UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

Universidad de Málaga

escuela de ingenierías industriales

Trabajo Fin de Grado

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Noise Removal in ECGs Using Deep Learning

Eliminación de Ruido en ECGs Usando Deep Learning

Autor:
Ricardo Fernández Borrego

Tutor: Francisco Manuel Castro Payán
Departamento: Arquitectura de Computadores
Área de conocimiento: Arquitectura y Tecnología de Computadores

Cotutora: Paula Ruiz Barroso
Departamento: Arquitectura de Computadores
Área de conocimiento: Arquitectura y Tecnología de Computadores

3 de octubre de 2025

Índice

Acrónimos y Abreviaturas xi

Resumen xiii

Abstract xv

1 Introducción 1
- 1.1 Motivación . . . 1
- 1.2 Objetivos . . . 3
- 1.3 Plan de Trabajo . . . 3
- 1.4 Estructura del Documento . . . 4

2 Estado del Arte 5

3 Herramientas Utilizadas 7
- 3.1 Python . . . 7
- 3.2 Librería TensorFlow . . . 8
- 3.3 Librería PyTorch . . . 8
- 3.4 Librería DeepDish . . . 8
- 3.5 Servidor . . . 8

4 Fundamentos Teóricos 9
- 4.1 Características de una Señal ECG . . . 9
- 4.2 Enfermedades Cardiovasculares . . . 11
  - 4.2.1 Trastornos del Ritmo o Arritmias . . . 11
  - 4.2.2 Bloqueos de Rama . . . 12
  - 4.2.3 Cardiopatía Isquémica . . . 13
- 4.3 Introducción a las Redes Neuronales . . . 14
- 4.4 Redes Neuronales Convolucionales . . . 15
- 4.5 Arquitectura de las CNNs . . . 15
  - 4.5.1 Capas . . . 15
    - 4.5.1.1 Capa Convolucional . . . 16
    - 4.5.1.2 Capa de Activación . . . 16
    - 4.5.1.3 Capa de Agrupación o *Pooling Layer* . . . 17
    - 4.5.1.4 Capa de Normalización . . . 18
    - 4.5.1.5 Capa de *Dropout* . . . 18

4.5.1.6 Capa Totalmente Conectada o *Fully-Connected Layer* 18
4.5.2 Funciones de pérdida . . . 18
4.5.3 Optimizadores . . . 20
4.5.3.1 Descenso del gradiente estocástico (*Stochastic Gradient Descent (SGD)*) . . . 20
4.5.3.2 Adam . . . 20
4.5.3.3 AdamW . . . 21
4.5.4 Hiperparámetros . . . 22

5 Metodología **25**
5.1 Datos de Entrada . . . 26
5.2 Arquitectura de la CNN . . . 29
5.2.1 Etapa Slowfast . . . 29
5.2.2 Etapa de Decodificación . . . 31
5.3 Entrenamiento de la Red . . . 31
5.3.1 Función de Pérdida Split-MSE . . . 32
5.4 Testeo de la Red . . . 33
5.4.1 Búsqueda de Umbral (*Threshold*) . . . 33
5.4.2 Test de Precisión . . . 34

6 Experimentos y Resultados **35**
6.1 Bases de Datos . . . 35
6.2 Resultados Experimentales . . . 36
6.2.1 MAE del Modelo de Reconstrucción de ECGs . . . 37
6.2.2 *Accuracy* del modelo Predictor . . . 38
6.3 Discusión de Resultados . . . 39

7 Conclusiones y Líneas Futuras **41**
7.1 Conclusiones . . . 41
7.2 Líneas Futuras . . . 42

Bibliografía **44**

Índice de Figuras

1.1 Esquema general del flujo de datos propuesto. En la sección a) puede verse como se añade ruido a las señales originales, sirviendo la señal resultante de entrada a un modelo de reconstrucción de ECGs. En la sección b) puede verse como la salida de dicho modelo pasa por un modelo predictor, que identificará (en comparación con la etiqueta) la presencia o no de anomalías. 2

4.1 Segmento de un ECG sin presencia de ruido, perteneciente a la base de datos MIT-BIH Arrhytmia Database. 9
4.2 Morfología normal de las ondas P, QRS y T. Fuente: Sociedad Española de Cardiología. 10
4.3 Muestras de cada tipo de ruido obtenidas de la base de datos MIT-BIH Noise Stress Database. La imagen a) muestra un ruido BW, la imagen b) muestra un ruido EM y la c) un ruido MA. 11
4.4 Arritmias supraventriculares. Imagen extraída de MIT-BIH Arrythmia Database. 12
4.5 Bloqueo de rama completo. Imagen extraída de MIT-BIH Arrythmia Database. 13

5.1 Flujo de Datos para filtrado de señales ECG. (a) La entrada es una derivación de ECG (Captura obtenida del visualizador lightWAVE de Physionet [1]). (b) La señal de entrada se recorta en tramos de 4 segundos. (c) Se añade ruido aleatorio a la señal. (d) La señal con ruido pasa por una CNN para su filtrado. (e) Se extrae la señal filtrada a la salida. 25
5.2 Flujo de Datos para predecir el futuro de señales ECG normales. (a) La entrada es una derivación de ECG (Captura obtenida del visualizador lightWAVE de Physionet [1]). (b) la muestra se divide en subsecuencias de 5 segundos.(c) cada señal de entrada se recorta en tramos de 4 segundos. (d) Las señales recortadas pasan por una CNN para predecir el siguiente segundo de la señal. (e) Se extrae la evolución de la señal a la salida. 26
5.3 División de un segmento de ECG en *input* y *label*. I representa la entrada, L_E la etiqueta expandida y L la etiqueta. 27

5.4 Datos de entrada antes y después, de la adición de ruido. Muestras extraídas de una partición en 512 fragmentos de un ECG. En la primera fila se presentan las muestras originales, mientras que la última fila contiene las muestras con ruido añadido. En la segunda fila se muestran los ruidos añadidos a la señal original. 28
5.5 Arquitectura general de la CNN propuesta. Estructura tipo U-Net formada por etapas Slow y Fast seguidas de una etapa de decodificación. 29
5.6 Arquitectura de la capa Slowfast. 4 bloques *slow* y 4 bloques *fast* entrelazados. A la salida de ambas ramas se realiza un *average pool*. . 30
5.7 Representación de las bandas interna y externa. La banda central incluye los segmentos entre picos R, mientras que las bandas exteriores recogen los picos de las ondas R y S. Imagen extraída de [2]. 32

6.1 Señal 108 del *dataset* MIT-BIH Arrhythmia Database. Pueden verse los dos canales capturados, siendo MLII el utilizado en este TFG. Captura obtenida del visualizador lightWAVE de Physionet [1]. . . . 36
6.2 Señal *ma* del *dataset* Noise Stress Database. Pueden verse los dos canales capturados, siendo *noise1* el utilizado en este TFG. Captura obtenida del visualizador lightWAVE de Physionet [1]. 36
6.3 Proceso de filtrado en tres de las muestras del *dataset*. La primera fila muestra las señales originales, la segunda, las imágenes con ruido BW añadido, y la tercera es una comparativa entre la salida del modelo y la señal original. 37
6.4 Comparativa entre señales originales y salida del *dataset*. En a) se muestran tres señales que previamente tenían ruido BW, en b) tres señales previamente con ruido tipo EM y en c) tres muestras con ruido MA a la entrada del modelo. 38

Índice de Tablas

5.1 Tamaño de cada una de las capas de cada bloque que compone las ramas *slow* y *fast*. 30

6.1 MAE medio obtenido para para cada tipo de ruido. Se ha calculado para el dataset MIT-BIH Arrhytmia Database con ruido BW, EM y MA. 38

6.2 Métricas de precisión del modelo predictor. La primera columna muestra la precisión en clasificación de muestras sin modificar, tanto normales como anómalas. En la segunda columna se muestra dicha precisión para muestras con ruido añadido. En la tercera columna se muestra lo propio para muestras con ruido, previamente reconstruidas por el modelo de reconstrucción de ECGs. 39

6.3 Métricas de precisión para cada tipo de ruido del modelo predictor. La primera columna muestra la precisión en clasificación de muestras reconstruidas a partir de muestras con ruido *baseline wander*, tanto normales como anomalías. En la segunda columna se presenta dicha precisión para muestras con ruido *electrode movement*. La tercera columna está constituida con los resultados para muestras reconstruidas con ruido tipo *muscle artifact*. 39

Acrónimos y Abreviaturas

Adam	Adaptive Moment Estimation
AE	Autoencoder
BW	Baseline Wander
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DAE	Denoising Autoencoder
DNN	Deep Neural Network
ECG	Electrocardiograma
EM	Electrode Movement
EMD	Empirical Mode Decomposition
FCN	Fully Convolutional Network
GPU	Graphic Processing Unit
GRU	Gated Recurrent Units
LSTM	Long Short-Term Memory
MA	Muscle Artifact
MAE	Mean Absolute Error
MSE	Mean Squared Error
NLP	Natural Languaje Processing
RC	Resistencia-Capacitancia

ReLU	Rectified Linear Unit
RL	Resistencia-Inductancia
RNN	Red Neuronal Recurrente
SGD	Stochastic Gradient Descent
TFG	Trabajo Fin de Grado
VMD	Variational Mode Decomposition

Resumen

El avance tecnológico y el creciente interés en campos como el Deep Learning y otras ramas de la computación ha provocado un aumento considerable en las contribuciones en el ámbito de señales de electrocardiogramas. A pesar de la existencia de múltiples enfoques para el análisis de este tipo de señales, ya sea mediante métodos tradicionales o *deep learning*, todos ellos necesitan partir de señales limpias y en entornos controlados. Esto limita su aplicabilidad en entornos realistas, como el análisis de señales recogidas fuera del hospital (prueba Holter).

Por ello, se han desarrollado propuestas basadas en Deep Learning, ámbito en el que este TFG pretende contribuir, por medio de un diseño capaz de reconstruir satisfactoriamente señales de ECG con ruido. Este diseño se realizará mediante el uso de diversos ruidos presentes en este tipo de medidas. Con este fin se ha utilizado una red neuronal convolucional con una etapa de decodificado, dados los buenos resultados obtenidos con este tipo de modelos para problemas similares.

Previo al entrenamiento de dicho modelo, se ha partido de una base de datos de ECGs sin ruido, se ha dividido cada muestra en segmentos de 4 segundos, se le ha añadido 3 tipos diferentes de ruido y se han presentado estas señales a la red. Posteriormente, con las señales ya filtradas, se ha comprobado el comportamiento de un modelo preentrenado de detección de anomalías en señales limpias. De esta forma, si el comportamiento es bueno, el proceso de limpieza de señales propuesto es correcto. Finalmente, se ha analizado la calidad de las reconstrucciones realizadas comparándolas con la señal original, por medio de la función de pérdida Error Absoluto Medio (MAE). Se ha obtenido un error del 4.66 % para ruidos tipo *baseline wander*, 6.31 % para ruidos tipo *electrode movement* y 5.23 % para ruidos tipo *muscle artifact*. Además, se ha comparado la precisión del segundo modelo frente a señales limpias versus señales reconstruidas por el modelo filtrador, con una reducción de precisión del 3.76 %.

Palabras Claves: MIT-BIH Arrhytmia Database, MLII, artefactos musculares, Inteligencia artificial, Pytorch, SlowFast, Eliminación de ruido, Red neuronal convolucional, ECG

Abstract

Technological advances and the growing interest in fields such as Deep Learning and other branches of computing have led to a significant increase in contributions to the field of electrocardiogram (ECG) filtering. Traditional approaches using either active or passive filters have proven effective in mitigating predictable or low-power noise. However, these solutions often struggle to reconstruct ECGs affected by sporadic or high-power noise, or they incur a high computational cost that limits their viability for real-time detection systems.

As a result, Deep Learning-based approaches have been developed, and this undergraduate thesis aims to contribute to this field by designing a model capable of reconstructing ECG signals with noise. The goal is for the proposed model to successfully reconstruct ECGs contaminated by various types of noise commonly found in such measurements. To achieve this, a convolutional neural network with a decoding stage has been used, given the promising results of such architectures in similar problems.

Prior to training the model, a noise-free ECG database was used. Each sample was divided into 4 seconds segments, three different types of noises were added, and these signals were introduced to the network. Subsequently, with the already filtered signals, it's the behavior was tested using a pretrained model, capable of detecting anomalies in clean signals. In this way, if the performance is good, the proposed signal-cleaning process is correct. Finally, the reconstruction quality was assessed by comparing the output with the original signal, yielding an error of 4.66 % for baseline wander noise, 6.31 % for electrode movement noise, and 5.23 % for muscle artifact noise. Additionally, the classification accuracy of the second model was compared between clean signals and those reconstructed by the filtering model, showing a precision reduction of only 3.76 %.

Key Words: MIT-BIH Arrhytmia Database, MLII, muscle artifacts,Artificial Inteligence, Pytorch, SlowFast, Noise Removal, Convolutional Neural Network, ECG

Capítulo 1

Introducción

Este capítulo se incluye como preámbulo al resto del documento. En él se contextualizará el trabajo, así como su motivación y los objetivos previos a su realización. Por último, se ha incluido un resumen de la estructura que seguirá este documento.

1.1. Motivación

La electrocardiografía es una herramienta fundamental en el diagnóstico de enfermedades cardiovasculares, proporcionando información crítica sobre la actividad eléctrica del corazón. Sin embargo, las señales de electrocardiogramas (ECGs) a menudo están contaminadas por diversos tipos de ruido, como el ruido muscular (MA, *muscle artifact*), interferencias provenientes de la red eléctrica y artefactos de movimiento, entre otros.

Dicha contaminación puede dificultar el análisis clínico preciso [3], especialmente en pacientes con enfermedades del aparato locomotor, como la enfermedad de Huntington o el Parkinson. Además, en sistemas de monitorización continua, la presencia de interferencias es más frecuente, lo que empeora la practicidad de dichos instrumentos. Por ello, una eliminación eficaz de estos artefactos es crucial para mejorar la calidad de la señal y facilitar diagnósticos más fiables.

