

Estudio sobre la Detección de Anomalías en Series de Datos

Grado en Ingeniería Informática



Trabajo de Fin de Grado
Junio de 2022

Autor
Eduardo Ortega Naranjo
Tutor
Francisco Mario Hernández Tejera

Estudio sobre la Detección de Anomalías en Series de Datos

Grado en Ingeniería Informática

Trabajo de Fin de Grado

Mención primera: Ingeniería de Software

Mención segunda: Computación

Autor

Eduardo Ortega Naranjo
www.theagilecrafter.com

Tutor

Francisco Mario Hernández Tejera
Departamento de Informática y Sistemas
Catedrático de Universidad

Las Palmas de Gran Canaria, junio de 2022

“Cuando lleguemos a ese río, cruzaremos ese puente.”

Julio César.

Resumen

La detección de anomalías es un problema de gran relevancia en la ciencia de datos debido a que, una vez interpretadas, pueden indicar problemas críticos como, por ejemplo, un fallo técnico en una máquina o un cambio de comportamiento en los hábitos de animales o personas.

El objetivo de este trabajo es realizar un estudio comparativo sobre la detección de anomalías en series de datos usando redes neuronales, centrándose en analizar el rendimiento de diferentes modelos y arquitecturas para determinar cuál es la mejor.

Para realizar el estudio, se seleccionan dos arquitecturas de red a las que se aplican diferentes hiperparámetros y se evalúan los modelos resultantes contra dos tipos de series temporales: periódicas y no periódicas.

Abstract

Anomaly detection is a highly relevant problem in data science because, once interpreted, anomalies can indicate critical problems, such as technical failures in a machine or a change in the behaviour of animals or people.

The aim of this survey is to carry out a comparative study on anomaly detection in datasets using neural networks, focusing on analysing the performance of different models and architectures to determine which one is the best.

To perform the study, two network architectures are selected to which different hyperparameters are applied. The resulting models are evaluated against two types of time series: periodic and non-periodic.

Agradecimientos

Me gustaría comenzar esta memoria agradeciendo a todas aquellas personas que de forma directa o indirecta han colaborado en la realización de este proyecto.

Empezando por mi tutor, Francisco Mario Hernández Tejera, por su apoyo, asesoramiento y paciencia a lo largo del desarrollo del Trabajo de Fin de Grado (TFG).

Al subdirector de Relaciones Institucionales e Igualdad, José Daniel Hernández Sosa, por su rápida respuesta a mis preguntas sobre el procedimiento administrativo para presentar el TFG.

Al usuario @pixabay por haber publicado las imágenes que he usado en la portada¹ y capítulos² de la memoria en www.pexels.com (licencia de dominio público³).

Por último, pero no menos importante, a mis amigos, mis padres, compañeros de clase y del trabajo por estar ahí.

A todos, muchas gracias.

¹ <https://www.pexels.com/photo/black-board-board-game-checkerboard-459275/>

² <https://www.pexels.com/photo/brown-and-black-wooden-chairs-inside-room-207691/>

³ <https://creativecommons.org/publicdomain/zero/1.0/>

Sobre la Imagen de Portada y Capítulos

En la portada de esta memoria se puede ver un tablero de ajedrez con dos filas de peones, agrupados por colores. Sin embargo, en una de esas filas hay un peón que es de diferente color al resto. Esto es una metáfora sobre el concepto de anomalía en un conjunto de datos.

Al inicio de cada capítulo, detrás del título, hay una imagen que muestra un aula universitaria clásica. Esta imagen se usa para que se perciba con mayor facilidad el inicio del capítulo. Esto se debe a que existe un elemento que rompe la continuidad de la lectura de forma llamativa entre capítulos contiguos.

Índice general

Resumen	4
Abstract	4
Agradecimientos	5
Sobre la Imagen de Portada y Capítulos	6
Capítulo 1. Introducción	9
1.1. Definición del Problema.....	9
1.1.1. ¿Qué son las Anomalías?	10
1.1.2. Tipos de Anomalías.....	10
1.2. Motivación y Objetivos Iniciales	12
1.3. Justificación de las Competencias Específicas Cubiertas.....	12
1.4. Aportaciones al Entorno Técnico	13
Capítulo 2. Aprendizaje Profundo para la Detección de Anomalías	15
2.1. Conceptos Previos.....	15
2.1.1. Aprendizaje Automático.....	15
2.1.2. Redes Neuronales Artificiales y Aprendizaje Profundo.....	16
2.2. La Detección de Anomalías	21
2.2.1. Complejidad del Problema	22
2.2.2. Métodos de Detección	24
Capítulo 3. Metodología de Trabajo	31
3.1. Metodología de Desarrollo	31
3.2. Entorno de Desarrollo.....	32
3.3. Librerías Utilizadas	33
3.4. Evaluación de las Redes Neuronales	34
3.4.1. Matriz de Confusión	34
3.4.2. Críticas hacia el Método de Evaluación Tradicional	36
3.4.3. Métrica elegida: Coeficiente de Correlación de Matthews (MCC)	38
3.5. Metodología de Entrenamiento	39
3.5.1. Preparación de los Datos.....	39
3.5.2. Elección de Arquitectura de Red	39
3.5.3. Validación Cruzada (k-Fold Cross-Validation).....	41
3.5.4. Ajuste del Umbral (Threshold Moving)	43

Capítulo 4. Descripción de los Datos	45
4.1. Aspectos Generales.....	45
4.2. Series Periódicas	45
4.3. Series Naturales	48
Capítulo 5. Desarrollo Experimental.....	51
5.1. Planificación Inicial.....	51
5.2. Descripción de Experimentos y Conjunto de Datos	52
5.3. Experimento con Modelo de Ranking	55
5.3.1. Arquitecturas de Red	55
5.3.2. Procedimiento	56
5.3.3. Resultados para las Series Periódicas.....	59
5.3.4. Resultados para las Series Naturales.....	66
5.4. Experimento con Autoencoder.....	73
5.4.1. Arquitecturas de Red	73
5.4.2. Procedimiento	75
5.4.3. Resultados para las Series Periódicas.....	76
5.4.4. Resultados para las Series Naturales.....	82
5.5. Estructura del Código Fuente	85
Capítulo 6. Conclusiones y Trabajo Futuro.....	89
6.1. Conclusiones del Estudio	89
6.2. Conclusiones Personales.....	90
6.3. Trabajo Futuro	91
Anexo A. Recorte de hiperparámetros MLP	93
Anexo B. Limitaciones de Google Colab.....	99
Anexo C. Tablas de Mejores Modelos	100
Bibliografía y Fuentes de Información.....	103

Capítulo 1. Introducción

1.1. Definición del Problema

El problema de la detección de anomalías consiste en identificar patrones en conjuntos de datos que no se ajustan al comportamiento esperado en relación con la mayoría de las observaciones. Estos patrones se conocen como anomalías, excepciones, valores atípicos (en inglés, *outliers*), observaciones discordantes, etc.

