EDGE AI aplicado a la telemedicina para el monitoreo de señales ECG

Autor: Martín Cavallo

Tutor: Prof. Dr. Jônata Tyska Carvalho

Ciudad de Rivera
Abril 2022

Índice

Resumen

Las enfermedades cardiovasculares son la principal causa de muerte en el mundo.[1] En tanto en Uruguay, son la primera causa de muerte en mujeres y la segunda en hombres.[2] Según la Organización Panamericana de la Salud, «los servicios de prevención y tratamiento de las enfermedades no transmisibles (ENT) se han visto gravemente afectados desde el comienzo de la pandemia de Covid-19 en la región de las Américas.» Y agrega que «desde que comenzó la pandemia, los servicios de salud de rutina fueron reorganizados o interrumpidos y muchos dejaron de brindar atención a las personas en tratamiento contra enfermedades como el cáncer, enfermedades cardiovasculares y diabetes».[3] Es así que la situación sanitaria llevó a una demora y un atraso significativo en la atención de los pacientes «no Covid-19». En general, ¿qué sucedidó con ellos?. En particular, ¿qué lugar han ocupado los pacientes cardiológicos cuando son los pacientes de la principal causa de muerte en el mundo?. La crisis de Covid-19 ha planteado demandas sin precedentes en los sistemas de salud a nivel global, destacando como nunca la necesidad de desarrollar nuevas tecnologías, autónomas y centradas en el ser humano, que puedan optimizar drásticamente la gestión y el tratamiento del paciente. El uso de sensores biométricos y asistencia sanitaria remota permite un seguimiento cercano sin necesidad de consultas presenciales. El concepto de Internet de las cosas (IOT del inglés Internet of Things), promete aportar mucho valor en esta demanda.

Pero en la salud no basta con que el IOT conecte una variedad de dispositivos a la nube y que dependan de una conexión estable y permanente a internet. El IOT debe ir más allá y agregar inteligencia, seguridad y autonomía a los dispositivos en el borde, cerca de los usuarios, de forma de ofrecer confianza, fiabilidad y seguridad.

En este trabajo se objetivizó estudiar la implementación de inteligencia en un dispositivo IOT ya existente, dedicado a la telemedicina para el monitoreo de pacientes cardiológicos, que sea capaz de clasificar señales ECG en señales normales y anormales. De forma de poder demostrar los beneficios del Edge Computing, esto es, del procesamiento de datos localmente en el dispositivo. Este objetivo se logró con la aplicación de tiny machine learning (de aquí en adelante tiny ML), desarrollando un modelo de machine learning lo suficientemenete pequeño y liviano para correr en microcontroladores de recursos limitados y que es capaz de clasificar señales cardíacas en normales o anormales.

Para esto, se estudió el estado del arte actual de las aplicaciones de machine learning para la clasificación de señales ECG, haciendo mayor hincapié en aplicaciones de tipo tiny ML. Como resultado, se encontró una nueva arquitectura de red neuronal denominada Temporal Convolutional Network (de aquí en adelante TCN), basada en redes convolucionales, y que ha sido presentada como una nueva opción para el análisis de series temporales con mejor desempeño sobre los clásicos modelos destinados para este fin, como ser las de redes recurren-

[1] World Health Organization. *The top 10 causes of death. (2020)* Disponible en https://www.who.int/es/news-room/fact-sheets/detail/the-top-10-causes-of-death

[2] Comisión Honoraria para la Salud Cardiovascular. *Enfermedades cardiovasculares. (2021).* Disponible en https://semanadelcorazon.com.uy/2021/enfermedades-cardiovasculares/

[3] Organización Panamericana de la Salud. *La COVID-19 afectó el funcionamiento de los servicios de salud para enfermedades no transmisibles en las Américas. (2020).* Disponible en https://www.paho.org/es/noticias/17-6-2020-covid-19-afecto-funcionamiento-servicios-salud-para-enfermedades-no

tes. Yendo un paso más, se encontró además un trabajo lanzado a mediados del año 2021, que presenta un modelo de tiny ML basado en la arquitectura TCN denominado ECG-TCN, con el fin de clasificar señales ECG. Se decidió tomar este trabajo como punto de partida y darle una vuelta más de tuerca para que se ajuste a nuestro objetivo. En particular, se identificaron los siguientes requerimientos:

1. **Trabajar en una clasificación binaria, en lugar de una clasificación categórica:** El trabajo original realiza una clasificación de las señales ECG en 5 categorías. No es de nuestro interés identificar a lujo de detalles el tipo de arritmia que un paciente puede estar cursando, ya que esto será responsabilidad del médico cardiólogo y/o de modelos de clasificación e identificación de arritmias más potentes que corren en la nube, y que procesarán los datos luego de que el dispositivo IOT los envíe. Nos basta con que el dispositivo pueda clasificar las señales en señales normales y señales anómalas.

2. **Procesar un dataset con señales lo más crudas y reales posible:** El trabajo original trabaja sobre un dataset compuesto por señales ECG tomadas de un único paciente, y que consisten en un único pulso cardíaco, lo que significa que existe un pre procesamiento de las señales ECG antes de llegar al dataset. En nuestro caso, trabajamos con un dataset lanzado en el año 2020, compuesto de señales ECG de 10 segundos de duración, en el cual las señales anómalas pueden presentar su anomalía en cualquier instante de tiempo. Tal como sucedería en una aplicación real.

3. **Minimizar el riesgo de falsos negativos:** Una aplicación de machine learning en la salud, donde puede estar en riesgo la vida de una persona, no admite falsos resultados. En nuestro caso, los falsos positivos no tratan un caso de gravedad, dado que posteriormente las señales ECG serán enviadas a la nube y analizadas por un médico y/o algoritmos más potentes, y podrán así detectar que se trata de un falso positivo, sin mayores consecuencias. Mas un falso negativo puede ser catastrófico. Como se explicará más adelante, uno de los beneficios del Edge Computing es la disminución del consumo (ahorro energético, mayor autonomía) y de los requerimientos del ancho de banda, que se obtiene de procesar los datos localmente y no enviarlos a la nube. Una señal clasificada como negativa, significa que ha sido identificada como una señal normal y por lo tanto no se enviaría a la nube, con el fin de ahorrar energía. De tratarse de un falso negativo, significa que el paciente podría estar cursando una arritmia y que el dispositivo no es capaz de identificarla, pero peor aún, tampoco podrán identificarlas ni el doctor ni los algoritmos más potentes, poniendo así en riesgo la vida del paciente. Se buscó entonces, lograr una certeza del 100 % en la detección de señales normales y anormales, priorizando eliminar falsos negativos.

Realizando las modificaciones pertinentes al modelo ECG-TCN según nuestras necesidades, se precedió a entrenar dicho modelo con nuestro dataset. Analizando los resultados obtenidos, se encontró que a priori no se satisface el objetivo 3. A partir de aquí, se trabajó sobre métodos de optimización del modelo en búsqueda de satisfacer dicho objetivo. Finalmente se realizó un procedimiento de optimización mediante la aplicación de pesos a las muestras de nuestro dataset, de forma de entrenar modelos de forma iterativa y brindar mayor pesos a aquellas muestras que representan falsos positivos y falsos negativos. De esta

forma, se logró encontrar un modelo cuyas predicciones contienen una zonas en el intervalo [0,0, 0,5] donde no se encuentran falsos negativos.

Para la implementación de los modelos y el proceso de entrenamiento, se hizo uso del framework TensorFlow y Keras. Posteriormente, los modelos seleccionados fueron convertidos mediante TensorFlow Lite para poder correrlos en un microcontrolador. Para esto último, se hizo uso de la librería de C++ TensorFlow Lite Micro.

Finalmente, para poder inferir nuestro dataset en el modelo que corre en un microcontrolador, fue necesario desarrollar un setup que permita entregar las muestras a nuestro modelo y además medir el consumo energético del mismo.

Palabras claves

Inteligencia Artificial - Machine Learning - Tiny Machine Learning – Temporal Convolutional Network - Internet of things o Internet de las cosas – Edge computing – Cloud computing – Edge Artificial Inteligence - Microcontroladores

1. Introducción

El término eHealth define al conjunto de Tecnologías de la Información y la Comunicación (TICs) que se aplican en la atención médica y en los servicios de la salud en materia de prevención, diagnóstico, tratamiento, seguimiento, gestión y administración, con el fin de mejorar la eficiencia del sistema sanitario, optimizar recursos y mejorar la accesibilidad. eHealth abarca una amplia variedad de posibilidades y áreas, como ser registros electrónicos de la salud, historia clínica electrónica, telesalud, telemedicina, asistencia sanitaria virtual, sistemas de información del hospital (HIS), sistema de información de laboratorio (LIS), entre otros.

La telemedicina es un concepto que se encuentra dentro de la telesalud y ésta se encuentra dentro de eHealth. La telemedicina puede definirse como la prestación de servicios de salud en donde la distancia es uno de los principales factores críticos, pero no el único. Según un informe publicado por la Revista Médica del Uruguay (Chá Ghiglia, 2020),

> el término telemedicina se refiere a la parte clínica de la telesalud (prevención - diagnóstico - tratamiento - monitoreo), corresponde a la práctica médica, realizada a distancia, en tiempo real o diferido, por intermedio del uso de las TICs, donde uno de los actores es integrante del equipo de la salud y el otro es un médico o paciente, o ambos.

Y concluye que

> la telemedicina puede contribuir a la mejora de los servicios de salud cuando su uso se planifica en función de las necesidades de los usuarios, los recursos y de la organización. Su implementación puede mejorar la accesibilidad a la atención médica en los diferentes niveles de atención con un menor gasto de tiempo y dinero para los pacientes y las organizaciones.

La telemedicina es una modalidad relativamente nueva. En Uruguay, se promulgó la ley N° 19.869 que aprueba los lineamientos generales para la implementación y el desarrollo de la telemedicina como prestación de los servicios de salud, en abril de 2020.

Uno de los conceptos que la telemedicina aplica como herramienta es el IOT, el cual juega un papel de gran importancia por las prestaciones que brinda y consecuentemente los beneficios que genera en el cuidado de la salud de los pacientes. Esto implica un amplia gama de dispositivos que se encuentran conectados a la nube, que pueden ser usados por los pacientes desde la comodidad de su hogar, o incluso durante sus actividades de rutina diarias, que registran mediante sensores distintos signos vitales del paciente como ser presión arterial, temperatura, actividad cardiaca, y que son transmitidos a la nube. Esto permite conocer el estado del paciente en todo momento, brindando la posibilidad de monitorear a distancia al paciente en tiempo real o registrar masivamente datos para su análisis y seguimiento por parte del médico, por ejemplo en instancias posteriores a una internación hospitalaria. Existen otras aplicaciones en los que los datos son también de acceso para el usuario, para que éste pueda llevar un cuidado y un control de su salud de manera independiente y rigurosa, como puede ser el caso de pacientes diabéticos, los cuales pueden monitorear el nivel de azúcares en sangre en todo momento y tomar medidas ante alguna alteración. Otro caso es el del personal de

urgencias, quien puede contar con dispositivos para transmitir EEG, EKG y otras lecturas, al personal del hospital mientras se desplazan. Los especialistas pueden aconsejar tratamientos inmediatos al personal de urgencia y el personal del hospital puede prepararse mejor para la llegada del paciente. También, puede facilitarse el acceso a la atención sanitaria a pacientes que se encuentran en áreas rurales, que no cuentan con centros hospitalarios ni doctores radicados en alrededores cercanos.

Galeno_Sys es un proyecto que se está desarrollando actualmente en nuestro país, y es un caso concreto de aplicación de IOT en la salud. Es un proyecto desarrollado por le empresa DataFlow SRL, a cargo del ingeniero electrónico Hugo Articardi, que ha sido incubado en la incubadora ThalesLab con el apoyo de la Agencia Nacional de Investigación e Innovación (ANII). Según se explica en el sitio web del proyecto,[4]

> Galeno_Sys es una herramienta de telemedicina para el monitoreo cardiológico remoto de pacientes en tratamiento o diagnóstico, basada en el concepto de eHealth y en la tecnología Internet of Things. Este sistema permite el monitoreo cardiológico de pacientes que estén hospitalizados, en internación domiciliaria o realizando ejercicios de rehabilitación, de forma inalámbrica e independiente de su ubicación geográfica, favoreciendo la accesibilidad, universalidad y calidad de un servicio centrado en el paciente. El sistema está compuesto por un dispositivo, un servidor y una plataforma web. Por medio de cuatro electrodos colocados en el tórax del paciente, el dispositivo toma la información electrocardiográfica y la envía al servidor. Esta información es accesible para el personal de salud por medio de la plataforma web que permite la gestión y visualización remota de las gráficas de hasta seis derivaciones del ECG de forma histórica y en tiempo real. Las principales aplicaciones de Galeno_Sys son: monitoreo cardiológico intrahospitalario, monitoreo cardiológico en internación domiciliaria, Wifi-Holter para registros prolongados en el tiempo, llevar atención de calidad a centros poco poblados sin especialistas radicados.

La tecnología y la medicina han sido cómplices en el crecimiento y auge de distintas tecnologías aplicadas en la salud. Los avances tecnológicos van modificando el concepto de salud y las necesidades sanitarias están influyendo en el desarrollo de la tecnología. La Inteligencia Artificial (IA) es otra de las tecnologías que fuertemente ha tomado terreno en la salud, en sus distintas áreas, como ser la asistencia sanitaria (prevención de enfermedades, diagnóstico, tratamiento y seguimiento de pacientes), investigación y formación académica. En el área de asistencia se encuentran ejemplos de algoritmos aplicados en la prevención y diagnóstico de cáncer de cuello uterino, próstata y piel, detección de cardiopatías en registros electrocardiográficos, predicción de riesgos de psicopatologías según el contenido de las publicaciones en redes sociales de un paciente. En conjunto con la robótica, se han desarrollado robots para seguimiento y acompañamiento de pacientes dotados de sistemas de visión artificial para su movilización y detección de emociones por reconocimiento facial, como también sistemas de escucha e interpretación de la voz. En el campo de la docencia y formación continua, la IA ofrece crear escenarios virtuales de entrenamiento y aprendizaje

[4] http://galenosys.com.uy/

que simulan una intervención real con gran realismo sin riesgo. Un caso típico es la simulación de cirugías ya sea para el aprendizaje de estudiantes como para el entrenamiento de profesionales ante cirugías complejas. Permite evaluar el progreso del estudiante ajustando pruebas sucesivas a los logros alcanzado previamente. En el área de investigación, en la actualidad hay en marcha numerosos estudios, la mayoría ensayos clínicos, que intentan buscar pruebas de la utilidad de la aplicación de la IA en salud. Empresas como Google, Apple y Amazon se encuentran trabajando en conjunto con distintas universidades para la utilización de terminales y software específico en el seguimiento y control de numerosas enfermedades y procesos (Ávila-Tomás, Mayer-Pujadas, y Quesada-Varela, 2021).

En la búsqueda de una idea a desarrollar como proyecto final de aprobación y finalización del Postgrado en Robótica e Inteligencia Artificial dictado por UTEC y FURG, surgió el contacto entre estudiantes y coordinadores académicos del PRIA, y la empresa GeneXus Consulting (empresa socia de DataFlow en el proyecto Galeno_Sys). Desde GeneXus Consulting manifestaron la inquietud por parte de la empresa de comenzar a implementar IA en sus proyectos y presentaron el proyecto Galeno_Sys como posibilidad de generar el primer suceso de este hito empresarial fusionándolo con nuestra necesidad de ejecutar un proyecto acorde al programa de estudio del PRIA. Fue así que surgieron 3 proyectos independientes pero en comunión entre sí y con Galeno_Sys. Entre ellos este proyecto, titulado «EDGE AI aplicado a la telemedicina para el monitoreo de señales ECG».

En este proyecto, se propone desarrollar una forma de extender las prestaciones que brinda actualmente el sistema Galeno_Sys, permitiendo que el dispositivo tenga la capacidad de clasificar las señales cardíacas en normales y anormales, localmente. Esto implica descentralizar el procesamiento de datos, al analizar los datos en primera instancia en el propio dispositivo electrónico. Este concepto se denomina Edge Computing, y significa que la computación y el procesamiento de datos ocurre parcial o totalmente en el borde (del inglés edge) de la red, es decir, localmente en el dispositivo o sensor.

En el año 2015 habían casi 4 billones de dispositivos IOT conectados a internet, actualmente hay 12,3 billones, y se proyecta que para el año 2025 hayan cerca de 30 billones de dispositivos IOT conectados.[5] El crecimiento exponencial de dispositivos IOT implica un crecimiento exponencial de tráfico de datos hacia y desde la nube. Como consecuencias, las arquitecturas basadas en Cloud Computing, esto es, arquitecturas que centralizan los datos y su procesamiento en servidores en la nube, corren el riesgo de volverse obsoletas para manejar una gran cantidad de datos y atender la demanda de tantos dispositivos. Esto puede resultar en tiempos de respuestas prolongados que son inadmisible para muchas aplicaciones IOT, en particular, en la salud.

Edge Computing propone delegar tareas de procesamiento en los dispositivos, obteniéndose las siguientes ventajas:

- **Ancho de banda:** En arquitecturas donde se cuenta con una gran cantidad de dispositivos conectados a una red local, o en casos donde los dispositivos generan datos en formatos pesados como ser imagen y video, se puede volver inviable enviar todos los

[5]IOT Analytics. *State of IoT 2021: Number of connected IoT devices growing 9 % to 12.3 billion globally, cellular IoT now surpassing 2 billion. (2021).* Disponible en textthttps://iot-analytics.com/number-connected-iot-devices/

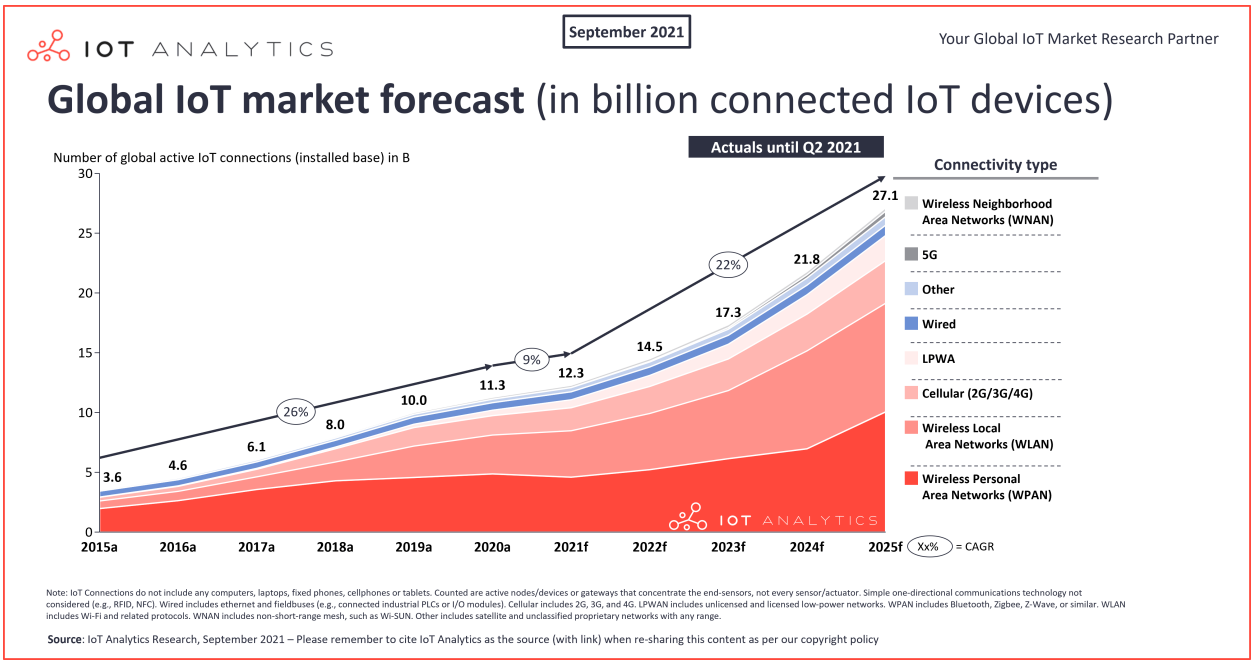

Figura 1: Proyección del crecimiento del mercado IOT global. Adaptado de IOT Analytics (https://iot-analytics.com/number-connected-iot-devices/)

datos a la nube debido a limitaciones del ancho de banda. Procesar datos localmente disminuye notoriamente el tráfico de datos y elimina el compromiso del ancho de banda.

- **Latencia:** La latencia es el tiempo de respuesta del dispositivo frente a una entrada de datos. Es decir, es el tiempo entre que se genera una nueva lectura de datos y que se genera una salida en respuesta a los mismos. Existen aplicaciones como ser los vehículos autónomos, en los que se debe minimizar los tiempos de respuesta, donde la respuesta debe ser en tiempo real. Por lo tanto, no se puede enviar datos a la nube y esperar una respuesta. Procesar datos localmente disminuye notoriamente el tiempo de respuesta de los dispositivos.

- **Costos:** Tanto el tráfico de datos a la nube como la cantidad de trabajo ejecutado en la nube tienen un costo económico y energético. Procesar los datos en el dispositivo, disminuye la tráfico de datos y la carga de trabajo en la nube, disminuyendo en consecuencia el costo y el consumo.

- **Fiabilidad:** Dispositivos que procesan datos por si mismos, se vuelven más robustos y fiables frente a aquellos que dependen de una conexión estable y permanente a internet para el envío de datos a la nube. Los primeros pueden seguir operando aunque carezcan de una conexión a internet, mientras que los segundos se vuelven inútiles.

- **Privacidad:** Evitar que los datos salgan del dispositivo y viajen a la nube por internet, disminuye los riesgos de hacking y violación de la privacidad.

Poder ofrecerle al proyecto Galeno_Sys la posibilidad de contar con las ventajas que Edge Computing brinda, fue lo que motivó a desarrollar la idea y la propuesta de este proyecto. Aplicado a Galeno_Sys, integrar en su ecosistema IOT un dispositivo capaz de clasificar señales ECG entre normales y anormales puede, por un lado, decidirse no enviar a la nube aquellas señales que son clasificadas con una alta probabilidad como normales, y así reducir el costo y el consumo requeridos para enviar datos a la nube. Por otro lado, puede ganarse tiempo en la detección de señales anormales y reducir el tiempo de asistencia al paciente. Actualmente, considerando que el paciente presenta señales cardíacas anormales, los doctores no serán notificados hasta que los datos arriben a la nube y sean procesados, viéndose este proceso demorado en caso de que el dispositivo no cuente con una conexión a internet. Esta demora puede reducirse si se dota de inteligencia al dispositivo. De todos modos, queda fuera del alcance de este proyecto definir el comportamiento del sistema frente a la detección con alta probabilidad de señales anormales. Sin embargo, algunas opciones podrían ser, enviar un SMS al doctor, notificar a algún miembro de la familia via bluetooth, o mostrar al paciente un mensaje de forma sutil y cautelosa de que el dispositivo no recolecta datos correctamente y que debe contactar al doctor indicándole el código «XX».