Con el objetivo de solventar esta problemática, tradicionalmente se han propuesto diversas soluciones para la reconstrucción de señales, basadas en filtros activos y pasivos [3, 4]. Si bien este tipo de implementaciones ha demostrado su utilidad para ruidos simples o de poca potencia, no son capaces de eliminar ruidos más complejos como los artefactos musculares, si no que en todo caso mitigan sus efectos a costa de un gran gasto computacional y energético.

En este contexto, el auge de la Inteligencia Artificial permite la propuesta de nuevas estrategias de reconstrucción de señales. Los modelos basados en Inteligencia Artificial han demostrado extensamente su efectividad en tareas de reconocimiento de patrones, por lo que en los últimos años se han comenzado a realizar propuestas en el ámbito de la reconstrucción de señales de ECG [5, 6].

Por lo anteriormente expuesto, este TFG pretende implementar un modelo basado en técnicas de *deep learning*, capaz de reconstruir señales a partir de segmentos a los que se les han añadido varios tipos de ruido típicamente presentes en este tipo de señales. Para comprobar el correcto funcionamiento del modelo entrenado, se ha utilizado un segundo modelo [2] capaz de predecir el futuro de señales normales sin ruido y se ha pasado el test propuesto en [2] para medir la capacidad del segundo modelo de detectar la presencia de anomalías tras ser reconstruidas por el modelo propuesto. El proceso general propuesto se ilustra en la Figura 1.1.

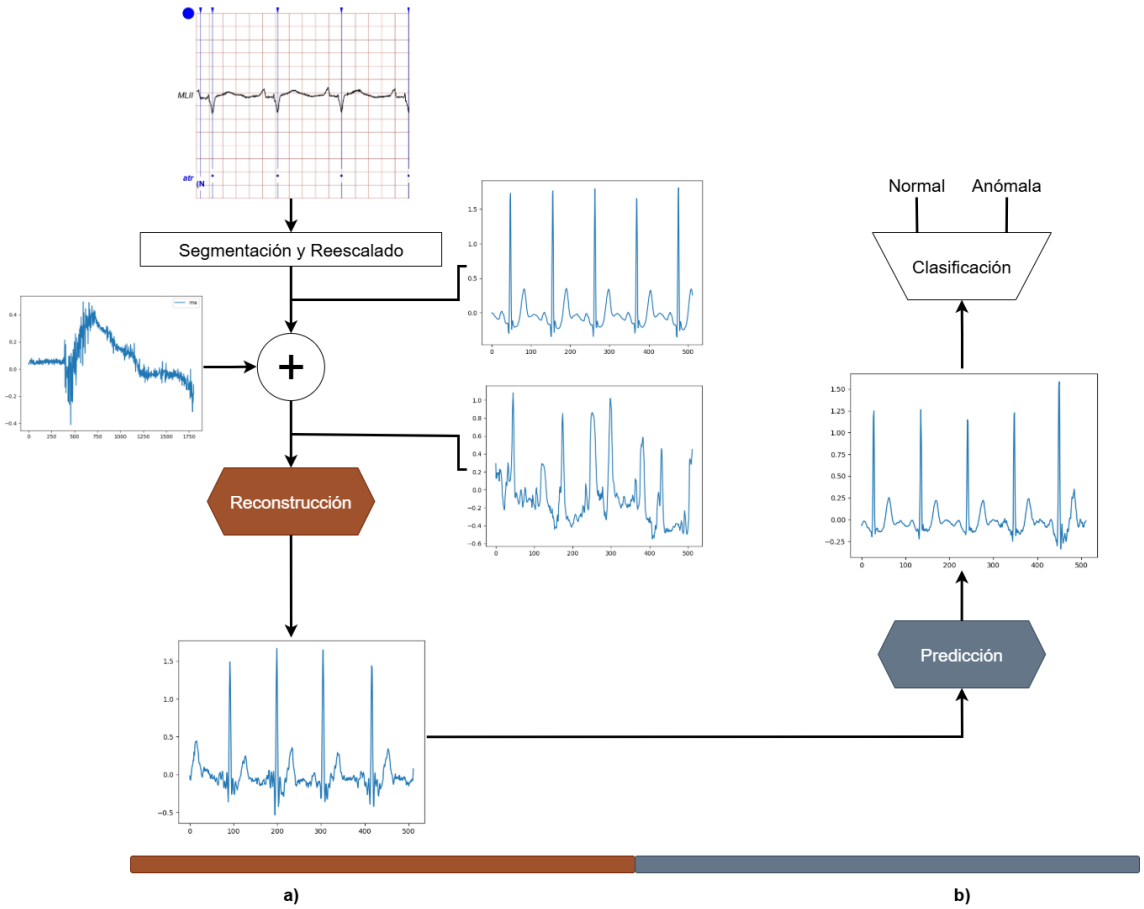

Figura 1.1: Esquema general del flujo de datos propuesto. En la sección a) puede verse como se añade ruido a las señales originales, sirviendo la señal resultante de entrada a un modelo de reconstrucción de ECGs. En la sección b) puede verse como la salida de dicho modelo pasa por un modelo predictor, que identificará (en comparación con la etiqueta) la presencia o no de anomalías.

1.2. Objetivos

El objeto de este TFG es el entrenamiento y validación de un modelo basado en técnicas de *deep learning* capaz de reconstruir señales provenientes de un ECG que presentan ruido. Para ello, se dividirá en segmentos una base de datos de ECGs y se les añadirá ruido. Posteriormente, se entrenará el modelo con la base de datos segmentada y modificada y se validará el modelo obtenido. Se ha dividido el flujo de trabajo para lograr el objetivo en los siguientes apartados:

- Segmentación de la base de datos y adición de ruido a cada muestra.
- Entrenamiento del modelo encargado de la reconstrucción de señales.
- Implementación del modelo predictor presentado en [2] para predecir el futuro de la señal reconstruida, con el objetivo de clasificar dicha señal como normal o anómala en función de la diferencia con la señal original.
- Comprobar el funcionamiento del modelo de reconstrucción de ECGs por medio de la métrica MAE (*Mean Absolute Error*) y el modelo implementado en el punto anterior.

1.3. Plan de Trabajo

En este apartado se enumerarán los pasos llevados a cabo para la realización de este trabajo:

1. Lectura de artículos relacionados para valorar distintas soluciones planteadas.
2. Aprendizaje del uso de herramientas y materias relacionadas con la rama de estudio: Python, PyTorch, Tensorboard, Deep Learning, etc.
3. Segmentación de la base de datos y adición de ruido a cada muestra.
4. Entrenamiento de un modelo encargado de la reconstrucción de señales.
5. Implementación del modelo predictor presentado en [2].
6. Comprobar el funcionamiento del modelo de reconstrucción de ECGs por medio de la métrica MAE (*Mean Absolute Error*) y el modelo implementado en el punto anterior.
7. Redacción del TFG y presentación.

1.4. Estructura del Documento

Este trabajo está dividido en una serie de capítulos, tras los cuales se encuentra la bibliografía. A continuación, se presenta un breve resumen sobre lo tratado en cada uno de los capítulos.

En el Capítulo 1, se realiza una introducción del TFG.

En el Capítulo 2, se presenta el estado del arte sobre la eliminación de ruido en ECGs.

En el Capítulo 3, se introducen las herramientas utilizadas para la realización del trabajo.

En el Capítulo 4, se desarrollan los fundamentos teóricos a partir de los cuales se desarrolla este TFG.

En el Capítulo 5, se explica la metodología aplicada para el desarrollo del trabajo.

En el Capítulo 6, se exponen los resultados obtenidos, así como el proceso para su obtención.

En el Capítulo 7, se enumeran las conclusiones obtenidas, así como posibles líneas futuras de ampliación de este TFG.

Capítulo 2

Estado del Arte

Tanto el filtrado como la reconstrucción de señales son áreas fundamentales en cualquier sistema de medición electrónico. En el ámbito médico, más concretamente en la obtención de electrocardiogramas (ECGs), los avances permiten tanto la realización de mediciones precisas, como la implementación de sistemas automatizados de monitorización y alerta continua.

A comienzos del siglo XX se realizaron las primeras contribuciones referentes al filtrado de señales mediante filtros electrónicos, destacándose los filtros paso alto basados en circuitos pasivos RC (Resistencia-Capacitancia) y RL (Resistencia-Inductancia) [7], o los filtros elimina banda [8], publicados en la década de 1920. Posteriormente, en 1930, se diseñó y publicó el filtro Butterworth [9], con respuesta más plana hasta la frecuencia de corte. En las últimas décadas, se han desarrollado enfoques más sofisticados; como el *Wavelet Denoising* [4], que utiliza la transformada wavelet para dividir la señal en detalles de baja y alta frecuencia, así como algoritmos de descomposición de señales como la *Empirical Mode Decomposition* (EMD), que descompone las señales en funciones oscilatorias básicas, y la *Variational Mode Decomposition* (VMD), que mejora el método EMD descomponiendo la señal en funciones oscilatorias llamadas modos [10]. Estos métodos permiten un filtrado más adaptativo y eficiente en contextos no lineales y no estacionarios.

Actualmente, existen modelos de electrocardiógrafos que incorporan diversos filtros con el objetivo de mitigar el efecto del ruido [11]. Entre ellos se encuentran los filtros paso-alto y elimina banda, comúnmente empleados para contrarrestar el efecto de la deriva de la línea de base y el ruido de la red eléctrica, respectivamente. Si bien este tipo de filtros pasivos han demostrado su utilidad contra los efectos de la interferencia de la red (y parcialmente contra la deriva de línea base), resultan ineficaces contra el resto de fuentes. El uso de filtros activos, como *Wavelet Denoising*, ha mostrado mejoras en la atenuación de ruidos de tipo BW (*baseline wander*) y EM (*electrode movement*). Por ejemplo, en [12] se implementó un algoritmo de

filtrado que combina un filtro *CEEDMAN* [13] con técnicas de *Wavelet Denoising*, logrando una atenuación eficaz del ruido de baja frecuencia, aunque con un elevado coste computacional.

En los últimos años, el auge de las técnicas de *Deep Learning* ha dado lugar a la publicación de diversas propuestas basadas en *Redes Neuronales Convolucionales* (CNNs), que tienen la capacidad de procesar una gran variedad de datos de entrada. La versatilidad de las CNNs permite obtener resultados satisfactorios a la hora de mitigar todos los ruidos anteriormente mencionados, sin el coste computacional en tiempo real asociado a los filtros activos.

En [6] se propone el uso de un *Denoising Autoencoder* (DAE) encargado de filtrar y extraer características de la señal de entrada, que consta de tres capas (entrada, codificación y salida). Se han utilizado como funciones de activación, *Rectified Linear Unit* (ReLU) y sigmoide. En cuanto a la estructura de capas, se proponen modelos con 252 nodos en cada capa y con 126 en la capa de codificación y 252 en el resto. Se parte de 4 muestras de distintas personas de la base de datos MIT-BIH Arrhytmia Database [14], a las que se añaden uno de 4 ruidos extraídos del *dataset* MIT-BIH Noise Stress Database [15]. El mejor modelo obtiene una precisión de 99.34 %, obtenidos sobre 2 de las muestras de la base de datos MIT-BIH Arrhytmia Database modificadas.

En [5], por otro lado, se propone el uso de un DAE basado en una estructura *Fully Convolutional Network* (FCN), conformada por 13 capas, comparando su comportamiento con un modelo basado en *Deep Fully Connected Neural Network* y otro basado en métodos de *denoising*. Para el entrenamiento se parte de las bases de datos utilizadas en este trabajo, MIT-BIH Arrhytmia Database (en su totalidad) y MIT-BIH Noise Stress Database. Los modelos consiguen un error cuadrático medio de 0.06, 0.09 y 0.07, respectivamente.

Frente a las propuestas anteriormente mencionadas, en este TFG se diseñará una red convolucional basada en SlowFast, con una etapa de decodificación tipo U-Net que permita reconstruir la señal. Para el entrenamiento se partirá de las bases de datos completas MIT-BIH Arrhytmia Database para muestras de ECGs y MIT-BIH Noise Stress Database para muestras de ruido. Se comprobará el funcionamiento del modelo por medio de la métrica Mean Absolute Error (MAE). Además, las muestras reconstruidas se ha utilizado como entrada de otro modelo capaz de predecir el futuro de la señal de ECG, con el fin de clasificar la señal predicha como normal o anómala.

Capítulo 3

Herramientas Utilizadas

En este capítulo se presentan las herramientas empleadas en el desarrollo del trabajo, en particular el lenguaje de programación Python y las librerías específicas para este contexto, además de información sobre el servidor utilizado.

3.1. Python

Python [16] es un lenguaje de programación gratuito y de código abierto, con una gran adopción en el ámbito de la inteligencia artificial. Se trata de un lenguaje de alto nivel multiparadigma, por lo que permite programación orientada a objetos, imperativa y funcional, brindando una gran flexibilidad en el desarrollo de soluciones complejas.

Otro de sus principales atributos es el extenso ecosistema de librerías disponibles, así como una de las comunidades de desarrolladores más activa. En el ámbito de la inteligencia artificial, se incluyen tanto librerías de álgebra lineal y manipulación de datos como Numpy, como frameworks para deep learning entre los que se encuentran PyTorch o TensorFlow.

Estas características, junto con su compatibilidad multiplataforma, contribuyen a su adopción como lenguaje en el ámbito de la inteligencia artificial.

3.2. Librería TensorFlow

TensorFlow [17] es una librería de código abierto, diseñada para *Machine Learning* y computación numérica. Esta librería implementa la visualización de las características de los modelos mediante *TensorBoard*.