La importancia de la detección de anomalías se debe a que estas pueden traducirse en información de vital importancia. Por ejemplo: la presencia de anomalías en el uso de tarjetas de crédito puede indicar un robo de identidad; en una imagen médica pueden indicar la presencia de un tumor; en el tráfico de una red informática pueden indicar que se está produciendo un ciberataque; etc. En la **Figura 1.1** se pueden ver otros ejemplos de aplicaciones prácticas de la detección de anomalías.

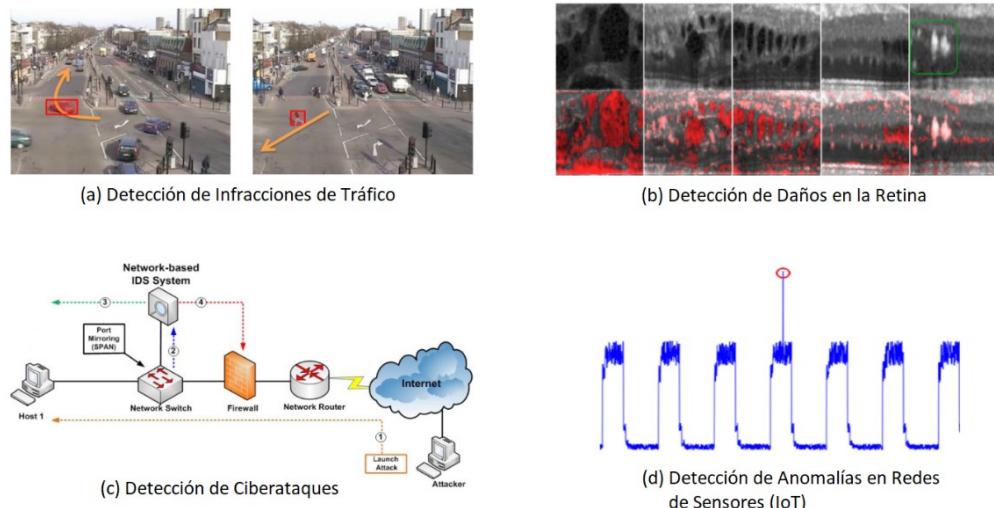


Figura 1.1: Aplicaciones prácticas de la detección de anomalías. Adaptado de [1].

1.1.1. ¿Qué son las Anomalías?

Las anomalías son patrones en conjuntos de datos que no se ajustan a la definición de **comportamiento normal**. En la **Figura 1.2** se puede ver un ejemplo de anomalías presentes en un conjunto de datos bidimensional. Las regiones N_1 y N_2 serían las regiones normales, ya que la mayoría de las observaciones residen en ellas, mientras que los puntos o_1 y o_2 , junto a la región O_3 , son las anomalías. [2]

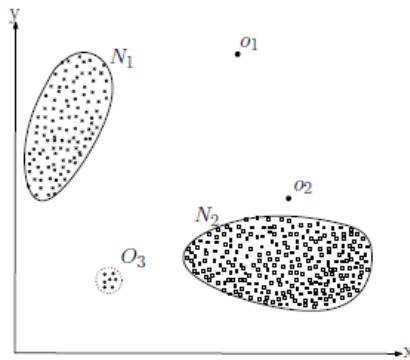


Figura 1.2: Anomalías presentes en un conjunto de datos bidimensional. Fuente [2].

Tal y como se explica en [2], las anomalías pueden aparecer por múltiples razones, como actividad maliciosa (fraude, ciberataques, etc.) o malfuncionamiento de los sistemas de recolección de datos, pero, sea cual sea el motivo, siempre son relevantes para el análisis. En ocasiones, las anomalías son incorporadas al modelo de normalidad, es decir, pasan a considerarse datos normales. Estas anomalías especiales se conocen como patrones emergentes, o novedades (en inglés, *novelty*).

Es importante no confundir las anomalías con el ruido. El ruido se define como un fenómeno en los datos que carece de interés para el análisis [2], es un obstáculo a la hora de realizar dicho análisis y lo deseable es eliminarlo.

1.1.2. Tipos de Anomalías

Las anomalías se clasifican en las siguientes tres categorías [2]:

- **Anomalías Puntuales:** Si una instancia de datos individual puede ser considerada como anómala con respecto al resto de datos, entonces la instancia se denomina punto anómalo. En la **Figura 1.2**, por ejemplo, los puntos o_1 y o_2 , junto a todos los puntos de la región O_3 , representan anomalías puntuales.

- **Anomalías Contextuales o Condicionales:** Si una instancia de datos es anómala en un contexto concreto, pero no de otro modo, entonces se denomina anomalía contextual. Las anomalías contextuales también son anomalías puntuales. En la *Figura 1.3*, por ejemplo, la temperatura t_1 y t_2 tienen el mismo valor, pero sólo t_2 se considera una anomalía, porque ocurre en un contexto diferente. La noción de un contexto debe especificarse como parte de la formulación del problema, cada instancia de datos se define utilizando los siguientes atributos:
 - **Atributos Contextuales:** se utilizan para determinar el contexto para esa instancia. Por ejemplo, en la *Figura 1.3*, el tiempo es un atributo contextual que determina la posición de una instancia en la secuencia.
 - **Atributos de Comportamiento:** definen las características no contextuales de una instancia. Por ejemplo, en la *Figura 1.3*, el valor de la temperatura sería el atributo de comportamiento.

