will stop the real-time record), and then the file will be automatically saved to. In the txt file, the file name is the date and time when the test started.



9. Start / Stop Single Record: Start the record, click the button again (the button is to stop the single record), and the file will be automatically saved to. In the txt file, notes need to be filled in the text box before the file is saved. This function is suitable for data analysis function for different time periods or actions.



10. Save the screenshot: The line chart of the current chart is saved in JPG format for data analysis. The name of the screenshot is the time of the screenshot. The Screenshot feature can use only one between single-icon or multi-chart mode, without saving screenshots of both modes.



Note: All saved data and images are saved in the MyoFile folder on the desktop, which contains three subfolders with real-time records, screenshots, and single records.

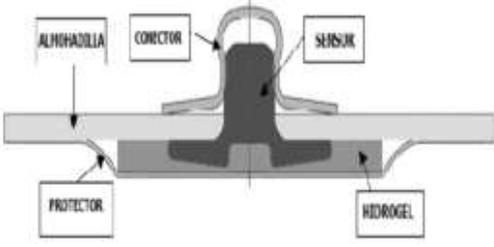


Anexo 4. Hoja técnica y de seguridad de electrodos BioProtech

	FICHA TÉCNICA Y DE SEGURIDAD DE PRODUCTO TERMINADO	
	ELECTRODOS	
VIGENCIA: 2013-12-10	CÓDIGO: FT-GC-19	VERSIÓN: 02

SECCIÓN N°1: Identificación del producto y la compañía	
Nombre Comercial del Producto	Electrodos de uso médico BIO PROTECH, electrodos desechables.
Marca	BIO PROTECH
Presentaciones Comerciales	Bolsa por 3,5 y 50 unidades en caja por 60, 100, 600, 10.000. Caja de embalaje por 600, 1000, 6000 y 10.000 unidades.
Titular del Registro Sanitario	IMPOMEDICAS DE COLOMBIA LTDA
Registro Sanitario	INVIMA 2007DM-0001158
Vigencia del Registro Sanitario	2017-11-30
Clasificación según el INVIMA	Dispositivo médico riesgo I
Código ATC:	No aplica por ser dispositivo médico
Dirección LIFE CARE SOLUTIONS SAS	Autopista Medellín Kilómetro 2.5, 900 Mts Entrada Parcelas, Centro Empresarial OIKOS CIEM, bodega 45, Cota, Cundinamarca.
Fabricante	BIO PROTECH INC. Corea del Sur
Teléfono Colombia	+ 57 (1) 8773929 – 8776417 Ext.101
Fax	+57 (1) 8776409
Web	www.lifecaresolutions.com.co
Contactos vía e-mail	Calidad calidad@lifecaresolutions.com.co Logística Johannes.j@lifecaresolutions.com.co Atención al cliente recepcion@lifecaresolutions.com.co
SECCIÓN N°2: Especificaciones del producto	
Especificaciones de contenido	Envase individual no estéril de bolsa de polietileno, empacada en grupos de 10, 20, 50, 100, 200, 300 y 400 unidades.
Uso	Para la monitorización del ECG para recibir tratamiento médico. Con componentes de muy alta calidad

	FICHA TÉCNICA Y DE SEGURIDAD DE PRODUCTO TERMINADO	
	ELECTRODOS	
VIGENCIA: 2013-12-10	CÓDIGO: FT-GC-19	VERSIÓN: 02

<p>aseguran una señal fiable para monitorización cardiaca. El cloruro de plata (Ag / AgCl) elemento sensor ofrece la mejor sensibilidad, el hidrogel tiene muy bajo nivel de impedancia y adhesivo sólido tanto en la piel de paciente y el elemento de detección de manera que el efecto de movimiento se puede reducir significativamente. La adhesividad de la espuma es lo suficientemente fuerte para usar en pruebas de esfuerzo y. Además de eso no genera residuos, es hipoalergénico.</p>	
Frecuencia de uso	No reutilizable
Carga microbiológica	No estéril
<p>Especificaciones de diseño</p> 	<p>① Hidrogel Excelente conductividad y adhesión para detectar pequeña corriente entre la piel y el electrodo</p> <p>② Sensor Elaborado con Ag / AgCl. Por lo tanto, reduce el ruido por impacto y tiene excelente conductividad. Se puede transmitir la señal eléctrica en la superficie del cuerpo.</p> <p>③ Conector Excelente conductividad, compuesto en carbono y acero inoxidable.</p> <p>④ Almohadilla La materia prima de la almohadilla es una espuma de Polietileno, paño, cinta clara y microporos. Por lo que se adhiere adecuadamente en la piel debido a la gran flexibilidad</p> <p>⑤ Protector Un lado del forro está recubierto de silicona. Por lo que hace a la almohadilla fácilmente separable del mismo.</p>
Tiempo de vida útil	3 años bajo condiciones recomendadas de almacenamiento 72 Horas de monitoreo prolongado
<p>Referencias Electrodo ECG, Modelos para adultos y pediátrico TELE: 715, 716, 718 ,815 ,816 ,818 ,816C ,818 ,816C</p>	

	FICHA TÉCNICA Y DE SEGURIDAD DE PRODUCTO TERMINADO	
	ELECTRODOS	
VIGENCIA: 2013-12-10	CÓDIGO: FT-GC-19	VERSIÓN: 02

,726, 825,915, 916, 716C, 717, 717C, 716B, 732, 832, 718C ,717S,728S.

SECCIÓN N°3: Composición, Información sobre los componentes

Broche metálico: Niquel-plata
Censor: Plata, cloruro de plata
Almohadilla: Espuma P.E. y adhesivo
Revestimiento: Película transparente de poliestireno y sílica
Pegatina: Papel, adhesivo
Gel: Hidráulico y fibra tejida, para mayor adherencia y reducción de interferencias

SECCIÓN N°4: Identificación de los riesgos.

· **Riesgo a la salud:** Puede causar heridas accidentales, una vez esté conectado a la fuente de energía eléctrica puede causar quemaduras.

· **Riesgos Generales:** Heridas involuntarias.

Precauciones: Corrosión cuando se expone a sustancias químicas o alcalinas. No debe exponerse a las siguientes sustancias químicas durante más de CUATRO HORAS: Cloruro de aluminio, cloruro de bario, bicloruro de mercurio, cloruro de calcio, ácido carbólico, sosa clorinada, ácido cítrico, solución de Darwin, cloruro ferroso, lisol, cloruro de mercurio, sales de mercurio, fenol, permanganato potasio, tiocianato potásico, hipoclorito sódico, agua destilada, cloruro estaño, ácido tartárico y las siguientes sustancias químicas no deben ser empleadas nunca con estos productos: hipoclorito de sodio, agua regia, agua destilada, cloruro férrico, ácido sulfúrico, ácido clorhídrico y yodo.

SECCIÓN N°5: Medidas de Primeros Auxilios.

· **Inhalación:** No relevante por ser sólido pero los humos producidos en incendio requieren ventilación y oxigenación.

· **Contacto con la piel:** No hay riesgo bajo condiciones normales de uso.

· **Ingestión:** El producto es atóxico pero puede causar obstrucción del tracto digestivo en caso de ingestión accidental de sus partes, lo cual implicaría intervención médica.

· **Contacto con los ojos:** El producto bajo condiciones normales de uso y almacenamiento no produce volátiles o sustancias que afecten los ojos.

	FICHA TÉCNICA Y DE SEGURIDAD DE PRODUCTO TERMINADO	
	ELECTRODOS	
VIGENCIA: 2013-12-10	CÓDIGO: FT-GC-19	VERSIÓN: 02

SECCIÓN N°6: Medidas para lucha contra el fuego.

· **Riesgos específicos:** Generación de humos y sustancias volatilizadas tóxicas.

· **Método específico a emplear:** Se puede emplear indistintamente agua atomizada, espuma, polvo seco, dióxido de carbono o arena para sofocar a combustión. Se recomienda utilizar agua atomizada en vez de agua a chorro, usar agua atomizada o nebulizada para enfriar los envases expuestos al fuego.

· **Protección en caso de incendio:** No se debe entrar en la zona del incendio sin el equipo protector adecuado, incluyendo protección respiratoria. Tomar las precauciones habituales en caso de incendio químico. Evite que el agua (sobrante) de extinción del fuego afecte al entorno.

· **Equipos de protección personal para el combate del fuego:** Extintores de agua atomizada, espuma, polvo seco o dióxido de carbono. Ropa usual para control de incendios (guantes de carnaza, botas de cuero con punta metálica y saco y pantalón para protección de incendios), máscara para vapores orgánicos y gafas herméticas.

SECCIÓN N°7: Medidas para controlar derrames o fugas.

· **Medidas de emergencia a tomar si hay derrame del producto:** No aplica por ser un sólido.

· **Equipo de protección personal para atacar la emergencia:** No aplica por ser un sólido.

SECCIÓN N°8: Manipulación y Almacenamiento.

· **Recomendaciones sobre manipulación segura:** Seguir la instrucciones de uso, usar el equipo fuente de energía eléctrica bajo condiciones establecidas en el protocolo de funcionamiento y desinfectar adecuadamente después del uso.

· **Condiciones de almacenamiento:** El producto se puede almacenar a temperaturas entre -0°C a 60°C y humedades relativas de 0% a 90%. Evitar la exposición a fuentes de calor y contacto con solventes orgánicos. No almacenar al aire libre ni a la luz solar.

SECCIÓN N°9: Control de Exposición/Protección Especial.

· **Medidas de higiene general:** El producto debe ser manipulado bajo condiciones asépticas usuales de manejo de dispositivos médicos.

· **Equipo de protección personal:** No requiere un equipo de protección para su manipulación.

· **Protección Respiratoria:** No requiere un equipo de protección respiratoria para su manipulación.

	FICHA TÉCNICA Y DE SEGURIDAD DE PRODUCTO TERMINADO	
	ELECTRODOS	
VIGENCIA: 2013-12-10	CÓDIGO: FT-GC-19	VERSIÓN: 02

- **Protección de manos:** No requiere protección especial para las manos.
- **Protección ocular:** No requiere de protección ocular para su manipulación.
- **Protección de la piel y cuerpo:** No se requiere de protección especial.

SECCIÓN N°10: Propiedades Físicas y Químicas.

- **Estado Físico:** Sólido
- **Apariencia y olor:** metal recubierto con cable, inodoro
- **Punto de inflamabilidad:** No inflamable pero pueden arder las partes plásticas y de papel con dificultad al contacto con llama directa.

SECCIÓN N°11: Estabilidad y reactividad

- **Estabilidad:** Estable bajo condiciones usuales de almacenamiento.
- **Productos de descomposición peligrosos:** Por descomposición térmica produce gases orgánicos.
- **Condiciones a evitar:** Contacto con fuentes de calor mayores a 70°C y/o llama directa
- **Sustancias a evitar:** Oxidantes como el hipoclorito de sodio o el permanganato de potasio, el agua puede ayudar en la oxidación, los solventes orgánicos tales como acetona, éter etílico o metiletil cetona, soluciones de fenol pueden ser dañar las partes plásticas.
- **Reactividad:** Mediana a soluciones acuosas y oxidantes que entren en contacto por periodos de varias horas.

SECCIÓN N°12: Información Toxicológica.

- **El ensayo de irritación de la piel (humana):** No aplica
- **Experiencia en el hombre:** Uso común en electrocirugía.
- **Condiciones médicas generalmente agravadas por la exposición:** No hay efectos que evidencien riesgos en tanto no se encuentre conectados a la fuente de energía eléctrica.

	FICHA TÉCNICA Y DE SEGURIDAD DE PRODUCTO TERMINADO	
	ELECTRODOS	
VIGENCIA: 2013-12-10	CÓDIGO: FT-GC-19	VERSIÓN: 02

SECCIÓN N°13: Información Ecológica.
Efectos sobre el medio ambiente:

No hay efectos adversos considerables por exposición al agua o al ambiente. Los productos de degradación térmica pueden ser gases tóxicos como el monóxido de carbono que pueden deteriorar la fauna y flora circundante en condiciones de contacto masivo con el ambiente.

SECCIÓN N°14: Consideraciones sobre Disposición Final.

El producto puede tratarse como un sólido no peligroso. La técnica de incineración puede aplicarse para la disposición final.

SECCIÓN N°15: Información sobre Transporte.

El producto debe ser transportado en vehículos que cuenten con higiene suficiente para transportar medicamentos y dispositivos médicos. No se debe transportar el producto en caso de riesgo de contacto con solventes orgánicos o fuentes de calor.

SECCIÓN N°16: Información Reglamentaria

- Importación y comercialización: Decreto 4725 de 2005 del Ministerio de Protección Social de Colombia.
- Almacenamiento y acondicionamiento: Resolución 4002 de 2007 del Ministerio de Protección Social de Colombia.
- ISO 13485 :2000 Norma Internacional aplicada al sistema de calidad de la fabricación la calidad de los Dispositivos médicos

SECCIÓN N°17: Información Adicional.


T715

T716

T716B

	FICHA TÉCNICA Y DE SEGURIDAD DE PRODUCTO TERMINADO	
	ELECTRODOS	
VIGENCIA: 2013-12-10	CÓDIGO: FT-GC-19	VERSIÓN: 02

  		
<u>T716C</u> <u>T717</u> <u>T718</u>		
  		
<u>T718C</u> <u>T815</u> <u>T816</u>		
  		
<u>T816C</u> <u>T832</u> <u>T832C</u>		
ELABORADO POR	Laura Franco, Gestor de Calidad	2013-12-10
REVISADO POR	Edgar Calderón, Director Técnico	2013-12-10

Anexo 5. Código de programación en Python para el desarrollo del sistema de clasificación de movimientos de una mano

```
In [ ]: import pandas as pd
        from google.colab import drive
        import os

        # Montar Google Drive
        drive.mount('/content/drive')

        # Especifica la ruta de la carpeta que contiene los archivos CSV en Google Drive
        ruta_drive = '/content/drive/My Drive/TESIS/DATOS_rev3'

        # Verificar si la carpeta existe
        if os.path.exists(ruta_drive):
            # Obtener la lista de archivos CSV en la carpeta
            archivos_csv = [archivo for archivo in os.listdir(ruta_drive) if archivo.endswith('.csv')]

            if archivos_csv:
                # Iterar sobre los archivos CSV
                for i, archivo_csv in enumerate(archivos_csv, 1):
                    try:
                        # Leer el archivo CSV en un DataFrame
                        df = pd.read_csv(os.path.join(ruta_drive, archivo_csv))

                        # Almacenar el DataFrame con el nombre df{i}
                        nombre_df = f'df{i}'
                        globals()[nombre_df] = df

                        # Mostrar el DataFrame
                        print(f'DataFrame {nombre_df}:')
                        display(df)
                    except Exception as e:
                        print(f'Error leyendo {archivo_csv}: {e}')
            else:
                print('No se encontraron archivos CSV en la carpeta especificada.')
        else:
            print(f'La ruta especificada no existe: {ruta_drive}')

Mounted at /content/drive
DataFrame df1:
```

	时间	通道1均值	通道1采集值	通道1功率值	通道1肌肉强度	通道2均值	通道2采集值	通道2功率值	通道2肌肉强度
0	3:57:10	1618.0	1665.0	92.0	0.0	1586.0	1696.0	82.0	0.0
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	3:57:10	1618.0	1746.0	99.0	0.0	1586.0	1690.0	88.0	0.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	3:57:10	1618.0	1746.0	99.0	0.0	1586.0	1690.0	88.0	0.0
...
607825	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
607826	4:02:14	1618.0	1660.0	139.0	0.0	1586.0	1618.0	176.0	8.0
607827	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
607828	4:02:14	1618.0	1767.0	134.0	0.0	1586.0	1739.0	176.0	8.0
607829	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

607830 rows × 9 columns

DataFrame df2:

	时间	通道1均值	通道1采集值	通道1功率值	通道1肌肉强度	通道2均值	通道2采集值	通道2功率值	通道2肌肉强度
0	3:38:05	1583.0	1682.0	148.0	0.0	1400.0	1545.0	133.0	1.0
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	3:38:05	1583.0	1682.0	148.0	0.0	1400.0	1545.0	133.0	1.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	3:38:05	1583.0	1701.0	153.0	0.0	1400.0	1559.0	130.0	1.0
...
601965	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
601966	3:43:06	1583.0	1812.0	159.0	0.0	1400.0	1559.0	128.0	0.0
601967	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
601968	3:43:06	1583.0	1528.0	151.0	0.0	1400.0	1534.0	128.0	0.0
601969	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

601970 rows × 9 columns

DataFrame df3:

	时间	通道1均值	通道1采集值	通道1功率值	通道1肌肉强度	通道2均值	通道2采集值	通道2功率值	通道2肌肉强度
0	4:02:25	1618.0	1594.0	154.0	0.0	1586.0	1735.0	159.0	5.0
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	4:02:25	1618.0	1594.0	154.0	0.0	1586.0	1735.0	159.0	5.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4:02:25	1618.0	1768.0	151.0	0.0	1586.0	1740.0	156.0	5.0
...
681127	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
681128	4:08:05	1618.0	1839.0	198.0	0.0	1586.0	1875.0	240.0	23.0
681129	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
681130	4:08:05	1618.0	1839.0	198.0	0.0	1586.0	1875.0	240.0	23.0
681131	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

681132 rows x 9 columns

```
In [ ]: import pandas as pd

# Iterar sobre los DataFrames desde df1 en adelante
for i in range(1, len(archivos_csv) + 1):
    nombre_df = f'df{i}'

    # Verificar si el DataFrame existe
    if nombre_df in globals():
        # Eliminar la primera columna (índice 0, time)
        df_modificado = globals()[nombre_df].drop(columns=[globals()[nombre_df].columns[0]])

        # Nombres de las columnas a modificar
        nombres_columnas = ['A0_val', 'A0_Val_flt', 'A0_Pow', 'A0_str_lvl', 'A1_val']

        # Cambiar los nombres de las columnas
        df_modificado.columns = nombres_columnas

        # Almacenar el DataFrame modificado de nuevo en globals
        globals()[nombre_df] = df_modificado

        # Mostrar el DataFrame modificado
        print(f'{nombre_df} modificado:')
        display(globals()[nombre_df])
    else:
        print(f'{nombre_df} no encontrado')
```

df1 modificado:

	A0_val	A0_Val_fit	A0_Pow	A0_str_lvl	A1_val	A1_Val_fit	A1_Pow	A1_str_lvl
0	1618.0	1665.0	92.0	0.0	1586.0	1696.0	82.0	0.0
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1618.0	1746.0	99.0	0.0	1586.0	1690.0	88.0	0.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1618.0	1746.0	99.0	0.0	1586.0	1690.0	88.0	0.0
...
607825	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
607826	1618.0	1660.0	139.0	0.0	1586.0	1618.0	176.0	8.0
607827	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
607828	1618.0	1787.0	134.0	0.0	1586.0	1739.0	176.0	8.0
607829	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

607830 rows × 8 columns

df2 modificado:

	A0_val	A0_Val_fit	A0_Pow	A0_str_lvl	A1_val	A1_Val_fit	A1_Pow	A1_str_lvl
0	1583.0	1682.0	148.0	0.0	1400.0	1545.0	133.0	1.0
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1583.0	1682.0	148.0	0.0	1400.0	1545.0	133.0	1.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1583.0	1701.0	153.0	0.0	1400.0	1559.0	130.0	1.0
...
601965	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
601966	1583.0	1812.0	159.0	0.0	1400.0	1559.0	128.0	0.0
601967	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
601968	1583.0	1528.0	151.0	0.0	1400.0	1534.0	128.0	0.0
601969	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