TensorBoard permite visualizar en tiempo real aspectos importantes del entrenamiento de modelos, como pueden ser: la función de pérdida, la precisión, el histograma de pesos y salidas del modelo, entre otros.

3.3. Librería PyTorch

PyTorch [18] es una librería de aprendizaje automático de código abierto, siendo una de las herramientas más utilizadas en el desarrollo de modelos de inteligencia artificial.

Ofrece la ejecución de operaciones inmediata (*eager execution*), así como soporte completo para GPU (CUDA). Su estructura de datos principal es el tensor con soporte para CPU, cuyo formato es similar al array. Es ampliamente utilizado en el ámbito científico, y de forma creciente en el industrial, gracias a su API simple y a su flexibilidad.

3.4. Librería DeepDish

Deepdish [19] es una librería especializada en el almacenamiento estructurado de datos, siendo especialmente útil en el manejo de estructuras de datos complejas, como diccionarios anidados. Permite leer y guardar objetos de Python en formato HDF5, que es ampliamente utilizado en ciencia de datos.

3.5. Servidor

Para realizar las tareas de procesamiento de datos, entrenamiento y validación, se ha utilizado un servidor con procesador Intel Xeon Silver 4314 de 64 núcleos a 2.4 GHz, 500 Gb de RAM y acceso a dos GEFORCE RTX 3090. En cuanto a software, se han ejecutado en Python 3.10 con PyTorch 2.10 en CUDA 11.8.

Capítulo 4

Fundamentos Teóricos

En este capítulo se abordarán los fundamentos teóricos necesarios para la ejecución de este TFG. Concretamente, los referentes al análisis de ECGs, los tipos de ruido y de enfermedades cardiovasculares más comunes, así como el campo de las *Redes Neuronales*, enfocado en las *Redes Neuronales Convolucionales* (CNNs).

4.1. Características de una Señal ECG

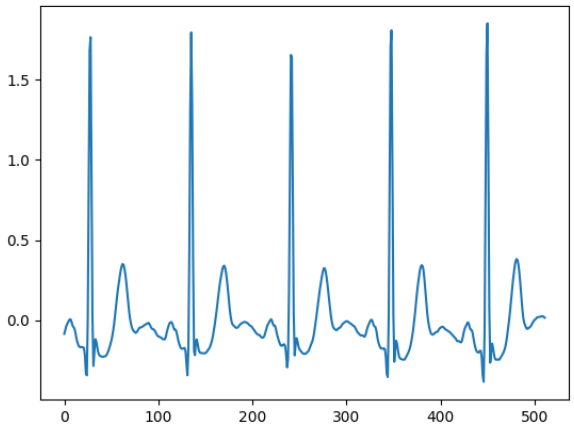

Figura 4.1: Segmento de un ECG sin presencia de ruido, perteneciente a la base de datos MIT-BIH Arrhytmia Database.

Un electrocardiograma (ECG) representa la actividad eléctrica del corazón en el tiempo, medida mediante un conjunto de electrodos situados en puntos concretos del cuerpo. Dicha información, dividida en derivaciones según los electrodos utilizados

para la medida, es crucial en la detección de afecciones cardíacas. En la Figura 4.1 puede apreciarse un segmento sin ruido de un ECG.

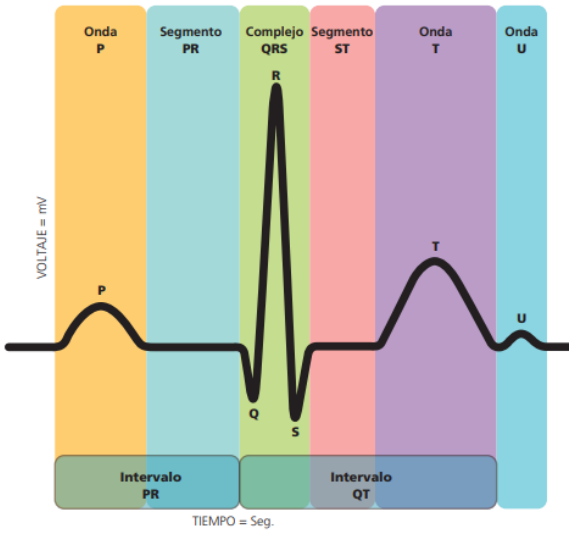

Figura 4.2: Morfología normal de las ondas P, QRS y T. Fuente: Sociedad Española de Cardiología.

En cuanto al análisis de un ECG, cabe destacar un conjunto de ondas relevantes, ilustradas en la Figura 4.2:

- Onda P: representa la despolarización auricular (activación de las aurículas).
- Complejo QRS: representa la despolarización ventricular (activación de los ventrículos).
- Onda T: representa la repolarización ventricular (relajación de los ventrículos).

En cuanto a la tipología de ruidos presentes en un ECG, se van a analizar tres tipos: *baseline wander* (BW), *electrode movement* (EM) y *mucle artifact* (MA). La deriva de la línea de base (BW) es un ruido con componentes de muy baja frecuencia ($<$0.5 Hz) provocado por la respiración del paciente o por una deficiencia en el contacto del electrodo. El ruido por movimiento del electrodo (EM) tiene su origen en un acoplamiento insuficiente entre el electrodo y el cuerpo, lo que puede distorsionar la morfología de las ondas características del ECG (ondas P, QRS y T). Finalmente, los artefactos musculares (MA), por otra parte, son causados por actividad eléctrica en el aparato muscular del paciente (movimientos musculares), ya sea voluntaria o involuntariamente, encontrándose frecuencialmente entre 20 y 200 Hz. En la Figura 4.3 pueden verse muestras aisladas de cada tipo de ruido.

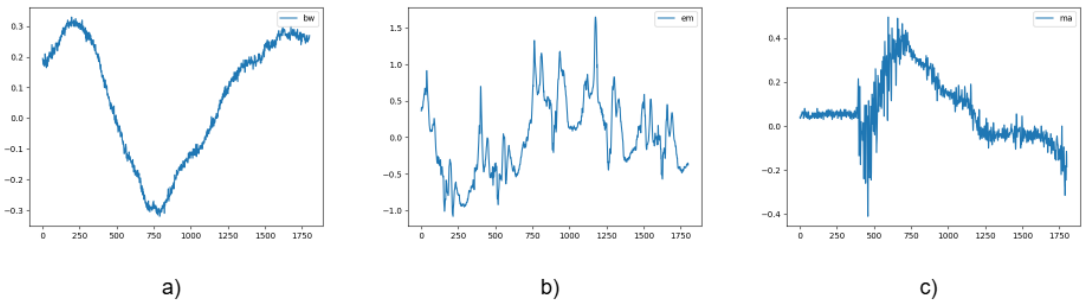

Figura 4.3: Muestras de cada tipo de ruido obtenidas de la base de datos MIT-BIH Noise Stress Database. La imagen a) muestra un ruido BW, la imagen b) muestra un ruido EM y la c) un ruido MA.

4.2. Enfermedades Cardiovasculares

El término enfermedades cardiovasculares refiere al conjunto de alteraciones del corazón y vasos sanguíneos. Una de las herramientas más utilizada para la detección de dichas anomalías es el análisis de ECGs, cuyas características se han detallado en la sección anterior. Las afecciones principales pueden dividirse en: trastornos del ritmo o arritmias, bloqueos de rama y cardiopatía isquémica, entre otras.

4.2.1. Trastornos del Ritmo o Arritmias

Las arritmias son alteraciones del ritmo cardíaco, cuya frecuencia normal se encuentra entre 60 y 100 latidos por minuto, de forma sincronizada y con ritmo constante. Estas anomalías pueden aparecer por una generación defectuosa del impulso eléctrico, una localización errónea de dicho impulso o una alteración en los caminos de conducción del impulso.

Existen distintas clasificaciones para cada tipo de arritmia; ya sea por su origen, supraventriculares o ventriculares; por su frecuencia, taquicárdicas o bradicárdicas; o por su modo de presentación, crónicas o paroxísticas.

En cuanto a las arritmias más frecuentes, se encuentran:

- **Bloqueos Auriculoventriculares**. Son un conjunto de trastornos del sistema de conducción eléctrico, caracterizados por un retraso o la no conducción del estímulo generado de las aurículas a los ventrículos. Si es de primer grado, puede identificarse en un ECG por la prolongación del intervalo PR con complejos QRS estrechos, mientras que en bloqueos de mayor grado presentan bloqueo de algunas ondas P.

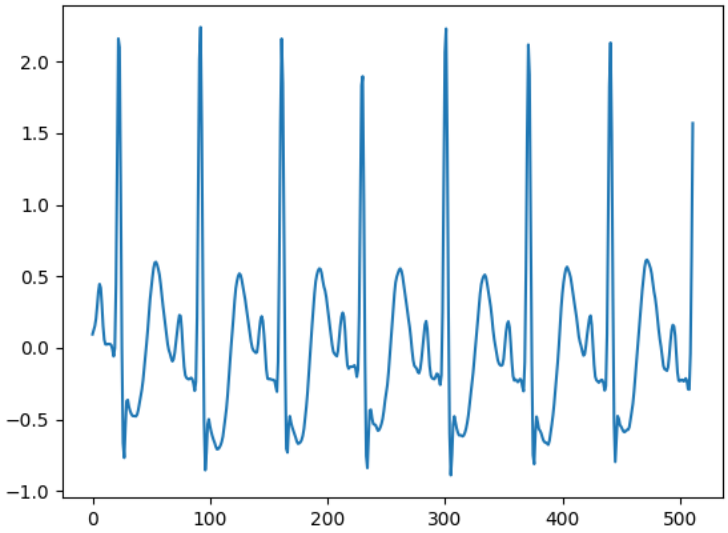

Figura 4.4: Arritmias supraventriculares. Imagen extraída de MIT-BIH Arrythmia Database.

- **Enfermedad del Nodo Sinusal**. Se caracteriza por una alteración en la frecuencia o conducción de la señal eléctrica que origina el latido. Puede detectarse en un ECG observando variaciones en el intervalo P-P.
- **Flutter Auricular**. En este trastorno se producen frecuencias muy elevadas en la aurícula que no es seguida por el ventrículo. En un ECG pueden apreciarse onda en "dientes de sierra".
- **Fibrilación Auricular**. En esta anomalía, se presenta una taquicardia supraventricular, así como una falta de contracción auricular. Para detectarla en un ECG, se observan intervalos R-R irregulares y la falta de ondas P.
- **Taquicardias Supraventriculares**. En los ECGs presentan frecuencias elevadas y regulares, así como complejos QRS estrechos. Este tipo de arritmia está ilustrada en la Figura 4.4.
- **Arritmias Ventriculares**. Se caracterizan por presentar un complejo QRS ancho.

4.2.2. Bloqueos de Rama

Los bloqueos de rama son anomalías en la transmisión del pulso eléctrico que afectan al correcto funcionamiento de los ventrículos. Estos trastornos afectan principalmente al complejo QRS y, en los casos más graves (bloqueo completo de rama), también afecta a la onda T, como puede apreciarse en la Figura 4.5.

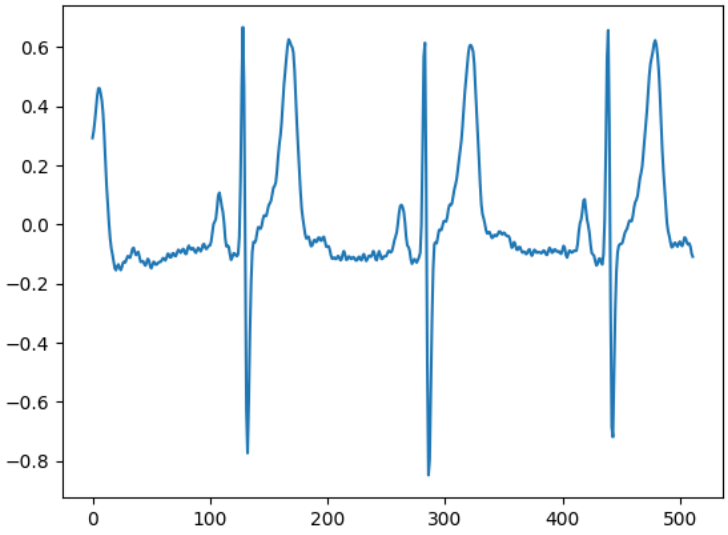

Figura 4.5: Bloqueo de rama completo. Imagen extraída de MIT-BIH Arrythmia Database.

Si el bloqueo de rama es completo, el impulso eléctrico al ventrículo no se conduce, lo que produce un complejo QRS mayor a 0.12 segundos. Por otro lado en los bloqueos de rama incompletos, el impulso se conduce con retraso, lo que hace que el impulso QRS se ensanche, con una duración entre 0.11 y 0.12 segundos.

4.2.3. Cardiopatía Isquémica

La cardiopatía isquémica (popularmente conocida como infarto) es un trastorno producido por la reducción del flujo sanguíneo al corazón, siendo una de las principales causas de muerte en el mundo [20].

Uno de los trastornos más frecuentes que provocan dicha enfermedad es el infarto agudo de miocardio, provocado normalmente por el desprendimiento de la placa de ateroma en una arteria coronaria. La principal alteración observable en un ECG es la elevación del segmento ST, evolucionando en ondas Q patológicas, inversión de las ondas T y normalización del segmento ST, observándose por último la pérdida de la onda R.

4.3. Introducción a las Redes Neuronales

Las redes neuronales artificiales son un modelo computacional, que sirven como base de la mayoría de propuestas en *Deep Learning*. Su funcionamiento tiene una base biológica, imitando de forma simplificada el funcionamiento de un cerebro humano. Las neuronas artificiales reciben de entrada datos numéricos, los procesan, y en función del valor transformado deciden qué valor transmitir. Estas neuronas pueden ser interconectadas en múltiples capas, creando arquitecturas capaces de identificar patrones complejos. En cuanto a los tipos de redes neuronales, destacan:

- **Redes Neuronales Recurrentes (RNNs)**. Su diseño está enfocado en el procesamiento secuencial de datos, para ello las salidas no solo dependen de la entrada, si no que también dependen del estado anterior. Debido a este comportamiento, las RNNs tienen problemas para mantener la información cuando la secuencia de entrada es muy larga. Para mejorar la memoria de la red, se han propuesto modificaciones como las *Long Short-Term Memory* (LSTM) [21], o las *Gated Recurrent Units* (GRU) [22], que incorporan puertas para decidir qué información olvidar o almacenar. Las LSTM incorporan tres tipos de puertas (*forget gate*, *input gate* y *output gate*), mientras que las GRU sólo incorporan dos (*reset gate* y *update gate*).