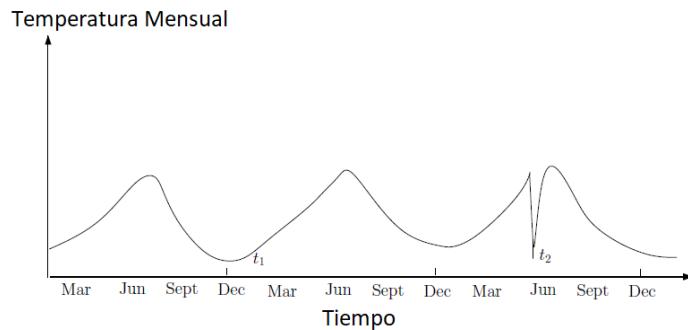


Figura 1.3: Serie temporal con anomalía contextual. Adaptado de [2].

- **Anomalías Colectivas o Grupales:** Si una colección de datos relacionadas es anómala con respecto al resto de datos, se denomina anomalía colectiva. Las instancias de datos individuales en la colección pueden no ser anomalías por sí solas, pero su aparición como grupo es anómala. Por ejemplo, en la *Figura 1.4*, la zona roja corresponde a una anomalía porque existe un valor bajo durante un largo periodo de tiempo.

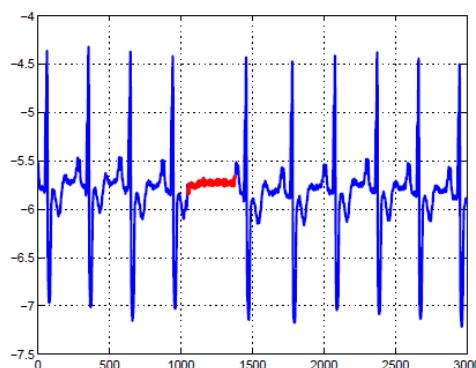


Figura 1.4: Anomalía colectiva en un electrocardiograma. Fuente [2].

1.2. Motivación y Objetivos Iniciales

Cuando buscaba el tema de mi trabajo de fin de grado, quería algo relacionado con la inteligencia artificial y la ciencia de datos, ya que son áreas complejas e interesantes de la informática y consideraba que necesitaba reforzar ese apartado de mi formación. Por otro lado, también tenía dudas sobre si me interesaría realizar un doctorado en el futuro y llevar mi carrera por el ámbito de la investigación.

Al comentar esto en la reunión inicial con mi tutor, me propuso hacer un estudio sobre la detección de anomalías en series de datos. Lo que más me atrajo de este tema fue su versatilidad. De este modo, los objetivos de este trabajo son:

- Suministrarme una formación especializada y complementaria en un tópico de interés requerido en el mercado.
- Enfrentarme a un proyecto completo, con análisis, diseño, desarrollo y evaluación.
- Desarrollar un estudio de detección de anomalías sobre series concretas y evaluar sus resultados.

1.3. Justificación de las Competencias Específicas Cubiertas

Las principales competencias del Grado en Ingeniería Informática que han sido cubiertas durante el desarrollo de este trabajo son las siguientes:

- **CII015:** *Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica.*

Esta competencia ha sido cubierta con la realización del trabajo, sobre todo en el Capítulo 2 y el Capítulo 3, al realizar tareas de investigación y estudio sobre redes neuronales.

- **CP04:** *Capacidad para evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan conducir a su resolución y recomendar, desarrollar e implementar aquella que garantice el mejor rendimiento de acuerdo con los requisitos establecidos.*

Esta competencia ha sido cubierta con la realización del trabajo, sobre todo en el Capítulo 3 y el Capítulo 5, al tener que crear los experimentos e implementar los algoritmos necesarios para evaluar las redes neuronales y al realizar recortes en los hiperparámetros iniciales para reducir el tiempo de entrenamiento.

1.4. Aportaciones al Entorno Técnico

Este Trabajo de Fin de Grado se centra en realizar un análisis del rendimiento de diferentes redes neuronales para encontrar el mejor modelo y arquitectura que permitan resolver un problema, en este caso: detectar anomalías en series de datos.

Por lo tanto, este proyecto se puede usar como guía básica para que futuros estudiantes (o personas interesadas) entiendan cómo entrenar, evaluar y comparar diferentes modelos de redes neuronales.

[Página intencionadamente dejada en blanco]



Capítulo 2. Aprendizaje Profundo para la Detección de Anomalías

2.1. Conceptos Previos

2.1.1. Aprendizaje Automático

El aprendizaje automático (en inglés, **Machine Learning**) es una rama de la inteligencia artificial que consiste en crear algoritmos que aprenden, es decir, algoritmos que realizan mejor su tarea a medida que adquieren experiencia. Estos algoritmos construyen un modelo matemático a partir de un conjunto de datos, conocido como **conjunto de entrenamiento** (en inglés, **training dataset**), para realizar predicciones o tomar decisiones sin ser explícitamente programados para ello. Existen tres tipos de aprendizaje automático:

- **Aprendizaje Supervisado:** en este caso, el algoritmo creará una función a partir del conjunto de entrenamiento. Dicho conjunto está formado por parejas de valores: un componente de la pareja representa los datos de entrada a la función y el otro componente representa la salida de la función para esa entrada. Dicha salida se denomina **etiqueta** (en inglés, **label**). El objetivo es que la función sea capaz de transformar cualquier dato de entrada en su etiqueta correspondiente, después de haber visto una serie de ejemplos (los datos etiquetados del entrenamiento). Este tipo de aprendizaje suele usarse para tareas de predicción (regresión) y clasificación.
- **Aprendizaje no supervisado:** en este caso, el conjunto de entrenamiento carece de etiquetas, lo cual implica que el modelo no sabe qué son los datos y se dedica a buscar similitudes entre las muestras para agruparlas. Esto se conoce como **clustering**.
- **Aprendizaje semisupervisado:** en este caso, el conjunto de entrenamiento posee datos etiquetados y datos no etiquetados. Normalmente, los datos etiquetados son una minoría.

2.1.2. Redes Neuronales Artificiales y Aprendizaje Profundo

Las **redes neuronales artificiales** o **RNA** (en inglés, *artificial neural networks* o *ANN*) son un paradigma de aprendizaje automático inspirado en la forma en que funcionan las neuronas biológicas y su interconexión.