2. Objetivos

2.1. Objetivo

El objetivo general de este trabajo ha sido desarrollar un modelo de machine learning capaz de clasificar señales ECG entre normales y anormales, enfocado en los mejores scores, esto es, próximos a 0 y 1, de forma de minimizar los riesgos de falsos positivos, y especialmente de falsos negativos. Y que sea lo suficientemente pequeño y liviano para correr en microcontroladores con recursos limitados. De forma de hacer posible su implementación en dispositivos IOT, dotarlos con la capacidad de procesar señales ECG localmente y permitirles gozar de los beneficios que el Edge Computing brinda.

2.2. Metas

- Encontrar un dataset acorde a nuestras necesidades. Esto es, un dataset compuesto con señales iguales en características a las señales adquiridas por el dispositivo Galeno_Sys.
- Estudiar el estado del arte en la clasificación de señales ECG mediante técnicas de machine learning, en particular, modelos de tiny ML.
- Obtener uno o más modelos que puedan clasificar señales normales y anormales, brindando resultados fiables, esto es, logrando predicciones lo más próximas a 0 y 1 posible, y reduciendo los riesgos de falsos negativos y falsos positivos.
- Convertir los modelos a tensorflow lite para que corran en microcontroladores, haciendo uso de TensorFlow Lite For Microcontrollers.
- Inferir muestras al modelo que corre en micocontroladores y medir el consumo energético del microcontrolador.

3. Revisión bibliográfica

3.1. Datasets

En todo trabajo de machine learning es importante contar con una fuente de datos fiable y representativa de la realidad. Es decir, los datos con los se entrenan los modelos de machine learning deberían ser tales que representen la totalidad de casos con los que el modelo pueda llegar a tratar en un entorno de producción, de forma de poder generalizar el funcionamiento y eliminar todo sesgo.

El dispositivo Galeno_Sys, realiza la adquisición de la señal ECG con las siguientes características: frecuencia de muestreo de 100 Hz, profundidad de bit de 8 bits, resolución 1/64 mV, muestras de 10 segundos de duración.

Teniendo estas características presente, se realizó la búsqueda de datasets públicos que puedan ser fuente de datos para nuestros modelos. A continuación, se presentan los dataset encontrados y utilizados en este trabajo.

PTB-XL, a large publicly available electrocardiography dataset (Wagner y cols., 2020) Se trata de un dataset público, compuesto por 21837 señales ECG de 12 derivaciones de 10 segundos de duración, adquiridas de 18885 pacientes. Es de los dataset más grandes hasta el momento, publicado en 2020, pero que reúne muestras tomadas entre octubre de 1989 y junio de 1996. Se encuentra balanceado respecto al sexo, donde el 52 % de los pacientes son de sexo masculino y el 48 % restante son de sexo femenino, y cubre un amplio rango de edades que va de 0 a 95 años.

Las señales fueron digitalizadas con una profundidad de 16 bits y una resolución de 1 μV/LSB. Originalmente, las señales fueron adquiridas a una frecuencia de muestreo de 400 Hz, pero el dataset se encuentra publicado en 2 versiones: una versión upsampled a 500 Hz, y una versión downsampled a 100 Hz.

Cada muestra del dataset tiene asociado un reporte en formato SCP-ECG, el cual asigna etiquetas dentro de 3 categorías: diagnóstico, forma y ritmo. Además brinda una probabilidad de certeza para cada etiqueta asignada. Dentro de la clasificación por diagnóstico, existen 44 tipos de diagnósticos diferentes. Dentro de la categoría forma, existen 19 tipos de forma dentro de los cuales 4 coinciden con el tipo de diagnóstico, y dentro de la categoría ritmo existen 12 tipos de ritmos que describen el ritmo cardiaco. Esto da un total de 71 etiquetas únicas de reporte SCP-ECG. Los reportes SCP-ECG surgen a partir de un reporte textual generado inicialmente para cada muestra, de los cuales:

- 67.13 % son reportes generador por la interpretación manual de la señal por parte de un cardiólogo.
- 31.2 % son reportes generados por la interpretación automática de dispositivos ECG.
    - 4.45 % validados por un cardiólogo.
    - 26.75 % información incompleta respecto a la validación humana.
- 1.67 % no cuentan con un reporte inicial.

Por último, los autores de este dataset, realizaron una división de las muestras en 10 folds (stratified fold) asignándole a cada muestra un número del 1 al 10. Esta división se realizó como recomendación ante la necesidad de partir el dataset en train, validation and test, indicando que los folds 9 y 10 están compuestos por muestras de alta calidad, clasificadas y validadas por al menos un cardiólogo, y que por tanto recomiendan usarlos como folds de validación y testeo respectivamente.

Finalmente, las muestras del dataset están publicadas en formato WFDB (WaveForm DataBase), formato propuesto por PhysioNet.

China Physiological Signal Challenge 2018 dataset (Liu y cols., 2018) Se trata del dataset utilizado durante la competición 1st China Physiological Signal Challenge 2018. Cuenta con 9831 registros de señales ECG de 12 derivaciones, tomados de 9458 pacientes de 11 hospitales diferentes, cuyas duraciones varían entre 6 y 60 segundos. Estos registros se dividen en 9 clases de señales, siendo una de ellas señales normales y las 8 restantes distintos tipos de señales anormales. Sin embargo, este dataset fue dividido en 2 partes, training y test, de donde solo existe acceso al dataset de training, mientras que el dataset de test fue reservado para evaluación de la competencia. El set de entrenamiento, contiene 6877 señales ECG tomadas de 3178 mujeres y 3699 hombres.

La frecuencia de muestreo para la adquisición de las señales fue de 500 Hz. Todos los registros de este dataset se encuentran provistos en formato MATLAB, en archivos .mat.

Tanto en el set de training como en el set de test, las señales ECG fueron divididas en 9 tipos de señales. En la mayoría de los casos, cada registro tiene solo una etiqueta, mientras que algunos registros tienen 2 y hasta 3 etiquetas. Existen 477 y 203 casos de etiquetas múltiples en el set de entrenamiento y en el test respectivamente.

El dataset cuenta con las siguientes señales y cantidades de muestras:

Acrónimo	Nombre	# muestras training	# muestras test
Normal	Normal	918	394
AF	Atrial fibrillation	1098	469
I-AVB	First-degree atrioventricular block	704	301
LBBB	Left bundle brunch block	207	90
RBBB	Right bundle brunch block	1695	729
PAC	Premature atrial contraction	574	250
PVC	Premature ventricular contraction	653	281
STD	ST-segment depression	826	354
STE	ST-segment elevated	202	86

Cuadro 1: Distribución del número de muestras para cada dataset, según el tipo de señal.

A 12-lead electrocardiogram database for arrhythmia research covering more than 10,000 patients (Zheng y cols., 2020) Este dataset fue creado bajo el auspicio de Chapman University y Shaoxing People's Hospital (Shaoxing Hospital Zhejiang University School of Medicine). Contiene registros de señales ECG de 12 derivaciones de 10 segundos de duración, adquiridos de 10646 paciente, incluyendo 5956 hombres y 4690 mujeres. De estos,

el 17 % tenía ritmo sinusal normal y el 83 % tenía al menos una anomalía. Los grupos etarios de prevalencia fueron 51-60, 61-70 y 71-80 años de edad, representando 19,82 %, 24,38 % y 16,9 % respectivamente.

Los registros fueron adquiridos a una frecuencia de muestreo de 500 Hz, con una profundidad de bit de 32 bits, y una resolución o valor de escalón de 4.88 μV. Los autores disponen de 2 versiones de datos, una versión de los registros originales y una segunda versión de los datos pre procesados para el filtrado y la reducción de ruido.

Este dataset contiene 11 ritmos cardíacos y 56 condiciones cardiovasculares adicionales, etiquetados por hasta 3 cardiólogos. Los ritmos cardíacos y el número de registros para cada caso se muestran a continuación.

Acrónimo	Nombre	# muestras
SB	Sinus Bradycardia	3889
SR	Sinus Rhythm	1826
AFIB	Atrial Fibrillation	1780
ST	Sinus Tachycardia	1568
AF	Atrial Flutter	445
SI	Sinus Irregularity	399
SVT	Supraventricular Tachycardia	587
AT	Atrial Tachycardia	121
AVNRT	Atrioventricular Node Reentrant Tachycardia	16
AVRT	Atrioventricular Reentrant Tachycardia	8
SAAWR	Sinus Atrium to Atrial Wandering Rhythm	7

Cuadro 2: Distribución del número de muestras, según el tipo de señal.

3.2. Trabajos relacionados

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling (Bai, Kolter, y Koltun, 2018) Se trata de un trabajo de investigación publicado en el año 2018, en el cual, los autores demuestran de forma empírica, que las arquitecturas de redes convolucionales pueden alcanzar mejores resultados que las arquitecturas de redes recurrentes en tareas de modelos secuenciales. Para esto realizaron una evaluación sistemática de arquitecturas convolucionales y recurrentes generales para modelos secuenciales. Las arquitecturas fueron evaluadas a través de un amplio rango de tareas que son comúnmente utilizadas como patrón o referencia en la evaluación de redes recurrentes, que comprenden tareas de modelado de música polifónica, modelado del lenguaje a nivel de palabras y de caracteres, y pruebas de test sintético. Y concluyeron, que la común asociación entre modelos secuenciales y redes recurrentes debería ser reconsiderada, y que las redes convolucionales deberían ser consideradas como un punto de partida y de gran potencial para tareas de modelos secuenciales.

Su motivación surgió de trabajos publicados por otros autores durante los años 2016 y 2017, donde indicaron que ciertas arquitecturas convolucionales alcanzan precisiones del estado del arte en tareas de síntesis de audio, modelado del lenguaje a nivel de palabras, y

traducción automática. Esto llevó a los autores aquí citados a preguntarse si estos resultados son específicos de las aplicaciones estudiadas o son resultados que se podrían generalizar.

En representación de redes convolucionales, los autores aplicaron una arquitectura de red convolucional denominada Temporal Convolutional Network (TCN), la cual es comparada con arquitecturas recurrentes canónicas como son las redes LSTM y GRU. Los resultados experimentales indicaron que los modelos TCN alcanzaron sustancialmente mejores resultados que las arquitecturas recurrentes genéricas como son las LSTM y GRU. Además, las redes TCN exhibieron un mayor capacidad de memoria que las arquitecturas recurrentes, siendo más simples y claras.

La clasificación de series temporales, como es el caso de la clasificación de señales ECG, es una tarea de modelos secuenciales. Por tanto, si bien este trabajo no aplica a tareas de clasificación de señales ECG, determina fuertemente el punto de partida de nuestro trabajo.

An ECG Signal Classification Method Based on Dilated Causal Convolution (Ma y cols., 2021) Este paper es uno de los pocos antecedentes encontrados en la aplicación de TCN en tareas de clasificación de señales ECG. Los autores de este paper partieron de la arquitectura TCN presentada en (Bai y cols., 2018), y le realizaron pequeñas modificaciones para mejorar los resultados obtenidos, nombrando a este nuevo modelos como *Improved Dilated Causal Convolutional Network*. No se detallan aquí las modificaciones mencionadas dado que aún no se ha realizado una descripción detalla de la composición de las redes TCN, la cual se hará en la sección siguiente.

Respecto a los datos, los autores trabajaron con el dataset MIT-BIH AFDB(Goldberger y cols., 2000). Está compuesto por 25 registros de señales ECG tomados de pacientes con fibrilación auricular. Cada registro tiene una duración de 10 horas, y contiene 2 señales ECG, cada una muestreada a una frecuencia de 250 Hz con 12 bits de resolución en un rango de ±10 mV. En primer lugar, los autores realizaron un procesamiento de filtrado para la eliminación del ruido. En segundo lugar, realizaron un proceso de normalización de las muestras. En tercer lugar, realizaron un proceso de segmentación en 2 etapas. Primero segmentaron las señales siguiendo las anotaciones originales según el tipo de señal, de forma de que obtuvieron 288 señales normales, 291 señales con fibrilación atrial y 14 señales con aleteo auricular. Luego, segmentaron las señales en señales de 4 segundos de duración y descartaron las señales que resultaron con una duración menor a los 4 segundos. De esta forma, obtuvieron un dataset con las siguientes características:

Tipo	# muestras
Normal	124808
Fibrilación auricular	83626
Aleteo auricular	1449
Total	209883

Cuadro 3: Distribución de las muestras luego del pre procesamiento

Por último, dividieron las muestras en 5 folds para aplicar *crossvalidation* durante el entrenamiento. Como resultado, obtuvieron una exactitud (*accuracy*) de 98,65 %, una sensibilidad (*sensitivity*) de 98,79 % y una especificidad (*specificity*) de 99,04 %. Comparado con

otras 6 publicaciones que trabajaron sobre el mismo dataset, obtuvieron los mejores resultados de exactitud y especificidad, y el segundo mejor resultado respecto a la sensibilidad. Y presentan también una mejor relación entre exactitud y tiempo de inferencia del dataset de testing, respecto a arquitecturas basadas en redes recurrentes y/o redes convolucionales convencionales.

En relación a nuestro trabajo, la publicación aquí citada permitió confirmar que la selección como punto de partida de un modelo TCN para el procesamiento de señales ECG, es una buena y prometedora elección. Si bien en esta publicación se implementa un modelo mejorado de TCN para el procesamiento de señales ECG, se enfoca en la detección de fibrilaciones auriculares solamente, debido al dataset utilizado, el cual carece de otros tipos de arritmias. En cambio, en nuestro trabajo se objetivizó la detección de cualquier arritmia posible, ampliando el rango de posibilidades, y no enfocarse solo en fibrilaciones auriculares. Y además, nuestro trabajo busca modelos que se puedan embeber en microcontroladores y correr con los más mínimos recursos posibles. Esto no fue interés de la publicación aquí citada.

Temporal Convolutional Networks for Anomaly Detection in Time Series (He y Zhao, 2019) Tal como su nombre lo indica, este paper expone la aplicación de redes TCN para la detección de anomalías en series temporales. Los autores, realizan esta aplicación en 3 campos diferentes, siendo uno de ellos las detección de anomalías en señales ECG. En particular, el enfoque del trabajo presentado en este paper difiere al nuestro en cuanto a que presentan una solución al problema de detección de anomalías en series temporales usando aprendizaje no supervisado, donde la red TCN es entrenada solo con muestras de señales ECG normales. De todos modos, este paper se encuentra sumamente resumido y omite detalles importantes, como ser, el dataset utilizado en la detección de anomalías en señales ECG. Si bien no aporta demasiado valor a nuestro trabajo, se reconoce como un antecedente en la aplicación de redes TCN para la clasificación de señales ECG.

Convolutional-Recurrent Neural Network on Low-Power Wearable Platform for Cardiac Arrhythmia Detection (Faraone y Delgado-Gonzalo, 2020) Este paper presenta un antecedente semejante al trabajo que se ha intentado realizar en nuestro proyecto, debido al marco de trabajo y a los objetivos perseguidos. Los autores identifican claramente los beneficios del *Edge Computing*, de poder embeber modelos de redes neuronales en dispositivos wearable de bajo consumo y de recursos limitados, que habilitan una gran cantidad de aplicaciones de cuidado de la salud a nivel personal como para tratamientos clínicos.

La plataforma utilizada fue el *System on Chip (SoC)* nRF52832 de Nordic Semiconductor. Presenta una arquitectura ARM Cortex-M4 a 64 MHz, equipada con 64 Kb de RAM y 512 Kb de Flash. Incluye Float Point Unit (FPU) y soporta instrucciones de tipo Single Instruction Multiple Data (SIMD). Para el desarrollo del firmware y la implementación del modelo, se hizo uso de CMSIS. CMSIS es una librería que provee de una hardware abstraction layer (HAL) para microcontroladores de aquitectura ARM Cortex. Contiene además una librería llamada CMSIS-NN para la implementación de redes neuronales en microcontroladores, que soporta un rango básico de topologías de capas, como ser convolutional layers, dense layers, and pooling layers, varias funciones de activación, incluyendo *tanh* y *sigmoid*, y una versión

modificada de $softmax$ con potencia de 2 en vez de e.

El dataset utilizado consiste de 8528 muestras de señales ECG de una única derivación, usado como dataset de referencia en la competencia *Computing in Cardiology 2017 Challenge*(Clifford y cols., 2017). Las señales ECG fueron muestreadas a una frecuencia de 300 Hz, y tienen una duración variable de entre 9 y 60 segundos. Cada registro está etiquetado con una de las 4 clases: señal normal, fibrilación atrial, ruidosa, otro ritmo. Estas clases se encuentran desbalanceadas, con una mayoría de señales normales, y débilmente etiquetadas, ya que la etiqueta está asociada a la señal entera y no se tiene información de exactamente dónde ocurre la arritmia. De este dataset, los autores tomaron 1528 muestras con igual cantidad de registros por clase como dataset de testing, y utilizaron las 7000 muestras restantes como dataset de training. Varios pasos de pre procesamiento fueron aplicados al dataset. Primero se aplicó una etapa de filtrado. Luego se realizó un proceso de downsampling a 107 Hz, en orden de disminuir la carga de trabajo final y unificar a la frecuencia de muestreo implementada por el dispositivo de adquisición usado para demostraciones internas. Posteriormente, las muestras se normalizaron, y se segmentaron en señales de 256 muestras, con solapamiento del 50 %.

Sobre el modelo, los autores tomaron un modelo de red neuronal convolucional-recurrente que extrajeron del trabajo desarrollado por otros autores, diseñado para detectar y clasificar arritmias cardíacas en señales ECG, y lo adaptaron para que pueda correr en microcontroladores de bajo consumo y recursos limitados. La red neuronal está formada por dos partes. La primera consiste de 7 capas convolucionales 1D con un kernel de 5 elementos, cada una seguida de una capa de average pooling con tamaño de kernel y stride igual a 2. El número de canales se mantiene múltiplo de 8 para maximizar la aceleración de CMSIS-NN. Una capa global de average pooling se aplica al final de esta primera etapa. La salida de esta primera etapa consiste de un tensores de dimension 128. La segunda etapa consiste de una red GRU de 64 unidades ocultas, y un dropout del 50 % aplicado a las compuertas internas. Finalmente una capa de salida de 4 neuronas con $softmax$ como función de activación. El modelo entero consiste de 194.596 parámetros. Considerando que estám cuantizados en 8 bits, el tamaño total de memoria ocupado es apenas menor a 200 Kb.

El entrenamiento fue realizado usando Keras y Tensorflow, categrorical cross-entropy como función de error o pérdida, y Adam como optimizador. Después de entrenar con 250 epochs, la exactitud del modelo sin cuantizar es de 86,1 % el dataset de testing, y de 85,7 % para el caso del modelo cuantizado a 8 bits.

El footprint final de la memoria Flash ronda los 210 Kb, el cual incluye los pesos, los bias, las rutinas de CMSIS-NN para correr la red y otras líneas de código para el setup inicial y configuración de la placa. En particular se trata de 195.6 Kb para los pesos, bias y cálculos intermedios. Para la memoria RAM, se utilizan 6,8 Kb para los cálculos intermedios de la red.

ECG-TCN: Wearable Cardiac Arrhythmia Detection with a Temporal Convolutional Network (Ingolfsson y cols., 2021) Por último, pero no menos importante, se encuentra este paper como otro antecedente a nuestro trabajo, donde los autores identifican claramente el rápido crecimiento y las ventajas del Edge Computing, y en particular su aplicación en soluciones para el cuidado de la salud y el monitoreo de bioseñales mediante

dispositivos wearables. Fue publicado en el año 2021, y tal como su título lo indica, implementa una arquitectura de red TCN que se ejecuta en dispositivos wearables con el fin de detectar arritmias cardíacas.

En concreto, los autores tomaron la arquitectura TCN presentada en (Bai y cols., 2018), realizaron algunas modificaciones que consideraron pertinentes, y entrenaron dicho modelo utilizando el dataset ECG5000. El modelo resultante, lo utilizaron en microcontroladores de bajo consumo y recursos limitados. Realizaron este último procedimiento utilizando frameworks distintos y plataformas distintas. Por un lado, utilizaron la placa de evaluación B-L475E-IOT01A STM32L4 Discovery Kit de ST Microelectronics, la cual está basada en un microcontrolador Cortex M4F con 1 Mb de memoria Flash y 128 Kb de memoria SRAM, a una frecuencia de 80 MHz. Con esta placa, exploraron 3 formas diferentes de implementar redes neuronales en microcontroladores Cortex M4F y presentaron sus ventajas y desventajas. Estas son: convertir un modelo TFLite con TFLite Micro, convertir un modelo TFLite con CUBE.AI, y convertir un modelo de Keras directamente con CUBE.AI. Por otro lado, utilizaron la placa GAPuino de GreenWave Technology, basada en su procesador GAP8. Es una plataforma de ultra bajo consumo basada en la arquitectura RISC-V, con doble dominio de cómputo. Uno de ellos destinados a tareas de control, con un núcleo CV32E40P y 512 Kb de memoria L2, y un dominio cluster de 8 núcleos CV32E40P para procesos computacionales en paralelo que demandan una alta carga de trabajo, y 80 Kb de memoria L1. La frecuencia de trabajo puede ir desde 32 KHz a 250 MHz.

Como fuente de datos, utilizaron el dataset ECG5000, el cual consiste en 5000 registros, donde cada registro es un pulso cardíaco y está formado por 140 muestras. Se encuentra particionado en training y test, donde train dataset contiene solo 500 registros y test dataset contiene 4500 registros. Originalmente, el dataset ECG5000 surge de un registro ECG de 20 horas de duración que forma parte del dataset BIDMC Congestive Heart Failure Database, y es el registro «chf07». Este registro fue pre procesado en 2 pasos. Primero se le extrajo cada pulso cardíaco, siendo en total 92584 pulsos cardíacos. Luego, todos los registros extraídos se hicieron de igual largo usando interpolación. Finalmente, 5000 pulsos cardiacos fueron seleccionados de forma estocástica. El paciente tenía fallas cardíacas severas y el tipo de clase de pulso fue anotado automáticamente. 5 tipos de pulsos fueron anotados: Normal, R-on-T Premature Ventricular Contraction, Premtaure Ventricular Contraction, Supraventricular Premature or Ectopic beat, y unclassified beat.

Para el desarrollo y entrenamiento del modelo, utilizaron TensorFlow y PyTorch para adaptarse a los diferentes frameworks de desarrollo. Para el entrenamiento, usaron categorical cross-entropy como función de pérdida o error, optimizador ADAM, learning rate de 0.001, batches de 20 muestras y 30 epochs.