- **Redes Neuronales Convolucionales (CNNs)**. Se caracterizan por tomar como entrada datos en forma matricial, permitiendo extraer características complejas de la entrada. Son ampliamente utilizadas en tareas de visión artificial, así como en el análisis de series temporales. Es el tipo de red utilizado en la realización de este TFG, por lo que a continuación se detallará en mayor profundidad su funcionamiento.

- **Transformers**. Son un tipo de Red Neuronal Profunda (DNN, *Deep Neural Network*) diseñada para procesar secuencias de datos [23], especialmente en el campo del procesamiento del lenguaje natural (NLP). A diferencia de las RNNs, que procesan datos de manera secuencial, utilizan mecanismos para procesar todos los elementos en paralelo. Para ello, convierten cada palabra (*token*) en un vector numérico, que se combina con una codificación posicional para no perder el orden de los elementos. Seguidamente, mediante un mecanismo *Self-Attention*, se mide la relevancia de cada palabra respecto al resto, y mediante un mecanismo de *Multi-Head Attention* captura la relación entre palabras. Por último, se procesa cada *token* mediante una red neuronal.

4.4. Redes Neuronales Convolucionales

Las CNNs, como se ha mencionado, son diseñadas para la análisis de imágenes o series temporales, para ello se sirven de convoluciones que extraen sus características. Con este propósito las primeras capas extraen características de bajo nivel (como patrones), mientras que las últimas extraen detalles de alto nivel.

En cuanto a su entrenamiento, debe normalizarse en primer lugar el *dataset*, además de definirse la arquitectura del modelo, es decir, los componentes internos que conformarán la red, así como la función de pérdida, el optimizador y los pesos iniciales, generalmente aleatorios. Una vez comienza el entrenamiento, cada ciclo puede dividirse en las siguientes etapas:

- **Propagación hacia delante** (*feedfordward*). En esta etapa la entrada pasa por la arquitectura de la red, produciéndose una salida.
- **Cálculo de pérdida** (*Loss*). La salida se compara con la etiqueta, mediante una función de pérdida.
- **Retropropagación del error** (*backpropagation*). Por medio de la regla de la cadena, se calcula el gradiente de la pérdida respecto a cada peso.
- **Actualización de Pesos**. Se utiliza una función de optimización para actualizar cada peso en función del gradiente obtenido.

Este proceso se realiza de forma iterativa hasta la convergencia de la red, obteniendo idealmente el resultado deseado.

4.5. Arquitectura de las CNNs

En este apartado se definirán los principales componentes en la arquitectura de una *Red Neuronal Convolucional*, como son las capas, las funciones de activación, las funciones de pérdida y los optimizadores.

4.5.1. Capas

Las capas son el bloque fundamental que compone una CNN, siendo el principal condicionante de la capacidad de aprendizaje de la red. Tanto su tipo como su número dependen del criterio de diseño seguido.

4.5.1.1. Capa Convolucional

Como su nombre indica, es el núcleo de las CNNs, extrayendo características de la entrada mediante una operación de convolución. Esta operación consiste en la convolución discreta S entre regiones de la entrada I y un filtro (*kernel*) K, de la forma:

$$S(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) \cdot K(m,n) \tag{4.1}$$

Donde tanto i como j son los índices de cada elemento de la entrada, mientras que m y n lo son para cada elemento del filtro (siendo su dimensión $M \times N$). Como resultado, se obtiene el mapa de activación $S(i,j)$. Por simplicidad, se ha mostrado la operación de convolución 2D, pero dicha operación debe ser adaptada al número de componentes de entrada a la capa. En cuanto a la configuración de la capa, caben destacar los siguientes elementos:

- Tamaño del filtro (*kernel size*): define las dimensiones del filtro a implementar.
- Paso (*stride*): representa el movimiento de la ventana de convolución por los datos de la entrada, es decir, el salto entre índices que habrá para cada operación. El aumento de este valor disminuye el número de características extraídas.
- Relleno (*padding*): Añade ceros en cada dimensión de la muestra, con el objetivo de controlar el tamaño de la salida.
- Profundidad (*depth*): número de filtros en cada capa. Cada uno producirá su propio mapa de activación, conformando en conjunto el volumen de activación.

Cada filtro es codificado por los pesos de una neurona. En el contexto de la *backpropagation*, cada filtro se aplica con los parámetros invertidos.

4.5.1.2. Capa de Activación

Esta capa se sitúa tras cada capa de convolución, con el objetivo de producir una adaptación en los datos. Para ello, se hace uso de diversas funciones de activación. Esta operación produce valores en un rango determinado, introduciendo una no linealidad que fomenta el aprendizaje de relaciones complejas entre la entrada y la salida. Cabe destacar las siguientes funciones:

- **ReLU (*Rectified Linear Unit*)**: toma el dato de entrada y da como salida un valor en el rango [0, ∞]. Es una función simple y eficiente, lo que la ha convertido en una de las funciones más empleadas. Esta función viene dada por:

$$f(x) = \max(0, x) \tag{4.2}$$

- **Sigmoide**: transforma la entrada en activaciones de rango [0, 1]. Su principal problema es la saturación de los valores en los extremos, haciendo que el gradiente se desvanezca en las capas iniciales, efecto conocido como *vanishing gradient*. Su salida es representada por la siguiente ecuación:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{4.3}$$

- **Tanh** (Tangente Hiperbólica): su comportamiento es similar al de la función sigmoide, ampliando el rango a [-1, 1]. Presenta el mismo problema para valores extremos, si bien es preferible al estar centrada en 0. Esta función viene dada por la siguiente ecuación:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{4.4}$$

- **Softmax**: para asignar los valores de activación, realiza una distribución normalizada, asignando valores en el rango [0, 1], sumando todas las activaciones 1. Este comportamiento permite su interpretación como una distribución probabilística entre las clases, por lo que se usa normalmente como activación en la última capa del clasificador. Viene representada por la siguiente ecuación:

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{4.5}$$

4.5.1.3. Capa de Agrupación o *Pooling Layer*

La capa de agrupación tiene por objetivo reducir el tamaño de los mapas de activación, mejorando la eficiencia del modelo y disminuyendo el riesgo de *overfitting*. Como parámetros de entrada toma el tamaño de la ventana, *stride*, *padding* y el tipo de operación.

Las operaciones más utilizadas son: máximo (*Max Pooling*), promedio (*Average Pooling*) y global (*Global Pooling*). El *Max Pooling* toma el valor máximo de cada región de la entrada, mientras que el *Average Pooling* toma el promedio. Por otro lado, el *Global Pooling* reduce a un único valor la dimensionalidad de la entrada, siendo su principal implementación antes de la capa de salida.

4.5.1.4. Capa de Normalización

La capa de normalización tiene por objetivo la estabilización y aceleración del entrenamiento, controlando la distribución de los datos internos de la red. Los métodos de normalización típicamente empleados son: normalización por lotes (*Batch Normalization*), normalización por neurona (*Layer Normalization*), normalización por canales (*Instance Normalization*) y por grupo de canales (*Group Normalization*). Para las CNNs, suele aplicarse la normalización por lotes.

4.5.1.5. Capa de *Dropout*

En esta capa se implementa la técnica *dropout* [24], lo que ayuda a reducir el *overfitting*. Consiste en la desactivación aleatoria de un porcentaje de neuronas durante el entrenamiento, lo que obliga a la red a no depender en exceso de neuronas específicas, mejorando la generalización. Esta técnica solo se aplica durante el entrenamiento, ya que en el *test* todas las neuronas se encuentran activas.

4.5.1.6. Capa Totalmente Conectada o *Fully-Connected Layer*

Es el componente principal de las redes neuronales tradicionales y también se usa en las redes convolucionales. Como su nombre indica, las neuronas de la capa tienen conexiones a todas las salidas de la capa anterior, conteniendo la mayoría de parámetros de la red. Esta composición hace que normalmente se sitúe en las últimas capas de red.

4.5.2. Funciones de pérdida

Las funciones de pérdida o de coste (*loss functions*), asocian un coste a cada salida en función de la calidad de la predicción respecto a la etiqueta. Este valor será utilizado por el optimizador, con el objetivo de minimizar el error.

En una CNN sus ámbitos de aplicación pueden dividirse en problemas de clasificación y de regresión. En los primeros penaliza las predicciones incorrectas, mientras que en los problemas de regresión representa la diferencia entre la predicción y la salida objetivo.

Las funciones de pérdida más utilizadas son:

- **Función de Pérdida de Entropía Cruzada** (*Cross-Entropy Loss Function*). Esta función calcula la diferencia entre la distribución probabilística predicha y la real, donde la distribución de etiquetas se codifica como un vector binario, siendo 1 la clase correcta y el resto 0. Su cálculo en un caso multiclase viene dado por:

$$\mathcal{L}(p, q) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} p_{ij} \log q_{ij} \tag{4.6}$$

  donde p_i, q_i son el valor real y la probabilidad a la salida de una capa de clasificación, N el número total de muestras y C el número total de clases.

- **Función de Pérdida del Error Absoluto Medio** (*MAE Loss Function*). Se utiliza principalmente en problemas de regresión. Mide la diferencia promedio entre los valores reales y las predicciones del modelo, tomando este error como valor absoluto, lo que le confiere robustez frente a valores atípicos. De esta forma, sirve como indicador genérico del funcionamiento del modelo. Su cálculo viene dado por la siguiente ecuación:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{j=1}^{N} |y_j - \hat{y_j}| \tag{4.7}$$

  donde y_j e $\hat{y_j}$ representan el valor real y predicho, respectivamente, mientras que N es el número total de muestras.

- **Función de Pérdida del Error Cuadrático Medio** (*MSE Loss Function*). Es la función de pérdida más utilizada para problemas de regresión. Es más sensible a predicciones atípicas, ya que se calcula como la suma de diferencias entre valor real y predicción al cuadrado, lo que supone que una única predicción puede penalizar gravemente el error, si la diferencia es elevada. Se calcula de la siguiente manera:

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N}\sum_{j=1}^{N} (y_j - \hat{y_j})^2 \tag{4.8}$$

  donde N representa el número total de muestras, y_j e $\hat{y_j}$ son, respectivamente, el valor real y predicho.

- **Función de Pérdida de Contraste** (*Contrastive Loss Function*). Esta función mide la similitud entre pares de muestras. Utiliza una etiqueta booleana que indica si dos muestras pertenecen a la misma clase. Se calcula mediante la siguiente ecuación:

$$\mathcal{L}(Y, D_w) = \frac{1}{2}\left[Y \cdot D_w^2 + (1 - Y) \cdot \text{máx}(0, m - D_w)^2 \right] \tag{4.9}$$

$$D_w = ||\hat{y}_1 - \hat{y}_2||_2 \tag{4.10}$$

donde Y es una etiqueta binaria que indica si el par pertenece a la misma clase, D_w la distancia euclidea entre el par de muestras $\hat{y}_1$ e $\hat{y}_2$, y m es un parámetro (mayor que 0) que penaliza en menor o mayor medida la no similitud entre muestras.

4.5.3. Optimizadores

En el proceso de entrenamiento, se utilizan optimizadores como algoritmo de actualización de pesos. Para ello, se calculan los gradientes de la función de pérdida para posteriormente aplicarlos en la actualización de parámetros del modelo.

4.5.3.1. Descenso del gradiente estocástico (*Stochastic Gradient Descent (SGD)*)

En la mayoría de los casos el método de optimización se basa en el descenso del gradiente como fundamento del algoritmo. Su función es minimizar una función objetivo, que es caracterizada según los pesos del modelo, de manera que los pesos se actualizan en el sentido contrario al gradiente de la función objetivo en relación a los parámetros.

El algoritmo basado en el descenso del gradiente en mini-lotes o mini-batch es el más popular para la optimización de deep learning. Se basa en la actualización en cada mini-batch de N muestras de entrenamiento. Como consecuencia se consigue una convergencia más estabilizada reduciéndose también la varianza de actualizaciones:

$$\theta \leftarrow \theta - \alpha \sum_{i=1}^{n} (h_\theta(x_i) - y_i) x_i \tag{4.11}$$

donde θ son los parámetros del modelo, α es la tasa de aprendizaje, n es el tamaño del mini-lote, $h_\theta(x_i)$ es el valor de salida del modelo para la muestra x_i de entrada e y_i es la etiqueta de la muestra i.

4.5.3.2. Adam

El algoritmo basado en la estimación del momento adaptativo (Adam), se basa en la optimización de la tasa de aprendizaje para cada peso del modelo. Se registra el promedio exponencialmente descendente de los gradientes cuadrados pasados v_t y m_t. Se utilizan las ecuaciones 4.13 y 4.12 para calcular v_t y m_t respectivamente.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.12}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.13}$$

donde β_1 y β_2 son constantes descendentes, m_{t-1} y v_{t-1} son el promedio descendente de los gradientes pasados y el promedio descendente de los gradientes pasados cuadrados, respectivamente. Para el paso $t-1$, g_t son los gradientes actuales. Los parámetros m_t y v_t son estimaciones del primer momento y segundo momento.