Descrito de forma muy simplificada, las neuronas biológicas son células que transmiten información a través de señales electroquímicas y están compuestas, principalmente, de tres partes (ver *Figura 2.1*): el soma (cuerpo celular), las dendritas (canales de entrada de información) y el axón (canal de salida de la información). Los impulsos nerviosos que emiten otras neuronas son captados por las dendritas, se procesan en el soma y se transmiten a través del axón hacia las neuronas contiguas mediante un impulso nervioso. A partir de esta idea, surgió el modelo de **neurona artificial**.

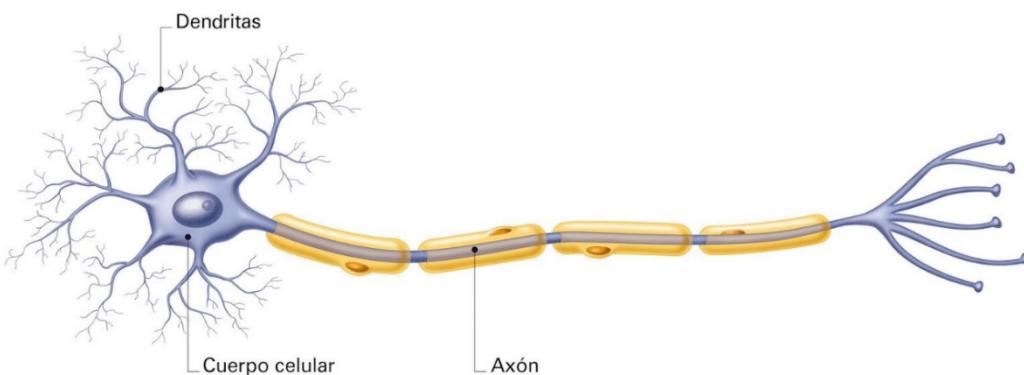


Figura 2.1: Estructura de una neurona humana. Fuente [3].

Las neuronas artificiales son unidades de cálculo que reciben un conjunto de datos como entrada (llamémoslo X) y realizan una suma ponderada de los mismos; si la suma supera cierto umbral (ϕ [*phi*]), la neurona genera un valor de salida, por ejemplo 1, y, si no lo supera, genera un valor distinto, por ejemplo 0. La ponderación viene dada por un peso (W) que está asociado a cada dato particular ($x_1, x_2, x_3, \dots, x_n$) del conjunto de entrada. Es decir, cada conexión de entrada a la neurona tiene asociado un valor ($w_1, w_2, w_3, \dots, w_n$) que sirve para definir con qué intensidad afecta ese dato a la neurona. Existe también un término adicional llamado **sesgo** (b) (en inglés, *bias*), que se añade a la suma ponderada antes de calcular la salida de la neurona. Los pesos y el sesgo son los parámetros que tiene que ajustar la red neuronal para aprender.

Para que la neurona calcule el valor de salida, necesita aplicar una **función de activación** (en inglés, *activation function*), esta función recibe el resultado de la suma calculada anteriormente y genera el valor de salida de la neurona. En la *Figura 2.2* se puede ver un diagrama con la estructura general de una neurona.

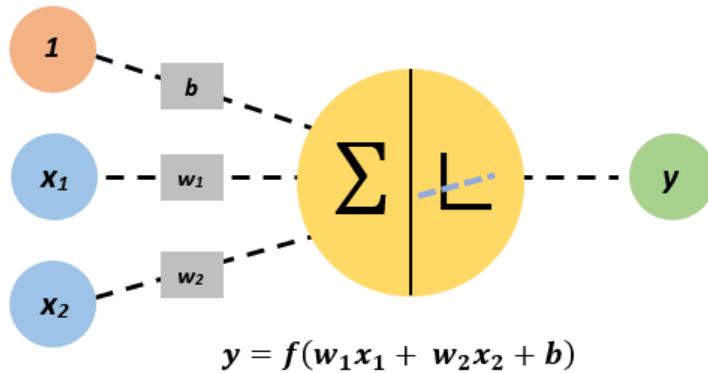


Figura 2.2: Diagrama estructural de una neurona. Adaptado de [4].

Algunas de las funciones de activación más populares son las siguientes (ver representaciones gráficas en la [Figura 2.3](#)):

- **Función escalón unitario** (en inglés, *heaviside step function*): es una función discontinua cuyo valor es 0 para cualquier entrada negativa y 1 para cualquier entrada positiva, incluido el cero.

$$\text{heaviside}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \quad (2.1)$$

- **Función sigmoide:** es una función real que tiene una asíntota horizontal en 0 y otra en 1. Su valor tiende a 0 cuando la entrada de la función es pequeña y tiende a 1 cuando es grande. Se suele usar para representar probabilidades, ya que se mueven en ese rango.

$$\text{sigmoide}(x) = \frac{1}{1 + e^{-x}} \quad (2.2)$$

- **Función tangente hiperbólica (tanh):** es similar a la sigmoide, pero su rango varía entre -1 y 1.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.3)$$

- **Función unidad rectificada lineal (ReLU):** se comporta como una función lineal (la entrada es igual a la salida) cuando su entrada es positiva y vale 0 en cualquier otro caso.

$$\text{relu}(x) = \max(0, x) \quad (2.4)$$

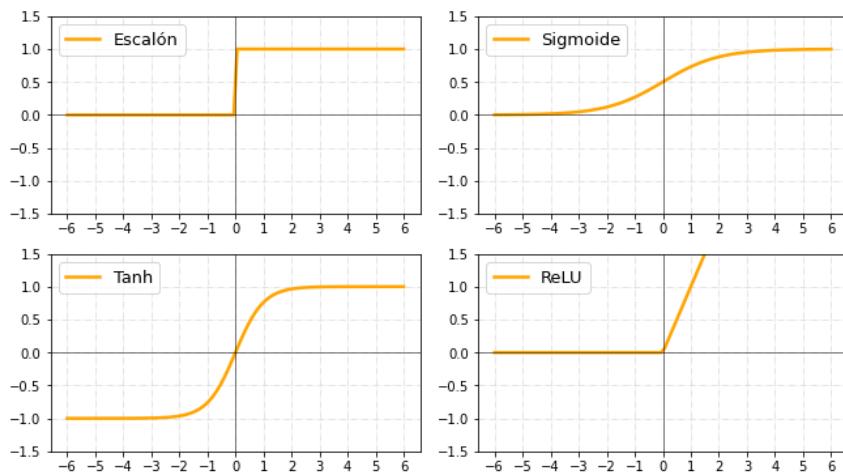


Figura 2.3: Representación gráfica de algunas funciones de activación.