601970 rows × 8 columns

df3 modificado:

	A0_val	A0_Val_fit	A0_Pow	A0_str_lvl	A1_val	A1_Val_fit	A1_Pow	A1_str_lvl
0	1618.0	1594.0	154.0	0.0	1586.0	1735.0	159.0	5.0
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	1618.0	1594.0	154.0	0.0	1586.0	1735.0	159.0	5.0
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	1618.0	1768.0	151.0	0.0	1586.0	1740.0	156.0	5.0
...
681127	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
681128	1618.0	1839.0	198.0	0.0	1586.0	1875.0	240.0	23.0
681129	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
681130	1618.0	1839.0	198.0	0.0	1586.0	1875.0	240.0	23.0
681131	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

681132 rows × 8 columns

```
In [ ]: import pandas as pd

# Función para crear nuevos DataFrames con columnas específicas
def crear_nuevo_df(df, sensor1, sensor2):
    nuevo_df = pd.concat([df[sensor1], df[sensor2]], axis=1)
    nuevo_df.columns = ['sensor 1', 'sensor 2']
    nuevo_df.dropna(how='any', inplace=True)
    return nuevo_df

# Lista de DataFrames originales
dfs_originales = [df1, df2, df3]

# Lista para almacenar los nuevos DataFrames
nuevos_dfs = []

# Crear nuevos DataFrames a partir de los DataFrames originales
for i, df in enumerate(dfs_originales, start=1):
    try:
        nuevo_df = crear_nuevo_df(df, 'A0_Pow', 'A1_Pow')
        nuevos_dfs.append(nuevo_df)
        print(f"Nuevo DataFrame new_df{i}:")
        display(nuevo_df)
    except KeyError as e:
        print(f"Error: {e} en DataFrame df{i}")
    except Exception as e:
        print(f"Ocurrió un error con df{i}: {e}")

# Almacenar los nuevos DataFrames en variables específicas
new_df1, new_df2, new_df3 = nuevos_dfs

Nuevo DataFrame new_df1:
```

	sensor 1	sensor 2
0	92.0	82.0
2	99.0	88.0
4	99.0	88.0
6	96.0	94.0
8	96.0	94.0
...
607820	140.0	171.0
607822	140.0	171.0
607824	139.0	176.0
607826	139.0	176.0
607828	134.0	176.0

303915 rows × 2 columns

Nuevo DataFrame new_df2:

	sensor 1	sensor 2
0	148.0	133.0
2	148.0	133.0
4	153.0	130.0
6	153.0	130.0
8	153.0	130.0
...
601960	143.0	130.0
601962	143.0	130.0
601964	159.0	128.0
601966	159.0	128.0
601968	151.0	128.0

300985 rows × 2 columns

Nuevo DataFrame new_df3:

	sensor 1	sensor 2
0	154.0	159.0
2	154.0	159.0
4	151.0	156.0
6	151.0	156.0
8	151.0	156.0
...
681122	207.0	246.0
681124	203.0	246.0
681126	203.0	246.0
681128	198.0	240.0
681130	198.0	240.0

340566 rows × 2 columns

```
In [ ]: #PREPROCESAMIENTO DE DATOS: mano extendida
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Definir la función del filtro pasa alto
def highpass_filter(data, cutoff_freq, fs, order=5):
    nyquist_freq = 0.5 * fs
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    filtered_data = filtfilt(b, a, data, axis=0)
    return filtered_data

# Definir la función del filtro pasa bajo
def lowpass_filter(data, cutoff_freq, fs, order=5):
    nyquist_freq = 0.5 * fs
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_data = filtfilt(b, a, data, axis=0)
    return filtered_data

# Frecuencia de muestreo de las señales en Hz
fs = 1000

# Frecuencia de corte del filtro pasa alto y pasa bajo
cutoff_freq_high = 5 # Frecuencia de corte del filtro pasa alto en Hz
cutoff_freq_low = 450 # Frecuencia de corte del filtro pasa bajo en Hz

# Aplicar el filtro pasa alto a las señales
filtered_df1_high = highpass_filter(new_df1.values, cutoff_freq_high, fs)

# Aplicar el filtro pasa bajo a las señales
filtered_df1 = lowpass_filter(filtered_df1_high, cutoff_freq_low, fs)

# Convertir el array filtrado a DataFrame
filtered_df1 = pd.DataFrame(filtered_df1, columns=new_df1.columns)

# Graficar las columnas del DataFrame original y del DataFrame filtrado
```

```
plt.figure(figsize=(12, 8))

# Graficar las columnas del DataFrame original
plt.subplot(2, 2, 1)
plt.plot(new_dfi.iloc[:, 0], color='blue')
plt.title('Original - Sensor 1')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(new_dfi.iloc[:, 1], color='orange')
plt.title('Original - Sensor 2')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

# Graficar las columnas del DataFrame filtrado
plt.subplot(2, 2, 3)
plt.plot(filtered_dfi.iloc[:, 0], color='blue')
plt.title('Filtrado - Sensor 1')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

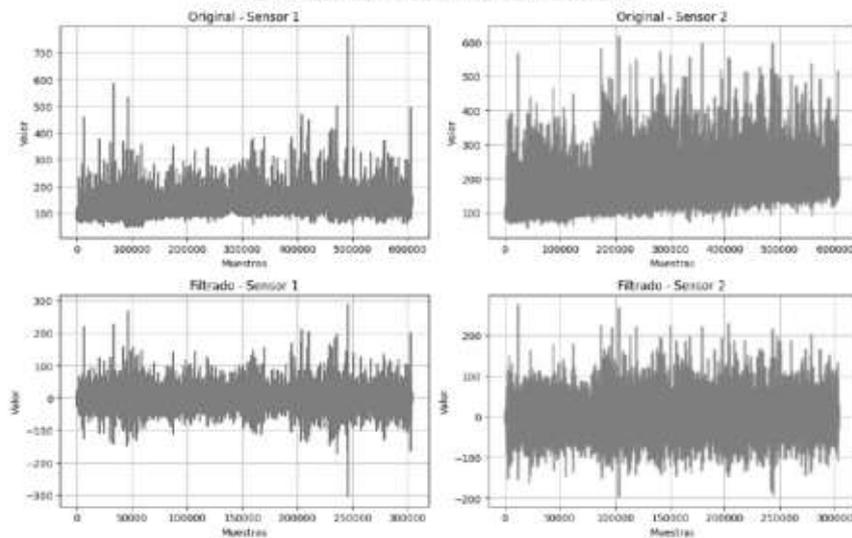
plt.subplot(2, 2, 4)
plt.plot(filtered_dfi.iloc[:, 1], color='orange')
plt.title('Filtrado - Sensor 2')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

# Agregar un título general para todas las subgráficas
plt.suptitle('PREPROCESAMIENTO DE DATOS: Mano extendida', fontsize=16)

plt.tight_layout(rect=[0, 0, 1, 0.96])

plt.tight_layout()
plt.show()
```

PREPROCESAMIENTO DE DATOS: Mano extendida



```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Definir la función del filtro pasa alto
def highpass_filter(data, cutoff_freq, fs, order=5):
    nyquist_freq = 0.5 * fs
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    filtered_data = filtfilt(b, a, data, axis=0)
    return filtered_data

# Definir la función del filtro pasa bajo
def lowpass_filter(data, cutoff_freq, fs, order=5):
    nyquist_freq = 0.5 * fs
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_data = filtfilt(b, a, data, axis=0)
    return filtered_data

# Frecuencia de muestreo de las señales en Hz
fs = 1000

# Frecuencia de corte del filtro pasa alto y pasa bajo
cutoff_freq_high = 5 # Frecuencia de corte del filtro pasa alto en Hz
cutoff_freq_low = 450 # Frecuencia de corte del filtro pasa bajo en Hz

# Aplicar el filtro pasa alto a las señales
filtered_df2_high = highpass_filter(new_df2.values, cutoff_freq_high, fs)

# Aplicar el filtro pasa bajo a las señales
filtered_df2 = lowpass_filter(filtered_df2_high, cutoff_freq_low, fs)

# Convertir el array filtrado a DataFrame
filtered_df2 = pd.DataFrame(filtered_df2, columns=new_df2.columns)

# Graficar las columnas del DataFrame original y del DataFrame filtrado
```

```
plt.figure(figsize=(12, 8))

# Graficar las columnas del DataFrame original
plt.subplot(2, 2, 1)
plt.plot(new_df2.iloc[:, 0], color='blue')
plt.title('Original - Sensor 1')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(new_df2.iloc[:, 1], color='orange')
plt.title('Original - Sensor 2')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

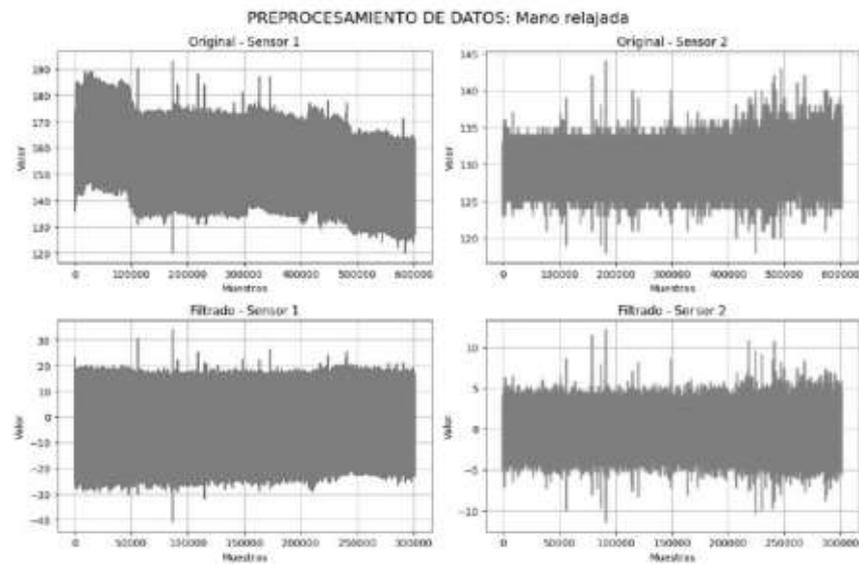
# Graficar las columnas del DataFrame filtrado
plt.subplot(2, 2, 3)
plt.plot(filtered_df2.iloc[:, 0], color='blue')
plt.title('Filtrado - Sensor 1')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

plt.subplot(2, 2, 4)
plt.plot(filtered_df2.iloc[:, 1], color='orange')
plt.title('Filtrado - Sensor 2')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

# Agregar un título general para todas las subgráficas
plt.suptitle('PREPROCESAMIENTO DE DATOS: Mano relajada', fontsize=16)

plt.tight_layout(rect=[0, 0, 1, 0.96])

plt.tight_layout()
plt.show()
```



```
In [ ]: #PREPROCESAMIENTO DE DATOS: mano cerrada
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Definir La función del filtro pasa alto
def highpass_filter(data, cutoff_freq, fs, order=5):
    nyquist_freq = 0.5 * fs
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    filtered_data = filtfilt(b, a, data, axis=0)
    return filtered_data

# Definir La función del filtro pasa bajo
def lowpass_filter(data, cutoff_freq, fs, order=5):
    nyquist_freq = 0.5 * fs
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_data = filtfilt(b, a, data, axis=0)
    return filtered_data

# Frecuencia de muestreo de Las señales en Hz
fs = 1000

# Frecuencia de corte del filtro pasa alto y pasa bajo
cutoff_freq_high = 5 # Frecuencia de corte del filtro pasa alto en Hz
cutoff_freq_low = 450 # Frecuencia de corte del filtro pasa bajo en Hz

# Aplicar el filtro pasa alto a Las señales
filtered_df3_high = highpass_filter(new_df3.values, cutoff_freq_high, fs)

# Aplicar el filtro pasa bajo a Las señales
filtered_df3 = lowpass_filter(filtered_df3_high, cutoff_freq_low, fs)

# Convertir el array filtrado a DataFrame
filtered_df3 = pd.DataFrame(filtered_df3, columns=new_df3.columns)
```

```
# Graficar las columnas del DataFrame original y del DataFrame filtrado
plt.figure(figsize=(12, 8))

# Graficar las columnas del DataFrame original
plt.subplot(2, 2, 1)
plt.plot(new_df3.iloc[:, 0], color='blue')
plt.title('Original - Sensor 1')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

plt.subplot(2, 2, 2)
plt.plot(new_df3.iloc[:, 1], color='orange')
plt.title('Original - Sensor 2')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

# Graficar las columnas del DataFrame filtrado
plt.subplot(2, 2, 3)
plt.plot(filtered_df3.iloc[:, 0], color='blue')
plt.title('Filtrado - Sensor 1')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

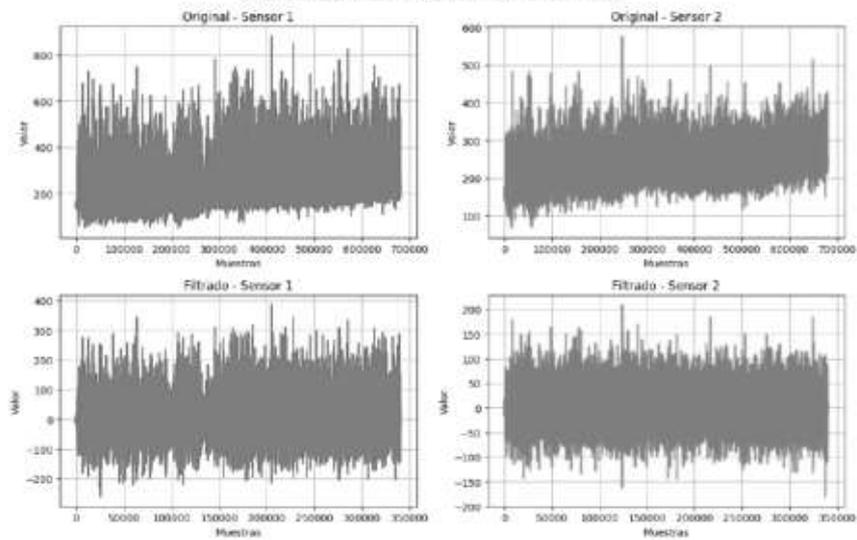
plt.subplot(2, 2, 4)
plt.plot(filtered_df3.iloc[:, 1], color='orange')
plt.title('Filtrado - Sensor 2')
plt.xlabel('Muestras')
plt.ylabel('Valor')
plt.grid(True)

# Agregar un título general para todas las subgráficas
plt.suptitle('PREPROCESAMIENTO DE DATOS: Mano cerrada', fontsize=16)

plt.tight_layout(rect=[0, 0, 1, 0.96])

plt.tight_layout()
plt.show()
```

PREPROCESAMIENTO DE DATOS: Mano cerrada



```
In [ ]: # Verificación de tipo de datos
print("Tipo de datos de filtered_df1:", type(filtered_df1))
print("Tipo de datos de filtered_df2:", type(filtered_df2))
print("Tipo de datos de filtered_df3:", type(filtered_df3))

Tipo de datos de filtered_df1: <class 'pandas.core.frame.DataFrame'>
Tipo de datos de filtered_df2: <class 'pandas.core.frame.DataFrame'>
Tipo de datos de filtered_df3: <class 'pandas.core.frame.DataFrame'>
```

```
In [ ]: import pandas as pd

# Etiquetado de DataFrame DE MANO EXTENDIDA
filtered_df1['out1'] = 1
filtered_df1['out2'] = 0
filtered_df1['out3'] = 0

# Etiquetado de DataFrame DE MANO RELAJADA
filtered_df2['out1'] = 0
filtered_df2['out2'] = 1
filtered_df2['out3'] = 0

# Etiquetado de DataFrame DE MANO CERRADA
filtered_df3['out1'] = 0
filtered_df3['out2'] = 0
filtered_df3['out3'] = 1

# Mostrar Los DataFrames FILTRADOS con ETIQUETAS
print("\nDataFrame filtrado con etiquetas (Mano Extendida):")
display(filtered_df1)

print("\nDataFrame filtrado con etiquetas (Mano Relajada):")
display(filtered_df2)

print("\nDataFrame filtrado con etiquetas (Mano Cerrada):")
display(filtered_df3)
```

DataFrame filtrado con etiquetas (Mano Extendida):

	sensor 1	sensor 2	out1	out2	out3
0	0.916130	-3.859158	1	0	0
1	7.994076	1.857525	1	0	0
2	7.311368	1.944258	1	0	0
3	4.874483	7.382758	1	0	0
4	3.713857	7.725765	1	0	0
...
303910	1.692748	-1.971121	1	0	0
303911	2.628138	-4.366235	1	0	0
303912	2.169546	1.447422	1	0	0
303913	3.056173	-0.034239	1	0	0
303914	-1.290507	-0.332775	1	0	0

303915 rows × 5 columns

DataFrame filtrado con etiquetas(Mano Relajada):

	sensor 1	sensor 2	out1	out2	out3
0	1.806134	-0.129768	0	1	0
1	1.481090	-0.030895	0	1	0
2	6.412430	-2.894440	0	1	0
3	5.752259	-2.831530	0	1	0
4	6.019568	-2.678491	0	1	0
...
300980	-9.829168	2.154011	0	1	0
300981	-8.958978	2.141346	0	1	0
300982	6.196406	0.271119	0	1	0
300983	6.739532	0.288574	0	1	0
300984	-1.580486	0.380721	0	1	0

300985 rows × 5 columns

DataFrame filtrado con etiquetas (Mano Cerrada):

	sensor 1	sensor 2	out1	out2	out3
0	0.286998	-1.764070	0	0	1
1	0.480636	-1.704337	0	0	1
2	-2.664096	-4.633446	0	0	1
3	-2.308406	-4.434016	0	0	1
4	-2.571269	-4.540736	0	0	1
...
340561	3.467864	1.900470	0	0	1
340562	0.014644	2.535753	0	0	1
340563	0.835810	2.823626	0	0	1
340564	-3.665264	-2.687651	0	0	1
340565	-2.725554	-2.078509	0	0	1

340566 rows × 5 columns

```
In [ ]: import pandas as pd

# Unir los DataFrames filtered_df1, filtered_df2 y filtered_df3
df_target = pd.concat([filtered_df1, filtered_df2, filtered_df3], ignore_index=True)

# Mostrar el DataFrame resultante
display(df_target)
```

	sensor 1	sensor 2	out1	out2	out3
0	0.916130	-3.859158	1	0	0
1	7.994076	1.857525	1	0	0
2	7.311368	1.944258	1	0	0
3	4.874483	7.382758	1	0	0
4	3.713857	7.725765	1	0	0
...
945461	3.467864	1.900470	0	0	1
945462	0.014644	2.535753	0	0	1
945463	0.835810	2.823626	0	0	1
945464	-3.665264	-2.687651	0	0	1
945465	-2.725554	-2.078509	0	0	1