Para evitar el sesgo a 0 del primer y segundo momento (debido a su inicialización con valor nulo), se aplica a su estimación la siguiente corrección:

$$\hat{m}_t = \frac{m_t}{1 - {\beta_1}^t} \tag{4.14}$$

$$\hat{v}_t = \frac{v_t}{1 - {\beta_2}^t} \tag{4.15}$$

donde m_t y v_t son las estimaciones obtenidas en 4.12 y 4.13, β_1 y β_2 constantes descendentes, t el paso actual, y tanto $\hat{m}_t$ como $\hat{v}_t$, las estimaciones no sesgadas del primer y segundo momento.

Por último, la regla de actualización de Adam viene dada por la siguiente ecuación:

$$\theta_{t+1} \leftarrow \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{4.16}$$

donde θ_{t+1} son los parámetros en el paso siguiente, θ_t son los parámetros actuales, α es la tasa de aprendizaje, $\hat{m}_t$ y $\hat{v}_t$ los parámetros calculados en 4.14 y 4.15, y ϵ es un término de suavizado.

4.5.3.3. AdamW

Este algoritmo de optimización es una modificación del algoritmo descrito en el apartado anterior, mejorando la regularización de pesos (*weight decay*).

Este tipo de regulación viene dado por la fórmula:

$$\mathcal{L}_{total} = \mathcal{L} + \lambda \cdot |\theta|^2 \tag{4.17}$$

donde $\mathcal{L}$ es el valor calculado por la función de pérdida, θ los parámetros calculados por el optimizador, y λ un hiperparámetro que controla la regularización (*weight decay*).

El algoritmo AdamW incluye el hiperparámetro λ separándolo de los gradientes, asegurando que la regularización no intervenga con los mecanismos adaptativos del optimizador. Por tanto, la actualización de pesos viene dada por la siguiente ecuación:

$$\theta_{t+1} \leftarrow \theta_t - \alpha \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \cdot \theta_t \right) \tag{4.18}$$

donde θ_{t+1} son los parámetros en el paso siguiente, θ_t son los parámetros actuales, α es la tasa de aprendizaje, $\hat{m}_t$ y $\hat{v}_t$ los parámetros calculados en 4.14 y 4.15, ϵ es un término de suavizado y λ el hiperparámetro *weight decay*.

4.5.4. Hiperparámetros

A continuación, de forma breve, se explican algunos de los hiperparámetros de las CNNs. Se trata de unas variables utilizadas para definir conceptos de alto nivel de las redes para caracterizar la capacidad de aprendizaje o la complejidad del modelo. Antes del entrenamiento se deben definir cuáles serán estos parámetros, y luego se pueden ir afinando, generalmente mediante una validación cruzada.

- **Arquitectura de la Red.** En la arquitectura de la red se incluyen tanto el número de capas como el tipo de las mismas, así como el número de filtros en cada una de ellas.
- **Tasa de Aprendizaje (*Learning Rate*).** La tasa de aprendizaje define la velocidad de actualización de los pesos. Este parámetro debe ser elegido de manera óptima, ya que una tasa de aprendizaje tanto demasiado alta como demasiado baja podría provocar que el modelo tenga una capacidad de aprendizaje baja, al fallar la optimización. Por un lado, que sea baja hace que el entrenamiento sea más lento y se dificulte llegar al mínimo local. Por otra parte, un valor elevado acelera la convergencia, pero si es excesivamente alto podría no converger.

- **Penalización *Weight Decay*.** El *weight decay* es un parámetro incluido en la regla de actualización de pesos que produce que los pesos caigan exponencialmente a cero si no se incluye otra actualización. Ayuda a reducir el *overfitting*.
- **Tamaño del Lote (*Batch Size*).** El tamaño del lote indica el número de muestras introducidas a la red durante cada iteración del entrenamiento.
- **Criterio de Parada (*Stop Criterion*).** El criterio de parada es el criterio que se aplica para dar por concluido un entrenamiento. Algunos de estos criterios pueden ser: el número de iteraciones del entrenamiento, alcanzar un valor determinado de precisión de la red o el haber detectado sobreajuste.
- **Tasa de *dropout* (*Dropout Rate*).** La tasa de *dropout* es el porcentaje de neuronas que son descartadas temporal y aleatoriamente durante el entrenamiento para evitar el sobreajuste.

Capítulo 5

Metodología

En este capítulo se expondrá la metodología aplicada en los experimentos realizados en este TFG, en los que se aborda el filtrado de señales ECG mediante un modelo basado en una CNN. El flujo de datos propuesto se puede observar en la Figura 5.1: (a) representa la señal ECG original; *(b)* se divide la derivación del ECG en múltiples muestras; *(c)* se añade un ruido aleatorio; *(d)* se introducen las muestras de entrada en la CNN para su filtrado; y *(e)* se obtiene como salida la señal filtrada.

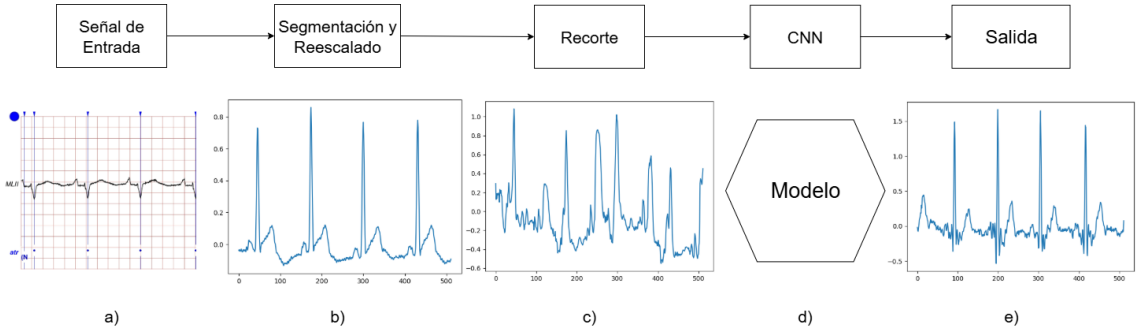

Figura 5.1: Flujo de Datos para filtrado de señales ECG. (a) La entrada es una derivación de ECG (Captura obtenida del visualizador lightWAVE de Physionet [1]). (b) La señal de entrada se recorta en tramos de 4 segundos. (c) Se añade ruido aleatorio a la señal. (d) La señal con ruido pasa por una CNN para su filtrado. (e) Se extrae la señal filtrada a la salida.

Una vez filtrado el ruido de los datos con el modelo propuesto, para comprobar la efectividad del sistema, se van a usar las señales limpiadas como entrada a un sistema de detección de anomalías en ECGs [2], cuyo flujo de datos es similar al propuesto para el modelo de filtrado de ruido, como puede apreciarse en la Figura 5.2: (a) representa la señal ECG original; *(b)* se particiona la derivación del ECG en múltiples muestras; *(c)* cada muestra se recorta, con diferencia de 1 segundo entre entrada y etiqueta; *(d)* se introducen las muestras de entrada en la CNN; y *(e)* se obtiene como salida la predicción de un segundo futuro de la señal. De esta forma, si

los resultados obtenidos por este detector de anomalías son similares usando nuestras señales filtradas y las señales originales, quiere decir que nuestro sistema es capaz de reconstruir las señales correctamente.

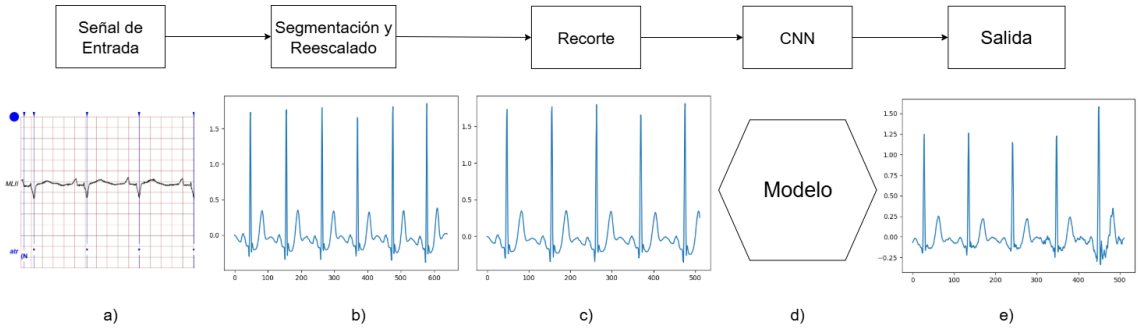

Figura 5.2: Flujo de Datos para predecir el futuro de señales ECG normales. (a) La entrada es una derivación de ECG (Captura obtenida del visualizador lightWAVE de Physionet [1]). (b) la muestra se divide en subsecuencias de 5 segundos.(c) cada señal de entrada se recorta en tramos de 4 segundos. (d) Las señales recortadas pasan por una CNN para predecir el siguiente segundo de la señal. (e) Se extrae la evolución de la señal a la salida.

5.1. Datos de Entrada

Para la realización de este TFG, se ha tomado como señal de entrada una modificación de la derivación II del ECG (MLII). Se ha escogido este tipo de señal ya que es la más utilizada de cara al monitoreo continuo y la detección de arritmias.

Para unificar el formato del conjunto de datos, cada muestra obtenida del *dataset* se ha dividido en secuencias de 5 segundos. En cada división, se realiza un preprocesamiento, mediante un filtro de mediana 3×3 y un filtro pasa-banda. Seguidamente, se modifica la velocidad de muestreo de las muestras (*resample*) para que se ajusten a una nueva frecuencia, pasando de 360 Hz a 128 Hz, así como para la ventana elegida (siendo el número total de muestras 640 para una ventana de 5 segundos). Como puede verse en la Figura 5.3, el modelo predictor toma como entrada el segmento I, mientras que la etiqueta será el segmento L. Además, se ha utilizado una etiqueta expandida L_E, de tal forma que la entrada y etiqueta tengan el mismo tamaño.

En el caso de los datos de entrada para el modelo de reconstrucción de ECGs, para cada secuencia de entrada se toma un tipo de ruido aleatorio entre *baseline wander*, *electrode movement* y *muscle artifact*, además de elegir una relación señal-ruido (SNR) en decibelios aleatoria, siendo este valor típicamente calculado de la forma:

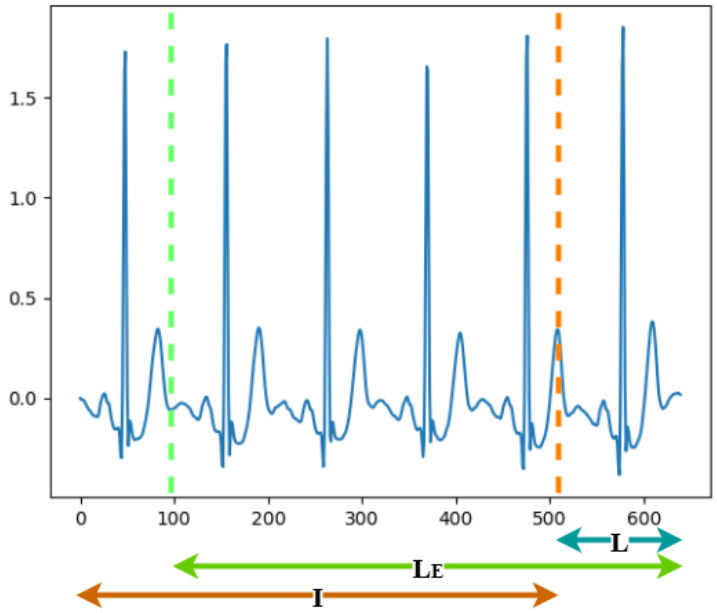

Figura 5.3: División de un segmento de ECG en *input* y *label*. I representa la entrada, L_E la etiqueta expandida y L la etiqueta.

$$SNR_{dB} = 10log_{10}\left(\frac{X}{S}\right) \tag{5.1}$$

donde X representa la potencia de la señal sin ruido, mientras que S es la potencia de la señal de ruido. Una vez elegido tanto el tipo de ruido (que será obtenido de una base de datos) como su SNR, se toma un segmento aleatorio del ruido con la misma duración temporal que la señal sin ruido. Seguidamente se compara la resolución de ambas señales y si es necesario se realiza un reescalado para que el número de muestras del ruido coincida con el segmento de ECG original (*resample*). Para que el SNR de la señal resultante sea el requerido, el ruido será ponderado por un coeficiente α. Este coeficiente será obtenido partiendo de la ecuación 5.1, calculándose de la siguiente forma:

$$\alpha = \sqrt{\frac{\sum_{i=1}^{N} s_i^2}{10^{\frac{SNR_{dB}}{10}} \cdot \sum_{i=1}^{N} n_i^2}} \tag{5.2}$$

donde s_i es el valor medido de la señal limpia en el instante i, n_i su equivalente de la señal de ruido, N el número total de muestras y SNR_{dB} la relación señal-ruido escogida aleatoriamente (en decibelios). Cabe destacar que se ha utilizado por simplicidad el sumatorio del valor al cuadrado de cada muestra de la señal como medida de potencia.

Por último, el ruido resultante se suma con el segmento de señal original, formando una muestra distorsionada por un tipo concreto de ruido. El segmento I de esta señal (tal y como se muestra en la Figura 5.3) será la entrada, mientras que el segmento I de su análoga sin ruido, la etiqueta.

En la Figura 5.4 pueden apreciarse tres segmentos de ECGs de una misma persona de la base de datos, tanto antes como después de añadir el ruido, además del ruido que se ha añadido en cada caso.