Una neurona, por sí misma, puede considerarse una forma muy simple de red neuronal; esta arquitectura se conoce como **perceptrón** o **perceptrón simple** y tiene la desventaja de que sólo puede resolver problemas lineales. Agrupando varias de estas neuronas, colaborando entre sí para producir un estímulo de salida, se forma una red neuronal con mayor capacidad de aprendizaje.

La estructura que se usa para agrupar neuronas se llama **capa** (en inglés, *layer*) y lo más común es agrupar varias capas para formar **redes multicapa** (en inglés, *multilayer neural network*). Todas las neuronas (también llamadas unidades o nodos) presentes en una capa reciben los mismos datos de entrada, pero producen diferentes salidas, ya que los parámetros (los pesos y el sesgo) se inicializan de forma aleatoria al comienzo del aprendizaje y se van modificando durante el entrenamiento hasta conseguir el resultado esperado. La salida de una capa es la entrada de la siguiente, y, a diferencia de las neuronas individuales, una capa proporciona más de un valor en su salida.

A la primera capa de la red, se la llama **capa de entrada** (en inglés, *input layer*); a la última se la llama **capa de salida** (en inglés, *output layer*); y las capas que se encuentran entre ellas, en caso de que existan, se llaman **capas ocultas** (en inglés, *hidden layers*). De forma general, cuantas más capas se le añadan a una red, mayor será su capacidad para aprender. Esta profundidad en la cantidad de capas es lo que se conoce como **aprendizaje profundo** (en inglés, *deep learning*). En la **Figura 2.4** se puede ver un diagrama de la estructura general de una red neuronal con una capa oculta.

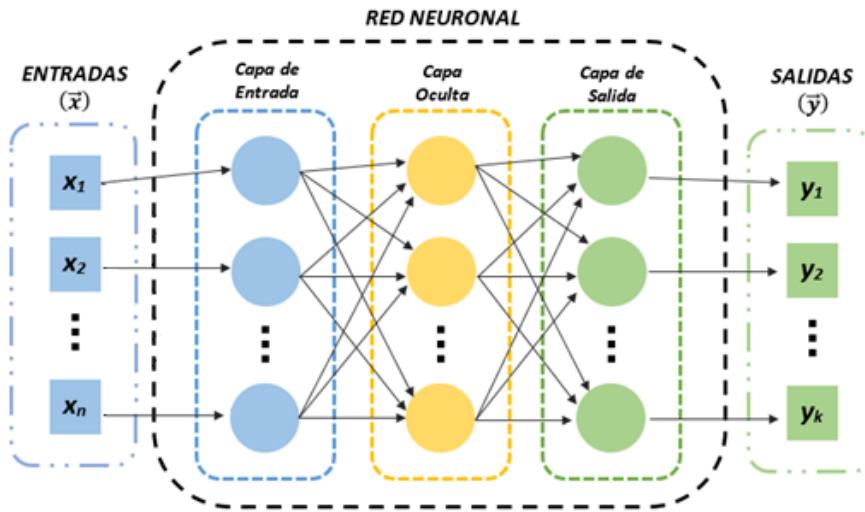


Figura 2.4: Diagrama estructural de una red neuronal con una capa oculta. Adaptado de [5].

Los parámetros de la red se calculan automáticamente durante el entrenamiento, pero existen otras variables que se deben ajustar manualmente para obtener el modelo óptimo, estas se conocen como **hiperparámetros** y, algunos de ellos, son: la cantidad de neuronas en cada capa; la función de activación que se usará en las neuronas de cada capa; el número de capas ocultas; la **función objetivo**, **función de pérdida** o **función de coste** del entrenamiento (en inglés, *objective function*, *loss function* o *cost function*); el **ratio de aprendizaje** (en inglés, *learning rate*); etc.

La **función objetivo** es una medida del error existente entre el valor que el modelo de red predice y el valor real (la etiqueta). Esta medida es la que usará el algoritmo de aprendizaje para actualizar los parámetros de la red. El algoritmo **backpropagation**, que usa el método de descenso según el gradiente, es el algoritmo para entrenar las redes en el aprendizaje supervisado. El error de cada capa se propaga recursivamente hacia atrás, comenzando por la capa de salida, hacia todas las neuronas de la capa anterior y se usa para calcular el gradiente de la función objetivo.

El vector gradiente ($\vec{\nabla}f$) es un vector de direcciones que proporciona la dirección hacia la cual asciende la pendiente de la función objetivo (f) [6], es decir, un vector que indica hacia dónde se incrementa el error. Para disminuirlo, se modifican los parámetros de la red siguiendo el sentido opuesto al gradiente. Para definir cómo afecta dicho gradiente a la actualización de los parámetros en cada iteración, se usa el **ratio de aprendizaje** (α [*alpha*]). En la ecuación (2.5) se muestra cómo se actualizan los parámetros (θ [*theta*]) de una red.

El valor del ratio de aprendizaje es muy importante. Si es muy pequeño, ralentizará la ejecución del algoritmo, mientras que, si es muy grande, puede impedir que se alcance la zona de mínimo coste, impidiendo que el algoritmo converja y causando un bucle infinito.

$$\theta = \theta - \alpha \nabla f \quad (2.5)$$

Algunos de los problemas que se pueden dar al entrenar redes neuronales son el **sobreajuste** (en inglés, *overfitting*) y el **subajuste** (en inglés, *underfitting*). Ambos hacen referencia al fracaso de la red a la hora de generalizar el conocimiento que pretendemos que adquiera.