945466 rows × 5 columns

```
In [ ]: #CALCULO DE CARACTERIZACION DE SEÑALES UTILIZANDO RMS
import pandas as pd
import numpy as np

# Vector de valores RMS
lista_rms = []
```

```

# Definir el número de muestras por grupo
muestras_por_grupo = 256

# Calcular la cantidad de grupos
num_grupos = len(df_target) // muestras_por_grupo

# Crear un nuevo DataFrame para almacenar los valores RMS de cada sensor
rms_df = pd.DataFrame()

# Iterar sobre cada grupo de muestras
for i in range(num_grupos):
    # Obtener el índice de inicio y fin del rango de muestras para el grupo actual
    inicio = i * muestras_por_grupo
    fin = (i + 1) * muestras_por_grupo

    # Seleccionar las muestras (índice) del grupo actual para cada sensor
    muestras_sensor1 = df_target.iloc[inicio:fin, 0] # Primera columna del DataFrame
    muestras_sensor2 = df_target.iloc[inicio:fin, 1] # Segunda columna del DataFrame

    # Calcular el valor RMS para cada sensor en el grupo actual
    rms_sensor1 = np.sqrt(np.mean(muestras_sensor1 ** 2))
    rms_sensor2 = np.sqrt(np.mean(muestras_sensor2 ** 2))

    lista_rms.append({'RMS_sensor1': rms_sensor1, 'RMS_sensor2': rms_sensor2})

# Convertir la lista de valores RMS a un DataFrame
rms_df = pd.DataFrame(lista_rms)

# Mostrar el nuevo DataFrame con los valores RMS de cada sensor
display(rms_df)

```

	RMS_sensor1	RMS_sensor2
0	8.769987	6.590129
1	8.243974	7.201933
2	7.058098	6.446942
3	8.064815	8.628547
4	6.885153	6.703171
...
3688	60.549096	23.690152
3689	56.551862	33.558910
3690	68.642486	38.525359
3691	28.177800	16.445084
3692	11.654748	10.378648

3693 rows × 2 columns

```

In [ ]: #CALCULO DE CARACTERIZACION DE SEÑALES UTILIZANDO MAV
import pandas as pd
import numpy as np

# Definir el tamaño de la ventana
ventana = 256

```

```

# Inicializar listas para almacenar los resultados
mav_sensor1 = []
mav_sensor2 = []

# Iterar sobre las señales en ventanas de 256 valores
for i in range(0, len(df_target), ventana):
    ventana_actual = df_target.iloc[i:i+ventana]

    # Calcular MAV solo si la ventana tiene el tamaño completo
    if len(ventana_actual) == ventana:
        mav_sensor1.append(np.mean(np.abs(ventana_actual['sensor 1'])))
        mav_sensor2.append(np.mean(np.abs(ventana_actual['sensor 2'])))
    else:
        # Si la ventana no está completa se ignora
        break

# Crear un nuevo DataFrame con los valores MAV de cada sensor
df_mav = pd.DataFrame({'MAV_sensor1': mav_sensor1, 'MAV_sensor2': mav_sensor2})

# Mostrar el DataFrame resultante
display(df_mav)

```

	MAV_sensor1	MAV_sensor2
0	7.061932	5.318291
1	6.632129	5.999914
2	5.504084	5.384108
3	6.569393	6.850616
4	5.218653	5.469662
...
3688	40.676255	20.068986
3689	47.886241	27.435115
3690	56.499539	30.122944
3691	19.869349	13.975936
3692	9.928198	8.451789

3693 rows × 2 columns

```

In [ ]: #CALCULO DE CARACTERIZACION DE SEÑALES UTILIZANDO MNF
import numpy as np
import pandas as pd
from scipy.signal import welch

# Parámetros de la señal
fs = 1000 # Frecuencia de muestreo en Hz

signal1 = df_target['sensor 1']
signal2 = df_target['sensor 2']

# Crear un DataFrame temporal
df_temp = pd.DataFrame({'Sensor1': signal1, 'Sensor2': signal2})

# Función para calcular la MNF de una señal
def calcular_MNF(signal, fs):
    freqs, power = welch(signal, fs)

```

```

total_power = np.sum(power)
if total_power == 0:
    mnf = np.nan # Asignar NaN si la suma de potencia es cero
else:
    mnf = np.sum(freqs * power) / total_power
return mnf

# Función para dividir una señal en segmentos y calcular la MNF para cada segmento
def calcular_MNF_por_segmento(signal, fs, segment_size):
    mnf_values = []
    num_segments = len(signal) // segment_size
    for i in range(num_segments):
        start_idx = i * segment_size
        end_idx = start_idx + segment_size
        segment = signal[start_idx:end_idx]
        mnf = calcular_MNF(segment, fs)
        mnf_values.append(mnf)
    return mnf_values

segment_size = 256 # Tamaño del segmento

# Calcular la MNF para cada segmento de cada columna del DataFrame
mnf_values_sensor1 = calcular_MNF_por_segmento(df_temp['Sensor1'], fs, segment_size)
mnf_values_sensor2 = calcular_MNF_por_segmento(df_temp['Sensor2'], fs, segment_size)

# Crear un nuevo DataFrame con los valores de MNF
df_mnf = pd.DataFrame({'MNF_sensor1': mnf_values_sensor1, 'MNF_sensor2': mnf_values_sensor2})

# Mostrar el nuevo DataFrame
display(df_mnf)

```

	MNF_sensor1	MNF_sensor2
0	72.910665	35.178880
1	87.056492	38.884882
2	90.547839	37.661604
3	59.239248	23.170731
4	85.364899	38.756986
...	-	-
3688	8.262027	21.326963
3689	16.696923	14.168895
3690	10.443172	31.134657
3691	12.342299	24.181880
3692	19.941335	38.257630

3693 rows × 2 columns

```

In [ ]: #CALCULO DE CARACTERIZACION DE SEÑALES UTILIZANDO CRUCES POR CERO
import pandas as pd
import numpy as np

# Definir una función para calcular los cruces por cero
def cruces_por_cero(serie):
    return np.sum(np.diff(np.sign(serie)) != 0)

```

```

# Definir el tamaño de la ventana
ventana = 256

# Inicializar listas para almacenar los resultados
cz_sensor1 = []
cz_sensor2 = []

# Iterar sobre las señales en ventanas de 256 valores
for i in range(0, len(df_target), ventana):
    ventana_actual = df_target.iloc[i:i+ventana]

    # Calcular cruces por cero solo si la ventana tiene el tamaño completo
    if len(ventana_actual) == ventana:
        cz_sensor1.append(cruces_por_cero(ventana_actual['sensor 1']))
        cz_sensor2.append(cruces_por_cero(ventana_actual['sensor 2']))
    else:
        # Si la ventana no está completa se ignora
        break

# Crear un nueva DataFrame con los valores de cruces por cero de cada sensor
df_cz = pd.DataFrame({'CZ_sensor1': cz_sensor1, 'CZ_sensor2': cz_sensor2})

# Mostrar el DataFrame resultante
display(df_cz)

```

	CZ_sensor1	CZ_sensor2
0	37	25
1	40	20
2	44	21
3	40	21
4	43	21
...
3688	14	11
3689	7	5
3690	7	20
3691	7	9
3692	10	18

3693 rows x 2 columns

```

In [ ]: #crea un dataframe que contenga los dataframes rms_df, df_mav, df_mnf, df_cz
new_df_rms_mnf = pd.concat([rms_df, df_mav, df_mnf, df_cz], axis=1)
display(new_df_rms_mnf)

```

	RMS_sensor1	RMS_sensor2	MAV_sensor1	MAV_sensor2	MNF_sensor1	MNF_sensor2	CZ_s
0	8.769987	6.590129	7.061932	5.318291	72.910665	35.178880	
1	8.243974	7.201933	6.632129	5.999914	87.056492	38.884882	
2	7.058098	6.446942	5.504084	5.384108	90.547839	37.661604	
3	8.064815	8.628547	6.569393	6.850616	59.239248	23.170731	
4	6.885153	6.703171	5.218653	5.469662	85.364899	38.756986	
...
3688	60.549096	23.690152	40.676255	20.068986	8.262027	21.326963	
3689	56.551862	33.558910	47.886241	27.435115	16.696923	14.168895	
3690	68.642486	38.525359	56.499539	30.122944	10.443172	31.134657	
3691	28.177800	16.445084	19.869349	13.975936	12.342299	24.181880	
3692	11.654748	10.378648	9.928198	8.451789	19.941335	38.257630	

3693 rows × 8 columns

```
In [ ]: #ETIQUETADO DE DATAFRAMES DE CARACTERISTICAS EXTRAIDAS
import pandas as pd

# Segmentos a analizar
grupo_size = 256

# Crear una lista para almacenar los promedios calculados
promedios = []

# Iterar sobre los datos en grupos de tamaño grupo_size
for i in range(0, len(df_target), grupo_size):
    grupo = df_target.iloc[i:i + grupo_size]

    # Calcular el promedio para cada grupo solo si tiene exactamente 256 filas
    if len(grupo) == grupo_size:
        # Se toma las columnas de etiquetas del dataframe
        promedio_grupo = grupo.iloc[:, 2:5].mean().round().tolist()
        promedios.append(promedio_grupo)

# Crear un nuevo DataFrame con los promedios calculados
if promedios:
    outs_df = pd.DataFrame(promedios, columns=df_target.columns[2:5])
    display(outs_df)
else:
    print("No hay grupos completos de tamaño", grupo_size)
```

	out1	out2	out3
0	1.0	0.0	0.0
1	1.0	0.0	0.0
2	1.0	0.0	0.0
3	1.0	0.0	0.0
4	1.0	0.0	0.0
...
3688	0.0	0.0	1.0
3689	0.0	0.0	1.0
3690	0.0	0.0	1.0
3691	0.0	0.0	1.0
3692	0.0	0.0	1.0

3693 rows × 3 columns

```
In [ ]: #crea un dataframe que contenga al dataframe new_df_rms_mnf, outs_df
final_df = pd.concat([new_df_rms_mnf, outs_df], axis=1)
final_df = final_df.dropna()
display(final_df)
```

	RMS_sensor1	RMS_sensor2	MAV_sensor1	MAV_sensor2	MNF_sensor1	MNF_sensor2	CZ_s
0	8.769987	6.590129	7.061932	5.318291	72.910665	35.178880	
1	8.243974	7.201933	6.632129	5.999914	87.056492	38.884882	
2	7.058098	6.446942	5.504084	5.384108	90.547839	37.661604	
3	8.064815	8.628547	6.569393	6.850616	59.239248	23.170731	
4	6.885153	6.703171	5.218653	5.469662	85.364899	38.756986	
...
3688	60.549096	23.690152	40.676255	20.068986	8.262027	21.326963	
3689	56.551862	33.558910	47.886241	27.435115	16.696923	14.168895	
3690	68.642486	38.525359	56.499539	30.122944	10.443172	31.134657	
3691	28.177800	16.445084	19.869349	13.975936	12.342299	24.181880	
3692	11.654748	10.378648	9.928198	8.451789	19.941335	38.257630	

3693 rows × 11 columns

```
In [ ]: #NORMALIZACION DE DATOS
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# Inicializar el objeto MinMaxScaler
scaler = MinMaxScaler()
```

```
# Normalizar los datos del DataFrame
df_normalized = pd.DataFrame(scaler.fit_transform(final_df), columns=final_df.columns)

# Mostrar el DataFrame normalizado
display(df_normalized)
```

	RMS_sensor1	RMS_sensor2	MAV_sensor1	MAV_sensor2	MNF_sensor1	MNF_sensor2	CZ_sensor1	CZ_sensor2
0	0.016014	0.047933	0.016807	0.054213	0.506052	0.297000	0.444914	0.397145
1	0.012115	0.053630	0.013165	0.063119	0.611369	0.336169	0.444914	0.397145
2	0.003324	0.046599	0.003605	0.055073	0.637362	0.323240	0.444914	0.397145
3	0.010787	0.066916	0.012633	0.074233	0.404267	0.170084	0.444914	0.397145
4	0.002042	0.048985	0.001186	0.056191	0.598775	0.334817	0.444914	0.397145
...
3688	0.399845	0.207180	0.301676	0.246932	0.024738	0.150597	0.100000	0.100000
3689	0.370214	0.299085	0.362778	0.343171	0.087537	0.074943	0.100000	0.100000
3690	0.459840	0.345336	0.435772	0.378288	0.040977	0.254256	0.100000	0.100000
3691	0.159882	0.139709	0.125345	0.167326	0.055116	0.180771	0.100000	0.100000
3692	0.037399	0.083214	0.041098	0.095153	0.111691	0.329539	0.100000	0.100000

3693 rows x 11 columns

```
In [ ]: # Calcula la matriz de correlación
correlation_matrix = df_normalized.corr()

# Muestra la matriz de correlación
print("Matriz de Correlación:")
display(correlation_matrix)
```

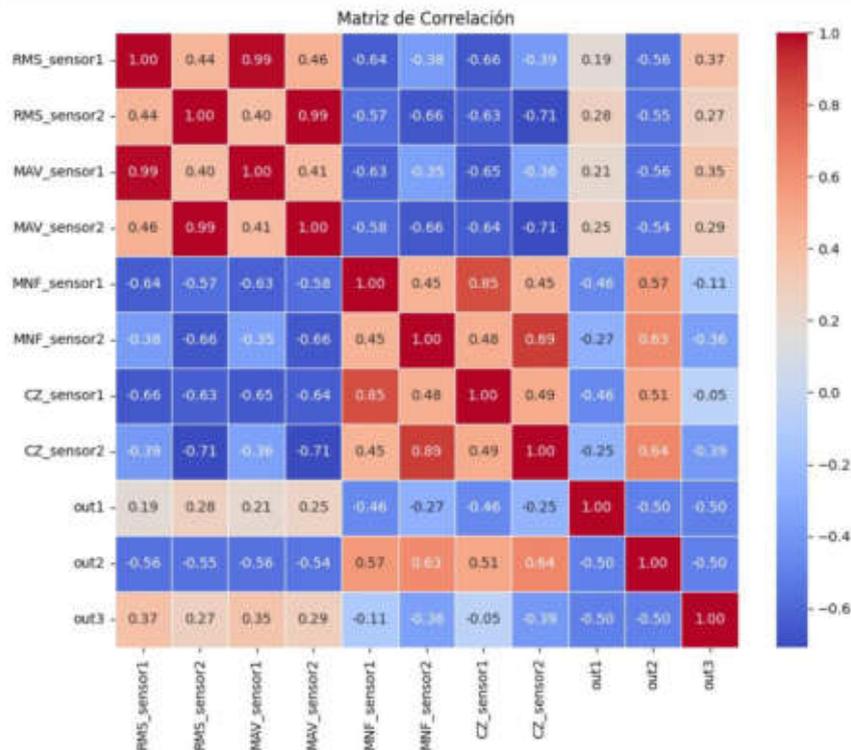
Matriz de Correlación:

	RMS_sensor1	RMS_sensor2	MAV_sensor1	MAV_sensor2	MNF_sensor1	MNF_sensor2	CZ_sensor1	CZ_sensor2
RMS_sensor1	1.000000	0.444914	0.990859	0.459081	-0.638601	-0.377938	-0.662847	-0.387077
RMS_sensor2	0.444914	1.000000	0.397145	0.994982	-0.571584	-0.664396	-0.633180	-0.707326
MAV_sensor1	0.990859	0.397145	1.000000	0.409925	-0.627392	-0.350862	-0.651747	-0.355560
MAV_sensor2	0.459081	0.994982	0.409925	1.000000	-0.575468	-0.659467	-0.641633	-0.710176
MNF_sensor1	-0.638601	-0.571584	-0.627392	-0.575468	1.000000	0.451321	0.846085	0.445937
MNF_sensor2	-0.377938	-0.664396	-0.350862	-0.659467	0.451321	1.000000	0.846085	0.445937
CZ_sensor1	-0.662847	-0.633180	-0.651747	-0.641633	0.846085	0.476111	1.000000	0.890261
CZ_sensor2	-0.387077	-0.707326	-0.355560	-0.710176	0.445937	0.890261	0.890261	1.000000
out1	0.185157	0.276786	0.209587	0.254293	-0.456910	-0.271877	0.185157	0.276786
out2	-0.559747	-0.551038	-0.561778	-0.543367	0.570468	0.629071	-0.559747	-0.551038
out3	0.374590	0.274251	0.352191	0.289074	-0.113557	-0.357191	0.374590	0.274251

```
In [ ]: #grafico de matriz de correlacion
import seaborn as sns
import matplotlib.pyplot as plt

# Calcular la matriz de correlación
correlation_matrix = df_normalized.corr()

# Crear el mapa de calor
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=
plt.title('Matriz de Correlación')
plt.show()
```



```
In [ ]: #Guardar datos del preprocesamiento (seguimiento interno)
import pandas as pd

# nombre del archivo de Excel
nombre_archivo = 'data_procesada_normalizada_rev3.xlsx'

# Guarda el DataFrame en un archivo Excel
df_normalized.to_excel(nombre_archivo, index=False)

# el archivo se ha guardado correctamente
print(f"El DataFrame se ha guardado en '{nombre_archivo}' correctamente.")

El DataFrame se ha guardado en 'data_procesada_normalizada_rev3.xlsx' correctamente.
```

```
In [ ]: #GRAFICA DE DATASET
import matplotlib.pyplot as plt
```

```
import numpy as np

# Crear una lista de colores
colors = plt.cm.viridis(np.linspace(0, 1, len(df_normalized.columns)))

# Número de columnas
num_columns = len(df_normalized.columns)

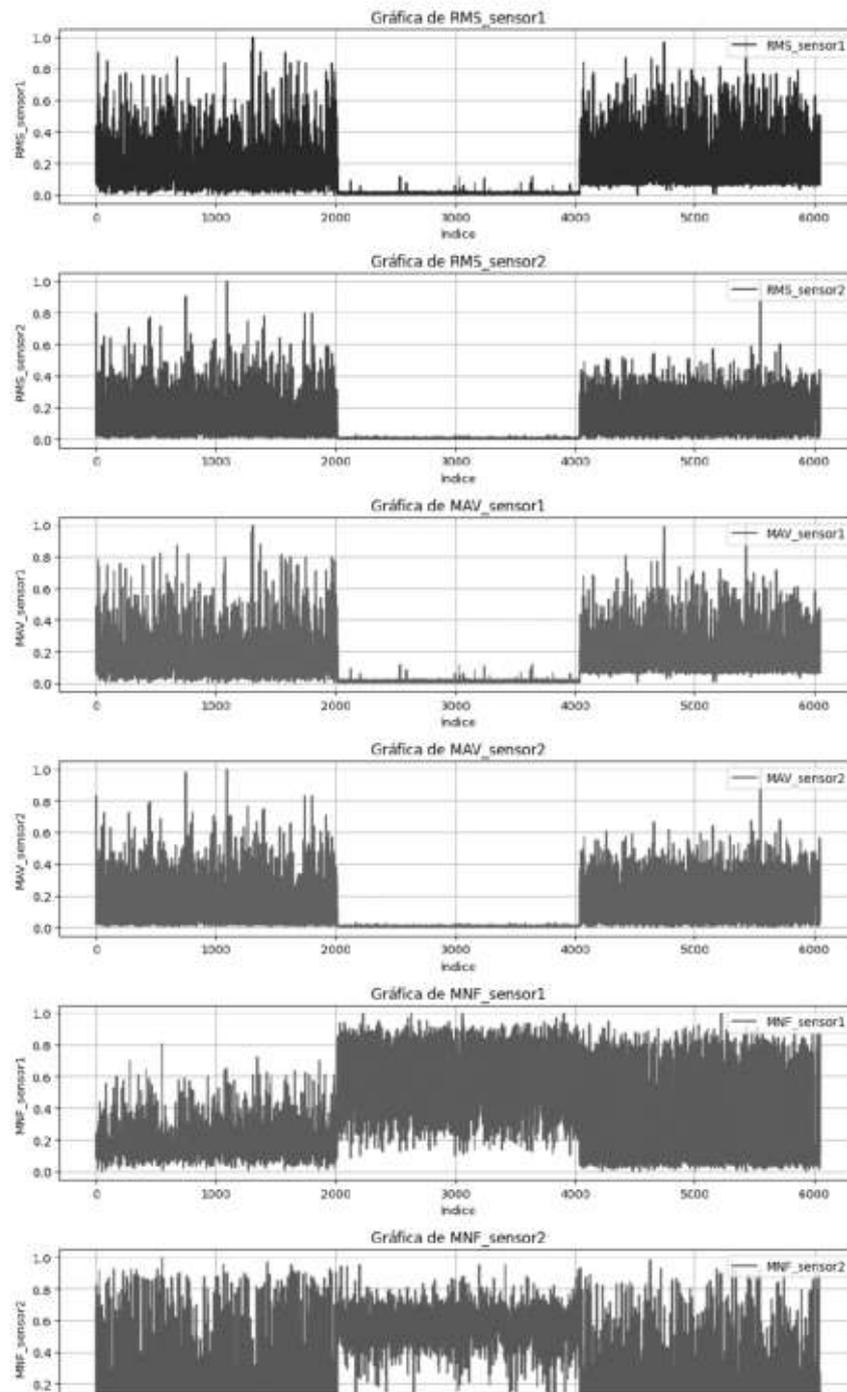
# Crear una figura con subplots para cada columna
fig, axes = plt.subplots(nrows=num_columns, ncols=1, figsize=(10, num_columns * 3))

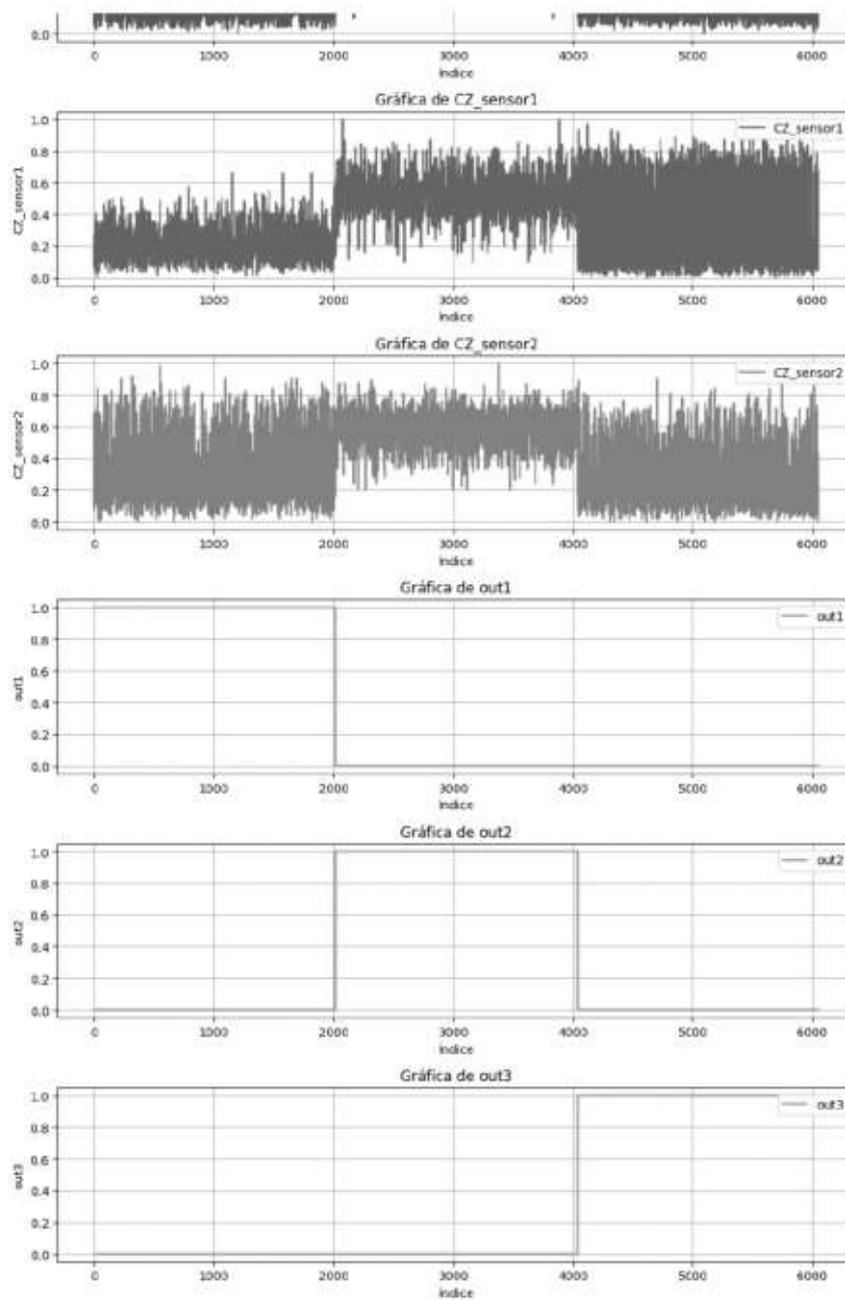
# Iterar sobre cada columna del DataFrame y graficarla
for i, (column, color) in enumerate(zip(df_normalized.columns, colors)):
    axes[i].plot(df_normalized[column], label=column, color=color)
    axes[i].set_title(f'Gráfica de {column}')
    axes[i].set_xlabel('Índice')
    axes[i].set_ylabel(column)
    axes[i].grid(True)
    axes[i].legend(loc='upper right')

# Agregar un título general para toda la figura
fig.suptitle('Gráficas de cada columna del DataFrame Normalizado', fontsize=16)

# Ajustar automáticamente el diseño para evitar solapamientos y dejar espacio para
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Gráficas de cada columna del DataFrame Normalizado





```
In [ ]: #Seguimiento interno
print(df_normalized.columns)
Index(['RMS_sensor1', 'RMS_sensor2', 'MAV_sensor1', 'MAV_sensor2',
       'MNF_sensor1', 'MNF_sensor2', 'CZ_sensor1', 'CZ_sensor2', 'out1',
       'out2', 'out3'],
      dtype='object')
```

```

In [ ]: #Se retiran los datos de RMS_sensor1, RMS_sensor2, CZ_sensor1, CZ_sensor2 del dataset
df_normalized.drop(['RMS_sensor1', 'RMS_sensor2', 'CZ_sensor1', 'CZ_sensor2'], axis=0)
print(df_normalized.columns)

Index(['MAV_sensor1', 'MAV_sensor2', 'MNF_sensor1', 'MNF_sensor2', 'out1',
       'out2', 'out3'],
      dtype='object')

In [ ]: #MODELO SECUENCIAL FINAL CON DROPOUT
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.utils import plot_model
from sklearn.metrics import confusion_matrix
import seaborn as sns
from IPython.display import Image

# Verificar la estructura de los datos de entrada
if df_normalized.shape[1] != 7:
    raise ValueError("El DataFrame df_normalized no tiene la estructura esperada (7 columnas)")

# Verificar la presencia de valores nulos en los datos de entrada
if df_normalized.isnull().any().any():
    raise ValueError("El DataFrame df_normalized contiene valores nulos")

# Dividir los datos en características (X) y etiquetas (y)
X = df_normalized.iloc[:, :4] # Características (primeras 4 columnas)
y = df_normalized.iloc[:, 4:] # Etiquetas (últimas 3 columnas)

# Dividir los datos en conjuntos de entrenamiento, validación y prueba
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

# Crear el modelo de red neuronal secuencial
model_secuencial = Sequential()
model_secuencial.add(Dense(8, input_dim=4, activation='relu'))
model_secuencial.add(Dropout(0.1)) # Añadir dropout con tasa del 10%
model_secuencial.add(Dense(8, activation='relu'))
model_secuencial.add(Dropout(0.1)) # Añadir dropout con tasa del 10%
model_secuencial.add(Dense(3, activation='sigmoid'))

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

# Entrenar el modelo con datos de entrenamiento y validar con datos de validación
history = model_secuencial.fit(X_train, y_train, validation_data=(X_val, y_val), epochs=100)

# Evaluar el modelo con datos de prueba
loss, accuracy = model_secuencial.evaluate(X_test, y_test)
print(f'Pérdida en datos de prueba: {loss}')
print(f'Precisión en datos de prueba: {accuracy}')

# Graficar pérdida durante el entrenamiento
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdidas durante el entrenamiento')
plt.xlabel('Época')
plt.ylabel('Pérdidas %')
plt.legend()
plt.grid(True)
plt.show()

```

```

# Graficar precisión durante el entrenamiento
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Curva de aprendizaje')
plt.xlabel('Época')
plt.ylabel('Precisión %')
plt.legend()
plt.grid(True)
plt.show()

# Realizar predicciones con datos de prueba
y_pred = model_secuencial.predict(X_test)
y_pred_round = y_pred.round()

# Obtener la matriz de confusión
cm = confusion_matrix(y_test.values.argmax(axis=1), y_pred_round.argmax(axis=1))

# Normalizar la matriz de confusión
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

# Graficar la matriz de confusión normalizada
plt.figure(figsize=(8, 6))

sns.heatmap(cm_normalized, annot=True, cmap='Blues', fmt='.2f', xticklabels=['Exter
plt.xlabel('Clases Predichas')
plt.ylabel('Clases Reales')
plt.title('Matriz de confusión normalizada')
plt.show()

# Visualizar el modelo
plot_model(model_secuencial, to_file='model_plot_SEC.png', show_shapes=True, show_l
Image(filename='model_plot_SEC.png')
# Guardar el modelo
model_secuencial.save('modelo_secuencial_v3.h5')

```

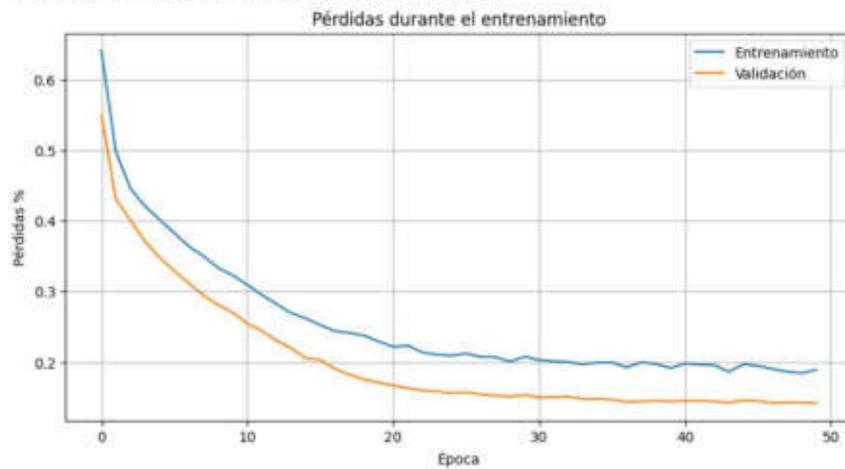
```
Epoch 1/50
485/485 [=====] - 2s 3ms/step - loss: 0.6403 - accuracy:
0.4313 - val_loss: 0.5493 - val_accuracy: 0.5752
Epoch 2/50
485/485 [=====] - 1s 2ms/step - loss: 0.4973 - accuracy:
0.6221 - val_loss: 0.4305 - val_accuracy: 0.7471
Epoch 3/50
485/485 [=====] - 1s 2ms/step - loss: 0.4453 - accuracy:
0.6719 - val_loss: 0.4000 - val_accuracy: 0.7256
Epoch 4/50
485/485 [=====] - 1s 2ms/step - loss: 0.4204 - accuracy:
0.7008 - val_loss: 0.3706 - val_accuracy: 0.7669
Epoch 5/50
485/485 [=====] - 1s 2ms/step - loss: 0.4013 - accuracy:
0.7117 - val_loss: 0.3475 - val_accuracy: 0.7884
Epoch 6/50
485/485 [=====] - 1s 3ms/step - loss: 0.3824 - accuracy:
0.7279 - val_loss: 0.3289 - val_accuracy: 0.7950
Epoch 7/50
485/485 [=====] - 2s 3ms/step - loss: 0.3636 - accuracy:
0.7440 - val_loss: 0.3114 - val_accuracy: 0.8116
Epoch 8/50
485/485 [=====] - 2s 3ms/step - loss: 0.3498 - accuracy:
0.7475 - val_loss: 0.2942 - val_accuracy: 0.8182
Epoch 9/50
485/485 [=====] - 1s 3ms/step - loss: 0.3333 - accuracy:
0.7679 - val_loss: 0.2807 - val_accuracy: 0.8413
Epoch 10/50
485/485 [=====] - 1s 2ms/step - loss: 0.3227 - accuracy:
0.7789 - val_loss: 0.2700 - val_accuracy: 0.8579
Epoch 11/50
485/485 [=====] - 1s 2ms/step - loss: 0.3093 - accuracy:
0.7948 - val_loss: 0.2546 - val_accuracy: 0.8579
Epoch 12/50
485/485 [=====] - 1s 2ms/step - loss: 0.2950 - accuracy:
0.7985 - val_loss: 0.2440 - val_accuracy: 0.8694
Epoch 13/50
485/485 [=====] - 1s 2ms/step - loss: 0.2821 - accuracy:
0.8129 - val_loss: 0.2305 - val_accuracy: 0.8793
Epoch 14/50
485/485 [=====] - 1s 2ms/step - loss: 0.2694 - accuracy:
0.8332 - val_loss: 0.2199 - val_accuracy: 0.8876
Epoch 15/50
485/485 [=====] - 1s 2ms/step - loss: 0.2619 - accuracy:
0.8389 - val_loss: 0.2052 - val_accuracy: 0.8909
Epoch 16/50
485/485 [=====] - 1s 2ms/step - loss: 0.2523 - accuracy:
0.8420 - val_loss: 0.2030 - val_accuracy: 0.8992
Epoch 17/50
485/485 [=====] - 1s 2ms/step - loss: 0.2439 - accuracy:
0.8472 - val_loss: 0.1904 - val_accuracy: 0.9074
Epoch 18/50
485/485 [=====] - 1s 2ms/step - loss: 0.2413 - accuracy:
0.8530 - val_loss: 0.1823 - val_accuracy: 0.9107
Epoch 19/50
485/485 [=====] - 1s 3ms/step - loss: 0.2375 - accuracy:
0.8524 - val_loss: 0.1754 - val_accuracy: 0.9107
Epoch 20/50
485/485 [=====] - 2s 3ms/step - loss: 0.2292 - accuracy:
0.8629 - val_loss: 0.1710 - val_accuracy: 0.9207
Epoch 21/50
485/485 [=====] - 2s 3ms/step - loss: 0.2216 - accuracy:
0.8662 - val_loss: 0.1666 - val_accuracy: 0.9174
Epoch 22/50
```

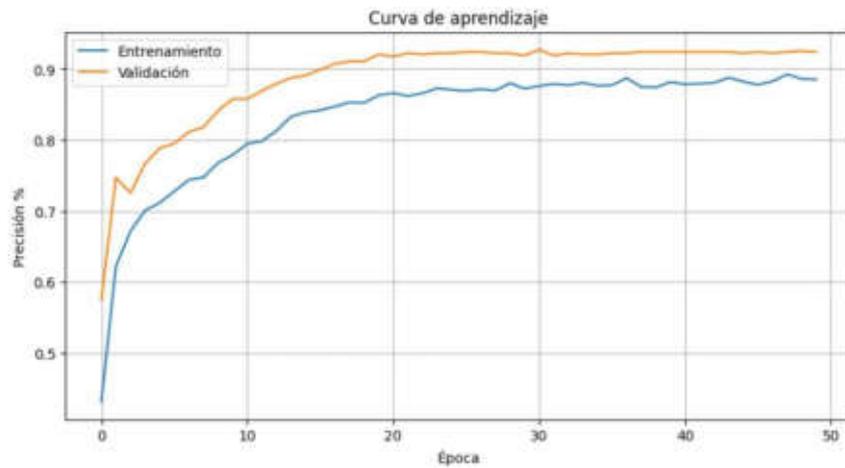
```
485/485 [=====] - 1s 3ms/step - loss: 0.2234 - accuracy:
0.8621 - val_loss: 0.1630 - val_accuracy: 0.9223
Epoch 23/50
485/485 [=====] - 1s 2ms/step - loss: 0.2134 - accuracy:
0.8658 - val_loss: 0.1598 - val_accuracy: 0.9207
Epoch 24/50
485/485 [=====] - 1s 2ms/step - loss: 0.2107 - accuracy:
0.8728 - val_loss: 0.1586 - val_accuracy: 0.9223
Epoch 25/50
485/485 [=====] - 1s 2ms/step - loss: 0.2089 - accuracy:
0.8709 - val_loss: 0.1558 - val_accuracy: 0.9223
Epoch 26/50
485/485 [=====] - 1s 2ms/step - loss: 0.2122 - accuracy:
0.8693 - val_loss: 0.1572 - val_accuracy: 0.9240
Epoch 27/50
485/485 [=====] - 1s 2ms/step - loss: 0.2073 - accuracy:
0.8718 - val_loss: 0.1540 - val_accuracy: 0.9240
Epoch 28/50
485/485 [=====] - 1s 2ms/step - loss: 0.2071 - accuracy:
0.8697 - val_loss: 0.1523 - val_accuracy: 0.9223
Epoch 29/50
485/485 [=====] - 1s 2ms/step - loss: 0.2006 - accuracy:
0.8800 - val_loss: 0.1512 - val_accuracy: 0.9223
Epoch 30/50
485/485 [=====] - 1s 2ms/step - loss: 0.2077 - accuracy:
0.8724 - val_loss: 0.1532 - val_accuracy: 0.9190
Epoch 31/50
485/485 [=====] - 1s 2ms/step - loss: 0.2027 - accuracy:
0.8761 - val_loss: 0.1501 - val_accuracy: 0.9273
Epoch 32/50
485/485 [=====] - 1s 3ms/step - loss: 0.2011 - accuracy:
0.8790 - val_loss: 0.1504 - val_accuracy: 0.9190
Epoch 33/50
485/485 [=====] - 2s 3ms/step - loss: 0.1999 - accuracy:
0.8771 - val_loss: 0.1510 - val_accuracy: 0.9223
Epoch 34/50
485/485 [=====] - 2s 3ms/step - loss: 0.1968 - accuracy:
0.8807 - val_loss: 0.1474 - val_accuracy: 0.9207
Epoch 35/50
485/485 [=====] - 1s 2ms/step - loss: 0.1992 - accuracy:
0.8765 - val_loss: 0.1482 - val_accuracy: 0.9207
Epoch 36/50
485/485 [=====] - 1s 2ms/step - loss: 0.1992 - accuracy:
0.8773 - val_loss: 0.1467 - val_accuracy: 0.9223
Epoch 37/50
485/485 [=====] - 1s 2ms/step - loss: 0.1924 - accuracy:
0.8873 - val_loss: 0.1437 - val_accuracy: 0.9223
Epoch 38/50
485/485 [=====] - 1s 2ms/step - loss: 0.1997 - accuracy:
0.8749 - val_loss: 0.1442 - val_accuracy: 0.9240
Epoch 39/50
485/485 [=====] - 1s 2ms/step - loss: 0.1973 - accuracy:
0.8743 - val_loss: 0.1450 - val_accuracy: 0.9240
Epoch 40/50
485/485 [=====] - 1s 2ms/step - loss: 0.1914 - accuracy:
0.8817 - val_loss: 0.1441 - val_accuracy: 0.9240
Epoch 41/50
485/485 [=====] - 1s 2ms/step - loss: 0.1980 - accuracy:
0.8786 - val_loss: 0.1449 - val_accuracy: 0.9240
Epoch 42/50
485/485 [=====] - 1s 2ms/step - loss: 0.1962 - accuracy:
0.8794 - val_loss: 0.1451 - val_accuracy: 0.9240
Epoch 43/50
485/485 [=====] - 1s 2ms/step - loss: 0.1959 - accuracy:
```