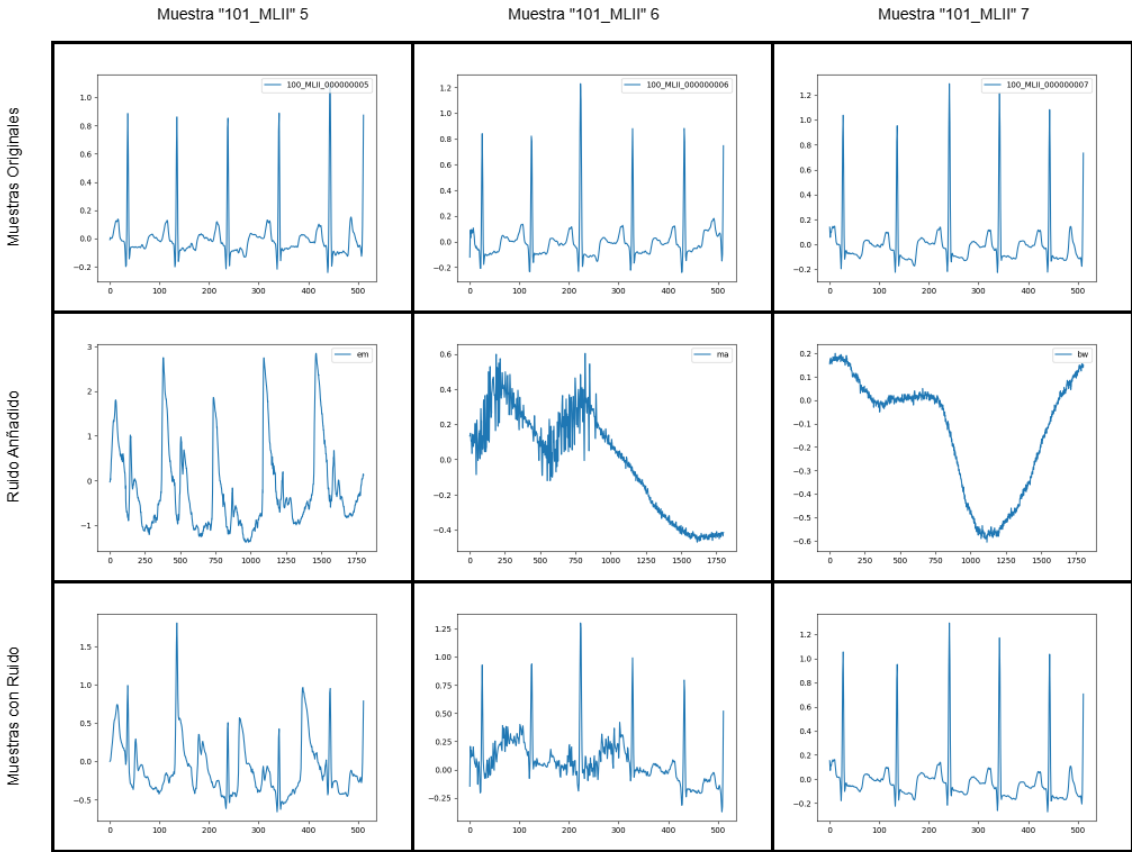

Figura 5.4: Datos de entrada antes y después, de la adición de ruido. Muestras extraídas de una partición en 512 fragmentos de un ECG. En la primera fila se presentan las muestras originales, mientras que la última fila contiene las muestras con ruido añadido. En la segunda fila se muestran los ruidos añadidos a la señal original.

5.2. Arquitectura de la CNN

La arquitectura de la CNN implementada sigue una estructura *slowfast* [25], con una rama de tipo *slow* y otra de tipo *fast*. Este tipo de redes analizan características diferentes de la entrada en cada rama. La rama *slow* extrae las características de baja frecuencia, mientras que la rama *fast* analiza los componentes de alta frecuencia. Este tipo de análisis mixto hace que sea un tipo de red ideal para analizar series temporales. Debido a que en origen las redes SlowFast se crearon para el análisis de información 3D, como los frames de un vídeo, es necesario adaptarla para el análisis de señales 1D (sustituyendo las convoluciones tridimensionales por unidimensionales), que son el tipo de señales utilizadas en el desarrollo de este TFG.

Seguido a esto, se incorpora un decodificador con 5 bloques, formando una red de tipo U-Net [26]. Este tipo de estructura ha sido elegida por la necesidad de reconstruir la señal limpia, lo que hace conveniente el uso de arquitecturas con decodificadores. En la Figura 5.5, puede verse el esquema general de la arquitectura de la red, cuyos elementos serán detallados a continuación.

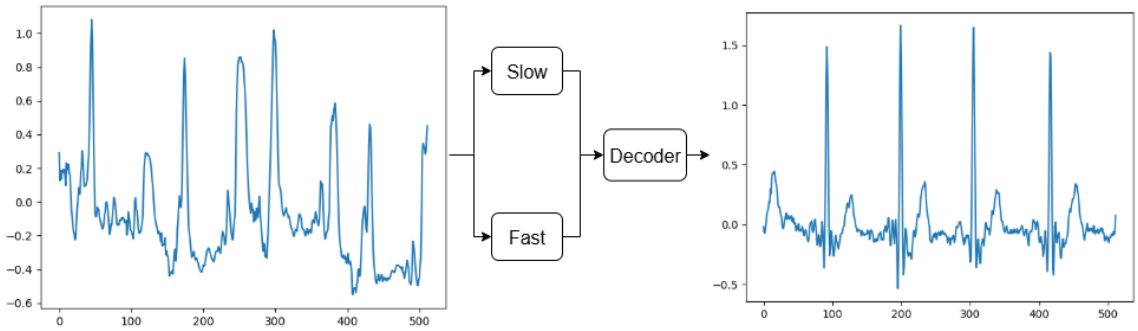

Figura 5.5: Arquitectura general de la CNN propuesta. Estructura tipo U-Net formada por etapas Slow y Fast seguidas de una etapa de decodificación.

5.2.1. Etapa Slowfast

En esta etapa la entrada se distribuye entre las ramas *slow* y *fast*. Cada una de ellas está compuesta por 4 bloques. Por lo tanto, podemos dividir el proceso en 4 etapas, teniendo todas ellas una estructura tipo ResNet.

En la primera etapa, la rama *slow* '*branch_slow_1*', se realiza un *max pooling* con *kernel* 1 y *stride* 2; seguido de una convolución '*conv1d*' con *kernel* 32, y *stride* 1 y sin *padding*; y una etapa '*bottleneck*' que aplica convoluciones 1×1, 3×1 y 1×1, en ese orden. En el primero de ellos se aplica *stride* 2 y en todos normalización por lotes (batch-norm) y ReLU como función de activación. La estructura de la rama *fast*

	Capa 1	Capa 2	Capa 3	Filtros
$branch\_slow\_1$	1×1	3×1	1×1	32
$branch\_slow\_2$	1×1	3×1	1×1	64
$branch\_slow\_3$	1×1	3×1	1×1	128
$branch\_slow\_4$	1×1	3×1	1×1	256
$branch\_fast\_1$	1×1	17×1	1×1	4
$branch\_fast\_2$	1×1	17×1	1×1	8
$branch\_fast\_3$	1×1	17×1	1×1	16
$branch\_fast\_4$	1×1	17×1	1×1	32

Tabla 5.1: Tamaño de cada una de las capas de cada bloque que compone las ramas *slow* y *fast*.

'$branch\_fast\_1$' es similar a la '$branch\_slow\_1$', con las siguientes modificaciones: primera convolución con *kernel* 8, *max pooling* con *stride* 1 y en el bloque '*bottleneck*' la convolución con *kernel* 3, se modifica para *kernel* 17.

En la siguiente etapa, la rama *slow* '$branch\_slow\_2$' toma como entrada la salida de la etapa *slow* anterior, mientras que la rama *fast* '$branch\_fast\_2$' toma como entrada la suma entre la salida *fast* anterior y la *slow* modificada para permitir dicha suma. Estas modificaciones consisten en una convolución de *kernel* 1 igualando los canales con la rama fast, seguido de normalización por lotes de 16 y una función ReLU, duplicando por último el tamaño de la señal (*upsample*). Ambas ramas están compuestas por un bloque '*bottleneck*' con la misma configuración mencionada anteriormente.

En las dos siguientes etapas la arquitectura es igual a la primera, solo que con *stride* 2. Cabe destacar que en la última etapa se realiza un *average pooling*, para formar la salida *slow* y *fast*. Estas salidas se concatenan para dar una salida única. En la Tabla 5.1 puede verse el tamaño de las capas de cada bloque.

Por último, el siguiente diagrama muestra de manera visual lo expuesto anteriormente:

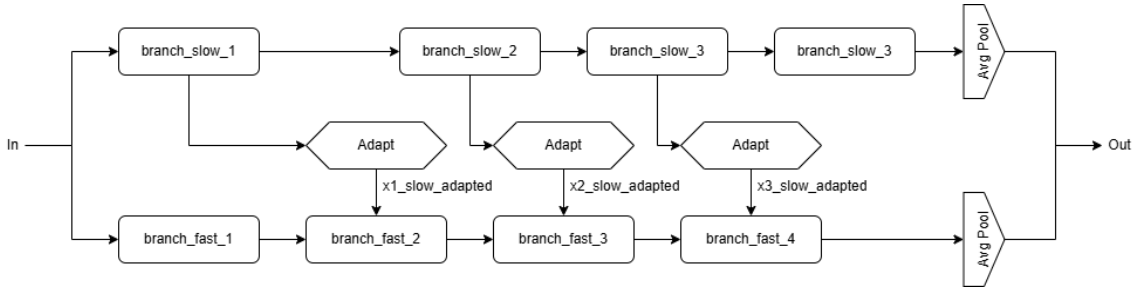

Figura 5.6: Arquitectura de la capa Slowfast. 4 bloques *slow* y 4 bloques *fast* entrelazados. A la salida de ambas ramas se realiza un *average pool*.

5.2.2. Etapa de Decodificación

Para este propósito, se han implementado 5 etapas de decodificación, formando una red tipo U-Net, de la que el primer tramo lo conforma la red SlowFast, el segundo tramo la decodificación. Este tipo de redes ya ha mostrado su efectividad en [5].

Cada bloque está compuesto los siguiente elementos ordenados:

- **Deconvolución**: proceso inverso a la convolución. En la primera etapa se aplica con *kernel* 27, *stride* 8 y *padding* 5; la segunda etapa presenta *kernel* 6, textitstride 2 y *padding* 2; la tercera etapa tiene *kernel* 3, textitstride 2 y *padding* 1; la cuarta *kernel* 3, textitstride 2 y *padding* 1; mientras que la quinta presenta *kernel* 3, textitstride 2 y *padding* 1.
- **Normalización por lotes** en todas las etapas.
- **Dropout**: el porcentaje será configurado según el tipo de entrenamiento.
- **Convolución**: con *kernel* 3 y *padding* 1.
- **Segunda normalización** basada en normalización por lotes al igual que en la normalización anterior.
- **Función de activación**: se ha elegido la función ReLU.

Por último, se implementa una capa *fully connected* linear antes de la salida.

5.3. Entrenamiento de la Red

Con el objetivo de entrenar los modelos, cabe diferenciar entre el proceso realizado para el modelo encargado de filtrar señales y el modelo encargado de predecir el futuro de la señal normal.

En el primer tipo de entrenamiento se parte del *dataset* modificado según se detalla en la Sección 5.1. En cuanto al proceso de entrenamiento, se parte desde cero, estableciendo un *dropout* de 0.2, un *learning rate* siguiendo una función coseno con un valor inicial de $1 \cdot 10^{-4}$ y un mínimo de $1 \cdot 10^{-7}$, y 200 épocas.

Para el segundo modelo, se ha utilizado como *dataset* la partición sin adición de ruido. Para el entrenamiento del modelo, se parte de un modelo preentrenado con una base de datos alterna (MIT-BIH Normal Sinus Rhythm), por lo que solo se realizará un proceso de *fine-tuning*. Para este cometido, se ha establecido un *dropout*

de 0.2, un *learning rate* siguiendo una función coseno con un valor inicial de $1 \cdot 10^{-4}$ y un mínimo de $1 \cdot 10^{-8}$, y 250 épocas.

En ambos modelos se ha empleado un optimizador AdamW, así como la función de pérdida *Split-MSE*, que será desarrollada en la sección 5.3.1, con rango interno de [-0.4, 0.4] y pesos 5 y 1. El proceso de entrenamiento se ha realizado en un servidor con procesador Intel Xeon Silver 4314 de 64 núcleos a 2.4 GHz, 500 Gb de RAM y acceso a dos GEFORCE RTX 3090. En cuanto a software, se han ejecutado en Python 3.10 con PyTorch 2.10 en CUDA 11.8.

5.3.1. Función de Pérdida Split-MSE

Como función de pérdida se ha elegido la función Split-MSE, detallada en [2], ya que está diseñada para su uso con ECGs. En concreto, se divide la señal en 3 secciones horizontales. Posteriormente, se calcula la función de pérdida MSE sobre las secciones exteriores y sobre la sección interior de forma independiente, como puede verse en la figura 5.7. La función de pérdida resultante será conformada por la suma ponderada de ambas, tal y como puede verse en la siguiente ecuación:

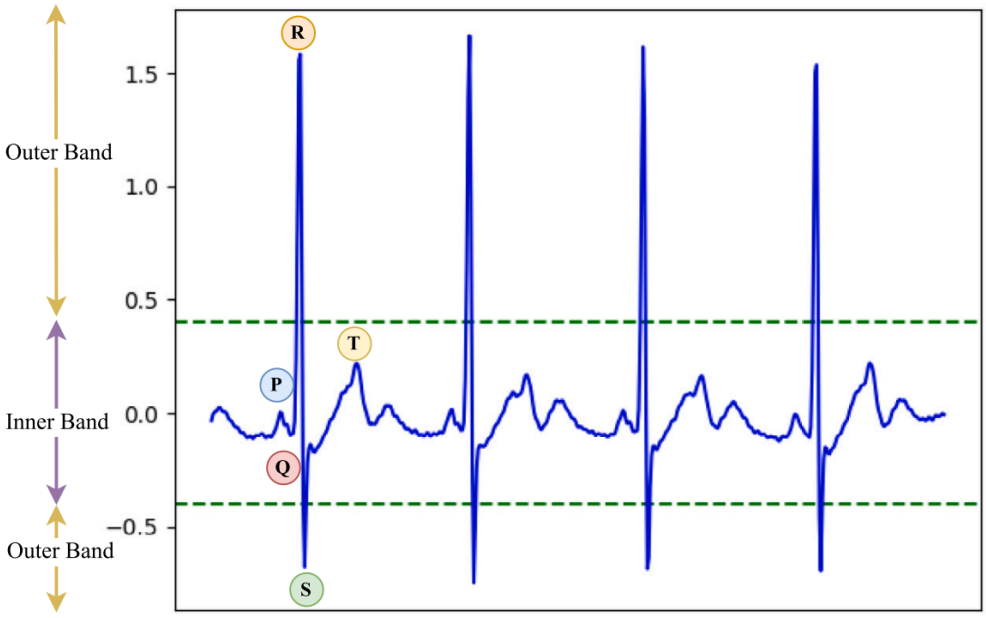

Figura 5.7: Representación de las bandas interna y externa. La banda central incluye los segmentos entre picos R, mientras que las bandas exteriores recogen los picos de las ondas R y S. Imagen extraída de [2].

$$\mathcal{L}_{split}^{MSE} = w_1 \cdot \mathcal{L}_{inner}^{MSE} + w_2 \cdot \mathcal{L}_{outer}^{MSE} \tag{5.3}$$

en donde $\mathcal{L}_{inner}^{MSE}$ y $\mathcal{L}_{outer}^{MSE}$ son el valor de pérdida MSE de las secciones interna y externa (respectivamente), w_1 y w_2 son los coeficientes que ponderan cada uno de los valores de pérdida.