Se dice que se produce *overfitting* cuando la red obtiene muy buenos resultados durante el entrenamiento, pero no cumple las expectativas cuando se enfrenta a datos nunca vistos anteriormente. En otras palabras, la red recuerda los ejemplos perfectamente, pero no ha aprendido nada de ellos. Por otro lado, cuando se produce *underfitting*, la red no es capaz de aprender de los ejemplos ni de recordarlos (en este caso, hay que modificar la arquitectura de la red, añadir más datos o ambas cosas).

Una táctica que se usa para evitar el *overfitting* es añadir capas de ***dropout*** a la red. Esta técnica consiste en seleccionar y retirar neuronas, junto a sus conexiones, durante la fase de entrenamiento (ejemplo en la [**Figura 2.5**](#)). La selección de neuronas es aleatoria.

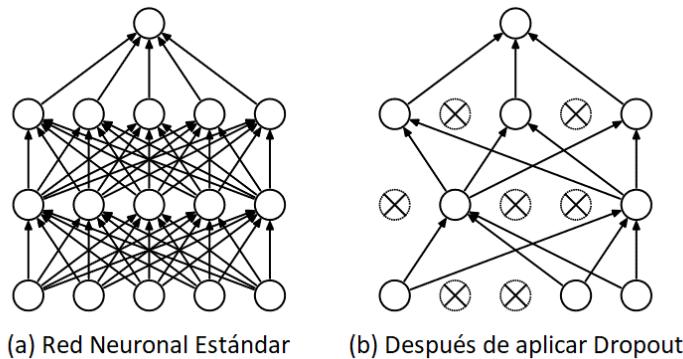


Figura 2.5: Ejemplo de red neuronal antes (a) y después (b) de aplicar dropout. Adaptado de [7].

Según se comenta en [7], en el caso más simple, cada neurona **se conserva** con una probabilidad fija (p) independiente de las otras neuronas. Cuando $p = 1$, implica que no se retiran neuronas, mientras que, valores más bajos de p implican más neuronas retiradas. Los valores de p recomendados en [7] para las neuronas en las capas ocultas están en el rango de 0.5 a 0.8. En el caso de las neuronas de la capa de entrada, se recomiendan valores entre 0.8 y 1.

En la [**Figura 2.6**](#) se puede ver cómo funcionaría una neurona con *dropout*. A la izquierda: una neurona durante el entrenamiento está presente con probabilidad p y está conectada a otras neuronas de la siguiente capa con pesos w . A la derecha: cuando finaliza el entrenamiento, la neurona está siempre presente con sus pesos multiplicados por p . La salida de la red es la misma que se esperaría en el entrenamiento [7].

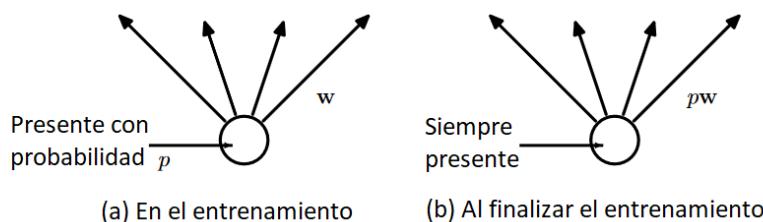


Figura 2.6: Ejemplo de comportamiento de neurona con dropout. Adaptado de [7].

En la práctica, el ratio de *dropout* (p) se interpreta de forma contraria a la teoría descrita en los artículos científicos, es decir, se interpreta como la probabilidad de descartar una neurona.

Por lo tanto, cuando en un artículo se recomienda un ratio de 0.8 (retención del 80%), se convierte, en la práctica, en un 0.2 (descarte del 20%). (Fuente [8], verificado en [9] y [10])

De este modo, los valores de p en la práctica serían:

- entre 0.2 y 0.5 para las capas ocultas
- entre 0.0 y 0.2 para la capa de entrada

2.2. La Detección de Anomalías

El aprendizaje profundo para la detección de anomalías (en inglés, **Deep learning for anomaly detection** o simplemente **Deep anomaly detection**) tiene como objetivo el aprendizaje de las características de las anomalías usando redes neuronales.

El aprendizaje profundo ha demostrado tener capacidad para aprender representaciones de datos complejos como, por ejemplo, conjuntos de datos de altas dimensiones⁴ (en inglés, **high-dimentional data**), datos temporales, espaciales, etc., expandiendo los límites de las diferentes tareas de aprendizaje. En los últimos años, se han presentado varios métodos de detección de anomalías que han demostrado tener un mejor rendimiento que los métodos convencionales [11].

⁴ **La Dimensionalidad**, en estadística, hace referencia a la cantidad de atributos que posee un conjunto de datos. En una hoja de cálculo, cada fila representa una muestra y cada columna representa una dimensión. **Alta Dimensionalidad** significa que el número de dimensiones es asombrosamente alto y puede exceder el número de muestras. Por ejemplo, los chips de ADN, que miden la expresión de los genes, pueden contener decenas de cientos de muestras y cada muestra puede contener decenas de miles de genes [33].

2.2.1. Complejidad del Problema

La detección de anomalías se puede abordar como un problema de clasificación, pero, a diferencia de la mayoría de estas tareas, en las que se trabaja con patrones regulares, en la detección de anomalías se trabaja con eventos minoritarios y de rara frecuencia, lo cual hace que tenga las siguientes complejidades [11]:

- **Desconocimiento:** las anomalías se asocian con muchas incógnitas, por ejemplo, instancias que muestran cambios de comportamiento, cambios de distribución, cambios de estructura, etc. Las anomalías son desconocidas hasta que ocurren, como, por ejemplo, el fraude o los ataques de ciberseguridad.
- **Heterogeneidad:** las anomalías son irregulares, lo cual implica que dos tipos de anomalías pueden tener características diferentes. Por ejemplo, en el análisis de imágenes de videovigilancia, los eventos “robo de una bicicleta” y “robo de un cajero automático” se pueden considerar anomalías de tipo “robo”, pero son visualmente muy diferentes.
- **Rareza y desequilibrio de clases:** las anomalías son poco frecuentes en comparación con los datos considerados normales. Por lo tanto, conseguir una gran cantidad de datos anormales etiquetados es difícil, y es el principal motivo por el que no suele haber grandes conjuntos de entrenamiento disponibles para la mayoría de las aplicaciones.