Para las métricas de evalución, utilizaron accuracy y balanced accuracy $((Sensitivity + Sprecificity)/2)$ debido a que el dataset se encuentra desbalanceado. Los resultados obtenidos fueron un accuracy de 94,2 % y un balanced accuracy de 89,0 %. Por un lado, los autores compararon sus resultados con otros 8 métodos de clasificación basados en el dataset ECG5000, obteniendo en este caso las mejores métricas incluso con un modelo que tiene la menor cantidad de parámetros respeto a los otros 8 métodos. Por otro lado, compararon sus resultados con los resultado obtenidos en (Faraone y Delgado-Gonzalo, 2020) a pesar de utilizar datasets diferentes, y compararon los resultados obtenidos con ambas plataformas utilizadas. En resumen, la plataforma basada en el procesador GAP8 alcanzó los mejores

resultados respecto a las métricas y a la eficiencia energética, comparada con las plataformas basadas en arquitectura ARM Cortex M4F utilizada por los autores en este trabajo y en (Faraone y Delgado-Gonzalo, 2020)

3.3. Fundamentación teórica

3.3.1. Temporal Convolutional Networks

En este apartado, se presenta la arquitectura de redes TCN, según como fue presentada en (Bai y cols., 2018). Luego se presentarán las modificaciones realizadas en (Ma y cols., 2021) y en (Ingolfsson y cols., 2021).

Las redes TCN, son redes basadas en redes convolucionales, pero que tienen las siguientes características:

1. Las convoluciones son causales, lo que significa que solo elementos del pasado son tomados en cuenta. No existe información del futuro.

2. La arquitectura puede tomar una entrada de cualquier largo y mapearla a una salida del mismo largo, tal como sucede en las RNN.

Fueron diseñadas con el fin de combinar simplicidad, predicción auto regresiva y memoria de largo alcance.

Modelado de secuencias Antes de definir la estructura de la red, se remarca la naturaleza del modelado de secuencias. Supóngase, que dada una secuencia de entrada $x_0, ..., x_T$, se desea predecir la salida correspondiente $y_0, ..., y_T$ en cada instante de tiempo t. La limitante clave es que para predecir la salida y_t para un instante de tiempo t dado, solamente se cuenta con los elementos de las entradas que han sido consideradas anteriormente: $x_0, ..., x_t$. Formalmente, una red de modelado de secuencia es una función $f : \mathcal{X}^{T+1} \to \mathcal{Y}^{T+1}$ donde:

$$\hat{y}_0, ..., \hat{y}_T = f(x_0, ..., x_T) \tag{1}$$

si satisface la limitante causal por la cual y_t depende solamente de $x_0, ..., x_t$ y no de ninguna entrada futura $x_{t+1}, ..., x_T$.

Este formalismo deja afuera aquellos dominios de aplicaciones donde la secuencia de entrada es utilizada entera para predecir cada una de las salidas.

Convoluciones causales Las TCN están basadas en redes convolucionales 1D. Además, tal como se mencionó al inicio de esta sección, las TCN deben cumplir 2 principios. Por un lado, el hecho de que la red produce una salida del mismo largo que la entrada, y por otro lado, el hecho de que no puede haber fuga de información desde el futuro y hacia el pasado. Para alcanzar el primer principio, cada capa oculta implementa padding a la secuencia de entrada, pero solo al comienzo de la misma, agregando tantos ceros como el resultado de la diferencia $(k - 1)$, siendo k el tamaño del filtro o kernel. De esta forma, la activación de cada capa oculta y de la capa de salida, será de igual largo a la secuencia de entrada. Para alcanzar el segundo principio, las TCN implementan convoluciones causales, convoluciones

donde una salida en el instante de tiempo t involucra solamente el elemento de entrada en el instante de tiempo t y elementos anteriores a este. La figura 2 ilustra este comportamiento.

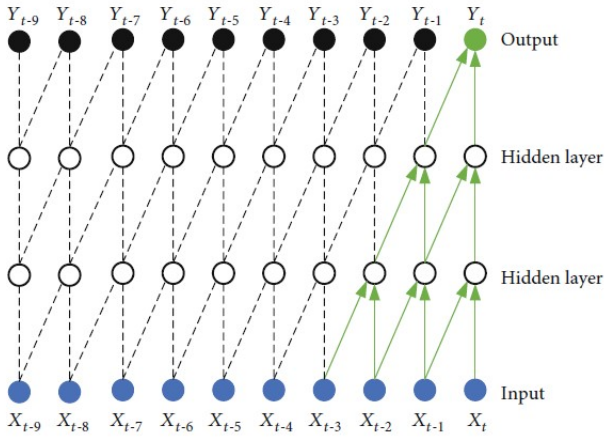

Figura 2: Imagen tomada de (Ma y cols., 2021) donde se representa la estructura una convolución causal.

La desventaja de la arquitectura descripta hasta aquí, es que en orden de requerir un efecto de memoria de largo plazo en el tiempo, se necesitaría una red extremadamente profunda, incrementando el tamaño de la red, lo que es contradictorio por ejemplo en nuestro caso, donde buscamos redes lo suficientemente pequeñas para embeber en dispositivos de recursos limitados.

Convoluciones dilatadas Una convolución causal simple como la mencionada en el párrafo anterior, es capaz de mirar elementos del pasado de forma proporcional a la profundidad de la red. Esto hace desafiante el diseño de aplicaciones que requieran retenciones de información de largo períodos de tiempo. Para evitar este compromiso y lograr memoria de largo alcance sin necesidad de incrementar la profundidad de la red, se emplean convoluciones dilatadas, las cuales permiten obtener un campo receptivo exponencialmente más largo. Más formalmente, dada una secuencia de entrada $x \in R^n$ y un filtro o kernel $f : \{0, ..., k-1\} \rightarrow R$, la operación de convolución dilatada F sobre el elemento s de la secuencia es definida como

$$F(s) = (x *_d f)(s) = \sum_{i=0}^{k-1} f(i) x_{s-di} \tag{2}$$

donde d es el factor de dilatación, k es el tamaño del filtro o kernel, y $s - di$ es el índice de los elementos de la secuencia hacia el pasado, es decir, anteriores al elemento s, que son tomados en cuenta en la convolución. El factor de dilatación indica el salto que se aplica entre los elementos consecutivos. Cuando $d = 1$, la convolución dilatada se reduce a una convolución convencional. Utilizando factores de dilatación más grande, permite que cada elemento de la salida, tome en cuenta un mayor rango de elementos de la entrada, lo que se denomina como el campo receptivo de entrada.

Hay 2 formas de incrementar el tamaño del campo receptivo de entrada, o en inglés, receptive field size (RFS): incrementando el tamaño del filtro k, o incrementando la fac-

tor de dilatación d. En la práctica, lo más común es incrementar el factor de dilatación exponencialmente con la profundidad de la red.

A continuación, la figura 3 ilustra el concepto de convoluciones causales dilatadas.

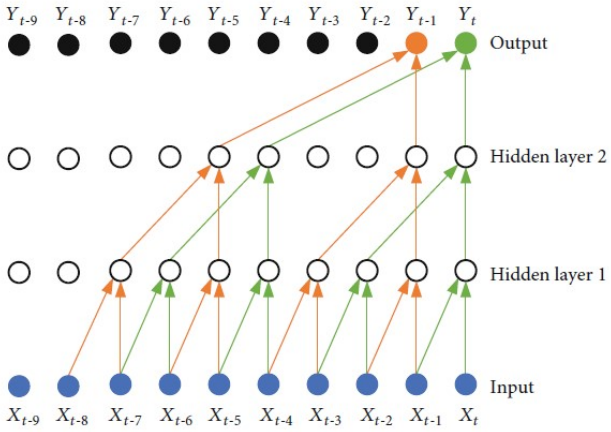

Figura 3: Imagen tomada de (Ma y cols., 2021) donde se representa la estructura de una convolución causal dilatada.

Bloques residuales Un bloque residual contiene una rama compuesta de una serie de transformaciones $\mathcal{F}$, y una rama que se encarga de sumar la salida de la rama anterior con la entada. Opcionalmente, esta última rama puede presentar una convolución 1x1 para igualar las dimensiones de la entrada y la salida antes de sumarlas entre si, elemento a elemento.

$$o = Activation(x + \mathcal{F}(x)) \tag{3}$$

El bloque residual es la unidad estructural base de toda TCN, de forma que las TCN se construyen conectando bloques residuales uno a continuación del otro en forma de pila o stack. De esta forma, se introduce un nuevo factor que permite incrementar RFS y es el número de bloques que contiene la red. Considerando un factor de dilatación que crece exponencialmente, dado por $d = 2^{L-1}$, donde L es el número de bloques residuales, el RFS se determina como:

$$RFS = 1 + 2(K_T - 1)(2^L - 1) \tag{4}$$

donde K_T es el tamaño del kernel o filtro.

Según la presentación en (Bai y cols., 2018), un bloque residual está compuesto por una capa de convolución dilatada, seguida por una capa de weight normalization, seguida de una capa de activación ReLu y posteriormente una capa dropout para regularización. Esta secuencia de funciones se aplica 2 veces, tal como se muestra en la figura 4.

Como se mencionó anteriormente, en una TCN la entrada y la salida pueden tener diferentes largos, por lo que se puede adicionar una convolución 1x1 para igualar dicha dimensión entre ambos tensores, previo a realizar la suma elemento a elemento.

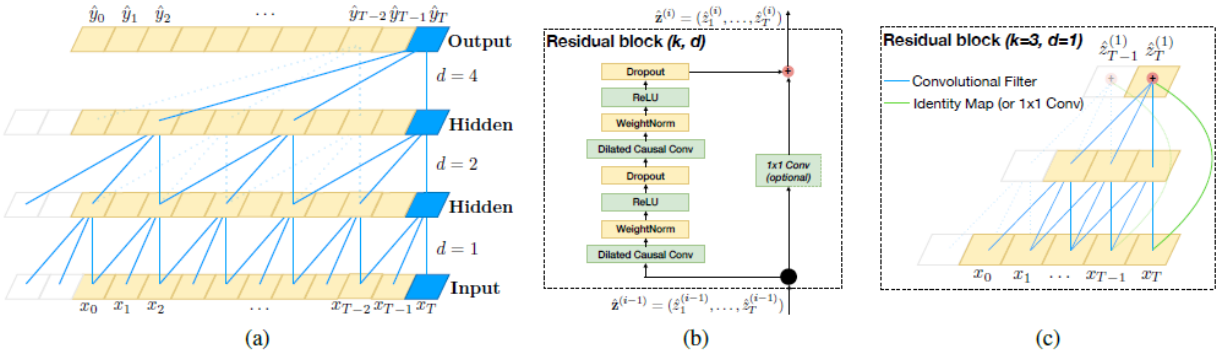

Figura 4: Imagen tomada de (Bai y cols., 2018) donde se representa la estructura de las TCN. (a) Convolución causal dilatada, con factores de dilatación $d = 1, 2, 3, 4$ y tamaño de filtro $k = 3$. (b) Bloque residual de una TCN. (c) Ejemplo de una conexión residual en una TCN.

Modificaciones propuestas en trabajos relacionados Una vez presentada la definición y las características de las TCN, se vuelve a citar a los trabajos relacionados con el fin de describir las modificaciones realizadas en cada uno de ellos, respecto a la arquitectura presentada en este apartado.

- En (Ma y cols., 2021), según los autores, lograron obtener mejores resultados luego de realizar las siguientes modificaciones:
  - Cada bloque residual contiene una única capa de convolución causal dilatada, una única capa de weight normalization, una única capa de activación ReLu, y una única capa dropout. Es decir, si bien respetaron los tipos de capa descriptas en (Bai y cols., 2018), emplearon solo una capa de cada tipo dentro del bloque residual en vez de dos.
  - Además, agregaron una nueva conexión residual entre la entrada de la red y la salida del último bloque residual.

  Estas modificaciones se ilustran en la figura 5.

- En (Ingolfsson y cols., 2021), los autores realizaron las siguientes modificaciones:
  - Uso de batch normalization en lugar de weight normalization.
  - Uso de batch normalization en la rama residual donde se inserta una convolución 1x1.
  - Uso de normal dropout en lugar de spatial dropout.

  Además, los autores agregaron una capa de convolución 1x1 a la entrada de la red TCN, es decir, antes del primer bloque residual. La figura 6 ilustra la arquitectura de la red implementada por los autores.

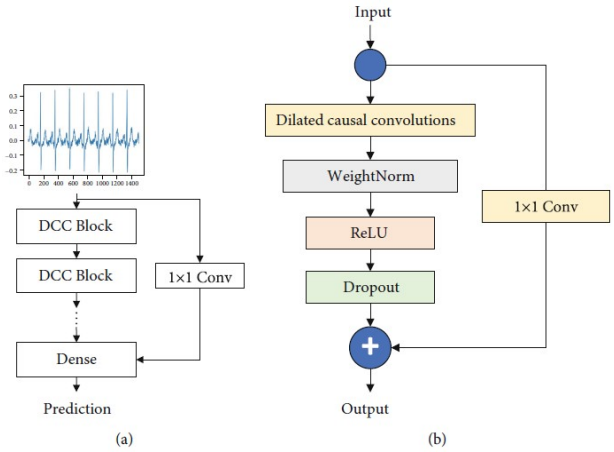

Figura 5: Imagen tomada de (Ma y cols., 2021). (a) Ilustración de la conexión residual agregada por los autores, que realiza un salto que comprende a todos los bloques residuales. (b) Bloque residual implementado por los autores.

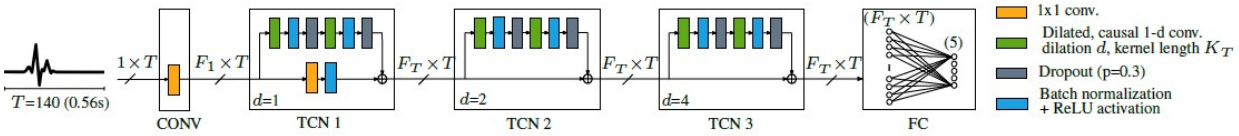

Figura 6: Imagen tomada de (Ingolfsson y cols., 2021) la cual ilustra la arquitectura TCN implementada por los autores. Donde $F_1 = 2$, $F_T = 11$, y $K_T = 11$.

3.3.2. TinyML

¿Qué es TinyML? En la web de la tinyML Foundation[6], puede encontrarse la siguiente definición:

> Tiny machine learning is broadly defined as a fast growing field of machine learning technologies and applications including hardware (dedicated integrated circuits), algorithms and software capable of performing on-device sensor (vision, audio, IMU, biomedical, etc.) data analytics at extremely low power, typically in the mW range and below, and hence enabling a variety of always-on use-cases and targeting battery operated devices.

TinyML es un área de rápido crecimiento cuyo objetivo es desarrollar las herramientas necesarias para poder implementar modelos de machine learning en dispositivos de ultra bajo consumo y recursos limitados. Con consumo ultra bajo, nos referimos a consumos en el orden del mW o menor. Con respecto a recursos limitados, nos referimos a memorias Flash de decenas de Kb o a lo sumo el Mb, memorias RAM de decenas de Kb y frecuencias de reloj por debajo del GHz.

En algunas publicaciones se refieren al tinyML como una nueva revolución dentro del machine learning. Luego de la explosión y el crecimiento de la inteligencia artificial debido

[6]https://www.tinyml.org/

a los avances tecnológicos de las últimas décadas, que permitieron grandes capacidades de cómputo, almacenamiento de grandes cantidades de datos, y desarrollo de modelos de millones de parámetros, surge en contrapartida una nueva área que busca desarrollar modelos de tan solo algunas decenas o centenas de Kb. ¿Por qué?. Porque poder correr modelos de machine learning en dispositivos en el edge, cerca de los sensores, permite el desarrollo de aplicaciones que gocen de los beneficios que el edge computing ofrece y que fueron mencionados anteriormente. Estos es, aplicaciones fiables y seguras, que brindad seguridad y rápidas respuestas, y energéticamente eficientes al evitar envío de información a través de comunicaciones inalámbrica. Esto ha despertado gran interés en la industria y en la academia.

Desafíos y metas para el crecimiento (C. R. Banbury y cols., 2020) (C. Banbury y cols., 2021) Si bien los avances logrados con el área de tinyML prometen desarrollar una nueva clase de aplicaciones inteligentes, el continuo progreso se ve limitado por la falta de recursos y herramientas generalizadas que permitan sistemáticamente medir, comparar, evaluar y mejorar la performance de estos sistemas, lo cual es necesario para avanzar y alcanzar un nivel de madurez en el área. En inglés se define como benchmarking.

La comunidad tinyML ha estado trabajando en la determinación de un benchmark suite, y en el proceso han identificado los siguientes desafíos:

- **Bajo consumo** El bajo consumo es una de las features de los sistemas tinyML. Por lo tanto, se debería poder determinar cuán eficiente energéticamente un dispositivo es. Sin embargo, hay algunas dificultades para medir el consumo de energía. Los dispositivos pueden consumir drásticamente diferentes cantidades de energía, por lo que para una misma aplicación no es posible mantener el mismo consumo para los distintos dispositivos. Determinar qué factores o parámetros se encuentran dentro del consumo de energía es difícil, además de que los periféricos o el firmware pueden impactar de distintas maneras entre dispositivos.

- **Memoria limitada** Mientras dispositivos de machine learning tradicional como ser smartphones o computadores, gozan de memorias en el orden de Gbs, los sistemas tinyML deben lidiar con recursos de memoria que son en el orden de 2 magnitudes más pequeñas. Desarrollar un benchmark suite general es difícil porque en algunos casos podría exceder la memoria de algunos dispositivos. Por el contrario, si se desarrollan becnhmarks individuales, habría que abarcar una amplia cantidad de dispositivos, por lo que habría que contemplar distintos niveles cuantización y precisión. Una variedad de benchmarks deberían elegirse para soportar la diversidad de casos.

- **Heterogeneidad de hardware** Los sistemas tinyML son todavía deversos en performance, potencia y capacidades. Esta heterogeneidad trae una serie de desafíos dado que el sistema bajo test podría no incluir características estándares, como un reloj del sistema o una interfaz de debug. Por lo tanto, la tarea de normalizar los resultados de performance a través implementaciones heterogéneas representa un desafío clave.

- **Heterogeneidad de software** Los sistemas tinyML están a menudo fuertemente ligados a sus stacks de inferencia y a sus herramientas de desarrollo. Para alcanzar la mayor eficiencia, los fabricantes frecuentemente desarrollan sus propias herramientas

para implementar y ejecutar modelos en sus propios hardware. Esto significa un desafío al diseñar un benchmark para estos sistemas, porque cualquier restricción, propuesta por el benchmark podría impactar negativamente en estos sistemas y obtener resultados no representativos. Por lo tanto, se debe lograr un balance entre optimización y portabilidad, y comparabilidad con representatividad.

MLPerf Tiny (C. R. Banbury y cols., 2020)(C. Banbury y cols., 2021) Distintos actores de la academia y la industria, han estado trabajando para poder generar un benchmark suite, que supere los desafíos que se mencionaron en el párrafo anterior. Para esto han seleccionado 4 áreas de aplicación priorizando la diversidad, de forma de poder contemplar tantos casos de uso como sea posible, el acceder a datasets y modelos open-source, y que sean relevantes para las aplicaciones del mundo real. En la tabla siguiente, se presentan los 4 casos de uso, el dataset propuesto, el modelo propuesto y la métrica target.

Use case	Dataset (Input Size)	Model (TFLite Model Size)	Quality Target (Metric)
Keyword Spotting	Speech Commands (49x10)	DS-CNN (52,5 Kb)	90 % (Top-1)
Visual Wake Words	VWW Dataset (96x96)	MobileNetV1 (325 Kb)	80 % (Top-1)
Image Classification	CIFAR10 (32x32)	ResNet (96 Kb)	85 % (Top-1)
Anomaly Detection	ToyADMOS (5x128)	FC-AutoEncoder (270 Kb)	0,85 (AUC)

Cuadro 4: Benchmarcks que forman parte de MLPerf Tiny

3.3.3. TensorFlow Lite for Microcontrollers

TensorFlow TensorFlow es una librería open-source de machine learning, desarrollada por Google y liberada al público en 2015. Con soporte para las plataformas desktop y server de Linux, Windows y macOS, ofrece una cantidad de herramientas para entrenamiento, optimización y uso de de modelos.

TensorFlow Lite Con la necesidad de contar con aplicaciones que corran modelos de machine learning en plataformas móviles, como ser Andorid e iOS, en 2017 Google comenzó a desarrollar TensorFlow Lite, una librería pensada para correr modelos de redes neuronales de forma eficiente y fácil en dispositivos móviles. Para reducir tamaño y complejidad, se eliminaron features que no son necesarias en estas plataformas. Por ejemplo, no soporta entrenamiento, solo correr inferencias de modelos que fueron previamente entrenados en la nube. Tampoco soporta todos los tipos de datos disponibles en TensorFlow. Adicionalmente, algunas de las operaciones no muy usadas frecuentemente fueron descartadas.

Como resultado, TensorFlow Lite puede ajustarse a tan solo unas pocas centenas de Kilobytes, haciendo más fácil su implementación en aplicaciones de tamaño limitado. También ofrece librerías de optimización para CPUs de la serie ARM Cortex-A, junto con soporte para la API Neural Network de Android. Otra ventaja clave es que tiene buen soporte para la cuantización de modelos a 8 bits.

TensorFlow Lite for Microcontrollers El equipo de Google notó que habían un montón de productos de Google y de fabricantes externos que podrían beneficiarse corriendo modelos

de machine learning en plataformas embebidas, sobre las cuales la librería TensorFlow Lite excedía aún las capacidades de estas plataformas. Para estas plataformas se necesita no exceder los 20 Kb. Así, en 2018 el equipo de Google comenzó a trabajar en una nueva versión de TensorFlow Lite destinada a sistemas embebidos, teniendo en cuenta los siguientes requerimientos:

- **No operating systems dependencies** Un modelo de machine learning es fundamentalmente una caja negra matemática donde los números son alimentados a la entrada y números son retornados como resultados. Acceso al resto del sistema no debería ser necesario para realizar estas operaciones. En la práctica, significa que la librería debería omitir opciones como ser cargar modelos desde un sistema de archivos u algún periférico. Por lo tanto, es posible escribir un framework de machine learning sin llamar al sistema operativo. Además, la mayoría de las plataformas objetivo, no cuentan con un sistema operativo.

- **No standard C or C++ library dependencies at linker time** Aparentemente, funciones de las librerías estándares de C y C++, como ser `sprintf()`, pueden fácilmente tomar hasta 20 Kb de memoria. Por tanto, se descartó el uso de estas librerías con el fin de reducir el consumo de memoria. Solo la librería `math.h` fue utilizada para los casos de funciones trigonométricas.