```

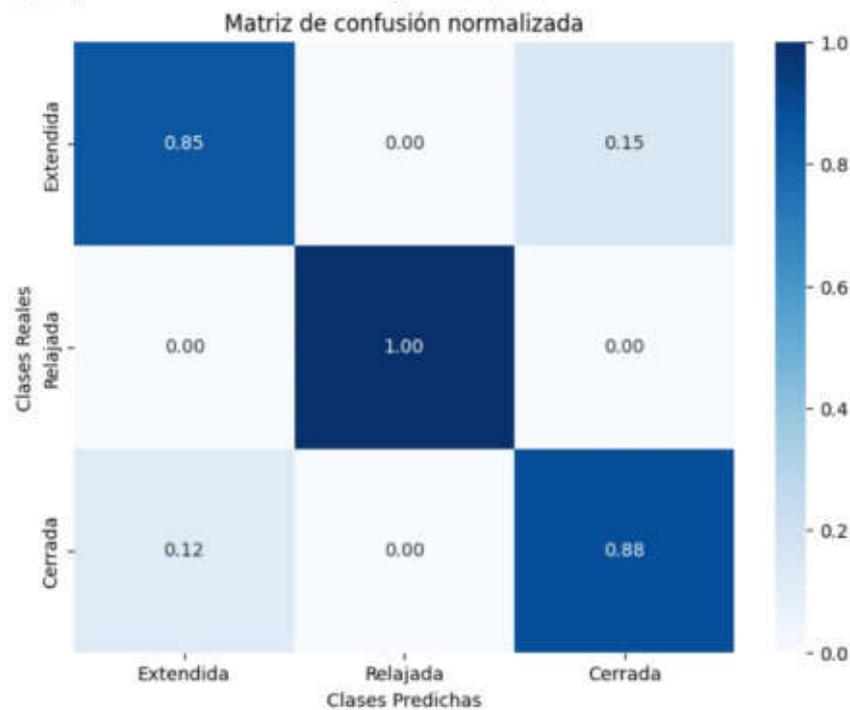
0.8804 - val_loss: 0.1443 - val_accuracy: 0.9240
Epoch 44/50
485/485 [=====] - 1s 2ms/step - loss: 0.1862 - accuracy:
0.8879 - val_loss: 0.1423 - val_accuracy: 0.9240
Epoch 45/50
485/485 [=====] - 2s 3ms/step - loss: 0.1969 - accuracy:
0.8823 - val_loss: 0.1458 - val_accuracy: 0.9223
Epoch 46/50
485/485 [=====] - 2s 3ms/step - loss: 0.1946 - accuracy:
0.8780 - val_loss: 0.1449 - val_accuracy: 0.9240
Epoch 47/50
485/485 [=====] - 1s 3ms/step - loss: 0.1901 - accuracy:
0.8827 - val_loss: 0.1421 - val_accuracy: 0.9223
Epoch 48/50
485/485 [=====] - 1s 2ms/step - loss: 0.1864 - accuracy:
0.8926 - val_loss: 0.1427 - val_accuracy: 0.9240
Epoch 49/50
485/485 [=====] - 1s 2ms/step - loss: 0.1843 - accuracy:
0.8860 - val_loss: 0.1426 - val_accuracy: 0.9256
Epoch 50/50
485/485 [=====] - 1s 2ms/step - loss: 0.1891 - accuracy:
0.8854 - val_loss: 0.1420 - val_accuracy: 0.9240
19/19 [=====] - 0s 2ms/step - loss: 0.1553 - accuracy: 0.
9092
Pérdida en datos de prueba: 0.15526123344898224
Precisión en datos de prueba: 0.9092409014701843

```





19/19 [=====] - 0s 1ms/step



```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
In [ ]: #MODELO RECURRENTE FINAL CON DROPOUT
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.utils import plot_model
from sklearn.metrics import confusion_matrix
import seaborn as sns
from IPython.display import Image

# Define la longitud de la ventana de contexto
window_size = 2

# Crear listas para almacenar las secuencias de características y etiquetas
X_sequences = []
y_sequences = []

# Recorrer los datos y crear secuencias de características y etiquetas
for i in range(len(df_normalized) - window_size):
    X_sequences.append(df_normalized.iloc[i:i+window_size, :4].values) # Secuencia
    y_sequences.append(df_normalized.iloc[i+window_size, 4:].values) # Siguiente

# Convertir las listas en arreglos numpy
X_RNR_window = np.array(X_sequences)
y_RNR_window = np.array(y_sequences)

# Dividir los datos en conjuntos de entrenamiento, validación y prueba
X_train_rnr, X_temp_rnr, y_train_rnr, y_temp_rnr = train_test_split(X_RNR_window, y_RNR_window,
                                                                    test_size=0.2, random_state=42)
X_val_rnr, X_test_rnr, y_val_rnr, y_test_rnr = train_test_split(X_temp_rnr, y_temp_rnr,
                                                                test_size=0.5, random_state=42)

# Crear el modelo de red neuronal secuencial
model_rnr = Sequential()

# Agregar una capa LSTM con 4 unidades y activación relu
model_rnr.add(LSTM(4, input_shape=(window_size, 4), activation='relu'))
model_rnr.add(Dropout(0.1)) # Apaga el 50% de las neuronas aleatoriamente durante

# Agregar capas densas con activación relu y dropout
model_rnr.add(Dense(8, activation='relu'))
model_rnr.add(Dropout(0.1))

model_rnr.add(Dense(8, activation='relu'))
model_rnr.add(Dropout(0.1))

# Agregar la capa de salida con 3 neuronas y función de activación sigmoideal para 1
model_rnr.add(Dense(3, activation='sigmoid'))

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

# Entrenar el modelo con datos de entrenamiento y validar con datos de validación
history = model_rnr.fit(X_train_rnr, y_train_rnr, validation_data=(X_val_rnr, y_val_rnr),
                       epochs=100)

# Evaluar el modelo con datos de prueba
loss, accuracy = model_rnr.evaluate(X_test_rnr, y_test_rnr)
print(f'Pérdida en datos de prueba: {loss}')
print(f'Precisión en datos de prueba: {accuracy}')

# Graficar pérdida durante el entrenamiento
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')
plt.title('Pérdida durante el entrenamiento')
plt.xlabel('Época')
plt.ylabel('Pérdida %')
plt.legend()
plt.grid(True)

```

```

plt.show()

# Graficar precisión durante el entrenamiento
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Curva de aprendizaje')
plt.xlabel('Época')
plt.ylabel('Precisión %')
plt.legend()
plt.grid(True)
plt.show()

# Obtener las predicciones del modelo para los datos de prueba
predicciones = model_rnr.predict(X_test_rnr)
predicciones_binarias = (predicciones > 0.5) # Convertir las probabilidades en valores binarios

# Calcular la matriz de confusión
matriz_confusion = confusion_matrix(y_test_rnr.argmax(axis=1), predicciones_binarias)

# Normalizar la matriz de confusión
matriz_confusion_normalizada = matriz_confusion.astype('float') / matriz_confusion.sum(axis=1)

# Crear una figura y un eje para la gráfica
plt.figure(figsize=(8, 6))

# Visualizar la matriz de confusión normalizada como un heatmap
sns.heatmap(matriz_confusion_normalizada, annot=True, cmap='Blues', fmt='.2f', xticklabels=classes, yticklabels=classes)
plt.xlabel('Clases Predichas')
plt.ylabel('Clases Reales')
plt.title('Matriz de Confusión Normalizada')
plt.show()

# Visualizar el modelo
plot_model(model_rnr, to_file='model_plot_RNN.png', show_shapes=True, show_layer_names=True)
Image(filename='model_plot_RNN.png')
# Guardar el modelo
model_rnr.save('modelo_recurrente_v3.h5')

```

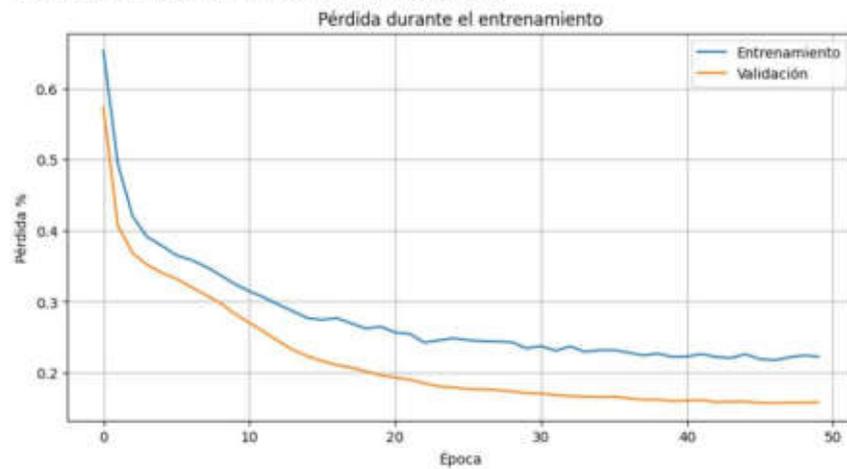
```
Epoch 1/50
152/152 [=====] - 4s 9ms/step - loss: 0.6532 - accuracy:
0.5745 - val_loss: 0.5730 - val_accuracy: 0.6661
Epoch 2/50
152/152 [=====] - 1s 4ms/step - loss: 0.4934 - accuracy:
0.6352 - val_loss: 0.4065 - val_accuracy: 0.7322
Epoch 3/50
152/152 [=====] - 1s 4ms/step - loss: 0.4202 - accuracy:
0.6534 - val_loss: 0.3683 - val_accuracy: 0.7273
Epoch 4/50
152/152 [=====] - 1s 4ms/step - loss: 0.3911 - accuracy:
0.6858 - val_loss: 0.3519 - val_accuracy: 0.7785
Epoch 5/50
152/152 [=====] - 1s 4ms/step - loss: 0.3789 - accuracy:
0.7003 - val_loss: 0.3405 - val_accuracy: 0.8397
Epoch 6/50
152/152 [=====] - 1s 4ms/step - loss: 0.3652 - accuracy:
0.7263 - val_loss: 0.3323 - val_accuracy: 0.7438
Epoch 7/50
152/152 [=====] - 1s 4ms/step - loss: 0.3589 - accuracy:
0.7298 - val_loss: 0.3207 - val_accuracy: 0.8430
Epoch 8/50
152/152 [=====] - 1s 4ms/step - loss: 0.3495 - accuracy:
0.7428 - val_loss: 0.3097 - val_accuracy: 0.8595
Epoch 9/50
152/152 [=====] - 1s 4ms/step - loss: 0.3377 - accuracy:
0.7637 - val_loss: 0.2982 - val_accuracy: 0.8694
Epoch 10/50
152/152 [=====] - 1s 4ms/step - loss: 0.3248 - accuracy:
0.7777 - val_loss: 0.2830 - val_accuracy: 0.8694
Epoch 11/50
152/152 [=====] - 1s 4ms/step - loss: 0.3148 - accuracy:
0.7881 - val_loss: 0.2705 - val_accuracy: 0.8645
Epoch 12/50
152/152 [=====] - 1s 4ms/step - loss: 0.3061 - accuracy:
0.7889 - val_loss: 0.2574 - val_accuracy: 0.8744
Epoch 13/50
152/152 [=====] - 1s 4ms/step - loss: 0.2959 - accuracy:
0.7990 - val_loss: 0.2445 - val_accuracy: 0.8860
Epoch 14/50
152/152 [=====] - 1s 4ms/step - loss: 0.2864 - accuracy:
0.8033 - val_loss: 0.2320 - val_accuracy: 0.8777
Epoch 15/50
152/152 [=====] - 1s 4ms/step - loss: 0.2767 - accuracy:
0.8108 - val_loss: 0.2231 - val_accuracy: 0.8893
Epoch 16/50
152/152 [=====] - 1s 4ms/step - loss: 0.2744 - accuracy:
0.8089 - val_loss: 0.2161 - val_accuracy: 0.8826
Epoch 17/50
152/152 [=====] - 1s 4ms/step - loss: 0.2768 - accuracy:
0.8067 - val_loss: 0.2107 - val_accuracy: 0.8843
Epoch 18/50
152/152 [=====] - 1s 5ms/step - loss: 0.2694 - accuracy:
0.8089 - val_loss: 0.2072 - val_accuracy: 0.8909
Epoch 19/50
152/152 [=====] - 1s 6ms/step - loss: 0.2621 - accuracy:
0.8193 - val_loss: 0.2017 - val_accuracy: 0.8926
Epoch 20/50
152/152 [=====] - 1s 6ms/step - loss: 0.2649 - accuracy:
0.8141 - val_loss: 0.1963 - val_accuracy: 0.8909
Epoch 21/50
152/152 [=====] - 1s 7ms/step - loss: 0.2565 - accuracy:
0.8184 - val_loss: 0.1931 - val_accuracy: 0.8926
Epoch 22/50
```

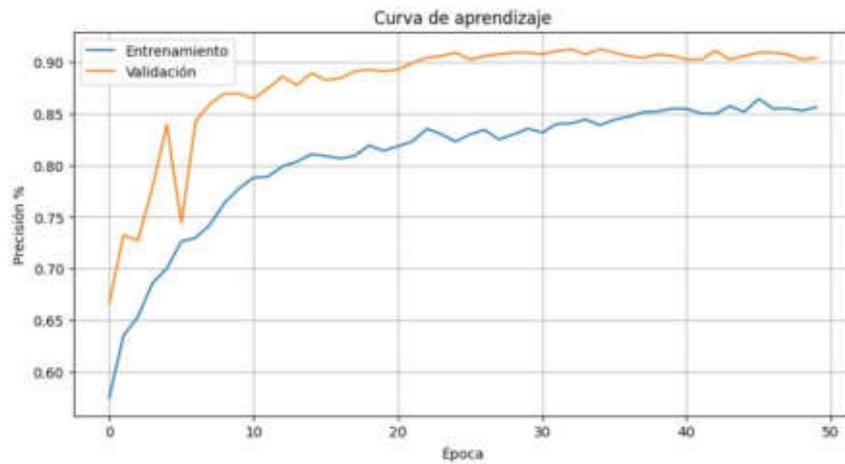
```
152/152 [=====] - 1s 6ms/step - loss: 0.2546 - accuracy:
0.8230 - val_loss: 0.1899 - val_accuracy: 0.8992
Epoch 23/50
152/152 [=====] - 1s 6ms/step - loss: 0.2423 - accuracy:
0.8354 - val_loss: 0.1849 - val_accuracy: 0.9041
Epoch 24/50
152/152 [=====] - 1s 4ms/step - loss: 0.2454 - accuracy:
0.8302 - val_loss: 0.1807 - val_accuracy: 0.9058
Epoch 25/50
152/152 [=====] - 1s 4ms/step - loss: 0.2486 - accuracy:
0.8230 - val_loss: 0.1791 - val_accuracy: 0.9091
Epoch 26/50
152/152 [=====] - 1s 4ms/step - loss: 0.2457 - accuracy:
0.8300 - val_loss: 0.1768 - val_accuracy: 0.9025
Epoch 27/50
152/152 [=====] - 1s 4ms/step - loss: 0.2441 - accuracy:
0.8343 - val_loss: 0.1765 - val_accuracy: 0.9058
Epoch 28/50
152/152 [=====] - 1s 4ms/step - loss: 0.2438 - accuracy:
0.8250 - val_loss: 0.1755 - val_accuracy: 0.9074
Epoch 29/50
152/152 [=====] - 1s 4ms/step - loss: 0.2429 - accuracy:
0.8298 - val_loss: 0.1737 - val_accuracy: 0.9091
Epoch 30/50
152/152 [=====] - 1s 4ms/step - loss: 0.2342 - accuracy:
0.8356 - val_loss: 0.1710 - val_accuracy: 0.9091
Epoch 31/50
152/152 [=====] - 1s 4ms/step - loss: 0.2374 - accuracy:
0.8316 - val_loss: 0.1706 - val_accuracy: 0.9074
Epoch 32/50
152/152 [=====] - 1s 4ms/step - loss: 0.2309 - accuracy:
0.8401 - val_loss: 0.1685 - val_accuracy: 0.9107
Epoch 33/50
152/152 [=====] - 1s 4ms/step - loss: 0.2371 - accuracy:
0.8405 - val_loss: 0.1669 - val_accuracy: 0.9124
Epoch 34/50
152/152 [=====] - 1s 4ms/step - loss: 0.2293 - accuracy:
0.8445 - val_loss: 0.1662 - val_accuracy: 0.9074
Epoch 35/50
152/152 [=====] - 1s 4ms/step - loss: 0.2316 - accuracy:
0.8389 - val_loss: 0.1656 - val_accuracy: 0.9124
Epoch 36/50
152/152 [=====] - 1s 4ms/step - loss: 0.2316 - accuracy:
0.8440 - val_loss: 0.1661 - val_accuracy: 0.9091
Epoch 37/50
152/152 [=====] - 1s 4ms/step - loss: 0.2281 - accuracy:
0.8471 - val_loss: 0.1639 - val_accuracy: 0.9058
Epoch 38/50
152/152 [=====] - 1s 4ms/step - loss: 0.2244 - accuracy:
0.8515 - val_loss: 0.1619 - val_accuracy: 0.9041
Epoch 39/50
152/152 [=====] - 1s 4ms/step - loss: 0.2268 - accuracy:
0.8521 - val_loss: 0.1620 - val_accuracy: 0.9074
Epoch 40/50
152/152 [=====] - 1s 4ms/step - loss: 0.2221 - accuracy:
0.8550 - val_loss: 0.1598 - val_accuracy: 0.9058
Epoch 41/50
152/152 [=====] - 1s 6ms/step - loss: 0.2227 - accuracy:
0.8548 - val_loss: 0.1605 - val_accuracy: 0.9025
Epoch 42/50
152/152 [=====] - 1s 6ms/step - loss: 0.2262 - accuracy:
0.8502 - val_loss: 0.1614 - val_accuracy: 0.9025
Epoch 43/50
152/152 [=====] - 1s 6ms/step - loss: 0.2220 - accuracy:
```