Además de lo mencionado, hay que tener en cuenta que hay tres tipos de anomalías (descritos en la Sección [1.1.2](#)) y que algunas dependen del contexto en que se encuentren para ser consideradas como tal. También pueden aparecer los tres tipos en el mismo conjunto de datos, lo que añade dificultad a su detección.

Según se explica en [2], los siguientes factores contribuyen a la complejidad del problema:

- Cuando las anomalías son fruto de acciones maliciosas, por ejemplo, los ataques de ciberseguridad, es frecuente que el atacante intente que parezcan datos normales, lo cual enturbia la definición de normalidad.
- En muchos casos, la definición de normalidad evoluciona según se adquieran nuevos datos, de modo que una definición actual puede no ser válida en el futuro. De ahí la importancia de redefinir el modelo cada cierto tiempo.
- La definición de anomalía también cambia según el dominio de los datos que se estén tratando. Por ejemplo, una variación de diez décimas en la temperatura corporal se puede considerar una anomalía en medicina, pero la misma variación puede ser normal en el contexto medioambiental.

- La presencia de ruido en los datos dificulta la distinción de las anomalías y, si el ruido es similar a las anomalías, también es difícil de eliminar.

La naturaleza compleja del problema lleva a una serie de desafíos que aún siguen sin resolver en gran medida [11]:

1. **Baja tasa de sensibilidad (*recall*) en la detección de anomalías.** Debido a que las anomalías son raras y heterogéneas, es difícil detectarlas todas. Esto implica que muchas instancias normales se reporten como anómalas (falso positivo) y que anomalías complejas se consideren instancias normales (falso negativo).
2. **Detección de anomalías en conjuntos de datos de alta dimensionalidad y/o no independientes.** A menudo, las anomalías muestran características anormales evidentes en conjuntos de baja dimensionalidad, como se mostró en la *Figura 1.2* y en la *Figura 1.3*, pero se vuelven imperceptibles cuando la dimensionalidad es alta. Lo más común en estos casos es recurrir a técnicas para reducir la dimensión, como el **análisis de componentes principales** (en inglés, *principal component analysis*) [11], seleccionando un subgrupo de atributos en lugar de trabajar con todos ellos. Pero, aunque sea algo esencial, encontrar relaciones entre los atributos y garantizar que el nuevo subgrupo conserva la información adecuada para detectar las anomalías sigue siendo un desafío. Además, es difícil detectar anomalías en instancias que pueden depender unas de otras, como las relaciones temporales, espaciales, basadas en grafos, etc.
3. **Aprendizaje eficiente de normalidad y anormalidad.** Debido al coste de recolectar y etiquetar grandes conjuntos de datos, la aplicación del aprendizaje supervisado en esta tarea suele ser impráctico. En la última década se han explorado como alternativas el aprendizaje no supervisado y el semisupervisado. El primer método se basa en gran medida en una suposición sobre la distribución de las anomalías y falla cuando los datos infringen dicha suposición. El segundo método trabaja con datos parcialmente etiquetados y trata de aprender a diferenciar bien una de las clases: o bien aprende la representación de normalidad, o bien la de anormalidad. Una variante de este método se conoce como **aprendizaje débilmente supervisado** (en inglés, *weakly-supervised*), donde se asume que hay etiquetas para las anomalías, pero son inexactas o incompletas.
4. **Detección de anomalías resistente al ruido.** Asumir que un conjunto de datos está limpio (carece de ruido) puede volver al modelo vulnerable frente a instancias ruidosas, clasificándolas como la clase opuesta.
5. **Detección de anomalías complejas.** La mayoría de los métodos existentes son para la detección de anomalías puntuales, y no funcionan bien para detectar anomalías condicionales (contextuales) o grupales.

- 6. Explicación de las Anomalías:** La mayoría de los métodos existentes se centran en la precisión a la hora de detectar anomalías e ignoran la tarea de proporcionar una explicación acerca de por qué se considera una anomalía.

2.2.2. Métodos de Detección

Los métodos de detección de anomalías con aprendizaje profundo se pueden clasificar jerárquicamente en tres categorías principales y once subcategorías atendiendo a la forma de modelado [11]. En la **Figura 2.7** se presenta una visión general de la taxonomía usada junto con la referencia a los desafíos (numerados del 1 al 6 en la sección anterior) que aborda cada método. A continuación, se explican dichas categorías, así como sus ventajas y desventajas.

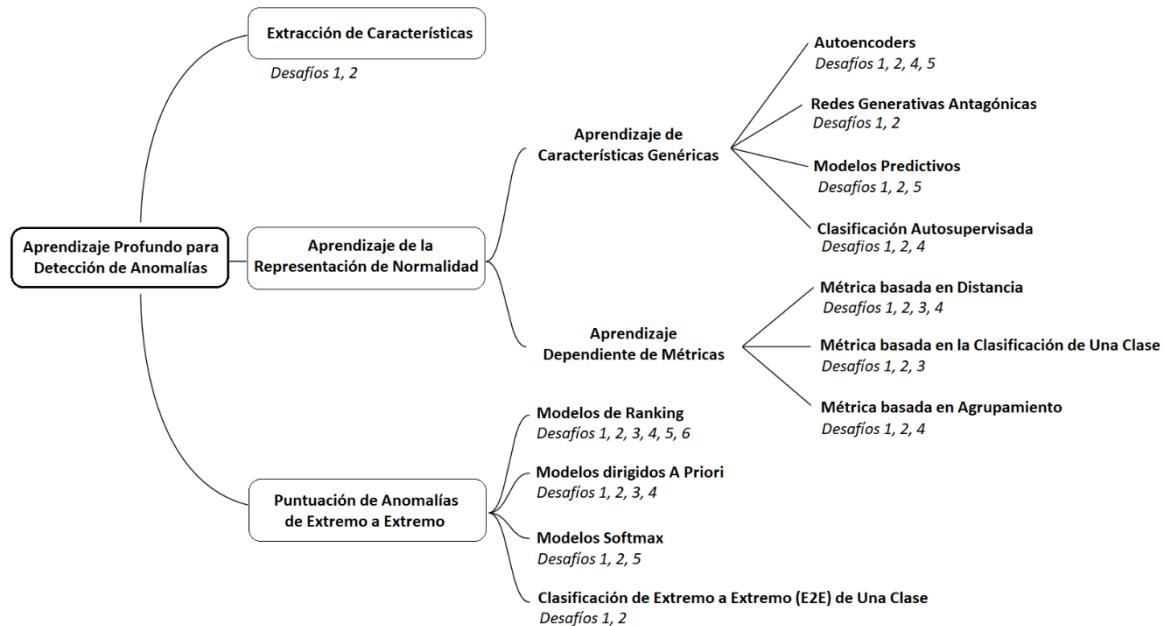


Figura 2.7: Taxonomía de las técnicas actuales de detección de anomalías. Adaptado de [11].