- **No floating-point hardware expected** Muchas plataformas embebidas no tienen soporte para aritmética de punto flotante en el propio hardware. Por lo que se debía evitar el uso crítico de flotantes. Esto significó enfocarse en modelos con parámetros de enteros de 8 bits, usando operaciones aritméticas de 8 bits. De todos modos, la librería cuenta con soporte para tipos flotantes si la plataforma es compatible.

- **No dynamic memory allocation** Muchas aplicaciones basadas en microcontroladores necesitan correr de forma continua durante meses o años. Asignando y desasignando memoria cíclicamente usando `malloc()/new()` y `free()/delete()`, es difícil garantizar que el heap no termine en un estado fragmentado que haga que la aplicación se rompa. Además, la memoria disponible en sistemas embebidos es escasa, por lo que planificar el uso de memoria es mucho más importante que en otras plataformas. Esto significa que aplicaciones embebidas a menudo evitan el uso de asignación dinámica de memoria. Debido a que la librería fue diseñada para usar en este tipo de aplicaciones, debió seguirse el mismo criterio. En la práctica, la librería espera que en la aplicación se le asigne un área de memoria estática de tamaño fijo al inicializar, llamada arena, en la cual podrá realizar las asignaciones de memoria temporales, como ser las activaciones internas de la red. Si la arena es demasiado pequeña, la librería devolverá un error y el desarrollador deberá compilar nuevamente con un tamaño de arena mayor, por lo que su tamaño se define de forma empírica.

- **It requieres C++11** TensorFlow Lite está escrita en su mayoría en C++, con el agregado de algunas APIs en C. De todos modos, no se basa en complejidades como templates, sino en el espíritu de un «mejor C» con clases para ayudar a modularizar el código. Además re escribir la librería en C hubiese tomado mucho trabajo y hubiese

sido un retroceso para los usuario de TensorFlow Lite. Además, cuando se relevó a las plataformas más populares, todas tenían soporte para C++ 11.

- **It expects 32-bit processors** La tendencia en los últimos años ha sido hacia los procesadores de 32 bits. Google decidió enfocar su desarrollo en los dispositivos de 32 bits porque además mantiene criterios como por ejemplo que el tipo de datos `int` en C es de 32 bits, al igual que en las plataformas móviles.

3.3.4. De TensorFlow a la memoria de un microcontrolador

TensorFlow Lite Converter Para poder correr un modelo de TensorFlow en un microcontrolador, es necesario convertirlo en un formato que permita guardarlo en la memoria del mismo.

El primer paso, es convertir el modelo de TensorFlow o Keras, en un modelo de TensorFlow Lite. Para esto se hace uso del TensorFlow Lite Converter, el cual provee una API en Python. Este conversor convierte el modelo de TensorFlow en un formato de archivo especial, diseñado para ser eficiente en memoria, y lo modifica para usar operaciones de TensorFlow Lite. Este formato se denomina *FlatBuffer*.

Post-training quantization En adición a crear un *FlatBuffer*, el conversor de TensorFlow Lite puede además aplicar optimizaciones al modelo. Estas optimizaciones generalmente reducen el tamaño del modelo, el tiempo que toma en correr, o ambas. Esto puede tener un costo de pérdida de exactitud y performance, pero muchas veces insignificativa. Uno de los métodos más utilizados de optimización es *quantization*. Por defecto, los pesos y bias en un modelo están almacenados como números de tipo `float` de 32 bits, de forma de tener una alta precisión durante el entrenamiento. Cuantización permite reducir la precisión de los números y por lo tanto pueden almacenarse como datos de tipo `int` de 8 bits, lo que significa una reducción del tamaño 4 veces menor. Incluso, las operaciones matemáticas con datos `int` son más fáciles de ejecutar que con datos `float`, por lo que un modelo cuantizado corre más rápido.

Esta técnica de cuantizar un modelo de TensorFlow ya entrenado, se denomina *Post-training quantization*. Permite reducir el tamaño mientras que mejora el tiempo de procesamiento de la CPU y la latencia, con una pequeña pérdida de exactitud del modelo mayoritariamente despreciable. El conversor de TensorFlow Lite, ofrece distintos métodos de cauntización con el fin de reducir el tamaño del modelo. Dentro de estos, nuestro interés está en aquellos que permite cuantizar el modelo a datos de tipo `int` de 8 bits.

1. **Dynamic range quantization** Es el método más simple de cuantización, en cual solamente cuantiza los pesos del modelo de datos tipo `float` a datos `int` de 8 bits, para su almacenamiento en memoria. Pero al momento de la inferencia, los pesos son convertidos nuevamente de `int` a `float`, y las operaciones son computadas usando datos de tipo `float`. Esta conversión de `int` a `float` se realiza una única vez y los pesos se almacenan en un cache de memoria para disminuir la latencia.

2. **Full integer quantization** Se puede mejorar la latencia, reducir el consumo de memoria, y compatibilizar con hardware que soporta solamente datos `int`, si todas las

operaciones matemáticas son con datos `int`.

Para una cuantización total, se requiere calibrar o estimar el rango (min, max) de todos los elementos de los tensores en el modelo, incluyendo las activaciones internas del modelo. Los tensores variables como ser las entradas, las activaciones intermedias y la salidas, no pueden ser calibrados a menos que se corran algunas inferencias. Como resultado, el conversor requiere de un dataset representativo para esta calibración. Este dataset, puede ser un subset del dataset de entrenamiento o de validación.

Dentro de este tipo de cuantización, existen 2 posibilidades:

a) **Integer with float fallback (using default float input/output)** Realiza la cuantización de todo el modelo, pero los tensores de la entrada y de la salida se mantienen como tensores de elementos de tipo `float` de forma de mantener la misma interfaz que el modelo original. Por lo tanto, no es compatible con dispositivos que solo soporten datos de tipo `int`.

b) **Integer only** Esta opción realiza una cuantización total, desde las operaciones y activaciones internas, incluyendo también las entradas y las salidas.

Es el tipo de cuantización utilizado típicamente en el caso de querer embeber un modelo de tensorflow en la memoria de un microcontrolaldor.

Convert to a C array El paso final, es preparar nuestro modelo para usar con la librería de C++ de TensorFlow Lite for Microcontrollers presentada en 3.3.3. Muchas plataformas de microcontroladores no tienen sistema de archivos nativo. La forma más fácil para incluir nuestro modelo en el programa es incluirlo como un array en C. Para esto existe una herramienta en Unix que permite generar un archivo fuente en C, que contenga el modelo como un array de elementos de tipo `char`.

Los siguientes comandos permiten instalar y correr `xxd` sobre un modelo de TensorFlow Lite cuantizado y devolver un archivo `.cc`.

```
# Install xxd if it is not available
!apt-get -qq install xxd

# Save the file as a C source file
!xxd -i tflite_model.tflite > tflite_model.cc
```

Dentro del archivo `tflite_model.cc` se encuentra definido el modelo de una forma similar al siguiente ejemplo.

```
unsigned char tflite_model_array[] = {
0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x12, 0x00,
0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00, 0x10, 0x00, 0x14, 0x00,
// ...
0x00, 0x00, 0x08, 0x00, 0x0a, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x09,
0x04, 0x00, 0x00, 0x00
};
unsigned int tflite_model_len = 2512;
```

Es importante, cambiar la declaración del array y agregar la palabra `const` para un manejo más eficiente de memoria por parte del compilador. Finalmente, el archivo generado puede incluirse en el proyecto en el que se está desarrollando le firmware del microcontrolador, junto con la librería de TensorFlow Lite for Microcontrollers.

TensorFlow Lite Interpreter Podría pensarse que para utilizar un modelo en el código del firmware de un microcontrolador, se debería implementar dicho modelo escribiendo código en C o C++, con todos los parámetros del modelo almacenados como arrays y la arquitectura expresada como una serie de llamadas a funciones que pasan las activaciones de una capa a la siguiente. Sin embargo, esto no es lo que ocurre en TensorFlow Lite ni en TensorFlow Lite for Microcontrollers. Lo que sucede realmente es que el modelo es interpretado, aunque esto suene contradictorio para quienes desarrollan código en un lenguaje compilado, como lo son C y C++. Interpretar el modelo es un enfoque diferente, y se basa en cargar una estructura de datos que define el modelo. En TensorFlow Lite, esta estructura es el FlatBuffer, mientras que en TensorFlow Lite for Microcontrollers, es el array de elementos de tipo `char` que se presentó en el párrafo anterior. En este enfoque, el código ejecutado es estático, solo los datos acorde al modelo cambian, y la información en el modelo controla cuáles operaciones son ejecutadas y cuáles son los parámetros a utilizar. Si por alguna razón, en una aplicación se deseara cambiar el modelo de machine learning utilizado, pero manteniendo la interfaz con el sistema, es decir, manteniendo el tamaño de las entradas y las salidas, basta con sobre escribir el array que contiene al modelo. El resto del código se mantiene como está.

Debido a que está diseñado para ser eficiente, el intérprete de TensorFlow Lite es apenas un poco más complicado de usar que la API de Keras. Para hacer predicciones con un modelo de Keras, se podría llamar al método `predict()`, pasándole un array de entradas. Con TensorFlow Lite, se necesita hacer lo siguiente:

1. Instanciar un objeto `Interpreter`.
2. Llamar a algunos métodos que asignan memoria para el modelo.
3. Escribir la entrada en el tensor de entrada.
4. Invocar el modelo.
5. Leer la salida desde el tensor de salida.

3.3.5. Introducción a la electrocardiografía

Para dar cierre a la fundamentación teórica, se presenta una introducción a la electrocardiografía debido a que es el área de aplicación en el cual hemos trabajado.

Cada vez que el corazón late establece un campo dieléctrico, debido a que los impulsos que preceden la contracción del corazón excitan las fibras musculares miocárdicas y producen corrientes eléctricas que se difunden por todo el cuerpo; el corazón actúa como un generador dentro de un conductor volumétrico y anisotrópico, motivando que la corriente generada se difunda a la superficie del conductor. Einthoven, fisiólogo belga, descubrió que la actividad eléctrica del corazón se puede recoger desde el exterior del cuerpo, dando origen al ECG o

electrocardiograma. El ECG es una gráfica de voltaje con respecto al tiempo que refleja la actividad eléctrica de la despolarización y repolarización del músculo cardiaco durante cada latido. Su registro se logra a través de electrodos ubicados en las extremidades y en el pecho.

El ECG o electrocardiograma de un latido (ciclo cardíaco) normal, consiste en una secuencia de ondas que Einthoven denominó P, Q, R, S, T, y U, de acuerdo con su orden de inscripción. La onda P corresponde al proceso de despolarización atrial, el complejo QRS al proceso de despolarización ventricular, y la onda T representa la repolarización ventricular. La onda U no tiene un origen definido y no siempre se registra. El intervalo o segmento PR, definido desde el inicio de la onda P hasta el inicio del complejo QRS, representa el tiempo desde el inicio de la activación auricular hasta la activación ventricular. El término del complejo QRS se denomina "punto J" y da inicio al segmento ST (periodo en que los ventrículos están aún despolarizados), que separa al complejo QRS con la onda T. El intervalo desde el inicio de la activación ventricular al término de la repolarización se denomina intervalo QT. La repolarización auricular no se registra en el electrocardiograma, ocupa parte del segmento PR y del complejo QRS, pero queda enmascarada por la alta magnitud del complejo QRS. La duración del complejo QRS, el intervalo PR y el intervalo ST dependen de la tasa de despolarización del corazón y son relativamente constante para un individuo, sin importar el nivel de ejercicio que practique.

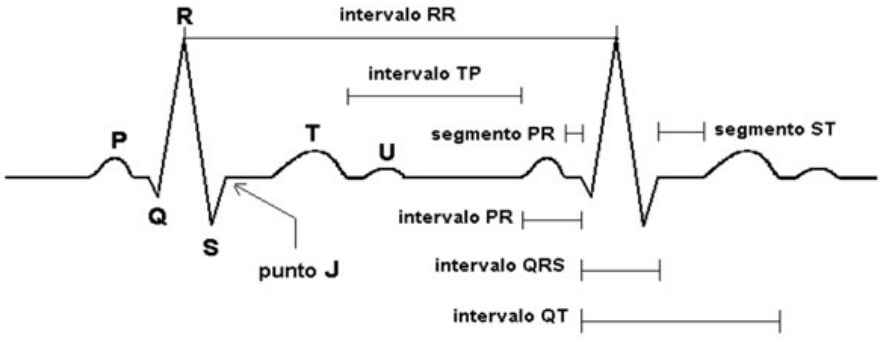

Figura 7: Ondas y segmentos del ECG

Derivaciones del ECG El campo eléctrico generado por el corazón se puede representar matemáticamente por un vector con una magnitud, una dirección y un sentido, que va cambiando a lo largo del ciclo cardíaco. Este vector se denomina vector de despolarización. Para registrar las diferentes proyecciones de este vector, es suficiente que se registren las fuerzas eléctricas cardiacas proyectadas sobre el plano frontal (arriba-abajo), y sobre el plano horizontal (derecha-izquierda). Como se ya se mencionó, esto se logra mediante la ubicación de electrodos en las extremidades y las paredes del pecho.

El electrocardiograma estándar consta de 12 derivaciones, que son el resultado de la exploración indirecta del corazón desde distintos planos. Para el registro de las 12 derivaciones se emplea un total de 10 electrodos. Estas 12 derivaciones se dividen en 6 derivaciones del plano frontal y 6 derivaciones del plano horizontal.

Las 6 derivaciones del plano frontal, se generan de 4 electrodos ubicados en las extremidades del paciente (el electrodo de la pierna derecha es el electrodo de tierra, y los otros 3

son electrodos exploradores), por lo que suelen llamarse también derivaciones de las extremidades. A su vez, de estas 6 derivaciones, 3 derivaciones son bipolares y 3 derivaciones son unipolares, tal como se describen a continuación:

- **Derivaciones bipolares:** Fueron creadas por William Einthoven y se generan a partir de los electrodos del brazo derecho, del brazo izquierdo, y de la pierna izquierda. Estas derivaciones registran la diferencia de potencial eléctrico entre dos puntos del cuerpo, y tienen su fundamento en la teoría del dipolo, de aquí su nombre. La base del corazón se conduce como polo negativo. Esa región basal se proyecta sobre el brazo derecho, por lo que dicho brazo constituye el polo negativo de las derivaciones bipolares. Recordemos que la onda de excitación marcha de base a punta y, al aproximarse al brazo izquierdo y la pierna izquierda, los convierte en polos positivos. Teniendo ya constituidos los 2 polos del dipolo, las 3 derivaciones de Einthoven se integran de la manera siguiente:

  1. **DI:** Es la diferencia de potencial entre el electrodo positivo del brazo izquierdo y el electrodo negativo del brazo derecho.
  2. **DII:** Es la diferencia de potencial entre el electrodo positivo de la pierna izquierda y el electrodo negativo del brazo derecho.
  3. **DIII:** Es la diferencia de potencial entre el electrodo positivo de la pierna izquierda y el electrodo negativo del brazo izquierdo.

  Einthoven consideró estas tres derivaciones bipolares (DI, DII, DIII) como un circuito cerrado, y estableció una relación conocida como *ley de Einthoven* que debe cumplirse siempre en un electrocardiograma, pues permite asegurar que está bien registrado. Esta relación establece que DI + DIII = DII y puede observarse en la figura 8.

  Sobre la base de la ley enunciada, Einthoven, partiendo del principio de que era igual colocar los electrodos en los brazos o en las piernas, que en las raíces de los miembros (hombro derecho, hombro izquierdo y pubis) y considerando que el tronco humano es una esfera conductora homogénea que tiene en su centro el corazón, describió su triángulo, cuyas características se ven en la figura 9.

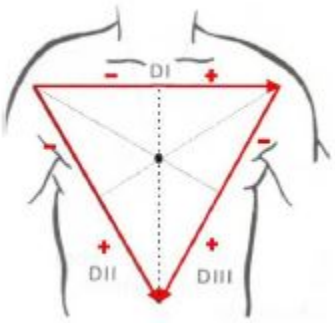

Figura 8: Ley de Einthoven

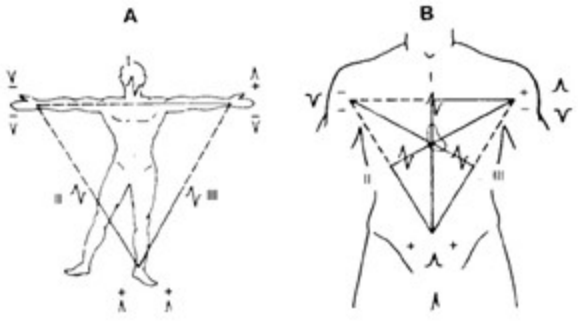

Figura 9: Triángulo de Einthoven

- **Derivaciones unipolares o aumentadas:** Estas derivaciones registran el potencial real neto en un punto de la superficie del cuerpo. Inicialmente, estas derivaciones fueron creadas por Frank Wilson y para su registro unió los tres vértices del triángulo de Einthoven (brazo derecho, brazo izquierdo y pierna izquierda), por medio de resistencias de 5000 Ω, a un solo punto denominado *terminal central de Wilson* (figura 10), con lo cual obtuvo en dicho punto un potencial eléctrico cero. Conectando el electrodo explorador al brazo derecho (R), al brazo izquierdo (L) y a la pierna izquierda (F), obtuvo los potenciales absolutos monopolares de dichos miembros, registrados respectivamente en las derivaciones VR, VL y VF. Luego, Goldberger modificó el sistema propuesto por Wilson, consiguiendo aumentar así la amplitud de las ondas hasta en un 50 %. Lo que Goldberger propuso fue medir el potencial neto de las extremidades del triángulo de Einthoven respecto a un terminal el cual consiste en unir mediante resistencias las otras dos derivaciones de extremidades a un punto común de modo que se obtiene un promedio de los dos potenicales. Así, las derivaciones VR, VL y VF se sustituyeron por las derivaciones aVR, aVL y aVF, donde la «a» significa ampliada o aumentada.

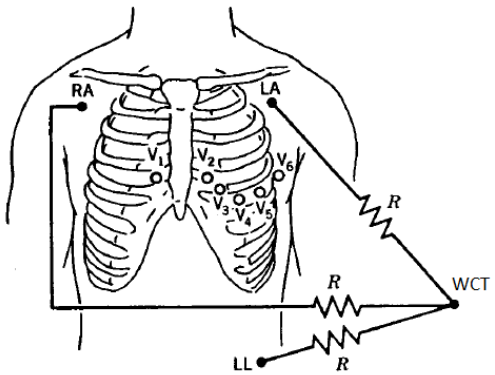

Figura 10: Terminal central de Wilson

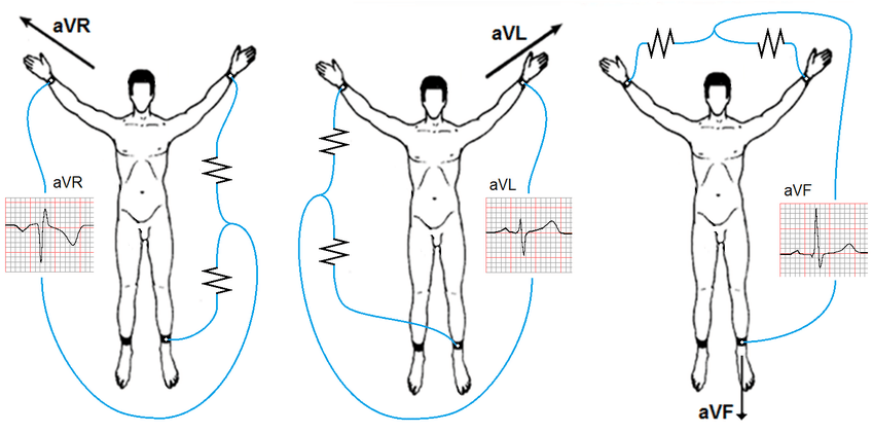

Figura 11: Derivaciones aumentadas

Las 6 derivaciones del plano horizontal se denominan derivaciones unipolares precordiales, pues registran el potencial neto de 6 ubicaciones del precordio con respecto al terminal central de Wilson. Estas se denominan V1, V2, V3, V4, V5 y V6, y la ubicación del electrodo explorador para cada una de ellas se muestra en la figura 12.

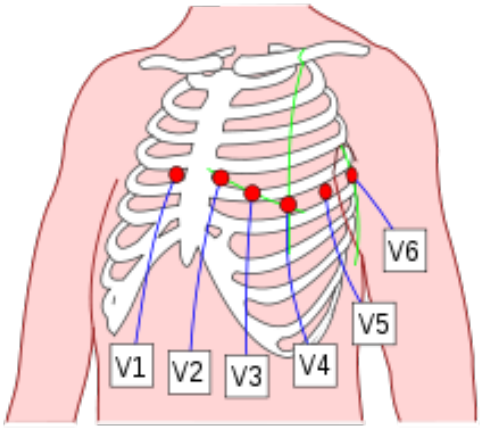

Figura 12: Derivaciones unipolares precordiales.

4. Metodología

4.1. Preparación del dataset

Como fuente de datos para el desarrollo de nuestro trabajo, se construyó un nuevo dataset a partir de los dataset presentados en 3.1. Cada uno de los dataset fue procesado individualmente con el fin de realizar tareas de data cleaning, segmentación, cambio de formato y etiquetado. Luego se realizó una unión parcial de las muestras de cada dataset formando un dataset único con las siguientes características:

- Señales ECG de 12 derivaciones de 10 segundos de duración. Medidas en mV.
- Frecuencia de muestreo de 100 Hz.
- Total de 23804 muestras.
- Etiquetadas en NORMALES (negativas) y NO NORMALES (positivas).
- Dataset balanceado: 50 % normales + 50 % anormales.

A continuación, se describe el trabajo realizado sobre cada dataset y la unión de los mismos para obtener el resultado final.

A 12-lead electrocardiogram database for arrhythmia research covering more than 10,000 patients Cronológicamente, este fue el primer dataset tratado, cuando se desconocía aún la existencia de los otros 2 datasets. Cuenta con 10646 registros los cuales se encuentran en formato `.csv`. Para su lectura se usó la librería de Python pandas, mediante el uso de `DataFrames`, y `numpy`. A continuación, se describen los pasos seguidos durante el procesamiento de los datos.

1. En primer lugar, se realizó un procesamiento de data cleaning, donde se leyó cada registro del dataset y se verificó la existencia de valores vacíos. De los 10646, se encontraron 18 registros que presentaban pérdida de información y por lo tanto fueron descartados, quedando un total de 10628 registros.

2. Como se mencionó en 3.1, son 11 las etiquetas de ritmos presentes en el dataset. Por lo tanto, en segundo lugar, se realizó un nuevo procedimiento de etiquetado acorde a la clasificación binaria que realizaríamos con nuestro modelo. Para esto, las señales que estaban etiquetadas con la siglas `SR` correspondientes a *Sinus Rythm*, fueron etiquetadas como señales normales, mientras que el resto de las señales que presentaban una de las 10 etiquetas restantes, representando algún tipo de anomalía, fueron etiquetadas como señales no normales. Como resultaron, se obtuvieron 1825 señales normales y 8803 señales anormales, representando el 17,17 % y el 82,83 % respectivamente.