Un mayor valor para w_1 implicará una mayor importancia para la banda central, correspondiente al segmento entre picos de la onda R, mientras que un mayor valor para w_2 hará lo propio para los segmentos exteriores, correspondientes a los picos de las ondas R y S.

5.4. Testeo de la Red

Tras completar los entrenamientos de ambos modelos, se ha realizado la búsqueda del umbral óptimo (*threshold*) entre las muestras normales y anómalas, para posteriormente calcular la precisión del modelo(*accuracy*).

5.4.1. Búsqueda de Umbral (*Threshold*)

La búsqueda de umbral tiene por objetivo encontrar el percentil óptimo que divide las señales normales (NSR) y anómalas (AN), estableciendo así un umbral de decisión. Con este fin se calcula en primer lugar una métrica compuesta por la multiplicación de las métricas MAE y MAE normalizada entre 0 y 1, siendo calculado como la función de pérdida detallada en la Sección 4, de esta forma la métrica estaría definida por:

$$\hat{\mathcal{L}}(y, \hat{y}) = \mathcal{L}(y, \hat{y}) \cdot \mathcal{L}_M(y, \hat{y}) \tag{5.4}$$

donde $\mathcal{L}(y, \hat{y})$ representa la métrica MAE, $\mathcal{L}_M(y, \hat{y})$ la métrica MAE normalizada, $\hat{\mathcal{L}}(y, \hat{y})$ la métrica obtenida, y e $\hat{y}$ la salida obtenida y su etiqueta, respectivamente.

Con el objetivo de clasificar cada muestra de *test* como normal '*(N)*' o anómala, se calcula su métrica según (5.4) y se almacena en el *array* correspondiente a su clase.

Una vez se han obtenido los valores correspondientes a cada una de las muestras evaluadas; se itera sobre percentiles, en el rango [0, 100] con paso 0.0001, hasta hallar el que maximice la precisión media de clasificación entre NSR y AN. Se obtendrá como resultado la mejor precisión obtenida, el umbral óptimo, y la precisión por clase en dicho umbral.

5.4.2. Test de Precisión

Este test (*accuracy*) tiene por objetivo evaluar la precisión real del modelo, mediante el umbral anteriormente calculado. Esta métrica se calcula tanto de forma global como para las clases AN y NSR de forma individual, además de analizar individualmente la precisión para cada clase anómala. La métrica de precisión se calcula de la siguiente forma:

$$Accuracy = \frac{\mathrm{TPR}_{AN} + \mathrm{TNR}_{AN}}{2} \tag{5.5}$$

donde TPR_{AN} y TNR_{AN} son, respectivamente, el porcentaje de AN mayor o igual al *treshold* calculado y NSR menor al *treshold*, ambos frente al total de su clase.

Por último, se obtiene como salida la precisión global y por clase, así como se almacenan estos datos junto con el treshold para un análisis posterior.

Capítulo 6

Experimentos y Resultados

En este capítulo se van a presentar las bases de datos utilizadas, así como los experimentos realizados para comprobar el rendimiento del modelo entrenado.

6.1. Bases de Datos

Para la realización de los experimentos se han utilizado dos bases de datos, el *dataset* MIT-BIH Arrhythmia Database como base de datos de ECGs, mientras que los ruidos se han extraido de MIT-BIH Noise Stress Database. Como base auxiliar para el pre-entreno del modelo predictor, se ha utilizado la base de datos MIT-BIH NSR, que contiene 18 grabaciones ECGs de larga duración.

La base de datos MIT-BIH Arrhythmia Database contiene 48 grabaciones de media hora de duración de ECG, obtenidos de 47 sujetos entre 1975 y 1979. Cada señal está digitalizada a una frecuencia de 360 muestras por segundo, y una resolución de 10 mV. En la Figura 6.1, puede verse una de las muestras del dataset.

La base de datos MIT-BIH Noise Stress Database contiene 3 señales de ruido, concretamente de tipo *baseline wander* (BW), *muscle artifact* (MA) y *electrode motion artifact* (EM). El resto del contenido son varias señales de MIT-BIH Arrhythmia Database con los tipos de ruido anteriormente mencionados añadidos a las señales ECG. En la Figura 6.2, puede verse una de las muestras de ruido.

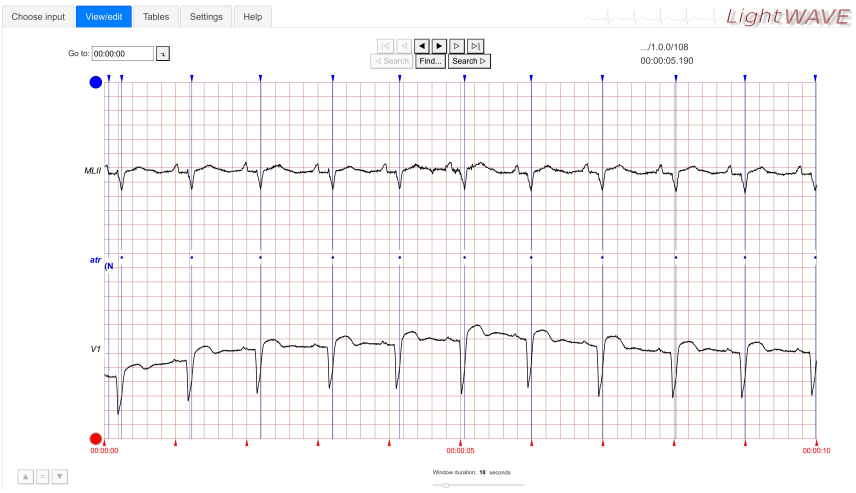

Figura 6.1: Señal 108 del *dataset* MIT-BIH Arrhythmia Database. Pueden verse los dos canales capturados, siendo MLII el utilizado en este TFG. Captura obtenida del visualizador lightWAVE de Physionet [1].

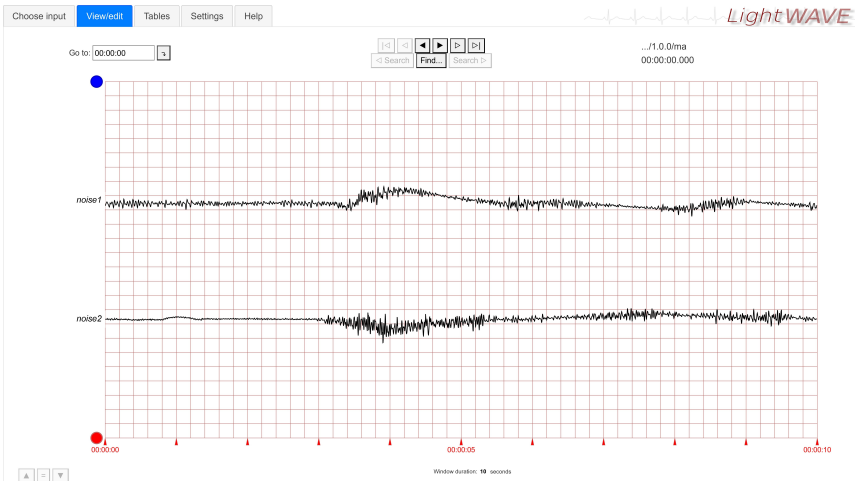

Figura 6.2: Señal *ma* del *dataset* Noise Stress Database. Pueden verse los dos canales capturados, siendo *noise1* el utilizado en este TFG. Captura obtenida del visualizador lightWAVE de Physionet [1].

6.2. Resultados Experimentales

A continuación se expondrán los resultados de los experimentos realizados en el desarrollo de este TFG. El primero de ellos es el cálculo del MAE del modelo de reconstrucción de ECGs, respecto a cada uno de los ruidos. El segundo es la comprobación de la calidad de la señal filtrada por el modelo, a través del test de precisión en el modelo predictor, tomando como entrada del segundo modelo la salida del primero.

6.2.1. MAE del Modelo de Reconstrucción de ECGs

El objetivo de este experimento, es comprobar la calidad del filtrado de señales mediante la métrica MAE (*Mean Absolute Error*). Para ello, se ha pasado por el modelo encargado de la reconstrucción el dataset *MIT-BIH Arrhythmia Database* (segmentando previamente cada muestra en tramos de 4 segundos), y se le ha añadido un sólo tipo de ruido para cada muestra, con SNR aleatorio. Con el conjunto de resultados a la salida del modelo, se calcula su MAE como se describe en 4.5.2, obteniéndose dicha métrica para cada tipo de ruido. En la Figura 6.3 puede verse el proceso desde la generación de las señales para el experimento hasta la salida del modelo para ruido BW. Por otro lado, En la Figura 6.4 puede verse la comparativa entre señal original y reconstruida por el modelo para cada tipo de ruido.

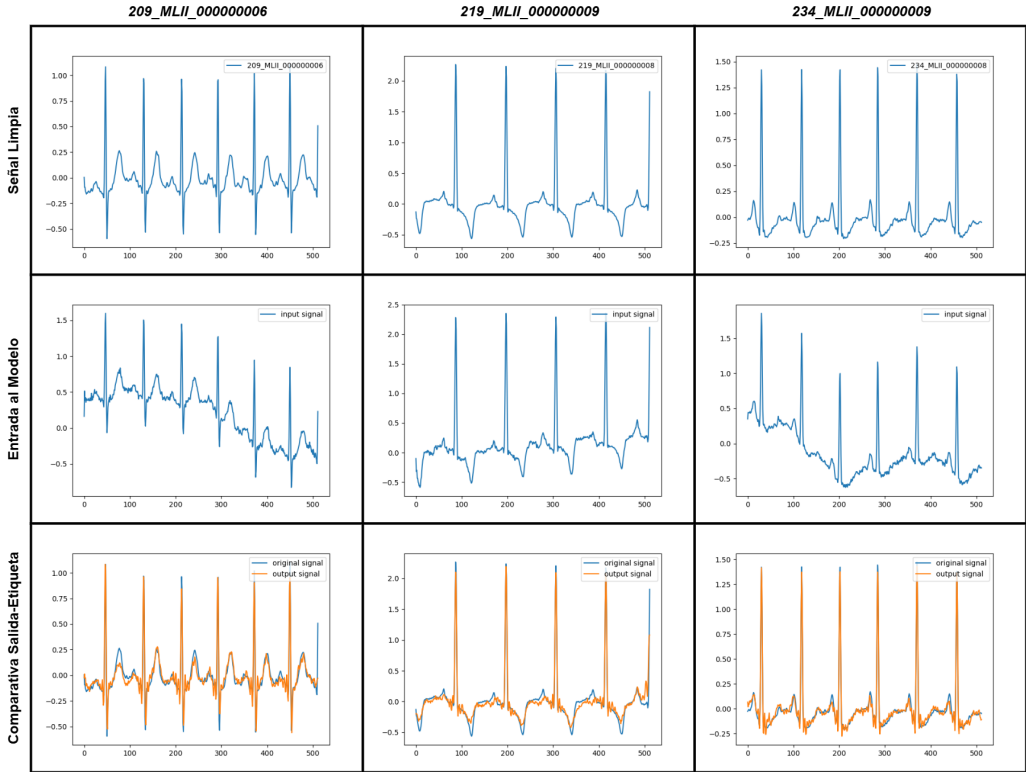

Figura 6.3: Proceso de filtrado en tres de las muestras del *dataset*. La primera fila muestra las señales originales, la segunda, las imágenes con ruido BW añadido, y la tercera es una comparativa entre la salida del modelo y la señal original.