```

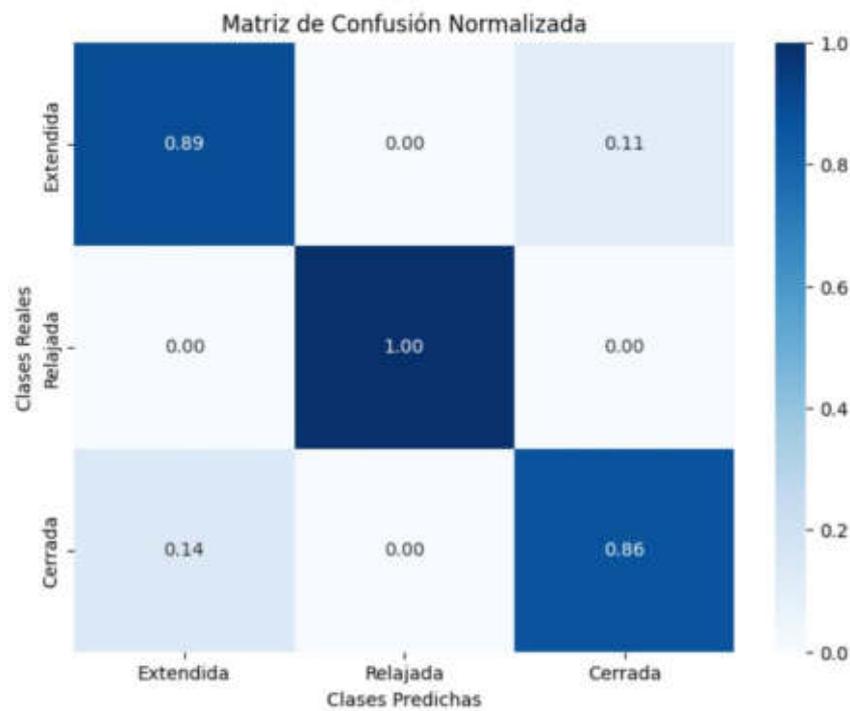
0.8498 - val_loss: 0.1585 - val_accuracy: 0.9107
Epoch 44/50
152/152 [=====] - 1s 6ms/step - loss: 0.2206 - accuracy:
0.8573 - val_loss: 0.1591 - val_accuracy: 0.9025
Epoch 45/50
152/152 [=====] - 1s 6ms/step - loss: 0.2259 - accuracy:
0.8519 - val_loss: 0.1592 - val_accuracy: 0.9058
Epoch 46/50
152/152 [=====] - 1s 5ms/step - loss: 0.2190 - accuracy:
0.8643 - val_loss: 0.1573 - val_accuracy: 0.9091
Epoch 47/50
152/152 [=====] - 1s 4ms/step - loss: 0.2175 - accuracy:
0.8548 - val_loss: 0.1573 - val_accuracy: 0.9091
Epoch 48/50
152/152 [=====] - 1s 4ms/step - loss: 0.2216 - accuracy:
0.8552 - val_loss: 0.1576 - val_accuracy: 0.9074
Epoch 49/50
152/152 [=====] - 1s 5ms/step - loss: 0.2243 - accuracy:
0.8529 - val_loss: 0.1579 - val_accuracy: 0.9025
Epoch 50/50
152/152 [=====] - 1s 6ms/step - loss: 0.2226 - accuracy:
0.8564 - val_loss: 0.1581 - val_accuracy: 0.9041
19/19 [=====] - 0s 2ms/step - loss: 0.1427 - accuracy: 0.
9175
Pérdida en datos de prueba: 0.142653688788414
Precisión en datos de prueba: 0.9174917340278625

```





19/19 [=====] - 0s 3ms/step



```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
In [ ]: #MODELO CONVOLUCIONAL FINAL CON DROPOUT
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Dropout
from tensorflow.keras.utils import plot_model
import seaborn as sns
from sklearn.metrics import confusion_matrix
from IPython.display import Image

# Verificar la estructura de los datos de entrada
if df_normalized.shape[1] != 7:
    raise ValueError("El DataFrame df_normalized no tiene la estructura esperada (?)

# Verificar la presencia de valores nulos en los datos de entrada
if df_normalized.isnull().any().any():
    raise ValueError("El DataFrame df_normalized contiene valores nulos")

# Dividir los datos en características (X) y etiquetas (y)
X_cnn = df_normalized.iloc[:, :4].values # Características (primeras 4 columnas)
y_cnn = df_normalized.iloc[:, 4:].values # Etiquetas (últimas 3 columnas)

# Dividir los datos en conjuntos de entrenamiento, validación y prueba
X_train_cnn, X_temp_cnn, y_train_cnn, y_temp_cnn = train_test_split(X_cnn, y_cnn, 1
X_val_cnn, X_test_cnn, y_val_cnn, y_test_cnn = train_test_split(X_temp_cnn, y_temp

# Redimensionar los datos para que tengan la forma adecuada para la capa Conv1D
X_train_cnn = X_train_cnn.reshape(X_train_cnn.shape[0], X_train_cnn.shape[1], 1)
X_val_cnn = X_val_cnn.reshape(X_val_cnn.shape[0], X_val_cnn.shape[1], 1)
X_test_cnn = X_test_cnn.reshape(X_test_cnn.shape[0], X_test_cnn.shape[1], 1)

# Crear el modelo de red neuronal convolucional (CNN)
model_cnn = Sequential()

# Agregar una capa convolucional 1D
model_cnn.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(4,

# Agregar una capa de agrupación (pooling)
model_cnn.add(MaxPooling1D(pool_size=2))

# Agregar una capa de aplanamiento
model_cnn.add(Flatten())

# Agregar una capa densa intermedia
model_cnn.add(Dense(64, activation='relu'))

# Agregar una capa de Dropout
model_cnn.add(Dropout(0.5)) # 50% de Dropout

# Agregar una capa densa para la salida
model_cnn.add(Dense(3, activation='sigmoid'))

# Compilar el modelo
model_cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy']

# Entrenar el modelo con datos de entrenamiento y validar con datos de validación
history = model_cnn.fit(X_train_cnn, y_train_cnn, validation_data=(X_val_cnn, y_val

# Evaluar el modelo con datos de prueba
loss, accuracy = model_cnn.evaluate(X_test_cnn, y_test_cnn)
print(f'Pérdida en datos de prueba: {loss}')
print(f'Precisión en datos de prueba: {accuracy}')

# Graficar pérdida durante el entrenamiento y validación
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Entrenamiento')
plt.plot(history.history['val_loss'], label='Validación')

```

```

plt.title('Pérdida durante el entrenamiento y validación')
plt.xlabel('Época')
plt.ylabel('Pérdidas %')
plt.legend()
plt.grid(True)
plt.show()

# Graficar precisión durante el entrenamiento y validación
plt.figure(figsize=(10, 5))
plt.plot(history.history['accuracy'], label='Entrenamiento')
plt.plot(history.history['val_accuracy'], label='Validación')
plt.title('Curva de aprendizaje')
plt.xlabel('Época')
plt.ylabel('Precisión %')
plt.legend()
plt.grid(True)
plt.show()

# Realizar predicciones en el conjunto de prueba
y_pred_cnn = model_cnn.predict(X_test_cnn)
y_pred_classes_cnn = np.argmax(y_pred_cnn, axis=1)
y_test_classes_cnn = np.argmax(y_test_cnn, axis=1)

# Calcular matriz de confusión
conf_matrix_cnn = confusion_matrix(y_test_classes_cnn, y_pred_classes_cnn)

# Normalizar la matriz de confusión
conf_matrix_cnn_normalized = conf_matrix_cnn.astype('float') / conf_matrix_cnn.sum(
axis=1)

# Graficar matriz de confusión normalizada
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_cnn_normalized, annot=True, fmt='.2f', cmap='Blues', xtickl
plt.title('Matriz de Confusión Normalizada')
plt.xlabel('Clases Predichas')
plt.ylabel('Clases Reales')
plt.show()

# Visualizar el modelo
plot_model(model_cnn, to_file='model_plot_CNN.png', show_shapes=True, show_layer_n
Image(filename='model_plot_CNN.png')
# Guardar el modelo
model_cnn.save('modelo_convolutacional_v3.h5')

```

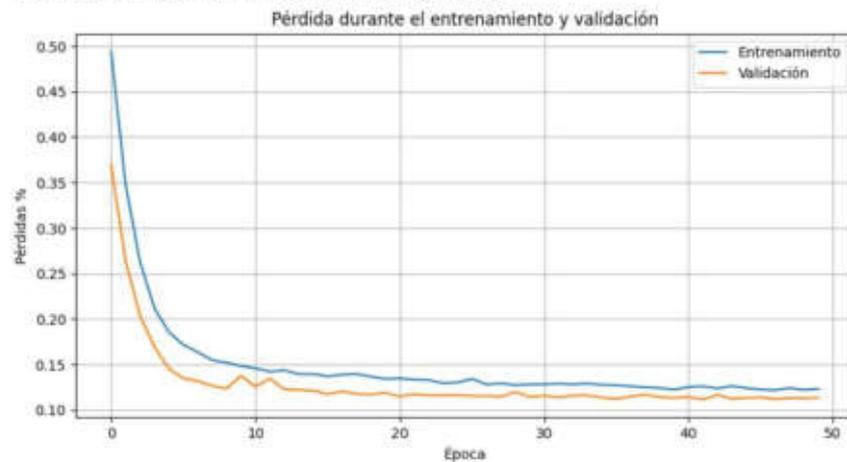
```
Epoch 1/50
485/485 [=====] - 2s 3ms/step - loss: 0.4944 - accuracy:
0.6428 - val_loss: 0.3701 - val_accuracy: 0.8099
Epoch 2/50
485/485 [=====] - 1s 2ms/step - loss: 0.3479 - accuracy:
0.7865 - val_loss: 0.2639 - val_accuracy: 0.8645
Epoch 3/50
485/485 [=====] - 1s 2ms/step - loss: 0.2632 - accuracy:
0.8546 - val_loss: 0.2035 - val_accuracy: 0.9107
Epoch 4/50
485/485 [=====] - 1s 2ms/step - loss: 0.2118 - accuracy:
0.8862 - val_loss: 0.1693 - val_accuracy: 0.9207
Epoch 5/50
485/485 [=====] - 1s 3ms/step - loss: 0.1855 - accuracy:
0.9042 - val_loss: 0.1452 - val_accuracy: 0.9322
Epoch 6/50
485/485 [=====] - 2s 3ms/step - loss: 0.1717 - accuracy:
0.9081 - val_loss: 0.1348 - val_accuracy: 0.9372
Epoch 7/50
485/485 [=====] - 2s 4ms/step - loss: 0.1635 - accuracy:
0.9096 - val_loss: 0.1318 - val_accuracy: 0.9306
Epoch 8/50
485/485 [=====] - 1s 3ms/step - loss: 0.1549 - accuracy:
0.9166 - val_loss: 0.1265 - val_accuracy: 0.9355
Epoch 9/50
485/485 [=====] - 1s 2ms/step - loss: 0.1520 - accuracy:
0.9176 - val_loss: 0.1235 - val_accuracy: 0.9388
Epoch 10/50
485/485 [=====] - 1s 2ms/step - loss: 0.1483 - accuracy:
0.9209 - val_loss: 0.1376 - val_accuracy: 0.9256
Epoch 11/50
485/485 [=====] - 1s 2ms/step - loss: 0.1458 - accuracy:
0.9189 - val_loss: 0.1256 - val_accuracy: 0.9273
Epoch 12/50
485/485 [=====] - 1s 2ms/step - loss: 0.1422 - accuracy:
0.9201 - val_loss: 0.1345 - val_accuracy: 0.9289
Epoch 13/50
485/485 [=====] - 1s 2ms/step - loss: 0.1436 - accuracy:
0.9207 - val_loss: 0.1226 - val_accuracy: 0.9306
Epoch 14/50
485/485 [=====] - 1s 2ms/step - loss: 0.1395 - accuracy:
0.9215 - val_loss: 0.1219 - val_accuracy: 0.9355
Epoch 15/50
485/485 [=====] - 1s 2ms/step - loss: 0.1394 - accuracy:
0.9257 - val_loss: 0.1213 - val_accuracy: 0.9339
Epoch 16/50
485/485 [=====] - 1s 2ms/step - loss: 0.1371 - accuracy:
0.9217 - val_loss: 0.1173 - val_accuracy: 0.9388
Epoch 17/50
485/485 [=====] - 1s 3ms/step - loss: 0.1388 - accuracy:
0.9203 - val_loss: 0.1204 - val_accuracy: 0.9339
Epoch 18/50
485/485 [=====] - 2s 3ms/step - loss: 0.1395 - accuracy:
0.9244 - val_loss: 0.1176 - val_accuracy: 0.9372
Epoch 19/50
485/485 [=====] - 2s 3ms/step - loss: 0.1366 - accuracy:
0.9259 - val_loss: 0.1170 - val_accuracy: 0.9372
Epoch 20/50
485/485 [=====] - 1s 3ms/step - loss: 0.1341 - accuracy:
0.9250 - val_loss: 0.1189 - val_accuracy: 0.9388
Epoch 21/50
485/485 [=====] - 1s 2ms/step - loss: 0.1346 - accuracy:
0.9238 - val_loss: 0.1148 - val_accuracy: 0.9388
Epoch 22/50
```

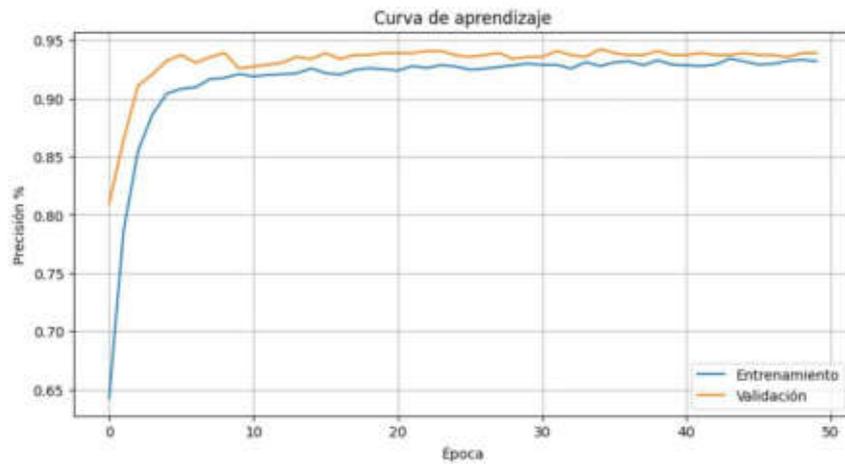
```
485/485 [=====] - 1s 2ms/step - loss: 0.1334 - accuracy:
0.9277 - val_loss: 0.1173 - val_accuracy: 0.9388
Epoch 23/50
485/485 [=====] - 1s 2ms/step - loss: 0.1328 - accuracy:
0.9261 - val_loss: 0.1160 - val_accuracy: 0.9405
Epoch 24/50
485/485 [=====] - 1s 2ms/step - loss: 0.1294 - accuracy:
0.9286 - val_loss: 0.1162 - val_accuracy: 0.9405
Epoch 25/50
485/485 [=====] - 1s 2ms/step - loss: 0.1302 - accuracy:
0.9273 - val_loss: 0.1161 - val_accuracy: 0.9372
Epoch 26/50
485/485 [=====] - 1s 2ms/step - loss: 0.1342 - accuracy:
0.9246 - val_loss: 0.1155 - val_accuracy: 0.9355
Epoch 27/50
485/485 [=====] - 1s 2ms/step - loss: 0.1280 - accuracy:
0.9255 - val_loss: 0.1154 - val_accuracy: 0.9372
Epoch 28/50
485/485 [=====] - 1s 2ms/step - loss: 0.1292 - accuracy:
0.9269 - val_loss: 0.1145 - val_accuracy: 0.9388
Epoch 29/50
485/485 [=====] - 1s 3ms/step - loss: 0.1273 - accuracy:
0.9284 - val_loss: 0.1199 - val_accuracy: 0.9339
Epoch 30/50
485/485 [=====] - 2s 3ms/step - loss: 0.1281 - accuracy:
0.9298 - val_loss: 0.1143 - val_accuracy: 0.9355
Epoch 31/50
485/485 [=====] - 2s 3ms/step - loss: 0.1282 - accuracy:
0.9288 - val_loss: 0.1157 - val_accuracy: 0.9355
Epoch 32/50
485/485 [=====] - 1s 3ms/step - loss: 0.1289 - accuracy:
0.9288 - val_loss: 0.1138 - val_accuracy: 0.9405
Epoch 33/50
485/485 [=====] - 1s 2ms/step - loss: 0.1281 - accuracy:
0.9255 - val_loss: 0.1158 - val_accuracy: 0.9372
Epoch 34/50
485/485 [=====] - 1s 2ms/step - loss: 0.1290 - accuracy:
0.9310 - val_loss: 0.1160 - val_accuracy: 0.9355
Epoch 35/50
485/485 [=====] - 1s 2ms/step - loss: 0.1277 - accuracy:
0.9277 - val_loss: 0.1136 - val_accuracy: 0.9421
Epoch 36/50
485/485 [=====] - 1s 2ms/step - loss: 0.1272 - accuracy:
0.9308 - val_loss: 0.1121 - val_accuracy: 0.9388
Epoch 37/50
485/485 [=====] - 1s 2ms/step - loss: 0.1260 - accuracy:
0.9319 - val_loss: 0.1147 - val_accuracy: 0.9372
Epoch 38/50
485/485 [=====] - 1s 2ms/step - loss: 0.1249 - accuracy:
0.9284 - val_loss: 0.1169 - val_accuracy: 0.9372
Epoch 39/50
485/485 [=====] - 1s 2ms/step - loss: 0.1242 - accuracy:
0.9327 - val_loss: 0.1142 - val_accuracy: 0.9405
Epoch 40/50
485/485 [=====] - 1s 2ms/step - loss: 0.1224 - accuracy:
0.9290 - val_loss: 0.1132 - val_accuracy: 0.9372
Epoch 41/50
485/485 [=====] - 1s 2ms/step - loss: 0.1252 - accuracy:
0.9284 - val_loss: 0.1143 - val_accuracy: 0.9372
Epoch 42/50
485/485 [=====] - 2s 3ms/step - loss: 0.1260 - accuracy:
0.9277 - val_loss: 0.1115 - val_accuracy: 0.9388
Epoch 43/50
485/485 [=====] - 2s 3ms/step - loss: 0.1238 - accuracy:
```