China Physiological Signal Challenge 2018 dataset En búsqueda de nuevas fuentes de datos que permitieran realizar un balanceamiento entre las señales normales y normales en el dataset anterior, fue que se halló el dataset China Physiological Signal Challenge 2018. Los registros se encuentran en formato de archivo `.mat` (MATLAB), por lo que para abrir y leer dichos archivos, fue necesario hacer uso de la función `loadmap()` perteneciente al módulo `io` de la librería de Python `scipy`. A continuación, se describen los pasos seguidos durante el procesamiento de los datos.

1. En primer lugar, se trabajó sobre el archivo de referencia del dataset el cual contiene la asignación de etiquetas para cada registro. Tal como se mencionó en 3.1, cada registro puede tener hasta 3 etiquetas asignadas, las cuales pueden ser de 9 tipos distintos de señales. De las 3 etiquetas, se tomó solamente la primera, descartándose las otras 2 etiquetas en caso de existir.

2. Dado que el objetivo de esta segunda búsqueda de datos, fue poder balancear la diferencia existente entre cantidad de señales normales y señales anormales resultante en el dataset anterior, se tomaron aquí solo las señales etiquetadas como señales normales. En total, se obtuvieron 918 señales normales de largo variables que van desde los 6 y hasta los 60 segundos.

3. En tercer lugar, se realizó un procedimiento de data cleaning, pero afortunadamente no se encontraron registros con pérdida de datos.

4. Dado que los registros en el dataset anterior son de 10 segundos de duración, y los registros en este dataset varían entre los 6 y los 60 segundos de duración, fue necesario segmentar los registros en este dataset para obtener así registros de 10 segundos de duración. En la práctica, esto significó segmentar registros en registros de 5000 valores, dado que la frecuencia de muestreo es de 500 Hz. Aquellos registros menores a los 10 segundos de duración fueron descartados. De este procedimiento, se obtuvieron 1201 registros normales de 10 segundos de duración, que serían adicionados al dataset anterior.

5. Por inspección, se dedujo que los registros en este dataset se encuentran en mV, mientras que los registros del dataset anterior, según los autores, se encuentran en μV. Por lo tanto, los registros segmentados obtenidos en el paso anterior, fueron multiplicados por 1000 para pasar de mV a μV.

6. En último lugar, se realizó un procesamiento de downsampling para bajar la frecuencia de muestreo de 500 Hz a 100 Hz. Este procedimiento se realizó en ambos dataset, ya que ambos dataset están constituidos por registros muestreados a 500 Hz, pero nuestro interés es trabajar con muestras a 100 Hz tal como muestrea el dispositivo de Galenos_Sys.

7. Como resultado final, se concatenaron ambos datasets, y se obtuvo un dataset compuesto de 11829 registros de 10 segundos de duración a una frecuencia de 100 Hz, donde 8803 son señales anormales y 3026 son señales normales, representando el 74,42 % y el 25,58 % respectivamente.

PTB-XL, a large publicly available electrocardiography dataset El dataset China Physiological Signal Challenge 2018, no fue suficiente para lograr un dataset balanceado. Se recurrió una vez más a realizar una nueva búsqueda de fuente de datos, y así fue que se halló la existencia de este dataset en el repositorio de Physionet. Para nuestra sorpresa, nos encontramos con un dataset mucho más completo, con más del doble de registros que los datasets anteriores, compuesto por registros de 10 segundos de duración, en 2 versiones de frecuencias de muestreos, una de ellas de 100 Hz, y con una mayor cantidad de señales etiquetadas como señales normales.

Los registros se encuentran disponibles en formato WaveForm DataBase, un formato propio de Physionet. Por lo que para su lectura se requiere instalar el paquete de Python `wfdb`. A continuación se presentan los pasos seguidos hasta obtener nuestro dataset final.

1. Tal como se describió en 3.1, este dataset presenta un etiquetado en formato SCP-ECG, donde para cada registro se asignan distintas etiquetas a las cuales se les asignó una probabilidad o certeza (likelihood). Estas etiquetas además se encuentran divididas en 3 categorías distintas que son: diagnóstico, forma y ritmo. Por lo que, en primer lugar, se decidió seleccionar todos los registros que tienen al menos una etiqueta de diagnóstico, eliminando aquellos registros que no tienen ninguna etiqueta dentro de la categoría diagnóstico. ¿Por qué?, porque es dentro de la categoría diagnóstico donde encontramos la clasificación de señales en señales normales o señales que presentan un diagnóstico anormal. De esta forma, se eliminaron en este primer paso 407 registros, pasando de 21837 a 21430.

2. En segundo lugar, se decidió considerar como señales normales a todas aquellas señales que están rotuladas únicamente con diagnóstico normal. Durante la revisión del dataset, se encontraron 445 casos de muestras con diagnóstico múltiple, incluyendo el diagnóstico normal. Sin importar el likelihood de cada diagnóstico, se decidió eliminar estas 445 muestras, de forma que las señales normales sean aquellas que solo están clasificadas únicamente como normales. De esta forma, el total de muestras bajó a 20985, de las cuales 11902 son señales que presentan alguna anomalía, y 9083 son señales normales.

3. Dado que este dataset es el mayor de las 3 opciones, que se encuentra próximo al balance entre señales normales y señales anormales, y que comparte las características de las señales con las que pretendemos trabajar, se decidió tomar este dataset como base y tomar registros de los datasets anteriores para poder lograr el balance al 50 % como corresponde. En particular, la diferencia entre señales anormales y señales normales es: $11902 - 9083 = 2819$. Así fue que se tomaron 1201 señales normales del dataset presentado en (Liu y cols., 2018) y 1618 del dataset presentado en (Zheng y cols., 2020).

4. Citando lo descripto en párrafos anteriores, las señales que se adicionaron se encontraban inicialmente en uV. En los registros de este dataset, las señales se encuentran en la unidad mV. Por tanto, primero se dividieron las señales entre 1000,0 para pasar de μV a mV.

5. Tal como se describió en 3.1, los autores de este dataset realizaron una partición del dataset en 10 stratified folds, recomendando los folds 9 y 10 para validación y testing respectivamente. Por tanto, las señales normales que se agregaron al dataset, debieron ser repartidas manualmente entre los 10 folds ya existentes. El resultado obtenido se muestra en la tabla a continuación.

Fold	# normal	# not normal
1	1181	1200
2	1207	1174
3	1225	1160
4	1168	1203
5	1177	1196
6	1174	1204
7	1204	1167
8	1174	1200
9	1198	1199
10	1194	1199

Cuadro 5: Dataset stratified folds

6. Una vez que se unieron los dataset, se realizó el procedimiento de crear matrices de `numpy` que almacenaran los elementos del dataset. De forma de que a partir de ahora se almacenan en archivos de formato `.npy`. Siguiendo la recomendación de los autores de (Wagner y cols., 2020), se tomaron los folds del 1 al 8 como folds de entrenamiento, y los folds 9 y 10 como fold de validación y testing respectivamente. Para las etiquetas, se designó el valor 0 para las señales normales, de aquí en más referenciadas como muestras negativas, y el valor 1 a las señales anormales, de aquí en más llamadas muestras positivas. La tabla 6 muestra la distribución del dataset obtenido.

Dataset	# total de muestras	# muestras positivas	# muestras negativas
Training	19014	9504	9510
Validation	2397	1199	1198
Test	2393	1199	1194
Total	23804	11902	11902

Cuadro 6: Distribución de las muestras del dataset final.

Tipo de señal	Etiqueta asignada
Normal	0
No normal	1

Cuadro 7: Etiquetas binarias

4.2. Métricas

Además de poder contar con datos para entrenar nuestros modelos de machine learning, es necesario también definir cómo vamos a evaluar la performance de estos modelos. La forma de hacerlo, es haciendo uso de las métricas que permiten mensurar dichas performances. La elección de las métricas depende en general del tipo de problema con el que se esté tratando y la naturaleza del mismo. En nuestro caso, estamos tratando un problema de clasificación binaria para señales ECG.

Durante el desarrollo de este trabajo, fue mutando el enfoque del mismo sobre la forma de evaluar la performance de los modelos entrenados y por ende las métricas empleadas. Inicialmente, al tratarse de un problema de clasificación de señales ECG, se entendía que el foco principal era obtener modelos que fueran buenos detectando señales anómalas, es decir, muestras positivas. Por tanto, se consideraba que nuestras métricas de evaluación debían ser *Recall o Sensitivity*, *Precision* y *F2*.

$$Recall/Sensitivity = \frac{TP}{TP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$F_2 = \frac{(1 + 2^2) * Recall * Precision}{(2^2 * Precision) + Recall} \tag{7}$$

Pero luego entendimos también que en nuestro caso de aplicación, es de gran relevancia que nuestros modelos sean buenos detectando señales ECG normales, es decir, muestras negativas. ¿Por qué? Porque de tratarse de una aplicación de edge AI o edge computing, y dadas las características y las prestaciones de este tipo de aplicaciones, que nuestros modelos detecten con certeza una señal ECG normal, significa que existe garantía para tomar la decisión de no enviar los datos a la nube y así ahorrar batería, prolongando la autonomía del dispositivo. Por tanto, también se emplearon las métricas *Specificity* y *Negative Predictive Value (NPV)*.

$$Specificity = \frac{TN}{TN + FP} \tag{8}$$

$$NPV = \frac{TN}{TN + FN} \tag{9}$$

Por último, considerando que nuestro dataset es un dataset balanceado, se incluyó la métrica *Accuracy* como una métrica que permita medir la performance de nuestro modelo de un modo genérico.

$$Accuracy = \frac{TN + TP}{TN + TP + FN + FP} \tag{10}$$

Por TN, TP, FN, FP entendemos *True Negatives*, *True Positives*, *False Negatives* y *False Positives* respectivamente. Pero cómo se define si una predicción es positiva o negativa. En nuestro caso, las predicciones de los modelos se encontrarán en el intervalo $[0, 0, 1, 0]$. En

general, se da por entendido que se definen como positivas todas aquellas predicciones que son $\geq 0,5$, de lo contrario se definen como predicciones negativas. Comúnmente, cuando se presentan las métricas, no se especifica qué umbral o límite se utiliza para separar las muestras positivas y negativas, y se asume que el umbral es $0,5$.

Pero en nuestro caso (veremos más adelante por qué), definimos también 3 métricas más utilizando 2 umbrales o thresholds distintos. Por un lado, definimos las métricas *Recall* y *NPV* con un threshold de $0,05$, y un *Precision* con un threshold de $0,95$.

$$Recall(0,05) = \frac{TP(0,05)}{TP(0,05) + FN(0,05)} \tag{11}$$

$$NPV(0,05) = \frac{TN(0,05)}{TN(0,05) + FN(0,05)} \tag{12}$$

$$Precision(0,95) = \frac{TP(0,95)}{TP(0,95) + FP(0,95)} \tag{13}$$

Si bien existen módulos en Python con funciones para el cálculo automático de estas métricas, en este trabajo se realizó el desarrollo de nuestra propia función para el cálculo de las métricas y la clasificación de las predicciones, realizando un registro de las predicciones de los modelos y la clasificación de las mismas en TN, TP, FN, FP.

4.3. Evaluación del estado del arte

Una vez que definimos nuestro dataset, las métricas de evaluación, y seleccionamos una arquitectura de red, que forma parte del estado del arte y que según los antecedentes brinda resultados satisfactorios en el procesamiento de series temporales, es momento de determinar dónde estamos parados, cuál es nuestro punto de partida. Para esto se decidió procesar nuestro dataset con el modelo definido en (Ingolfsson y cols., 2021), pero realizando algunas modificaciones preliminares, pertinentes a la adaptación del modelo a nuestra aplicación. Las modificaciones se listan a continuación.

- Dado que originalmente los autores trataron una clasificación categórica de 5 categorías, la capa de salida del modelo ECG-TCN cuenta con 5 neuronas. En nuestro caso, tratamos un problema de clasificación binaria y por lo tanto nuestra capa de salida se compone de una única neurona.
- En relación al punto anterior, se sustituyó la función de activación de la capa de salida `'softmax'` por la función de activación `'sigmoid'`.
- Del mismo modo, en la función de compilación se cambió el argumento de la función de pérdida `loss`, sustituyendo `'sparce_categorical_crossentropy'` por `'binary_crossentropy'`

Una vez realizadas las modificaciones en el modelo, el mismo se entrenó con muestras de la derivación DII de nuestro dataset de entrenamiento, y se realizaron inferencias del modelo en los dataset de validación y testing. Las métricas resultantes de inferir el modelo en el dataset de validación se presentan en la tabla 8, y para el caso del dataset de testing, en la tabla 9.

TN	FN	TP	FP	Recall	Precision	Specificity	NPV	F2	Acc
1140	515	684	58	0,570475	0,921833	0,951586	0,688822	0,617551	0,760951

Cuadro 8: Métricas estándares del modelo ECG-TCN en el dataset de validación

TN	FN	TP	FP	Recall	Precision	Specificity	NPV	F2	Acc
1120	556	643	74	0.53628	0.896792	0.938023	0.668258	0.583167	0,736732

Cuadro 9: Métricas estándares del modelo ECG-TCN en el dataset de testing

De los resultados presentados en las tablas 8 y 9, se observa que las métricas en el dataset de testing son apenas un poco menores que las métricas en el dataset de validación, pero que a nivel general, el modelo presenta un mismo comportamiento en ambos dataset, ya que dicha diferencia es despreciable, y la relación entre las métricas es la misma. A nivel general, el modelo puede predecir aproximadamente un 75 % de las muestras de forma correcta, presentando un mejor desempeño en la detección de señales negativas con un *Specificity* próximo al 95 %, mientras que no es tan bueno detectando señales positivas debido a un *Recall* que ronda el 55 %.

Para una aplicación de machine learning en el área de la salud, como es nuestro caso, estas métricas son poco alentadoras y hacen inviable la implementación de este modelo en un entorno de producción. ¿Por qué?, porque para las aplicaciones en el área de la salud, se desea que el modelo sea bueno detectando las muestras positivas, que son las muestras malignas, las que representan la enfermedad, de forma que se considera sumamente crítico cuando las predicciones son falsos negativos, dado que no se está identificando una muestra maligna, poniendo en riesgo la salud del paciente.

Para continuar con el análisis, se decidió dar un paso más y analizar la distribución de las predicciones que arrojó el modelo para entender cómo se comporta y poder buscar una manera de optimizar su rendimiento.

La figura 13 presenta los histogramas de las predicciones para las muestras positivas y las muestras negativas del dataset de validación. La figura 14 presenta los histogramas de las predicciones para las muestras positivas y las muestras negativas del dataset de testing.

Se observa en ambos casos, que la mayor concentración de predicciones se encuentra en donde se desea encontrar. Es decir, para las muestras negativas, se desea que las predicciones se acumulen en un entorno cercano de $0,0$, mientras que para las muestras positivas, se espera que las predicciones se concentren en un entorno cercano a $1,0$. Incluso, tal como lo demuestran los valores de *Specificity*, el modelo presenta un comportamiento muy bueno para las muestras negativas. Apenas pueden visualizarse valores en rojo por encima de $0,5$ en los histogramas de las muestras negativas. En las tablas 8 y 9, se lee que se trata de 58 y 74 falsos positivos para los dataset de validación y testing respectivamente. Pero sin embargo, se observa también en ambos casos, que existen muestras que se encuentran en el extremo opuesto de la distribución. Es decir, predicciones falsas que toman valores muy cercanos al peor valor posible, siendo $0,0$ para muestras positivas, y $1,0$ para muestras negativas. Por tanto, aunque el modelo sea muy bueno detectando muestras negativas, nunca podremos tener la seguridad de que una predicción próxima a 0 se trate de un true negative debido a la gran cantidad de falsos negativos. En el proceso de inferir el modelo en el dataset de

validación y en el de testing, se realizó también el registro de los valores máximos y mínimos, los cuales se presentan en la tabla 10.

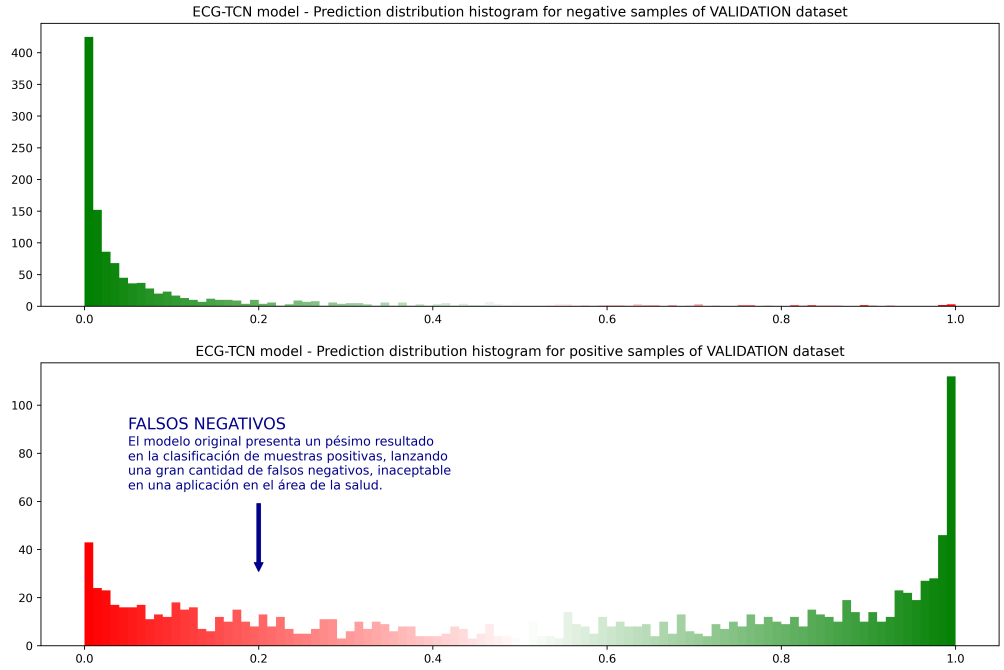

Figura 13: Distribución de predicciones en el dataset de validación para el modelo ECG-TCN.

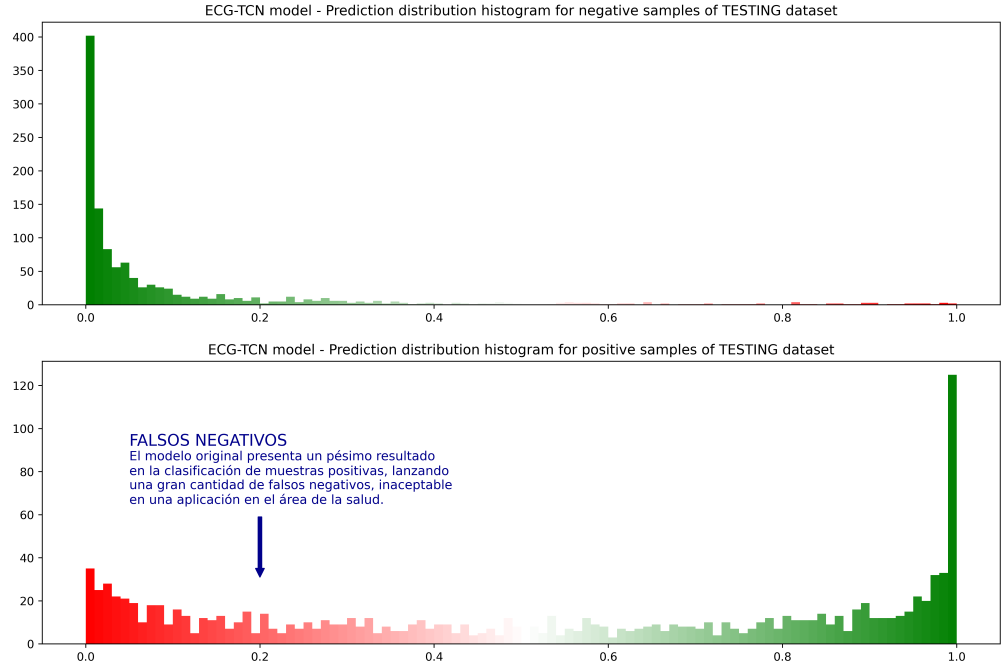

Figura 14: Distribución de predicciones en el dataset de testing para el modelo ECG-TCN.

Dataset	TN mínimo	FN mínimo	TP máximo	FP máximo
Validation	0,000134	0,000437	1,0	0,999406
Testing	0,000196	0,000305	1,0	0,9967

Cuadro 10: Máximos y mínimos de las predicciones de ECG-TCN en los datasets de validación y testing.

Para el caso de las muestras positivas, los valores de *Recall* en las tablas 8 y 9, denuncian que el modelo no es tan bueno detectando muestras positivas, situación que en los histogramas se puede confirmar al observar que existe una gran cantidad de falsos negativos por debajo de $0,5$, y que los mismos alcanzan valores extremadamente malos, cercanos a $0,0$, tal como se lee en la tabla 10.

Conclusiones A partir de los resultados lanzados por el modelo ECG-TCN presentado en (Ingolfsson y cols., 2021), luego de ser entrenado con el dataset elaborado en este trabajo y presentado en 4.1, y de las observaciones señaladas sobre los resultados de inferir el modelo en los dataset de validación y testing, se concluye que este modelo no logra resultados óptimos para la aplicación que aquí se trata. Pues las métricas indican que el modelo no es demasiado bueno detectando muestras positivas (muestras malignas), presentando una gran cantidad de falsos negativos, situación sumamente crítica en este tipo de aplicaciones. Más aún, no ofrece ninguna certeza respecto a las predicciones realizadas, dado que tanto para las muestras positivas como para las negativas presenta predicciones falsas, con casos extremadamente malos, donde la unión de distribuciones falsas obtenidas cubre prácticamente la totalidad del intervalo (0,0, 1,0). En resumen, no existe ningún caso de predicción donde podamos tener certeza de tratarse de un True Negative o un True Positive.

4.4. Método de optimización

Ante los resultados presentados en la sección anterior, surgió la pregunta de cómo se podría mejorar el comportamiento del modelo ECG-TCN siendo entrenado con el dataset presentado en 4.1. Idealmente, se desea que un modelo no presente falsos negativos ni falsos positivos. Matemáticamente hablando, esto significa que el máximo del conjunto de las predicciones de las muestras negativas sea menor a $0,5$, mientras que el mínimo del conjunto de predicciones de las muestras positivas sea mayor o igual a $0,5$. Viendo los valores presentados en la tabla 10, pensar en lograr llevar el modelo a un comportamiento ideal, sería un desafío grande y muy pretencioso.