A continuación, se calcula el MAE promedio entre las señales reconstruidas y la originales mencionado en 4.5.2. Cada uno de los *dataset* utilizados para el test cuenta con 26624 segmentos modificados de la base de datos MIT-BIH Arrhytmia Database. Los resultados se muestran en la Tabla 6.1

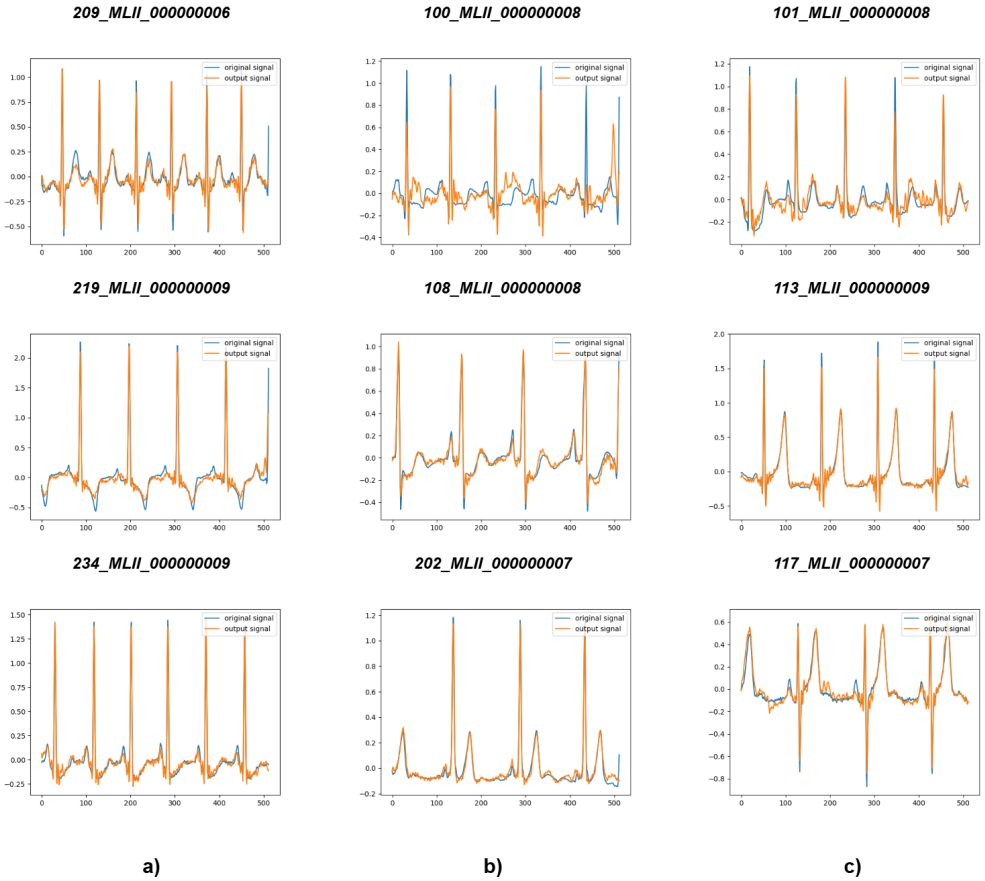

Figura 6.4: Comparativa entre señales originales y salida del *dataset*. En a) se muestran tres señales que previamente tenían ruido BW, en b) tres señales previamente con ruido tipo EM y en c) tres muestras con ruido MA a la entrada del modelo.

	Ruido BW	Ruido EM	Ruido MA
MAE	0.0466	0.0631	0.0523

Tabla 6.1: MAE medio obtenido para para cada tipo de ruido. Se ha calculado para el dataset MIT-BIH Arrhytmia Database con ruido BW, EM y MA.

6.2.2. *Accuracy* del modelo Predictor

Para realizar este test se parte del modelo predictor, sobre el que se comprobará su capacidad para distinguir entre ECGs normales y con anomalías. Con este cometido, se han utilizado 3 tipos distintos de *datasets*. El primero está compuesto por muestras sin modificar del *dataset* MIT-BIH Arrhytmia Database. El segundo *dataset* lo conforman segmentos de la misma base de datos, modificados con ruido aleatorio de tipo BW, EM y MA. El tercer dataset sirvió de preentreno para el modelo predictor, compuesto por muestras normales. En cuanto al test, se ha calculado el *threshold* óptimo y la precisión (*accuracy*) tal y como se describe en el punto 5.4. En la tabla 6.2 pueden apreciarse los resultados obtenidos para cada conjunto de datos, mientras que en la tabla 6.3 se muestran los resultados obtenidos con muestras

reconstruidas a partir de señales con un único tipo de ruido.

	Originales	Con Ruido	Reconstruidas
Percentil	0.0195	0.0205	0.0303
Accuracy	86.72 %	69.37 %	82.96 %
Accuracy NSR	84.97 %	94.23 %	84.91 %
Accuracy AN	88.47 %	44.51 %	81.01 %

Tabla 6.2: Métricas de precisión del modelo predictor. La primera columna muestra la precisión en clasificación de muestras sin modificar, tanto normales como anómalas. En la segunda columna se muestra dicha precisión para muestras con ruido añadido. En la tercera columna se muestra lo propio para muestras con ruido, previamente reconstruidas por el modelo de reconstrucción de ECGs.

	Ruido *BW*	Ruido *EM*	Ruido *MA*
Percentil	0.0255	0.0293	0.0311
Accuracy	83.22 %	83.21 %	81.68 %
Accuracy NSR	76.71 %	81.09 %	79.19 %
Accuracy AN	89.73 %	85.32 %	84.17 %

Tabla 6.3: Métricas de precisión para cada tipo de ruido del modelo predictor. La primera columna muestra la precisión en clasificación de muestras reconstruidas a partir de muestras con ruido *baseline wander*, tanto normales como anomalías. En la segunda columna se presenta dicha precisión para muestras con ruido *electrode movement*. La tercera columna está constituida con los resultados para muestras reconstruidas con ruido tipo *muscle artifact*.

6.3. Discusión de Resultados

A continuación, se analizarán los resultados obtenidos en cada uno de los *test*.

En el *test* mediante el cálculo de la métrica MAE para el modelo de reconstrucción de ECGs y cada uno de los ruidos, se han obtenido resultados similares. Para el ruido tipo *baseline wander* se ha obtenido una error medio del 0.0466. Para el tipo de ruido *electrode movement* se ha registrado una MAE del 0.0631, y para el ruido tipo *muscle artifact* un error del 0.0523. Al ser un ruido predecible de muy baja frecuencia, el ruido BW es el que menor perdida presenta, como era esperado. El error medio de todos los tipos de ruido es del 0.054.

En el *test* mediante la precisión del modelo clasificador se han obtenido resultados coherentes con el primer *test*. El modelo clasificador consigue una precisión media general del 86.72 % para señales originales, un 69.37 % para señales con ruido y un 82.96 % para señales reconstruidas. El segundo resultado puede ser engañoso, ya que la precisión desglosada en detección de señales normales (NSR) y anomalías (AN) es muy dispar (94.23 % y 44.51 %, respectivamente). Este desbalanceo puede ser debido a una predisposición del modelo a seleccionar este tipo de señales cuando la señal es

ruidosa. Cabe destacar que el modelo alimentado con señales reconstruidas presenta una mejora del 36.5 % en señales AN y 13.59 % general, respecto a cuando predice ese tipo de señales con ruido. Además, la degradación en la precisión respecto a señales originales es de sólo 0.06 % para señales normales, con un 3.76 % general.

En cuanto a la eficacia del modelo predictor para señales reconstruidas respecto a muestras con un sólo tipo de ruido, puede apreciarse que la precisión del conjunto es bastante similar para ruidos tipo BW y EM, con un 83.22 % y 83.21 %, respectivamente. Sin embargo, la precisión con ruidos BW para muestras NSR es de 76.71 % y de 89.73 % para anomalías, con una diferencia de precisión del 13.02 % entre tipo de muestras. Las muestras con ruido tipo EM presentan una precisión de 81.09 % y de 85.32 %, para muestras NSR y de anomalías respectivamente, los cuales están más balanceados.

Por otro lado, para ruidos de tipo MA, se ha obtenido una precisión general del 81.68 %, con resultados de 79.19 % para NSR y 84.17 % para anomalías. Analizando el conjunto de los datos, puede verse que la mayor y menor precisión para un tipo de muestra se presenta para ruidos de tipo BW, siendo las muestras con ruido EM las que presentan una precisión más homogénea. Las muestras con ruido MA son las que tienen menor precisión, si bien la diferencia no es significativa.

Puede concluirse que los resultados son satisfactorios, ya que el error se mantiene en un rango aceptable y no afecta significativamente al funcionamiento del modelo predictor.

Capítulo 7

Conclusiones y Líneas Futuras

En este capítulo se expondrán las conclusiones extraídas a partir de la realización de este TFG. Además, se mencionarán una serie de líneas futuras a seguir tras la realización de este trabajo.

7.1. Conclusiones

Para la realización de este TFG, se ha entrenado una CNN con capacidad para eliminar el ruido de ECGs. Con este propósito, se ha particionado una base de datos de ECGs para posteriormente añadirle ruido. Para medir la calidad del modelo propuesto, se ha utilizado el test de precisión (*accuracy*) como mecanismo de test, que por medio de métricas permite hallar la precisión del modelo. A su vez, se ha usado un modelo de detección de anomalías para comprobar la calidad de la señal reconstruida, al que se le ha realizado un proceso de *fine-tunning*. Tras la evaluación de los resultados, se han extraído las siguientes conclusiones:

- En términos generales, el filtrado de ECGs por medido de algoritmos de *deep learning*, reconstruye señales ECG satisfactoriamente.
- Se ha demostrado que las muestras reconstruidas por medio del modelo son utilizables por modelos clasificadores, sin pérdida de precisión relevante.
- Resulta beneficioso el uso de una red SlowFast como etapa de codificación, a la hora de entrenar un modelo robusto y preciso.
- El uso de la técnica *fine-tunning* permite adaptar un modelo general a un contexto específico, evitando el efecto indeseado de *overfitting* cuando el dataset presenta un número reducido de muestras.

- En términos generales, se ha comprobado la eficacia del optimizador AdamW a la hora de actualizar los pesos de un modelo en su etapa de entrenamiento.
- Respecto a la inferencia del tipo de ruido en una señal de ECG, se ha comprobado que el *electrode movement* es el más distorsionador.

7.2. Líneas Futuras

A continuación, se expondrán un conjunto de posibles líneas futuras de ampliación del trabajo expuesto en este TFG.

- Integración de un modelo de filtrado y clasificación en un equipo de monitorización cardíaca en tiempo real.
- Búsqueda e implementación de *datasets* con más tipos de ruido y/o mayor número de muestras.
- Evaluación de modelos entrenados a partir de arquitecturas distintas.
- Estudio de la eliminación de ruido en otras derivaciones.
- Aplicación de la metodología en otros campos de muestras temporales con patrones identificables.

Bibliografía

[1] *About LightWAVE*, https://physionet.org, Último acceso: 20 Jun., 2025 [Online]. Disponible en: https://physionet.org/lightwave/?db=mitdb/1.0.0.

[2] P. Ruiz-Barroso, F. M. Castro, J. Miranda, D.-A. Constantinescu, D. Atienza, and N. Guil, "Fade: Forecasting for anomaly detection on ecg," *Computer Methods and Programs in Biomedicine*, 2025.

[3] F. Azuaje, G. Clifford, and P. McSharry, *Advanced Methods and Tools for ECG Data Analysis*. IEEE, 2005.

[4] A. Buades, B. Coll, and J.-M. Morel, "A non-local algorithm for image denoising," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, 2005.

[5] H.-T. Chiang, Y.-Y. Hsieh, S.-W. Fu, K.-H. Hung, Y. Tsao, and S.-Y. Chien, "Noise reduction in ecg signals using fully convolutional denoising autoencoders," *IEEE Access*, 2019.

[6] S. Nurmaini, A. Darmawahyuni, A. N. S. Mukti, and B. T. Muhammad Naufal Rachmatullah, Firdaus Firdaus, "Deep learning-based stacked denoising and autoencoder for ecg heartbeat classification," *Electronics*, 2020.

[7] D. Sinar and G. K. Knopf, "Printed graphene derivative circuits as passive electrical filters," *Nanomaterials*, 2018.

[8] G. A. Campbell, "Physical theory of the electric wave-filter," *Bell System Technical Journal*, 1922.

[9] S. Butterworth, "On the theory of filter amplifiers," *Experimental Wireless & The Wireless Engineer*, 1930.

[10] K. Dragomiretskiy and D. Zosso, "Variational mode decomposition," *IEEE Transactions on Signal Processing*, 2014.

[11] *Conozca la configuración de los filtros de su MESI mTABLET ECG*, https://www.mesimedical.com, Último acceso: 23 Jun., 2025 [Online]. Disponible en: https://www.mesimedical.com/es/conozca-la-configuracion-de-los-filtros-de-su-mesi-mtablet-ecg/.

[12] Y. Xu, M. Luo, T. Li, and G. Song, "Ecg signal de-noising and baseline wander correction based on ceemdan and wavelet threshold," *Sensors*, 2017.

[13] M. Torres, M. Colominas, G. Schlotthauer, and P. Flandrin, "A complete ensemble empirical mode decomposition with adaptive noise," *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011.

[14] G. Moody and R. Mark, "The impact of the mit-bih arrhythmia database," *IEEE Eng in Med and Biol*, May-June 2001.

[15] M. GB, M. WE, and M. RG, "A noise stress test for arrhythmia detectors," *Computers in Cardiology*, 1984.

[16] *Welcome to Pyhton.org*, Python.org, Último acceso: 16 Sept., 2020 [Online]. Disponible en: https://www.python.org/.

[17] *TensorFlow*, TensorFlow.org, Último acceso: 16 Sept., 2020 [Online]. Disponible en: https://www.tensorflow.org/.

[18] *PyTorch documentation*, Python.org, Último acceso: 19 Jun., 2025 [Online]. Disponible en: https://docs.pytorch.org/docs/stable/index.html.

[19] *Welcome to deepdish's documentation!*, University of Chicago, Último acceso: 19 Jun., 2025 [Online]. Disponible en: https://deepdish.readthedocs.io/en/latest/.

[20] *The top 10 causes of death*, Organización Mundial de la Salud (OMS), Último acceso: 21 Sept., 2025 [Online]. Disponible en: https://www.who.int/en/news-room/fact-sheets/detail/the-top-10-causes-of-death.

[21] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[22] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, 2014.

[25] C. Feichtenhofer, H. Fan, J. Malik, and K. He, "Slowfast networks for video recognition," *Facebook AI Research (FAIR)*, 2019.

[26] R. Olaf, F. Philipp, and B. Thomas, "U-net: Convolutional networks for biomedical image segmentation," *Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany*, 2015.