```

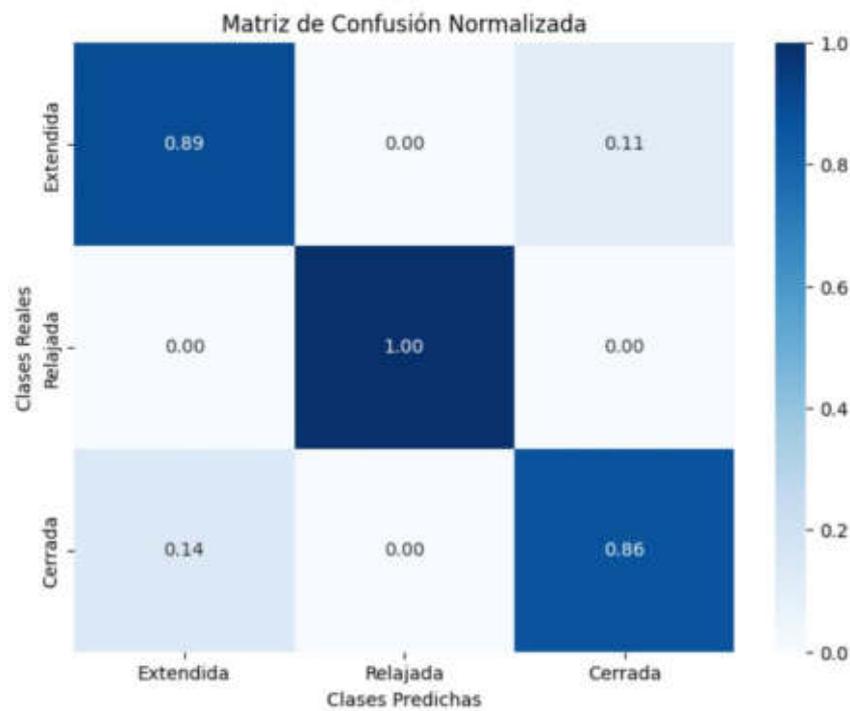
0.9290 - val_loss: 0.1165 - val_accuracy: 0.9372
Epoch 44/50
485/485 [=====] - 1s 3ms/step - loss: 0.1262 - accuracy:
0.9341 - val_loss: 0.1122 - val_accuracy: 0.9372
Epoch 45/50
485/485 [=====] - 1s 2ms/step - loss: 0.1240 - accuracy:
0.9317 - val_loss: 0.1133 - val_accuracy: 0.9388
Epoch 46/50
485/485 [=====] - 1s 2ms/step - loss: 0.1225 - accuracy:
0.9290 - val_loss: 0.1136 - val_accuracy: 0.9372
Epoch 47/50
485/485 [=====] - 1s 2ms/step - loss: 0.1217 - accuracy:
0.9296 - val_loss: 0.1119 - val_accuracy: 0.9372
Epoch 48/50
485/485 [=====] - 1s 2ms/step - loss: 0.1241 - accuracy:
0.9321 - val_loss: 0.1132 - val_accuracy: 0.9355
Epoch 49/50
485/485 [=====] - 1s 2ms/step - loss: 0.1220 - accuracy:
0.9331 - val_loss: 0.1128 - val_accuracy: 0.9388
Epoch 50/50
485/485 [=====] - 1s 2ms/step - loss: 0.1232 - accuracy:
0.9319 - val_loss: 0.1137 - val_accuracy: 0.9388
19/19 [=====] - 0s 2ms/step - loss: 0.1254 - accuracy: 0.
9191
Pérdida en datos de prueba: 0.1253615915775299
Precisión en datos de prueba: 0.9191418886184692

```





19/19 [=====] - 0s 2ms/step



```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

```
In [ ]: #MOSTRAR VARIABLES INTERNAS DEFINIDAS (USO INTERNO)
        %whos
```

Variable	Type	Data/Info
Conv1D	type	<class 'keras.src.layers.<...>oluti
onal.conv1d.Conv1D'>		
Dense	type	<class 'keras.src.layers.core.dens
e.Dense'>		
Dropout	type	<class 'keras.src.layers.<...>izati
on.dropout.Dropout'>		
Flatten	type	<class 'keras.src.layers.<...>shapi
ng.flatten.Flatten'>		
Image	type	<class 'IPython.core.display.Imag
e'>		
LSTM	type	<class 'keras.src.layers.rnn.lstm.L
STM'>		
MaxPooling1D	type	<class 'keras.src.layers.<...>_pool
ing1d.MaxPooling1D'>		
MinMaxScaler	type	<class 'sklearn.preproces<...>sing.
_data.MinMaxScaler'>		
Sequential	type	<class 'keras.src.engine.sequentia
l.Sequential'>		
X	DataFrame	MAV_sensor1 MAV_se<...>\n[60
54 rows x 4 columns]		
X_RNR_window	ndarray	6044x10x4: 241760 elems, type 'floa
t64', 1934080 bytes (1.844482421875 Mb)		
X_cnn	ndarray	6054x4: 24216 elems, type 'float64
', 193728 bytes (189.1875 kb)		
X_sequences	list	n=6044
X_temp	DataFrame	MAV_sensor1 MAV_se<...>\n[12
11 rows x 4 columns]		
X_temp_cnn	ndarray	1211x4: 4844 elems, type 'float64',
38752 bytes		
X_temp_rnr	ndarray	1209x10x4: 48360 elems, type 'float
64', 386880 bytes (377.8125 kb)		
X_test	DataFrame	MAV_sensor1 MAV_se<...>\n[6
06 rows x 4 columns]		
X_test_cnn	ndarray	606x4x1: 2424 elems, type 'float64
', 19392 bytes		
X_test_rnr	ndarray	605x10x4: 24200 elems, type 'float6
4', 193600 bytes (189.0625 kb)		
X_train	DataFrame	MAV_sensor1 MAV_se<...>\n[48
43 rows x 4 columns]		
X_train_cnn	ndarray	4843x4x1: 19372 elems, type 'float6
4', 154976 bytes (151.34375 kb)		
X_train_rnr	ndarray	4835x10x4: 193400 elems, type 'floa
t64', 1547200 bytes (1.47552490234375 Mb)		
X_val	DataFrame	MAV_sensor1 MAV_se<...>\n[6
05 rows x 4 columns]		
X_val_cnn	ndarray	605x4x1: 2420 elems, type 'float64
', 19360 bytes		
X_val_rnr	ndarray	604x10x4: 24160 elems, type 'float6
4', 193280 bytes (188.75 kb)		
accuracy	float	0.9191418886184692
archivo_csv	str	3.-2024-05-15_04-02-25_MC_REV2.csv
archivos_csv	list	n=3
ax	Axes	Axes(0.0510698,0.09124;0.944763x0.9
00747)		
axes	ndarray	11: 11 elems, type 'object', 88 byt
es		
butter	function	<function butter at 0x7d8e645270a0>
calcular_MNF	function	<function calcular_MNF at 0x7d8e5c7
9bd90>		
calcular_MNF_por_segmento	function	<function calcular_MNF_po<...>gment
o at 0x7d8e5c79beb0>		
cm	ndarray	3x3: 9 elems, type 'int64', 72 byte

```

s
cm_normalized          ndarray          3x3: 9 elems, type `float64`, 72 by
tes
color                 ndarray          4: 4 elems, type `float64`, 32 byte
s
colors                ndarray          11x4: 44 elems, type `float64`, 352
bytes
column                str          out3
conf_matrix_cnn       ndarray          3x3: 9 elems, type `int64`, 72 byte
s
conf_matrix_cnn_normalized ndarray          3x3: 9 elems, type `float64`, 72 by
tes
confusion_matrix      function          <function confusion_matrix at 0x7d8
e5750f6d0>
correlation_matrix    DataFrame          RMS_sensor1 <...>000 -
0.500000  1.000000
crear_nuevo_df        function          <function crear_nuevo_df at 0x7d8ea
89e7be0>
cruces_por_cero       function          <function cruces_por_cero at 0x7d8e
5c311750>
cutoff_freq_high      int          5
cutoff_freq_low       int          450
cz_sensor1            list          n=3693
cz_sensor2            list          n=3693
df                    DataFrame          A0_val  A0_Val_fl<...>[6811
32 rows x 8 columns]
df1                   DataFrame          A0_val  A0_Val_fl<...>[6078
30 rows x 8 columns]
df2                   DataFrame          A0_val  A0_Val_fl<...>[6019
70 rows x 8 columns]
df3                   DataFrame          A0_val  A0_Val_fl<...>[6811
32 rows x 8 columns]
df_cz                 DataFrame          CZ_sensor1  CZ_sens<...>\n[36
93 rows x 2 columns]
df_mav                DataFrame          MAV_sensor1  MAV_se<...>\n[36
93 rows x 2 columns]
df_mnf                DataFrame          MNF_sensor1  MNF_se<...>\n[36
93 rows x 2 columns]
df_modificado         DataFrame          A0_val  A0_Val_fl<...>[6811
32 rows x 8 columns]
df_normalized         DataFrame          MAV_sensor1  MAV_se<...>\n[60
54 rows x 7 columns]
df_sorted             DataFrame          RMS_sensor1  RMS_se<...>\n[369
3 rows x 11 columns]
df_target             DataFrame          sensor 1  sensor <...>[9454
66 rows x 5 columns]
df_temp              DataFrame          Sensor1  Sensor<...>[9454
66 rows x 2 columns]
dfs_originales        list          n=3
difference            Series          0      1.743641\n1      <...>gth:
3693, dtype: float64
drive                 module          <module `google.colab.dri<...>s/goo
gle/colab/drive.py`>
fig                   Figure          Figure(1000x3300)
filtered_df1          DataFrame          sensor 1  sensor <...>[3039
15 rows x 5 columns]
filtered_df1_high     ndarray          303915x2: 607830 elems, type `float
64`, 4862640 bytes (4.6373748779296875 Mb)
filtered_df2          DataFrame          sensor 1  sensor <...>[3009
85 rows x 5 columns]
filtered_df2_high     ndarray          300985x2: 601970 elems, type `float
64`, 4815760 bytes (4.5926666259765625 Mb)
filtered_df3          DataFrame          sensor 1  sensor <...>[3405
66 rows x 5 columns]

```

```

filtered_df3_high          ndarray          340566x2: 681132 elems, type 'float
64', 5449056 bytes (5.196624755859375 Mb)
filtfilt                   function          <function filtfilt at 0x7d8e641d7d9
0>
fin                         int              945408
final_df                   DataFrame          RMS_sensor1 RMS_se<...>\n[369
3 rows x 11 columns]
fs                          int              1000
grupo                      DataFrame          sensor 1 sensor<...>.0785
09      0      0      1
grupo_size                 int              256
highpass_filter            function          <function highpass_filter at 0x7d8e
6044dab0>
history                    History          <keras.src.callbacks.Hist<...>objec
t at 0x7d8def38c040>
i                           int              6043
inicio                    int              945152
labels                    list              n=2
lista_rms                  list              n=3693
loss                       float             0.1253615915775299
lowpass_filter             function          <function lowpass_filter at 0x7d8e6
044da20>
matriz_confusion           ndarray          3x3: 9 elems, type 'int64', 72 byte
s
matriz_confusion_normalizada ndarray          3x3: 9 elems, type 'float64', 72 by
tes
mav_sensor1                list              n=3693
mav_sensor2                list              n=3693
means                      list              n=2
mnf_values_sensor1         list              n=3693
mnf_values_sensor2         list              n=3693
model_cnn                  Sequential        <keras.src.engine.sequent<...>objec
t at 0x7d8e5c7d17b0>
model_rnn                  Sequential        <keras.src.engine.sequent<...>objec
t at 0x7d8def599210>
model_secuencial           Sequential        <keras.src.engine.sequent<...>objec
t at 0x7d8df9f70190>
muestras_por_grupo         int              256
muestras_sensor1           Series            945152 7.614738\n94515<...>ngth:
256, dtype: float64
muestras_sensor2           Series            945152 11.417162\n9451<...>ngth:
256, dtype: float64
new_df1                    DataFrame          sensor 1 sensor <...>[3039
15 rows x 2 columns]
new_df2                    DataFrame          sensor 1 sensor <...>[3009
85 rows x 2 columns]
new_df3                    DataFrame          sensor 1 sensor <...>[3405
66 rows x 2 columns]
new_df_rms_mnf             DataFrame          RMS_sensor1 RMS_se<...>\n[36
93 rows x 8 columns]
nombre_archivo             str              data_procesada_normalizada_rev3.xls
x
nombre_df                  str              df3
nombres_columnas           list              n=8
nombres_df                 list              n=3
np                          module            <module 'numpy' from '/us<...>kage
s/numpy/__init__.py'>
nuevo_df                   DataFrame          sensor 1 sensor <...>[3405
66 rows x 2 columns]
nuevos_dfs                 list              n=3
num_columns                int              11
num_grupos                 int              3693
os                          module            <module 'os' from '/usr/lib/python
3.10/os.py'>

```

```

outs_df                                DataFrame              out1 out2 out3\n0<...>\n[36
93 rows x 3 columns]
pd                                      module                <module 'pandas' from '/u<...>ages/
pandas/__init__.py'>
plot_model                              function              <function plot_model at 0x7d8e0147f
910>
plt                                      module                <module 'matplotlib.pyplo<...>es/ma
tplotlib/pyplot.py'>
predicciones                            ndarray               605x3: 1815 elems, type 'float32',
7260 bytes
predicciones_binarias                  ndarray               605x3: 1815 elems, type 'bool', 181
5 bytes
promedio_grupo                          list                  n=3
promedios                               list                  n=3693
rms_df                                   DataFrame              RMS_sensor1 RMS_sec<...>\n[36
93 rows x 2 columns]
rms_sensor1                             float64               11.654747564694294
rms_sensor2                             float64               10.378647694639229
ruta_drive                              str                   /content/drive/My Drive/TESIS/DATOS
_rev3
scaler                                  MinMaxScaler         MinMaxScaler()
segment_size                            int                   256
signal1                                 Series                0          0.916130\n1    <...>h: 94
5466, dtype: float64
signal2                                 Series                0          -3.859158\n1    <...>h: 94
5466, dtype: float64
sns                                      module                <module 'seaborn' from '/<...>ges/s
eaborn/__init__.py'>
train_test_split                         function              <function train_test_split at 0x7d8
e5750cc10>
ventana                                  int                   256
ventana_actual                           DataFrame              sensor 1 sensor<...>.0785
09  0  0  1
welch                                    function              <function welch at 0x7d8e641e0a60>
window_size                             int                   10
y                                         DataFrame              out1 out2 out3\n0<...>\n[60
54 rows x 3 columns]
y_RNR_window                             ndarray               6044x3: 18132 elems, type 'float64
', 145056 bytes (141.65625 kb)
y_cnn                                     ndarray               6054x3: 18162 elems, type 'int64',
145296 bytes (141.890625 kb)
y_pred                                   ndarray               606x3: 1818 elems, type 'float32',
7272 bytes
y_pred_classes_cnn                      ndarray               606: 606 elems, type 'int64', 4848
bytes
y_pred_cnn                              ndarray               606x3: 1818 elems, type 'float32',
7272 bytes
y_pred_round                            ndarray               606x3: 1818 elems, type 'float32',
7272 bytes
y_sequences                             list                  n=6044
y_temp                                   DataFrame              out1 out2 out3\n5<...>\n[12
11 rows x 3 columns]
y_temp_cnn                              ndarray               1211x3: 3633 elems, type 'int64', 2
9064 bytes
y_temp_rnr                              ndarray               1209x3: 3627 elems, type 'float64',
29016 bytes
y_test                                   DataFrame              out1 out2 out3\n8<...>\n[6
06 rows x 3 columns]
y_test_classes_cnn                      ndarray               606: 606 elems, type 'int64', 4848
bytes
y_test_cnn                              ndarray               606x3: 1818 elems, type 'int64', 14
544 bytes
y_test_rnr                              ndarray               605x3: 1815 elems, type 'float64',
14520 bytes

```

```

y_train          DataFrame          out1 out2 out3\n5<...>\n[48
43 rows x 3 columns]
y_train_cnn      ndarray           4843x3: 14529 elems, type `int64`,
116232 bytes (113.5078125 kb)
y_train_rnr      ndarray           4835x3: 14505 elems, type `float64`
, 116040 bytes (113.3203125 kb)
y_val            DataFrame          out1 out2 out3\n5<...>\n[6
05 rows x 3 columns]
y_val_cnn        ndarray           605x3: 1815 elems, type `int64`, 14
520 bytes
y_val_rnr        ndarray           604x3: 1812 elems, type `float64`,
14496 bytes

```

```

In [ ]: #LIMPIAR MEMORIA ( USO INTERNO)
        #BORRA TODO EN LA SECCION EXISTENTE
        #QUITAR EL COMENTARIO DE IMPORT
        #import gc
        gc.collect()

```

```

-----
NameError                                Traceback (most recent call last)
<ipython-input-29-124f411991a4> in <cell line: 5>()
      3 #QUITAR EL COMENTARIO DE IMPORT
      4 #import gc
----> 5 gc.collect()

NameError: name 'gc' is not defined

```

Anexo 6. Código de programación en Python para la validación y evaluación de rendimiento de los modelos desarrollados.

```

In [ ]: from google.colab import drive
from tensorflow.keras.models import load_model

# Montar Google Drive
drive.mount('/content/drive')

# Ruta al archivo del modelo en Google Drive
ruta_modelo_sec = '/content/drive/My Drive/TESIS/VALIDACION_Rev1/modelo_secuencial_v3.h5'

# Cargar el modelo
modelo_cargado_sec = load_model(ruta_modelo_sec)

# Verificar que el modelo se ha cargado correctamente
print(modelo_cargado_sec.summary())

Mounted at /content/drive
Model: "sequential_17"
-----
Layer (type)                 Output Shape              Param #
-----
dense_51 (Dense)             (None, 8)                 40
dropout_24 (Dropout)         (None, 8)                 0
dense_52 (Dense)             (None, 8)                 72
dropout_25 (Dropout)         (None, 8)                 0
dense_53 (Dense)             (None, 3)                 27
-----
Total params: 139 (556.00 Byte)
Trainable params: 139 (556.00 Byte)
Non-trainable params: 0 (0.00 Byte)
-----
None

In [ ]: from google.colab import drive
from tensorflow.keras.models import load_model

# Montar Google Drive
drive.mount('/content/drive')

# Ruta al archivo del modelo en Google Drive
ruta_modelo_rnr = '/content/drive/My Drive/TESIS/VALIDACION_Rev1/modelo_rrecurrente_v1 (1).h5'

# Cargar el modelo
modelo_cargado_rnr = load_model(ruta_modelo_rnr)

# Verificar que el modelo se ha cargado correctamente
print(modelo_cargado_rnr.summary())

Mounted at /content/drive
Model: "sequential_1"
-----
Layer (type)                 Output Shape              Param #
-----
lstm_3 (LSTM)                (None, 4)                 104
dropout_9 (Dropout)          (None, 4)                 0
dense_9 (Dense)              (None, 8)                 40
dropout_10 (Dropout)         (None, 8)                 0
dense_10 (Dense)             (None, 8)                 72
dropout_11 (Dropout)         (None, 8)                 0
dense_11 (Dense)             (None, 3)                 27
-----
Total params: 283 (1.11 KB)
Trainable params: 283 (1.11 KB)
Non-trainable params: 0 (0.00 Byte)
-----
None

In [ ]: from google.colab import drive
from tensorflow.keras.models import load_model

# Montar Google Drive
drive.mount('/content/drive')

# Ruta al archivo del modelo en Google Drive
ruta_modelo_cnn = '/content/drive/My Drive/TESIS/VALIDACION_Rev1/modelo_convolutional_v1.h5'

# Cargar el modelo
modelo_cargado_cnn = load_model(ruta_modelo_cnn)

```

```
# Verificar que el modelo se ha cargado correctamente
print(modelo_cargado_cnn.summary())

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount('/content/drive', force_remount=True).
Model: "sequential_8"
-----
Layer (type)                 Output Shape                 Param #
-----
conv1d (Conv1D)              (None, 2, 64)               256
max_pooling1d (MaxPooling1D) (None, 1, 64)               0
flatten (Flatten)            (None, 64)                   0
dense_14 (Dense)              (None, 64)                   4160
dropout_21 (Dropout)         (None, 64)                   0
dense_15 (Dense)              (None, 3)                    195
-----
Total params: 4611 (18.01 KB)
Trainable params: 4611 (18.01 KB)
Non-trainable params: 0 (0.00 Byte)
-----
None
```

```
26 [ ]: import pandas as pd
from google.colab import drive
import os

# Montar Google Drive
drive.mount('/content/drive')

# Especifica la ruta de la carpeta que contiene los archivos CSV en Google Drive
ruta_drive = '/content/drive/My Drive/TESIS/VALIDACION_Bev3/test_data.csv'

# Cargar el archivo Excel en un DataFrame
df_normalized = pd.read_csv(ruta_drive)
print(df_normalized.columns)

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount('/content/drive', force_remount=True).
Index(['RMS_sensor1', 'RMS_sensor2', 'MAV_sensor1', 'MAV_sensor2',
       'MNF_sensor1', 'MNF_sensor2', 'CZ_sensor1', 'CZ_sensor2', 'out1',
       'out2', 'out3'],
      dtype='object')
```

```
27 [ ]: #Muestra las columnas del dataframe de validacion
df_normalized.drop(['RMS_sensor1', 'RMS_sensor2', 'CZ_sensor1', 'CZ_sensor2'], axis=1, inplace=True)
print(df_normalized.columns)

Index(['MAV_sensor1', 'MAV_sensor2', 'MNF_sensor1', 'MNF_sensor2', 'out1',
       'out2', 'out3'],
      dtype='object')
```

```
28 [ ]: display(df_normalized)
```