Thresholds Lo que se propuso en este trabajo fue considerar dos umbrales o thresholds, uno inferior al que llamaremos TH_{LOW}, y otro superior al que llamaremos TH_{HIGH}, e intentar modificar las distribuciones de los falsos negativos y los falsos positivos, de forma que el mínimo del conjunto de falsos negativos se encuentre por encima de TH_{LOW}, y el máximo del conjunto de falsos positivos se encuentre por debajo de TH_{HIGH}. Si a su vez, se logra que el mínimo del conjunto de predicciones true negatives se encuentre por debajo de TH_{LOW}, y el máximo del conjunto de predicciones true positives se encuentre por encima de TH_{HIGH},

entonces podemos concluir que estamos en una situación en la cual el modelo disminuye el riesgo de obtener falsas predicciones. Para todas aquellas predicciones que estén por debajo de TH_{LOW}, podríamos determinar con gran probabilidad que se tratan de true negatives, y por ende, de señales normales. Mientras que para todas aquellas predicciones que estén por encima de TH_{HIGH}, tenemos gran probabilidad que se tratan de true positives, y por ende, de señales anómalas. Para aquellas predicciones que se encuentren entre ambos thresholds, no tendremos certeza de que tipo de señal se trata, no se pueden clasificar a priori, y deberán ser enviadas a la nube para el análisis de un especialista o de algoritmos más potentes. La figura 15 ilustra la división en las predicciones mediante la aplicación de thresholds.

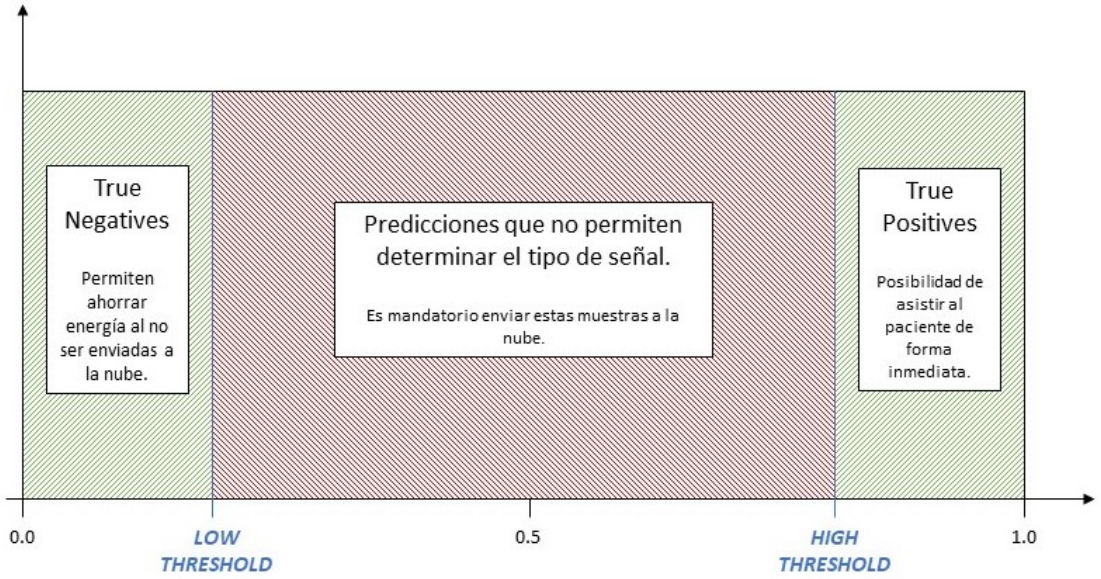

Figura 15: Thresholds.

Aplicando esta idea sobre la distribución de predicciones del modelo ECG-TCN presentada en la sección anterior, podrían considerarse valores de $TH_{LOW} = 0,05$ y $TH_{HIGH} = 0,95$ y proponernos mover aquellos falsos resultados como se indica con las flechas rojas presentes en la figura 16.

Visto desde el lado de la aplicación en la que se basa este trabajando, esta proposición permitiría lograr los objetivos de permitir gozar los beneficios del edge computing, de forma que aquellas señales que se encuentran debajo de 0,05 pueden ser clasificadas como normales (negativas), puede decidirse no enviarlas a la nube y ahorrar energía y ganar autonomía. Mientras que para aquellas señales detectadas como anormales (positivas) debido a que su predicción está por encima de 0,95, podrían permitir dar una primera asistencia al paciente o informar al centro hospitalario más cercano, además de ser enviadas a la nube para su posterior revisión. Para aquellas señales cuya predicción se encuentre entre los umbrales, no se tiene certeza de que tipo de señal se trata y por lo tanto, también deberían ser enviadas a la nube para una clasificación con modelos más potentes.

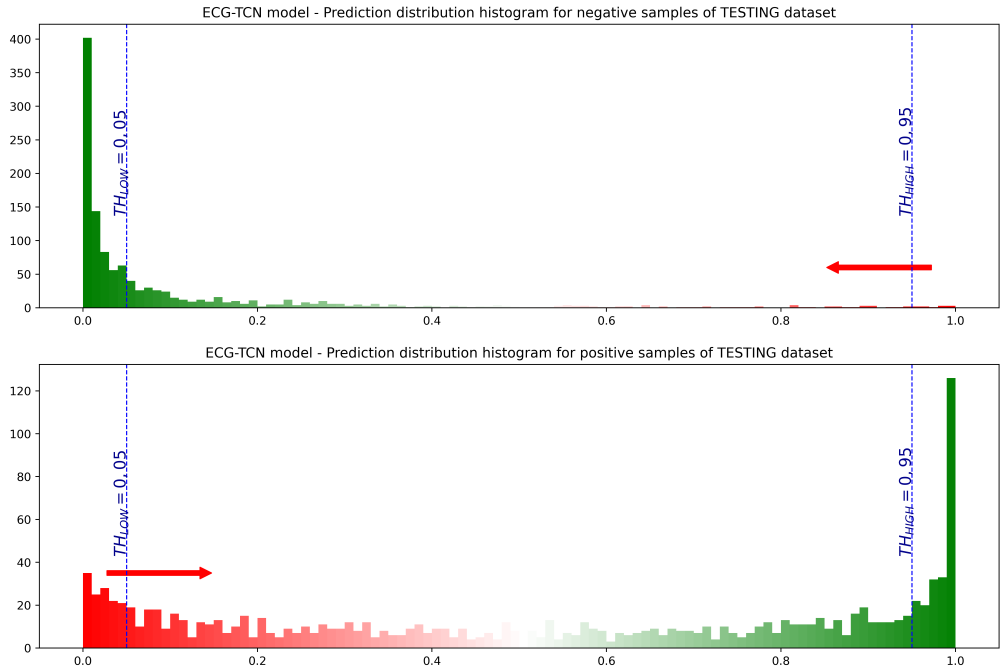

Figura 16: Thresholds aplicados a la distribución de predicciones en el dataset de testing para el modelo ECG-TCN.

Métricas Con la introducción de los thresholds en los párrafos anteriores, podemos ahora justificar el uso de las métricas mencionadas en 4.2 en las ecuaciones 11, 12 y 13. En la próxima sección se presenta el procedimiento propuesto para intentar que las falsas predicciones que están más allá de los umbrales se muevan hacia el otro lado del umbral. Era necesario poder medir el comportamiento, el desarrollo, y la evolución de este procedimiento. Por ello, se definieron las métricas presentadas en 4.2, dónde $TH_{LOW} = 0,05$ y $TH_{HIGH} = 0,95$. **El objetivo del método propuesto, es lograr tener las métricas** $NPV(0,05)$**,** $Recall(0,05)$ **y** $Precision(0,95)$ **iguales a** $1,0$**.**

Pesos El novedoso método de optimización aplicado en el intento de obtener los objetivos presentados en los párrafos anteriores, consiste en asignar pesos a las muestras del dataset de entrenamiento. Con pesos nos referimos a elementos numéricos que permiten ponderar el error de cada muestra en el error total del modelo. De forma que en el proceso de aprendizaje por backpropagation, el modelo le dará más importancia a las muestras que presentan mayor peso. De esta forma, estos pesos permiten manipular aquellas muestras cuyas predicciones, además de ser falsas, se encuentran del lado equivocado del threshold, asignándoles mayor relevancia y tratando de corregir su predicción.

En la práctica, se desarrolló un procedimiento iterativo, en el cual se realiza un proceso de entrenamiento aplicando crossvalidation al dataset de entrenamiento, en cada iteración. Se trata de 2 bucles cíclicos anidados, uno interno correspondiente al entrenamiento con crossvalidation, y uno externo definido en el método propuesto. Como es sabido, en cada iteración de crossvalidation, un nuevo modelos es creado y entrenado. Pero los pesos no se inicializan en cada ciclo de crossvalidation, ni tampoco en cada ciclo externo. Los pesos son

inicializados al comienzo del todo el procedimiento y se les asigna el valor 1,0.

En cada ciclo de crossvalidation, el modelo entrenado con los folds de entrenamiento, se infiere en el fold restante. Las predicciones arrojadas por el modelo son procesadas para el cálculo de las métricas y la clasificación de las mismas, y los resultados obtenidos se van guardando en los siguientes archivos:

- Un archivo `.csv` que contiene las siguientes columnas:
  - `iter` : Número de iteración externa.
  - `fold` : Número de iteración interna.
  - `target_reached` : Un flag booleano indicando si no se encontraron falsos negativos por debajo de TH_{LOW}. Equivalentemente, que $NPV(TH_{LOW}) = 1,0$.
  - `tn_min` : Mínimo del conjunto de predicciones true negatives.
  - `fn_min` : Mínimo del conjunto de predicciones false negatives.
  - `tp_max` : Máximo del conjunto de predicciones true positives.
  - `fp_max` : Máximo del conjunto de predicciones false positives.
  - `tn <= lt` : Número de predicciones true negatives que son menores o igual a TH_{LOW}.
  - `fn <= lt` : Número de predicciones false negatives que son menores o igual a TH_{LOW}.
  - `tp >lt` : Número de predicciones true positives que son mayores a TH_{LOW}.
  - `tp >= ht` : Número de predicciones true positives que son mayores o igual a TH_{HIGH}.
  - `fp >= ht` : Número de predicciones false positives que son mayores o igual a TH_{HIGH}.
  - `npv_lowThr` : NPV respecto a $TH_{LOW} = 0,05$.
  - `recall_lowThr` : *Recall* respecto a $TH_{LOW} = 0,05$.
  - `precision_highThr` : *Precision* respecto a $TH_{HIGH} = 0,95$.
  - `tn <0.5` : Número de predicciones true negatives que son menores a $0,5$.
  - `fn <0.5` : Número de predicciones false negatives que son menores a $0,5$.
  - `tp >= 0.5` : Número de predicciones true positives que son mayores o igual a $0,5$.
  - `fp >= 0.5` : Número de predicciones false positives que son mayores o igual a 0,5.
  - `recall` : *Recall* respecto a $0,5$.
  - `precision` : *Precision* respecto a $0,5$.
  - `specificity` : $Specificity$ respecto a $0,5$.
  - `npv` : NPV respecto a $0,5$.
  - `F2` : $F2$ respecto a $0,5$.

- 5 archivos `.csv` para el almacenamiento de las predicciones y la clasificación de las mismas:

  - Un archivo `predictions.csv` que almacena las predicciones obtenidas de inferir el fold de validación.
  - Un archivo `true_negatives.csv` que almacena los elementos de las predicciones que son true negatives respecto a 0.5.
  - Un archivo `false_negatives.csv` que almacena los elementos de las predicciones que son false negatives respecto a 0.5.
  - Un archivo `true_positives.csv` que almacena los elementos de las predicciones que son true positives respecto a 0.5.
  - Un archivo `false_positives.csv` que almacena los elementos de las predicciones que son false positives respecto a 0.5.

Luego se prosigue a realizar las modificaciones de los pesos. Para esto, primero se inspeccionan las métricas $NPV(TH_{LOW})$ y $Precision(TH_{HIGH})$. Si $NPV(TH_{LOW})$ es distinto de $1,0$, significa que existen falsos negativos por debajo de $TH_{LOW} = 0,05$, por lo que se realiza una búsqueda de dichos falsos negativos y una vez que se identifican, se modifica el peso asociado a dicha muestra multiplicándolo por un factor incremental definido como `WEIGHTS_MODIFICATION_RATIO`. Este factor puede tomar valores mayores a uno. Se probaron algunos valores posibles y el que lanzó mejores resultados fue `WEIGHTS_MODIFICATION_RATIO = 1.001`, lo que significa que el peso se incrementa en un 0,1 %. Del mismo modo, si $Precision(TH_{HIGH})$ es distinto de $1,0$, significa que existen falsos positivos por encima de $TH_{HIGH} = 0,95$. Se busca cuales muestras corresponden a estos falsos positivos, y su pesos asociado se multiplica también por `WEIGHTS_MODIFICATION_RATIO`. El valor de `WEIGHTS_MODIFICATION_RATIO` se implementó manualmente de forma empírica, y se observó que valores como $1,05$ o $1,1$ generan demasiada estocasticidad. El factor `WEIGHTS_MODIFICATION_RATIO` juega un rol similar al *paso de aprendizaje* o *learning rate* aplicado al método del gradiente descendiente que se implementa en el procedimiento de *backpropagation*. A continuación, un pseudocódigo de lo detallado en este párrafo.

```
LOW_THRESHOLD = 0.05
HIGH_THRESHOLD = 0.95
WEIGHTS_MODIFICATION_RATIO = 1.001

if metrics['npv_lowThresh'] != 1.0:
    for i in range(len(predictions)):
        if y_val_fold[i] == 1.0 and predictions[i] <= LOW_THRESHOLD:
            samples_weights[val_index[i]] *= WEIGHTS_MODIFICATION_RATIO

if metrics['precision_highThresh'] != 1.0:
    for i in range(len(predictions)):
        if y_val_fold[i] == 0.0 and predictions[i] >= HIGH_THRESHOLD:
            samples_weights[val_index[i]] *= WEIGHTS_MODIFICATION_RATIO
```

Finalmente, antes de pasar a la siguiente iteración, se corrió una inferencia en el dataset de validación con el modelo creado, computando nuevamente las métricas y almacenando en una versión separada de archivos todos los datos mencionados anteriormente, con las métricas y la clasificación de las predicciones para el dataset de validación.

Parámetros Para cerrar esta sección, en la tabla 11 se presentan los valores de los parámetros aplicados al procedimiento de entrenamiento y optimización explicado en esta sección.

Parámetro	Valor
Iteraciones	1000
Folds	8
TH_{LOW}	0,05
TH_{HIGH}	0,95
Factor incremental de los pesos	1,001

Cuadro 11: Parámetros empleados en el proceso de entrenamiento y optimización

4.5. Análisis de los resultados del método de optimización

El análisis de los resultados arrojados por el método de optimización fue realizado sobre el conjunto de datos recopilado al correr la inferencia sobre el dataset de validación y el dataset de testing. Estos dataset no intervinieron en el proceso de entrenamiento y optimización, ni fueron asignados pesos a sus muestras.

A continuación, se presentan y enumeran los análisis que se realizaron sobre los datos recopilados.

1. **Cantidad de folds por iteración que lograron el objetivo de no presentar falsos negativos por debajo de** TH_{LOW}. Para cada iteración de entrenamiento, se contabilizó cuántos modelos lograron tener métrica $NPV(0,05) = 1{,}0$. Este es un análisis grueso, considerando que si todos los modelos en una iteración lograran este objetivo, se puede confirmar que el método de optimización habría sido muy eficiente.

2. **Métricas estándares promedio por iteración** Para cada iteración, se calculó el promedio de las métricas de los 8 folds.

3. **Métricas customizadas con thresholds promedio por iteración** Para cada iteración, se calculó el promedio de las métricas $NPV(TH_{LOW})$, $Recall(TH_{LOW})$ y $Precision(TH_{HIGH})$

4. **Selección de mejores modelos** Se filtraron los modelos que obtuvieron métricas $NPV(0,05) = 1{,}0$ y $Recall(0,05) = 1{,}0$. Se optó por seleccionar modelos que detectan muestras negativas con 100 % de certeza, porque esto permite aplicar los beneficios del edge computing en el ahorro de energía y mejorar la autonomía del dispositivo. Luego, se realizó un ranking según la cantidad de predicciones true negatives que están por debajo de 0,05, siendo el mejor caso aquel modelo que predijo mayor número de true negatives menores a 0,05.

5. **Inferencia en dataset de testing** Para cada modelo seleccionado en el punto anterior, se corrió una inferencia en el dataset de testing, y se registraron las métricas de igual modo que en los procesos anteriores, generando los mismos tipos de archivo `.csv`.

6. **Selección del mejor modelo** A partir de las performance en los dataset de validación y testing, se seleccionó un ganador. Este fue el que mantuvo su performance en ambos dataset, logrando en ambos caso el mejor desempeño.

7. **Conversión y cuantización con TensorFlow Lite** El mejor modelo seleccionado en el paso anterior, se convirtió a TensorFlow Lite, aplicando cuantización de tipo **Full Integer (integer only) Quantization**.

8. **Inferencia en dataset de testing de modelos TFLite** El modelo convertido a TFLite en el punto anterior, se infirió también en el dataset de testing, y se registraron nuevamente las métricas de igual modo que en los procesos anteriores, generando los mismos tipos de archivo `.csv`, para la evaluación del desempeño del modelo, ahora en formato tflite.

9. **Conversión TensorFlow Lite for Microcontrollers** Finalmente, el modelo seleccionado, fue convertido también al formato de TensorFlow for Microcontrollers, esto es, en formato de un array en C, para poder embeberlo en un microcontrolador.

4.6. Modelo embebido en un microcontrolador

Luego de todo el trabajo realizado hasta este punto, era casi que una obligación embeber un modelo de TensorFlow Lite Micro en un microcontrolador y verlo funcionar.

Para poder llevar a cabo esta tarea, se debió encontrar respuesta a las siguientes preguntas:

1. ¿Cómo se incluye un modelos de TensorFlow Lite Microcontrollers en el firmware de un microcontrolador?

2. ¿Cómo funciona la librería de C++ de TensorFlow for Microntrollers, y cómo interacciona con el modelo?

3. ¿Cómo va a recibir el modelo las entradas a inferir?

4. ¿Cómo se registraran las predicciones realizadas por el modelo?

5. ¿Cómo se procederá a medir el consumo energético del microcontrolador durante la inferencia del modelo?

Para la primer pregunta, basta con incluir en el proyecto el archivo `.cc` generado con la herramienta `xxd`, en conjunto con su archivo header `.h` en el cual se definen las variables definidas en `.cc`, pero agregando la keyword `extern`.

```
extern const unsigned char tflite_model_array[];
extern const unsigned int tflite_model_len;
```

Luego, es necesario incluir la librería de C++ de TensorFlow Lite for Microcontrollers en el entorno de desarrollo con el que se esté trabajando. Esta librería está en constante desarrollo y se encuentra disponible en su repositorio de GitHub.[7] Sobre la plataforma de desarrollo, se decidió trabajar con dos entornos distintos. Por un lado la plataforma Arduino, y por el otro lado la plataforma ESP32 de Espressif Systems. Con respecto a la plataforma Arduino, su elección se debe a la gran flexibilidad y facilidad que ofrece Arduino a la hora de desarrollar firmware para sistemas embebidos, por la gran aceptación que tiene en la comunidad debido a ser open-source, y además ha sido la plataforma de referencia que han utilizado los principales responsables del desarrollo de TensorFlow Lite Micro, en su libro (Warden y Situnayake, 2019). Para el hardware, se seleccionó la placa Arduino Nano 33 BLE Sense que se ilustra en la figura 17. Con respecto a la plataforma ESP32, su elección se debe a que el dispositivo de Galeno_Sys está basado en un módulo o SoC ESP32. Se utilizó la placa DOIT DevKit V1 la cual se muestra en la figura 18, y para su programación se hizo uso del framework ESP-IDF, desarrollado por Espressif Systems. La tabla 12 contiene una descripción de las características de cada una de las placas utilizadas.

Placa	Arduino Nano 33 BLE Sense	DOIT DevKit V1
MCU	nRF52840	ESP32-WROOM-32
Arquitectura	32-bit ARM Cortex-M4	Xtensa 32-bit LX6
Fabricante	Nordic Semiconductors	Espressif Systems
Cores	1	2
Clock	64 MHz	240 MHz
Flash	1 MB	4 MB
SRAM	256 kb	520 KB
Framework	Arduino (Mbed OS)	ESP-IDF (FreeRTOS)

Cuadro 12: Características de las placas de evaluación.

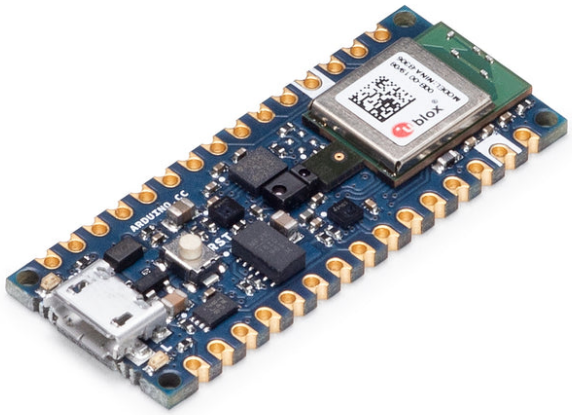

Figura 17: Arduino Nano 33 BLE Sense.

[7] https://github.com/tensorflow/tflite-micro

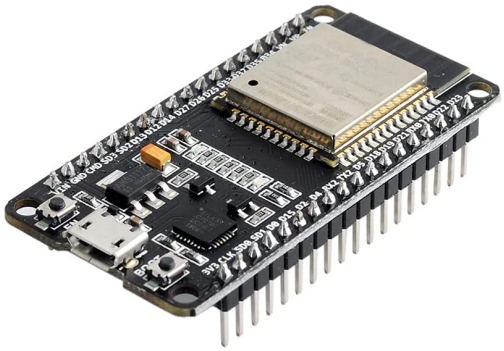

Figura 18: DOIT DevKit V1.

Para que el modelo pudiera inferir muestras y realizar predicciones corriendo en el propio microcontrolador, era necesario resolver cómo iba a recibir el modelo dichas muestras. Almacenar muestras en la propia memoria del microcontrolador es imposible. Se resolvió que las muestras serían transferida al microcontrolador mediante comunicación serial. Así fue que se desarrolló un script en Python que toma muestras del dataset de testing, y las envía a través del puerto serial al microcontrolador. Luego de algunos intentos, fue necesario tomar ciertos recaudos para poder lograr el monitoreo de la comunicación y verificar que la misma ocurre de forma eficiente:

- Las muestras se envían cuantizadas. Previo a enviar muestras al microcontrolador, se realizó la cuantización del dataset de testing según las características del modelo de tensorflow lite. Con esto se logra reducir la carga de datos a transferir. Por defecto, las muestras del dataset se encuentran en formato `float` ocupando 4 bytes cada elemento. Al cuantizar, cada elemento de una muestra se representa en un único byte, reduciendo en 4 la cantidad de datos a enviar por cada muestra del dataset. Así, cada muestra del dataset está compuesta por 1000 bytes. Esto se logró haciendo uso del intérprete de TensorFlow Lite.

- Las muestras se envían en batches de 50 elementos, tomando un total de 20 batches para completar el envío de una muestra del dataset.

- El microcontrolador gerencia la comunicación. Recibe y contabiliza los bytes recepcionados, los almacena en el tensor de entrada del modelo mediante un índice y le indica al script en Python, la posición a partir de la cual espera recibir datos.

- Se estableció un control de error básico mediante checksum por cada batch enviado. De forma que la suma de los elementos enviados en un batch, debe coincidir con la suma de los elementos recibidos por el microcontrolador.

- El microcontrolador debe devolver un string conteniendo la siguiente información: número de inferencia, tiempo de inferencia, salida cuantizada del modelo y salida real del modelo. Estos datos son almacenados en en un archivo `.csv` agregando también la etiqueta de la muestra y el índice de la muestra en el dataset de testing.

Para la medición de consumo, se decidió utilizar una de las placas mencionadas en (C. Banbury y cols., 2021), la placa X-NUCLEO-LPM01A de ST Microelectronics.

Figura 19: STM32 Nucleo expansion board for power consumption measurement X-NUCLEO-LPM01A.

Esta placa, permite alimentar sistemas de muy bajo consumo (hasta 50 mA), y medir el consumo del mismo. Puede usarse de forma standalone y controlarse mediante la interfaz compuesta por el display LCD, un botón y un joystick, puede conectarse a una PC vía USB y utilizarse con el software del propietario llamado STM32 CubeMonitor-Power, con el software de un fabricante tercero llamada EEMBC EnergyMonitor V2.0, o simplemente se puede comandar a través de la comunicación serial con comandos en formato ASCII. El fabricante pone a disposición la documentación de los comandos que permiten controlar el funcionamiento del dispositivo por comunicación serial, desde una aplicación custom. Fue así que se optó por desarrollar otro script en Python para el control del dispositivo a través de su lista de comandos ASCII.

Como resultado, se desarrolló un script en Python que implementa `multiprocessing` para poder correr dos tareas en paralelos:

1. Una tarea encargada de la comunicación con la placa X-NUCLEO-LPM01A, responsable de configurar inicialmente el funcionamiento del dispositivo, de iniciar y finalizar el proceso de muestreo del consumo energético, y de registrar los datos generados en el proceso de medición.

2. Una segunda tarea, encargada de comunicarse con el Arduino Nano 33 BLE Sense, enviar muestras del dataset a inferir, recibir los resultados de la inferencia y generar registro de los mismos. Esta tarea es inicializada desde la tarea anterior, y se mantiene en comunicación con la misma para indicarle el inicio y fin del muestreo.

Para finalizar, se muestra en la figura 20 una foto del prototipo fabricado para las pruebas. El Arduino es alimentado desde la placa X-NUCLEO-LPM01A con 3,3 V, y se comunica con la PC mediante un conversor Serial-USB, siendo necesario aislar el Arduino del conversor para evitar que exista un consumo de corriente residual por parte del conversor, el cual no se desea medir.

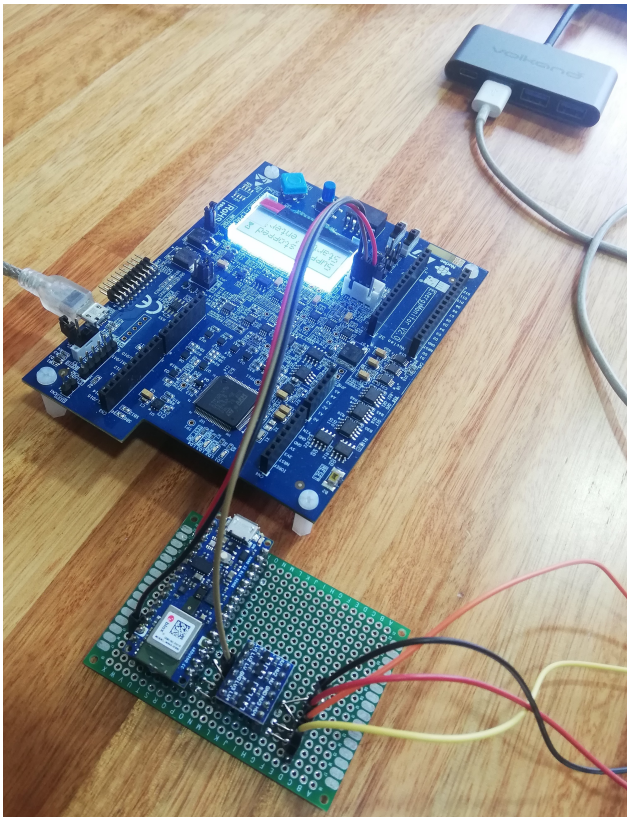

Figura 20: Prototipo desarrollado para las pruebas de inferencias y medición de consumo.

5. Resultados alcanzados

Siguiendo los pasos mencionados en 4.5, se presentan a continuación los resultados alcanzados luego de implementar la metodología descripta en la sección anterior.

Cantidad de folds por iteración que lograron el objetivo de no presentar falsos negativos por debajo de TH_{LOW} En la figura 21 se presenta el conteo de la cantidad de folds por iteración que lograron alcanzar un valor de $NPV(0,05) = 1{,}0$, lo que significa que no presentan falsos negativos por debajo de TH_{LOW}, pero si verdaderos negativos. Se observa que lo máximo que se llegó a obtener, fue 1 fold por iteración con $NPV(0,05) = 1{,}0$, pero se observa que a medida que avanzaron las iteraciones, fue más frecuente este resultado. Se podría llegar a pensar que con una mayor cantidad de iteraciones, se podrían obtener mejores resultados, pero por lo pronto no lo sabemos.

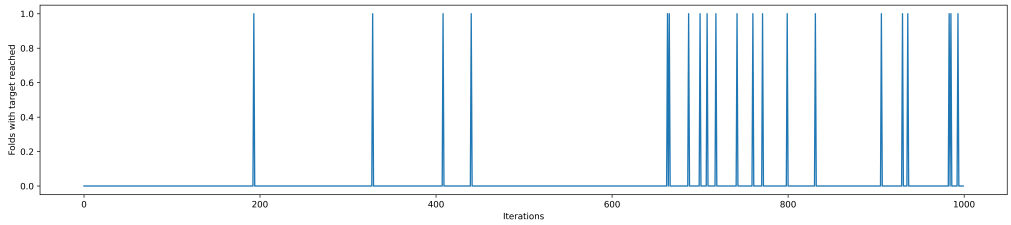

Figura 21: Cantidad de folds por iteración que lograron el objetivo planteado en la metodología.

Métricas estándares promedio por iteración Las figuras 22 a la 26 presentan la evolución de las métricas estándares promedio por iteración. A simple vista, se observa que no existe mejora alguna en ninguna de ellas, en términos generales, sino que mantienen un promedio constante, al cual se le suma una gran estocasticidad durante el proceso que genera oscilaciones. De todos modos, vale la pena aclarar que no es el objetivo de este proceso de optimización mejorar las métricas globales o generales del modelo, sino que como se mencionó anteriormente, el objetivo es minimizar el riesgo de falsos negativos. Se ilustran estas métricas, por el mero hecho que fueron las métricas utilizadas para evaluar inicialmente el modelo ECG-TCN.

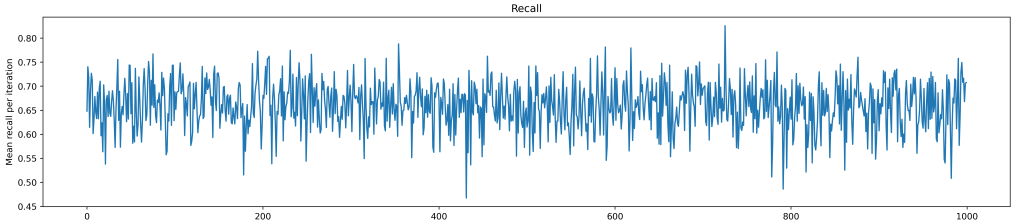

Figura 22: *Recall* o *sensitivity* promedio por iteración.

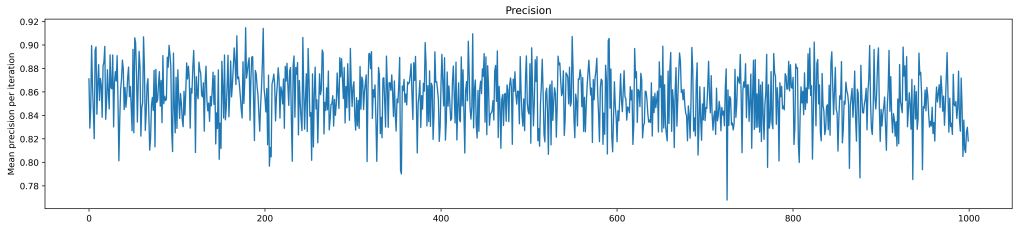

Figura 23: *Precision* promedio por iteración.

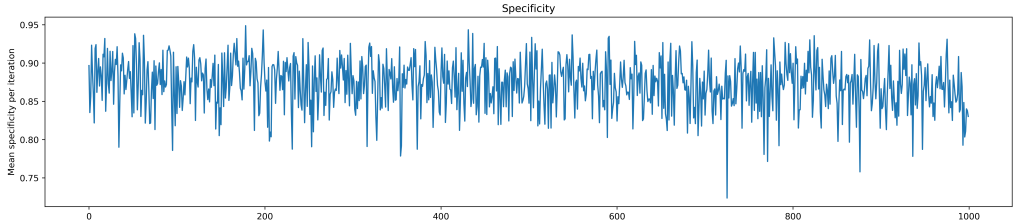

Figura 24: $Specificity$ promedio por iteración.

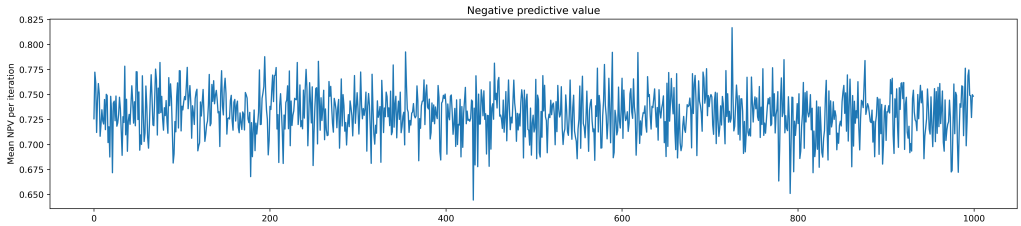

Figura 25: NPV promedio por iteración.

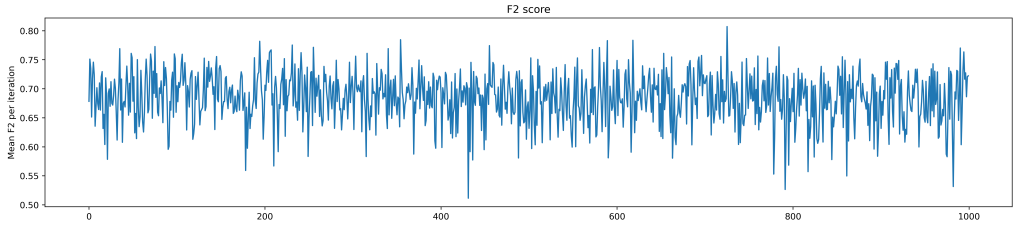

Figura 26: $F2socre$ promedio por iteración.

Métricas customizadas promedio por iteración Estas métricas son las que son realmente de nuestro interés, dado que son las que pueden reflejar si se logró optimizar nuestros modelos y lograr el objetivo. A modo general, no parece haber mejoras significativas. El $Recall(0,05)$ pareciera presentar una leve mejora en términos generales, mientras que $NPV(0,05)$ pareciera desmejorar, pero a todas se le suma también un una gran estocasticidad que genera oscilaciones.

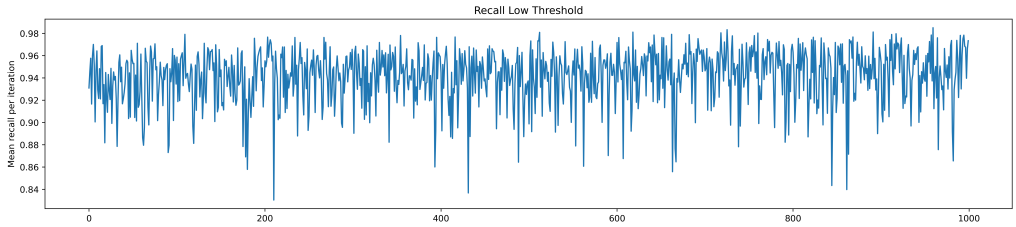

Figura 27: $Recall(0,05)$ o $sensitivity(0,05)$ promedio por iteración.

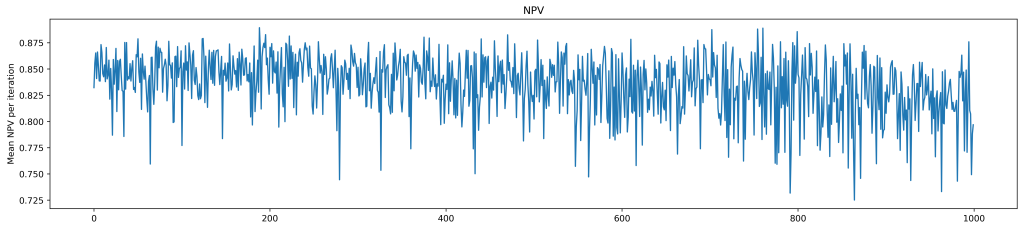

Figura 28: $NPV(0,05)$ promedio por iteración.

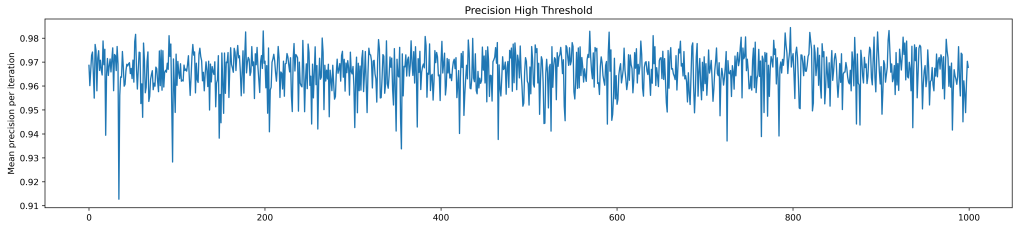

Figura 29: $Precision(0,95)$ promedio por iteración.

Selección de los mejores modelos De todos los modelos, se seleccionaron aquellos que en el dataset de validación lograron predecir muestras negativas con certeza del 100 %, es decir, que $Recall(TH_{LOW}) = 1{,}0$ y $NPV(TH_{LOW}) = 1{,}0$. Se obtuvo un total de 21 modelos. Posteriormente se ordenaron estos 21 modelos según la cantidad de predicciones true negatives inferiores a 0,05. La tabla 13 lista los 5 mejores modelos con sus respectivas métricas, y la tabla 14 presenta los valores máximos y mínimos de las predicciones positivas y negativas de los mismos.

Model	$TN \leq 0{,}05$	$FN \leq 0{,}05$	$TP > 0{,}05$	$TP \geq 0{,}95$	$FP \geq 0{,}95$	Precision(0,95)
719-8	15	0	1199	673	63	0.914402
664-8	12	0	1199	685	87	0.887306
994-8	10	0	1199	818	91	0.899890
441-7	10	0	1199	677	46	0.936376
688-7	8	0	1199	757	87	0.896919

Cuadro 13: Métricas customizadas con TH_{LOW} y TH_{HIGH} para los 5 mejores modelos TensorFlow en el dataset de validación.

Model	TN min	FN min	TP max	FP max
719-8	0.005460	0.050749	0.999995	0.999884
664-8	0.024497	0.050719	0.999999	0.999775
994-8	0.016059	0.053685	1.000000	1.000000
441-7	0.003198	0.052023	0.999989	0.999158
688-7	0.013679	0.057568	0.999972	0.999014

Cuadro 14: Valores máximos y mínimos de las predicciones positivas y negativas de los 5 mejores modelos TensorFlow en el dataset de validación.

En la tabla 13 se observa como la cantidad de predicciones negativas falsas es 0 por debajo de 0,05 tal como queremos, mientras que sí existen predicciones negativas verdaderas por debajo de 0,05. A diferencia de lo que ocurre en el modelo ECG-TCN sin optimizar, en este caso podemos tener certeza del 100 % que dichas señales cuyas predicciones son menores a 0,05, son señales normales. En la tabla 14, se observa en la columna *FN min*, que el mínimo valor del conjunto de predicciones negativas falsas, es mayor a 0,05, tal como se desea.

Inferencia en dataset de testing Para los 21 modelos resultantes en el paso anterior, se corrió una inferencia en el dataset de testing. Nuevamente se realizó una evaluación de los resultados obtenidos y se filtraron aquellos modelos que lograron mantener las métricas $Recall(TH_{LOW}) = 1{,}0$ y $NPV(TH_{LOW}) = 1{,}0$. De 21 modelos, 17 modelos mantuvieron dichas métricas. Luego, se ordenaron los 17 modelos según la cantidad de predicciones true negatives menores a 0,05. La tabla 15 lista los 5 mejores modelos con sus respectivas métricas, y la tabla 16 presenta los valores máximos y mínimos de las predicciones positivas y negativas de los mismos.

Model	$TN \leq 0{,}05$	$FN \leq 0{,}05$	$TP > 0{,}05$	$TP \geq 0{,}95$	$FP \geq 0{,}95$	Precision(0,95)
719-8	13	0	1199	658	60	0.916435
664-8	11	0	1199	688	6	0.888889
329-8	7	0	1199	457	41	0.917671
688-7	6	0	1199	767	95	0.889791
800-7	5	0	1199	595	70	0.894737

Cuadro 15: Métricas customizadas con TH_{LOW} y TH_{HIGH} para los 5 mejores modelos TensorFlow en el dataset de testing.

Model	TN min	FN min	TP max	FP max
719-8	0.005460	0.050749	0.999995	0.999884
664-8	0.024497	0.050719	0.999999	0.999775
994-8	0.016059	0.053685	1.000000	1.000000
441-7	0.003198	0.052023	0.999989	0.999158
688-7	0.013679	0.057568	0.999972	0.999014

Cuadro 16: Valores máximos y mínimos de las predicciones positivas y negativas de los 5 mejores modelos TensorFlow en el dataset de testing.

Selección del mejor modelo De las tablas 13 y 15, se observa que el modelo 719-8 no solo logra el objetivo de alcanzar las métricas $Recall(TH_{LOW}) = 1{,}0$ y $NPV(TH_{LOW}) = 1{,}0$, sino que además se posiciona en el primer lugar del ranking según la cantidad de predicciones true negatives por debajo de $0,05$ en ambos casos. Se concluye que el modelo mantiene su performance en ambos dataset y por lo tanto, **el modelo 719-8 es el mejor modelo**.

A continuación se presentan los histogramas de las predicciones del modelo 719-8 para los dataset de validación y testing, en los cuales se puede verificar de manera visual la ausencia de predicciones false negatives por debajo de 0,05, mientras que sí existen predicciones true negatives por debajo de dicho umbral, predicciones que presentan una certeza del 100 % debido a la ausencia de falsos negativos por debajo de 0,05. Además, se observa una mejora en las predicciones para las muestras positivas tal como se pretendía, ya que se concentran en gran cantidad en un entorno cercano a $1,0$, mientras que se encuentran muy pocos valores de falsos negativos por debajo de 0,5, y ninguno por debajo de 0,05, alcanzando el objetivo propuesto. El costo de esto, parece ser una desmejora en la distribución de las predicciones de las muestras negativas, dado que aumenta la cantidad de falsos positivos, y disminuye la cantidad de predicciones true negatives en un entorno cercano a $0,0$.

Figura 30: Histograma de predicciones del modelo 719-8 TensorFlow en el dataset de validación.

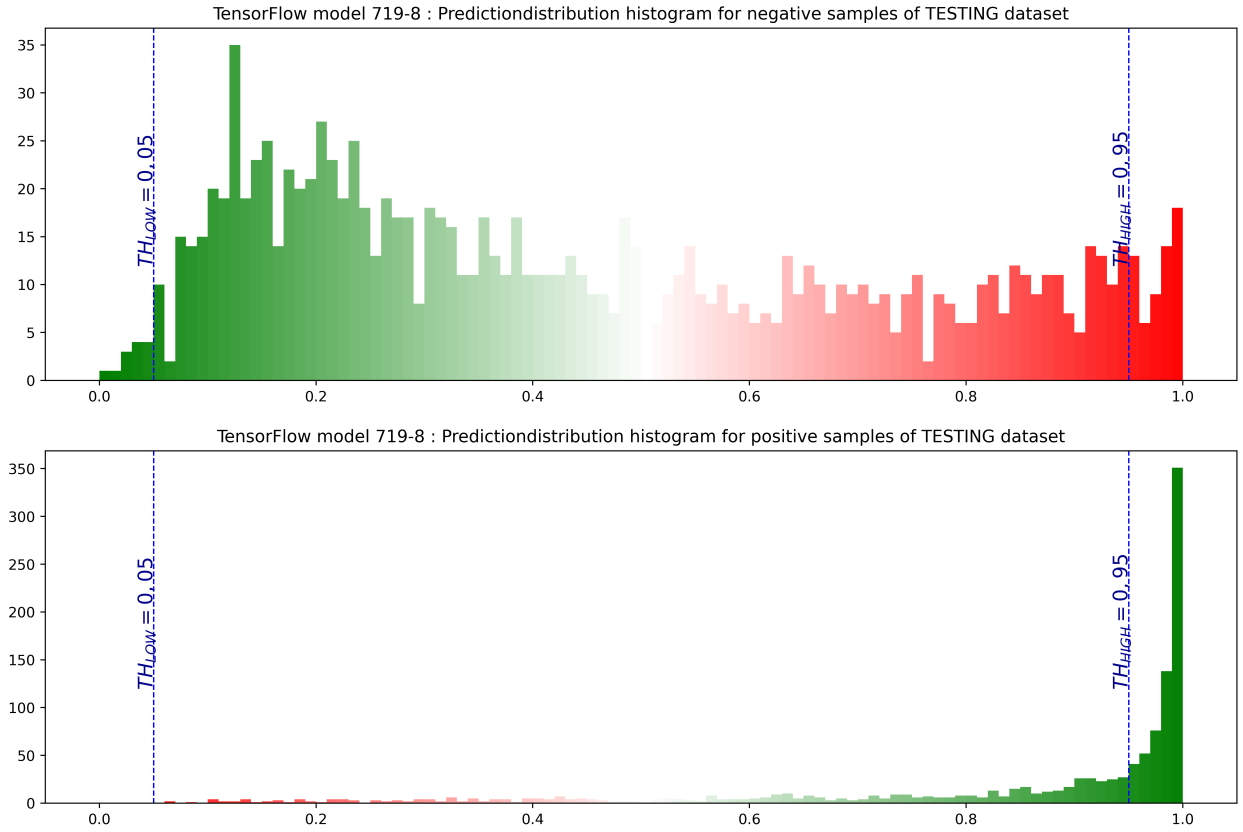

Figura 31: Histograma de predicciones del modelo 719-8 TensorFlow en el dataset de testing.