	MAV_sensor1	MAV_sensor2	MNF_sensor1	MNF_sensor2	out1	out2	out3
0	0.630908	0.366128	0.086811	0.128063	0	0	1
1	0.627387	0.384223	0.059711	0.045939	0	0	1
2	0.325905	0.212327	0.145045	0.181116	0	0	1
3	0.373271	0.371340	0.086071	0.096750	0	0	1
4	0.284523	0.278554	0.077805	0.182580	0	0	1
...
1510	0.136739	0.272178	0.087602	0.179044	0	0	1
1511	0.641077	0.354800	0.057967	0.062282	0	0	1
1512	0.504913	0.506442	0.080835	0.058541	0	0	1
1513	0.462934	0.115798	0.056900	0.099243	0	0	1
1514	0.305564	0.267965	0.005302	0.226574	0	0	1

1515 rows x 7 columns

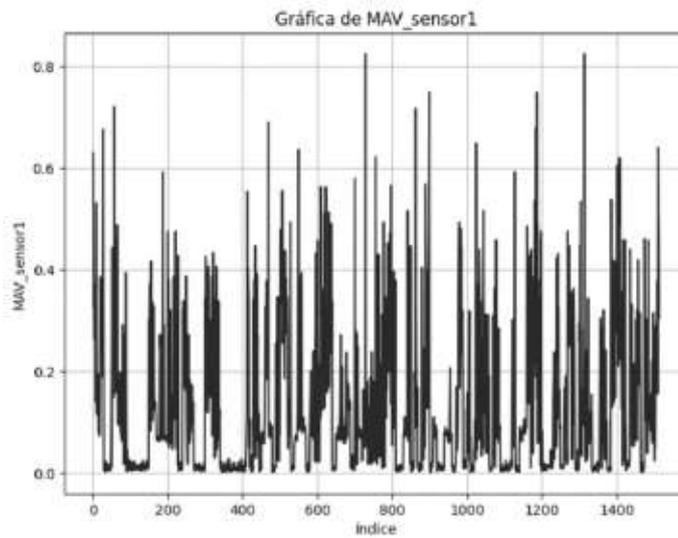
```
29 [ ]: import matplotlib.pyplot as plt

# Obtener el mapa de colores viridis
 cmap = plt.get_cmap('viridis')

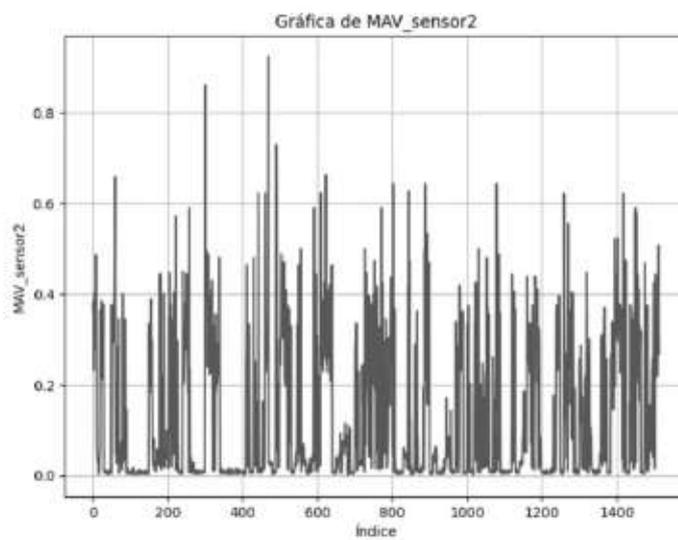
# Iterar sobre cada columna del DataFrame y graficarla
for i, column in enumerate(df_normalized.columns):
    plt.figure(figsize=(8, 6))
```

```
plt.plot(df_normalized[column], color=cmap(i / len(df_normalized.columns)))
plt.title(f'Gráfica de {column}')
plt.xlabel('Índice')
plt.ylabel(column)
plt.grid(True)
plt.subtitle('Gráficas de DataFrame para la validación cruzada')
plt.show()
```

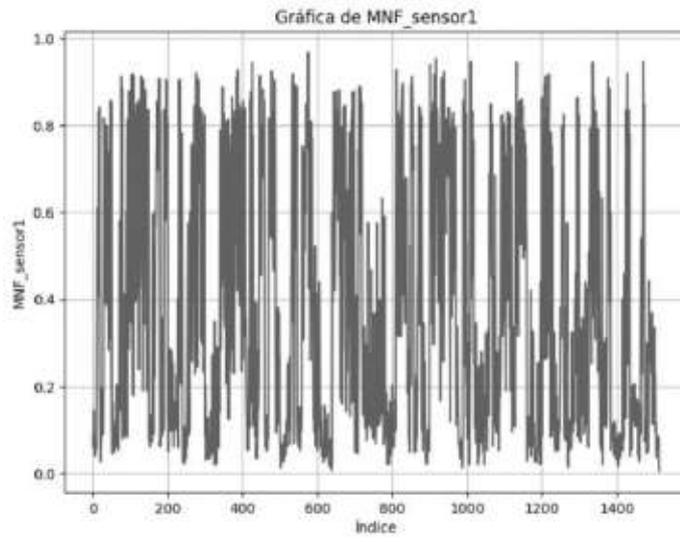
Gráficas de DataFrame para la validación cruzada



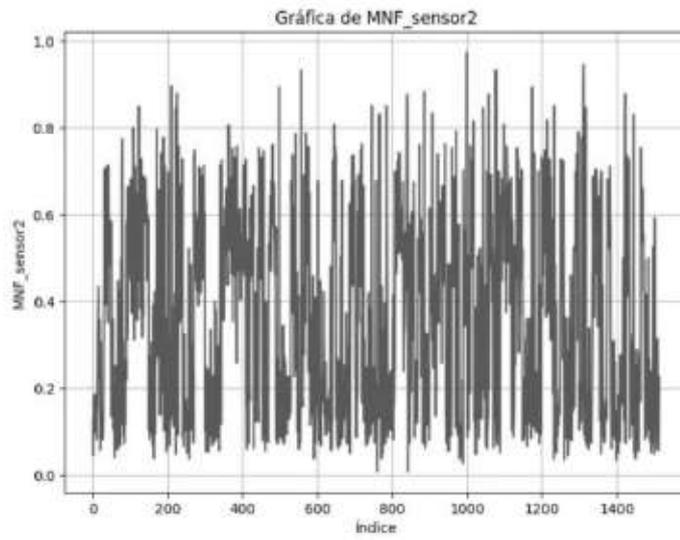
Gráficas de DataFrame para la validación cruzada



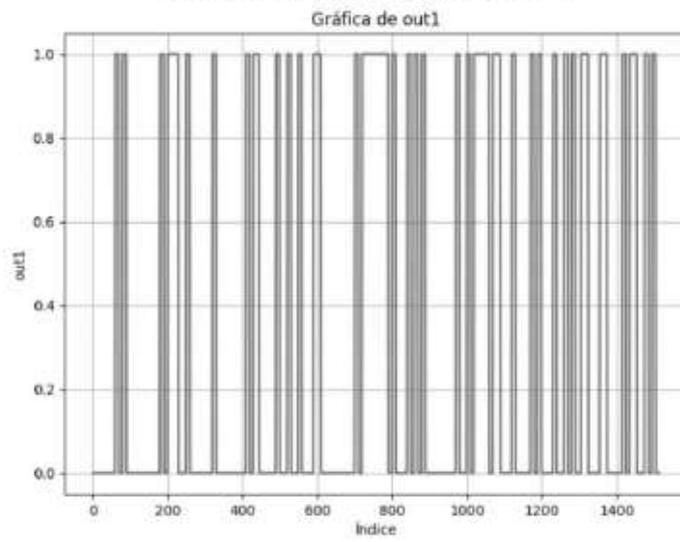
Gráficas de DataFrame para la validacion cruzada



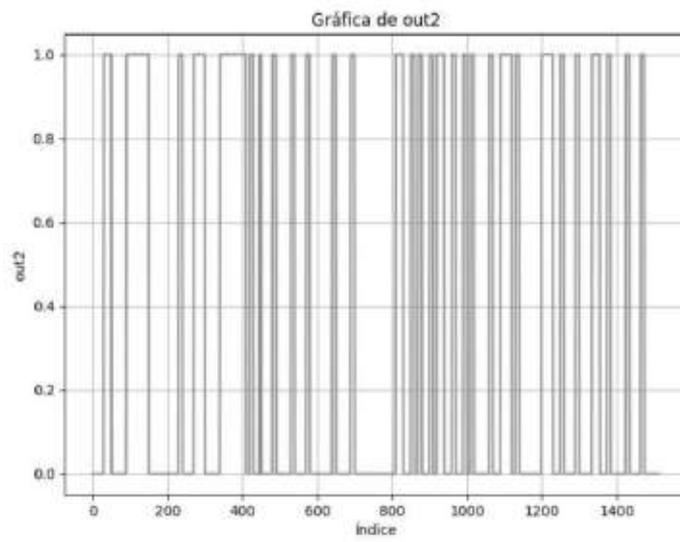
Gráficas de DataFrame para la validacion cruzada



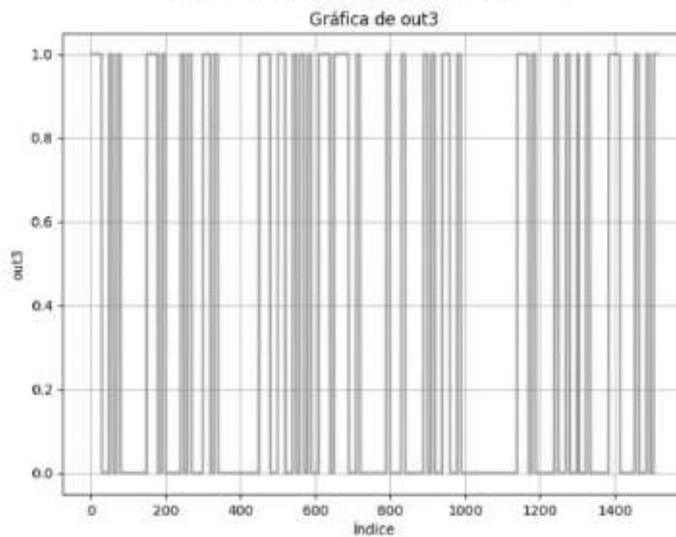
Gráficas de DataFrame para la validacion cruzada



Gráficas de DataFrame para la validacion cruzada



Gráficas de DataFrame para la validación cruzada



```
In [ ]: #SECUENCIAL
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

# Preparar los datos de entrada y salida
X_1 = df_normalized.iloc[:, :4].values # Datos de entrada
y_1 = df_normalized.iloc[:, 4:].values # Datos de salida

# Realizar las predicciones con el modelo cargado
predicciones_1 = modelo_cargado_sec.predict(X_1)

# Convertir las predicciones a etiquetas (redondear los valores)
predicciones_etiquetas_1 = predicciones_1.round()

# Construir la matriz de confusión
matriz_confusion_1 = confusion_matrix(y_1.argmax(axis=1), predicciones_etiquetas_1.argmax(axis=1))

# Normalizar la matriz de confusión
matriz_confusion_1_normalizada = matriz_confusion_1.astype('float') / matriz_confusion_1.sum(axis=1)[], np.newaxis]

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 8))
sns.heatmap(matriz_confusion_1_normalizada, annot=True, fmt='.2f', cmap='Blues', xticklabels=['Estendida', 'Relajada', 'Cerrada'],
            yticklabels=['Etiquetas Predichas', 'Etiquetas Reales'])
plt.xlabel('Etiquetas Reales')
plt.ylabel('Etiquetas Predichas')
plt.show()

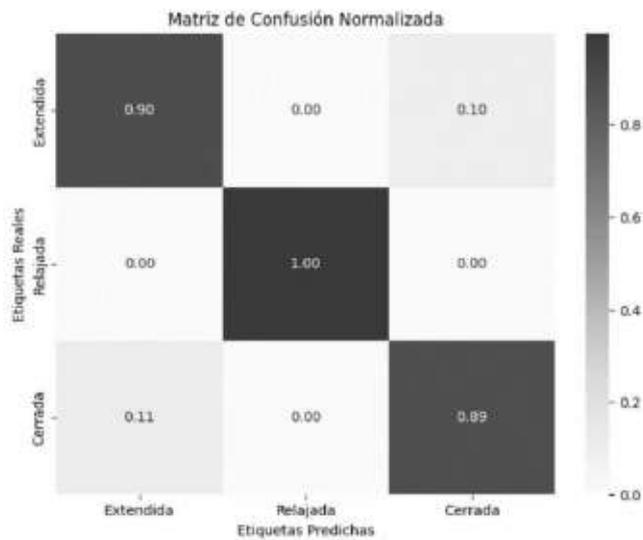
# Calcular la precisión
precision_1 = accuracy_score(y_1.argmax(axis=1), predicciones_etiquetas_1.argmax(axis=1))

# Calcular la pérdida (utilizando la precisión como medida)
perdida_1 = 1 - precision_1

# Calcular la sensibilidad (recall)
sensibilidad_1 = recall_score(y_1.argmax(axis=1), predicciones_etiquetas_1.argmax(axis=1), average='macro')

# Imprimir los valores en consola
print('Precisión: {precision_1}')
print('Pérdida: {perdida_1}')
print('Sensibilidad: {sensibilidad_1}')

48/48 [-----] - 0s 3ms/step
```



Precisión: 0.928852885288528
 Pérdida: 0.07194719471947197
 Sensibilidad: 0.928852885288528

```
In [ ]:
# RECURRENTES
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns

# Preparar los datos de entrada y salida
X = df.normalized.iloc[:, :4].values # Datos de entrada
y = df.normalized.iloc[:, 4].values # Datos de salida

# Preparar los datos de entrada con una ventana de contexto de 10 intervalos de tiempo
ventana_contexto = 2
X_ventana = []
for i in range(len(X) - ventana_contexto + 1):
    X_ventana.append(X[i:i + ventana_contexto])
X_ventana = np.array(X_ventana)

# Realizar las predicciones con el modelo cargado
predicciones = modelo_cargado_rnn.predict(X_ventana)

# Convertir las predicciones a etiquetas (redondear los valores)
predicciones_etiquetas = predicciones.round()

# Obtener las etiquetas reales (redondear los valores)
y_etiquetas = y[ventana_contexto - 1:]

# Construir la matriz de confusión
matriz_confusion = confusion_matrix(y_etiquetas.argmax(axis=1), predicciones_etiquetas.argmax(axis=1))

# Normalizar la matriz de confusión por clase
total_por_clase = matriz_confusion.sum(axis=1) # Total de predicciones verdaderas por clase
matriz_confusion_normalizada = matriz_confusion / total_por_clase[:, np.newaxis]

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(matriz_confusion_normalizada, annot=True, fmt=".2f", cmap='Blues', sticklabels=['Extendida', 'Relajada', 'Cerrada'])
plt.title('Matriz de Confusión Normalizada')
plt.xlabel('Etiquetas Predichas')
plt.ylabel('Etiquetas Reales')
plt.show()

# Calcular la precisión
precision = accuracy_score(y_etiquetas, predicciones_etiquetas)

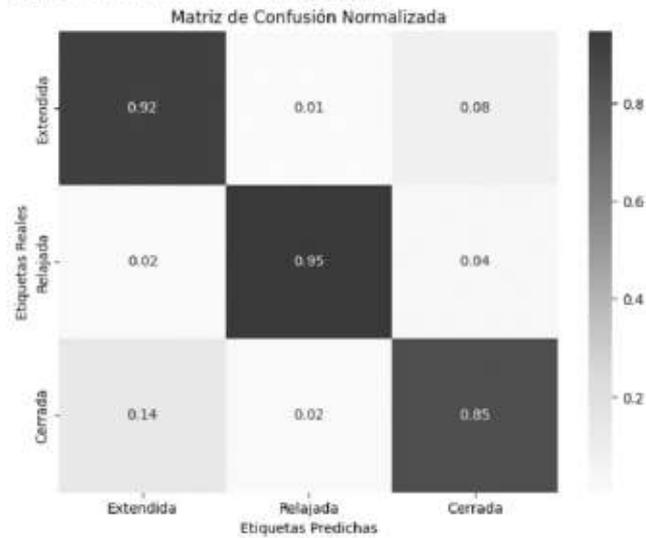
# Calcular la pérdida (utilizando la precisión como medida)
perdida = 1 - precision

# Calcular la sensibilidad (recall)
sensibilidad = recall_score(y_etiquetas, predicciones_etiquetas, average='macro')

# Imprimir los valores en consola
```

```
print(f'Precisión: {precision}')
print(f'Pérdida: {pérdida}')
print(f'Sensibilidad: {sensibilidad}')
```

48/48 [*****] - 1s 5ms/step



```
Precisión: 0.9815852047556143
Pérdida: 0.09841479524438568
Sensibilidad: 0.9815488119448515
```

```
In [ ]: #CONVENCIONAL
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns

# Preparar los datos de entrada y salida
X_2 = df_normalized.iloc[:, :4].values # Datos de entrada
y_2 = df_normalized.iloc[:, 4:].values # Datos de salida

# Realizar las predicciones con el modelo cargado
predicciones_2 = modelo_cargado_cnn.predict(X_2)

# Convertir las predicciones a etiquetas (redondear los valores)
predicciones_etiquetas_2 = predicciones_2.round()

# Construir la matriz de confusión
matriz_confusion_2 = confusion_matrix(y_2.argmax(axis=1), predicciones_etiquetas_2.argmax(axis=1))

# Normalizar la matriz de confusión
matriz_confusion_2_normalizada = matriz_confusion_2.astype('float') / matriz_confusion_2.sum(axis=1)[1, np.newaxis]

# Visualizar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(matriz_confusion_2_normalizada, annot=True, fmt='.2f', cmap='Blues', xticklabels=['Extendida', 'Relajada', 'Cerrada'],
            yticklabels=['Extendida', 'Relajada', 'Cerrada'])
plt.title('Matriz de Confusión Normalizada')
plt.xlabel('Etiquetas Predichas')
plt.ylabel('Etiquetas Reales')
plt.show()

# Calcular la precisión
precision_2 = accuracy_score(y_2.argmax(axis=1), predicciones_etiquetas_2.argmax(axis=1))

# Calcular la pérdida (utilizando la precisión como medida)
pérdida_2 = 1 - precision_2

# Calcular la sensibilidad (recall)
sensibilidad_2 = recall_score(y_2.argmax(axis=1), predicciones_etiquetas_2.argmax(axis=1), average='macro')

# Imprimir los valores en consola
print(f'Precisión: {precision_2}')
print(f'Pérdida: {pérdida_2}')
print(f'Sensibilidad: {sensibilidad_2}')
```

48/48 [*****] - 1s 5ms/step