Comparación con ECG-TCN En este punto, nos detenemos a comparar el modelo ECG-TCN y el modelo 719-8. Pero antes, volvemos a mencionar las características que esta aplicación trata debido a su tipo y naturaleza, y que han fijado los objetivos de este trabajo. En primer lugar, dijimos que por tratarse de una aplicación en el área de la salud, exige fuertemente un excelente desempeño en la identificación de muestras positivas y la disminución de falsos negativos, mientras que por tratarse de una aplicación de edge computing, exige poder tener certeza total en la identificación de muestras negativas para tomar la decisión de no enviar dicha muestra a la nube y ahorrar energía.

Modelo	Recall	Precision	Specificity	NPV	F2
ECG-TCN	0,53628	0,896792	0,938023	0,668258	0,583167
719-8	0,894912	0,697206	0,609715	0,852459	0,846882

Cuadro 17: Comparación de métricas estándares.

Modelo	NPV(0,05)	Recall(0,05)	Precision(0,95)
ECG-TCN	0,850967	0,890742	0,958678
719-8	1,0	1,0	0,916435

Cuadro 18: Comparación de métricas customizadas.

La tabla 17 presenta una comparación de las métricas estándares (threshold 0,5) de ambos modelos. Se observa notoriamente un aumento del *Recall* en el modelo optimizado, pasando de 0,53628 a 0,894912, lo que da a entender que existe una mejora en la detección de las muestras positivas. Del mismo modo, el aumento del NPV de 0,668258 a 0,852459 puede ser indicio de la disminución de falsos negativos.

La tabla 18 muestra claramente que se optimizó y alcanzó el objetivo de que las métricas *NPV(0,05)* y *Recall(0,05)* valgan 1,0, indicando la ausencia de falsos negativos por debajo de 0,05.

Para comprender un poco más los resultados obtenidos, se comparan los histogramas de las predicciones de cada modelo por tipo de señales. Observando los histogramas superiores de la figura 32, se observa que el modelo ECG-TCN sin optimizar, presenta una mejor distribución para las muestras negativas, acumulando la mayor cantidad de ellas en un entorno del 0,0. Mientras que el modelo 719-8 presenta una mayor cantidad de falsos positivos y disminuye la cantidad de true negatives cercanos a 0,0, disminuyendo la probabilidad de encontrar predicciones true negatives por debajo de nuestro umbral 0,05. Sin embargo, habíamos indicado en el modelo ECG-TCN, que el buen rendimiento sobre las muestras negativas de nada servía en nuestro caso debido a la cantidad de los falsos negativos y a la distribución de los mismos, que hacen que no exista seguridad para identificar muestras negativas. Se deduce que el proceso de optimización con los pesos, ha tenido un impacto sobre las muestras negativas poco deseable, el cual a priori podría resultar contraproducente. Pero en contra partida, comparando los histogramas inferiores de la figura 32, observamos que el proceso de optimización por pesos, ha tenido un impacto positivo y favorable sobre las predicciones de las muestras positivas, debido a que disminuye notoriamente el número de falsos negativos, los cuales además son mayores a nuestro umbral 0,05, no existen falsos negativos por debajo de 0,05, y la cantidad de predicciones positivas en el entorno de 1,0 aumentó casi hasta triplicar. Importante aquí es tener presente que las escalas de los ejes de las abscisas no son iguales en los histogramas. Dado que nos hemos enfocado en encontrar un modelo que principalmente pudiese predecir señales negativas con una certeza del 100 % para así hacer uso de los beneficios del edge computing, y poder generar un ahorro de energía, se valora positivamente el cambio generado a través del proceso de optimización.

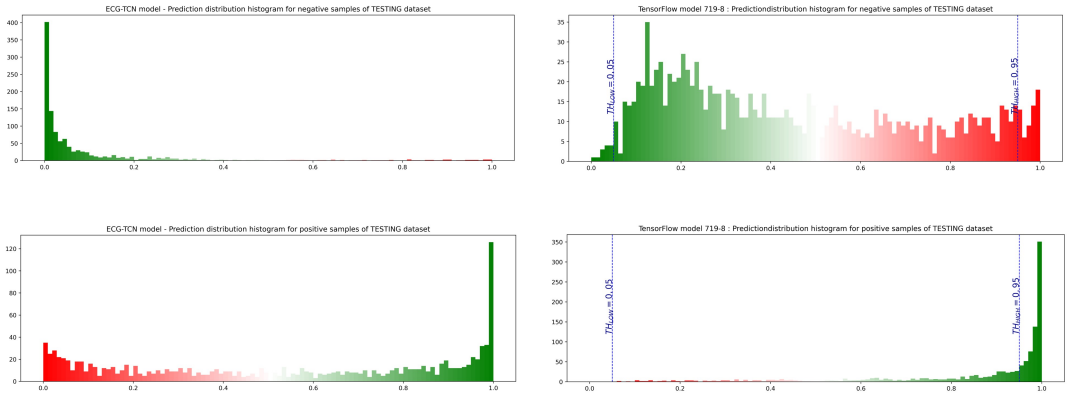

Figura 32: Comparación de histogramas. Los histogramas de la izquierda corresponden al modelo ECG-TCN, mientras que los histogramas de la derecha corresponden al modelo 719-8.

Para cerrar esta comparación, se muestran a continuación las gráficas de las curvas de *Recall*, *Precisión*, *Specificity* y NPV en función del threshold. Se puede observar en el caso del modelo 719-8 una mejora en el *Recall*, el cual baja muy lentamente a lo largo del intervalo de theshold, haciéndolo abruptamente muy cerca del 1,0 dando a entender que allí se concentra la mayor cantidad de muestras positivas. Mientras que el NPV no presenta caídas significativas, y alcanza el valor de 1,0 para threshold igual o menor a 0,05.

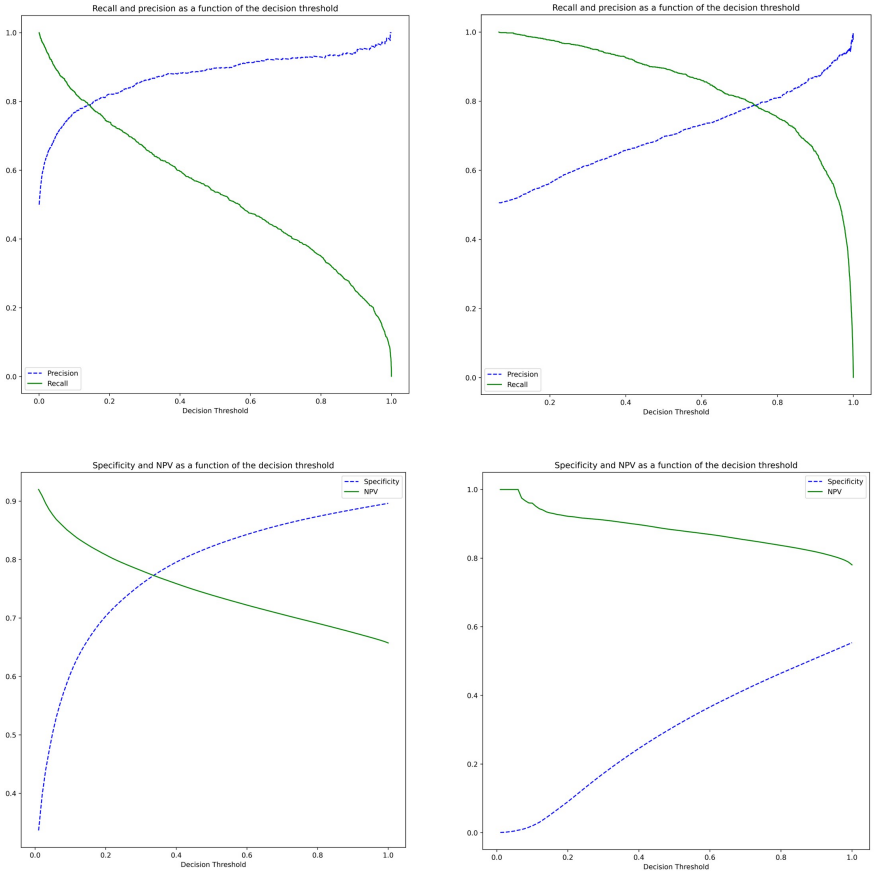

Figura 33: Curvas Recall-Precisin vs threshold y Specificity-NPV vs threshold. Las gráficas de la izquierda corresponden al modelo ECG-TCN, mientras que las gráficas de la derecha corresponden al modelo 719-8.

Conversión y cuantización con TensorFlow Lite El modelo 719-8 se convirtió en formato de TensorFlow Lite, haciendo uso del conversor de TensorFlow Lite. En el proceso de conversión, se aplicó *full integer only quantization*, de forma que tanto los pesos, como las activaciones internas, las entradas y la salida, son valores enteros de 8 bits. El modelo convertido fue inferido en el dataset de testing, y una vez más mantuvo su performance. La figura 34 presenta los histogramas de las predicciones del modelo TensorFlow Lite, mientras que las tablas 19 y 20 presentan las métricas obtenidas.

Model	$TN \leq 0{,}05$	$FN \leq 0{,}05$	$TP > 0{,}05$	$TP \geq 0{,}95$	$FP \geq 0{,}95$	Precision(0,95)
719-8	14	0	1199	534	36	0.936842

Cuadro 19: Métricas customizadas con TH_{LOW} y TH_{HIGH} para el modelo 719-8 de TensorFlow Lite en el dataset de testing.

Model	TN min	FN min	TP max	FP max
719-8	0.015625	0.074219	0.996094	0.996094

Cuadro 20: Valores máximos y mínimos de las predicciones positivas y negativas del modelo 719-8 TensorFlow Lite en el dataset de testing.

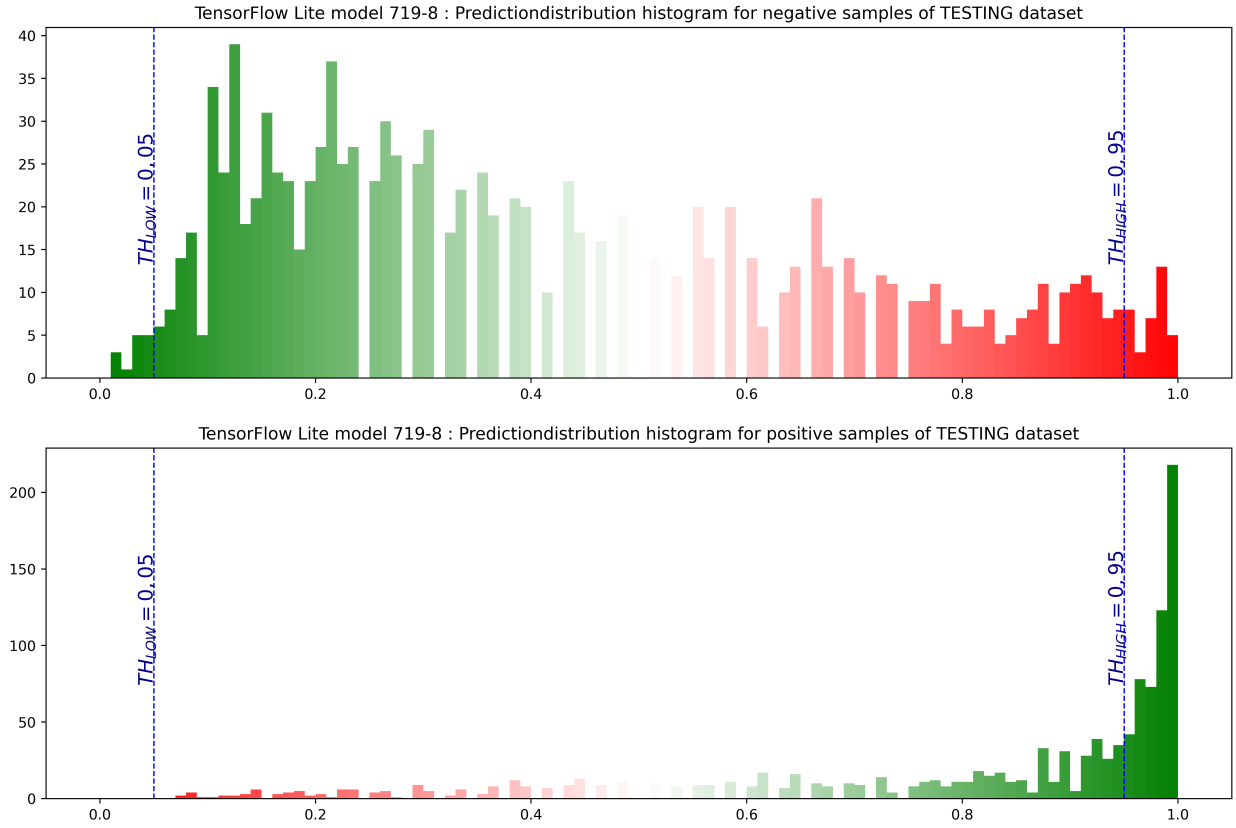

Figura 34: Histograma de predicciones del modelo 719-8 TensorFlow Lite para el dataset de testing.

Recursos El modelo en formato `tflite` fue convertido a un array de bytes en `C` haciendo uso de la herramienta de unix `xxd`, tal como se mencionó en 3.3.4. Luego se integró en el firmware desarrollado para cada una de las placas utilizadas junto con la librería de C++ de TensorFlow Lite Micro. Por último, empíricamente se determinó el tamaño del área de memoria llamada areana o tensor arena, presentada en 3.3.3. La tabla 21 ilustra los recursos del microcontrolador consumidos por el modelo.

Recurso	Memoria	Tamaño	Arduino Nano 33 BLE Sense	DIOT DevKit V1
Modelo	SRAM	36176 bytes	14,13 %	6,96 %
Tensor arena	SRAM	61 KB	23,83 %	11,73 %
API	Flash	16-20 KB	1,6 % - 2,0 %	0,4 % - 0,5 %

Cuadro 21: Recursos de memoria consumidos por el modelo.

Consumo energético Al procesar el dataset de testing con el modelo TensorFlow Lite, el mismo clasificó 14 señales normales con valores de predicciones inferiores a $0,05$ para cada caso. Estas son las señales normales que el modelo puede detectar como normales con certeza, y para las cuales el modelo permite ahorrar energía al evitar que el dispositivo envíe estos datos a la nube.

Considerando que el dataset de testing tiene 1194 muestras negativas, las 14 muestras representan el $1,17\,\%$, por lo que 1 de cada 100 muestras negativas no sería enviada a la nube y el dispositivo podría ahorrar energía.

A continuación se plantea el caso de un dispositivo IOT ficticio para dimensionar, a groso modo, el ahorro energético que puede generarse por cada muestra que no es enviada a la nube. Para esto se considera que el dispositivo IOT está compuesto por un Arduino Nano 33 BLE Sense, un módulo celular NB-IoT o LTE Cat-M1 y una batería de 3,7 V y 1200 mAh.

Supóngase, que el módulo celular tiene un consumo máximo promedio de 250 mA durante la transmisión de datos, y una tasa de transferencia de 20 kbps. Considerando que el paquete de datos a enviar tiene un tamaño de 13 KB, idealmente la transmisión de esta cantidad de datos hacia la nube sería realizada en 650 ms. Considerando también que el módulo se alimenta directamente de la batería, la potencia consumida por el módulo es $0,250A \times 3{,}7V = 0,925W$, lo que da un consumo total de energía de $0,925W \times 0{,}650s = 0,60125J$. Para el caso de las muestras normales detectadas con una predicción inferior a $0,05$, se genera entonces un ahorro de $0,60125J$ por muestra.

La batería por su parte tiene una capacidad total de $3,7V \times 1200mAh = 4,44Wh$, lo que se traduce en $4,44Wh \times 3600s = 15984J$. Si dividimos el consumo durante una transmisión entre la capacidad de la batería, tenemos que $0,60125J \div 15984J = 3,76 \times 10^{-5}$. Esto significa que por cada muestra normal no enviada a la nube se ahora un $3,76 \times 10^{-3}\,\%$ de la capacidad total de la batería.

6. Conclusiones y trabajos futuros

Se concluye que se logró optimizar el modelo ECG-TCN original a favor de los requerimientos establecidos por el tipo de aplicación tratado. El método de optimización que consiste en la aplicación de pesos a las muestras del dataset de entrenamiento para ponderar el error de las mismas y forzar un cambio en la distribución de las predicciones, permitió alcanzar el objetivo de mejorar la clasificación de muestras positivas y el objetivo de predecir muestras negativas con una certeza del 100 % con el fin de disminuir el consumo y generar un ahorro de energía.

Este método logró desplazar aquellas predicciones falsas negativas que se encontraban en un entorno cercano al $0,0$, de forma que sean mayores a un umbral de $0,05$, manteniendo aún predicciones negativas verdaderas menores a $0,05$.

A modo general, el foco de nuestro proyecto era poder dotar de inteligencia a un dispositivo IOT para que pueda tomar decisiones por sí mismo y aprovechar los beneficios que el edge computing brinda. Este objetivo se cumplió satisfactoriamente al encontrar un modelo de machine learning capaz de correr en el microcontrolador del dispositivo IOT, que logra detectar señales normales con la certeza y la seguridad de optar por no enviar dichas señales a la nube y generar un ahorro de energía que prolongue la autonomía de la batería.

Comparado con los dos trabajos presentados en 3.2 que implementan modelos de tiny machine learning para la clasificación de señales ECG, la solución lograda en este trabajo se obtuvo desde una perspectiva que considera fuertemente la puesta en marcha del modelo en un ambiente de producción. Haciendo un fuerte hincapié en los datos utilizados para el entrenamiento, para lograr un resultado confiable, siendo el dataset muy superior a los dataset utilizados en los antecedentes presentados, por la cantidad de muestras que contiene, la diversidad de las mismas, y que provienen de una gran cantidad de pacientes.

Como trabajos futuros que pueden seguir a partir de la culminación de este proyecto, se identifican los los siguientes casos:

- Investigar en mayor profundidad el impacto del valor del ratio de modificación de los pesos y del número de iteraciones en el proceso de optimización.
- Implementación de un sistema de control cerrado para la modificación de los pesos en función del impacto que los mismos generan, de forma controlada y automática.
- Optimización o búsqueda automática de los thresholds utilizados en las métricas customizadas.
- Clasificación de las señales ECG a partir de la espectrografía de las mismas en imágenes 2D.
- Realizar procesos de optimización de hiper parámetros del modelo mediantes técnicas de grid search o hyperband search, basados en las métricas customizadas.

Referencias

Ávila-Tomás, J. F., Mayer-Pujadas, M. A., y Quesada-Varela, V. J. (2021, ene.). La inteligencia artificial y sus aplicaciones en medicina ii: importancia actual y aplicaciones practicas. *Elsevier*, *53*(1), 81-88. Descargado de `https://www.elsevier.es` doi: 10.1016/j.aprim.2020.04.014

Bai, S., Kolter, J. Z., y Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Banbury, C., Reddi, V. J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., ... others (2021). Mlperf tiny benchmark. *arXiv preprint arXiv:2106.07597*.

Banbury, C. R., Reddi, V. J., Lam, M., Fu, W., Fazel, A., Holleman, J., ... others (2020). Benchmarking tinyml systems: Challenges and direction. *arXiv preprint arXiv:2003.04821*.

Chá Ghiglia, M. M. (2020, dic.). Telemedicina. *Revista Médica del Uruguay*, *36*(4), 411-417. Descargado de `https://revista.rmu.org.uy/ojsrmu311/index.php/rmu/article/view/647` doi: 10.29193/RMU.36.4.9

Clifford, G. D., Liu, C., Moody, B., Li-wei, H. L., Silva, I., Li, Q., ... Mark, R. G. (2017). Af classification from a short single lead ecg recording: The physionet/computing in cardiology challenge 2017. En *2017 computing in cardiology (cinc)* (pp. 1–4).

Faraone, A., y Delgado-Gonzalo, R. (2020). Convolutional-recurrent neural networks on low-power wearable platforms for cardiac arrhythmia detection. En *2020 2nd ieee international conference on artificial intelligence circuits and systems (aicas)* (pp. 153–157).

Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., ... Stanley, H. E. (2000). Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, *101*(23), e215–e220.

He, Y., y Zhao, J. (2019). Temporal convolutional networks for anomaly detection in time series. En *Journal of physics: Conference series* (Vol. 1213, p. 042050).

Ingolfsson, T. M., Wang, X., Hersche, M., Burrello, A., Cavigelli, L., y Benini, L. (2021). Ecg-tcn: Wearable cardiac arrhythmia detection with a temporal convolutional network. En *2021 ieee 3rd international conference on artificial intelligence circuits and systems (aicas)* (p. 1-4). doi: 10.1109/AICAS51828.2021.9458520

Liu, F., Liu, C., Zhao, L., Zhang, X., Wu, X., Xu, X., ... others (2018). An open access database for evaluating the algorithms of electrocardiogram rhythm and morphology abnormality detection. *Journal of Medical Imaging and Health Informatics*, *8*(7), 1368–1373.

Ma, H., Chen, C., Zhu, Q., Yuan, H., Chen, L., y Shu, M. (2021). An ecg signal classification method based on dilated causal convolution. *Computational and mathematical methods in medicine*, *2021*, 6627939. Descargado de `https://europepmc.org/articles/PMC7872762` doi: 10.1155/2021/6627939

Wagner, P., Strodthoff, N., Bousseljot, R.-D., Kreiseler, D., Lunze, F. I., Samek, W., y Schaeffter, T. (2020). Ptb-xl, a large publicly available electrocardiography dataset. *Scientific data*, *7*(1), 1–15.

Warden, P., y Situnayake, D. (2019). *Tinyml: Machine learning with tensorflow lite on arduino and ultra-low-power microcontrollers*. O'Reilly Media.

Zheng, J., Zhang, J., Danioko, S., Yao, H., Guo, H., y Rakovski, C. (2020). A 12-lead electrocardiogram database for arrhythmia research covering more than 10,000 patients. *Scientific Data*, *7*(1), 1–8.