Categoría 1. Extracción de Características (Feature Extraction)

El objetivo de los métodos pertenecientes a esta categoría es usar el aprendizaje profundo para reducir la dimensionalidad de los datos y así exponer más fácilmente las anomalías. Es decir, en este caso, la red neuronal empleada no clasifica las instancias, simplemente transforma los datos, que se usarán como fuente de información para otro método de detección independiente.

Según se menciona en [11], las principales ventajas de estos métodos son: **(i)** el aprendizaje profundo demuestra mayor capacidad que los métodos tradicionales, como el PCA (*principal component analysis*) o la proyección aleatoria (en inglés, *random projection*), a la hora de extraer semántica y relaciones no lineales entre características; **(ii)** existen varios modelos de redes neuronales profundas previamente entrenados para reducir dimensionalidad, lo cual facilita la tarea de clasificación.

Por otra parte, la principal desventaja es que los modelos previamente entrenados están limitados a tipos de datos específicos, es decir, no son modelos versátiles.

Categoría 2. Aprendizaje de la representación de Normalidad (*Feature representation of Normality*)

Los métodos pertenecientes a esta categoría, a diferencia de la anterior, combinan la extracción de características y la puntuación de las anomalías. Según la taxonomía anterior, estos métodos se dividen en dos subgrupos: aprendizaje de características genéricas y aprendizaje dependiente de métricas.

Categoría 2.1. Aprendizaje de Características Genéricas (*generic normality feature learning*)

Los métodos en esta categoría aprenden las características de los datos usando una función objetivo genérica, es decir, una función que no ha sido diseñada específicamente para detectar anomalías. Los métodos adoptados se pueden clasificar en cuatro grupos: reconstrucción de datos, modelos generativos, modelos predictivos y clasificación autosupervisada [11].

Reconstrucción de Datos (Autoencoders): los Autoencoders (AE) son redes neuronales que copian su entrada en su salida. Para ello, aprenden a comprimir y descomprimir los datos en una representación intermedia de dimensionalidad menor a la original. Se usan frecuentemente para reducir la dimensionalidad y eliminar el ruido en los datos [12].

El uso de esta técnica en la detección de anomalías se basa en la premisa de que las instancias de datos normales se pueden reconstruir mejor que las anómalas a partir de la representación intermedia, debido a que la red está forzada a aprender regularidades en los datos para minimizar el error de reconstrucción [11].

La principal ventaja de los AE es que se pueden adaptar a una gran variedad de datos y no requieren que estos estén etiquetados. Por otro lado, la principal desventaja es que las representaciones intermedias son un resumen genérico de las regularidades de los datos, así que no están optimizadas para detectar irregularidades.

Redes Generativas Antagónicas (GAN): este tipo de redes pertenecen al conjunto de modelos generativos y son capaces de producir contenido, como, por ejemplo, texto, imágenes, música, etc. [13]. En la detección de anomalías, cuando una GAN recibe una instancia de datos x , intentará generar una instancia z a partir de la representación latente que ha aprendido, de modo que z y x sean lo más similares posibles.

Esta técnica en la detección de anomalías se basa en la premisa de que las instancias de datos normales se pueden generar mejor que las instancias anómalas a partir de la representación latente que aprende la GAN [11].

La principal ventaja de este método es que las GAN tienen una gran capacidad a la hora de generar instancias realistas, lo cual mejora la detección de instancias anómalas, porque estarán mal generadas. Por otro lado, la principal desventaja es que el entrenamiento de la red puede sufrir varios problemas, por ejemplo, las instancias generadas pueden estar fuera del área donde se concentran las instancias normales (es decir, se producen *outliers*), la red puede no converger, etc. [11].

Modelos Predictivos: estos métodos intentan predecir la instancia actual a partir de las anteriores usando una ventana temporal como contexto. Las instancias se tratan como elementos de una secuencia, por ejemplo, fotogramas de video en una secuencia de video. El error en la predicción se interpreta como la puntuación de la anomalía [11].

La principal ventaja de este método es que existen varias técnicas que pueden ser adaptadas para implementarlo (por ejemplo, regresión, redes LSTM, autoencoders, etc.). La principal desventaja, es que este método está limitado a secuencias de datos y la predicción puede ser computacionalmente compleja de calcular [11].

Clasificación Autosupervisada: este enfoque se basa en el análisis de características cruzadas (*cross-feature analysis*). Se evalúa la normalidad de una instancia por su consistencia con un conjunto de modelos predictivos que han aprendido a predecir una característica en base al valor de las otras. La consistencia de una instancia se puede medir, por ejemplo, usando el promedio de predicciones correctas según la predicción de todas las características [11].

Categoría 2.2. Aprendizaje Dependiente de Métricas (*anomaly measure-dependent feature learning*)

Los métodos en esta categoría aprenden las características de los datos usando una función objetivo que ha sido diseñada específicamente para detectar anomalías. Las métricas más populares son: métrica basada en distancia, basada en la clasificación de una clase y basada en agrupamiento (*clustering*) [11].

Métrica basada en Distancia (*distance-based measure*): existen varias funciones para calcular distancias, algunas de ellas son: los k vecinos más próximos, el promedio de los k vecinos más próximos, distancia relativa, etc. Una limitación de estos métodos es que no funcionan bien cuando los datos tienen alta dimensionalidad. Este método se basa en la premisa de que las anomalías están situadas lejos de sus vecinos, mientras que las instancias normales están cerca [11].

La principal ventaja de estos métodos es que funcionan bien con datos de baja dimensionalidad y funcionan mejor que los enfoques tradicionales con datos de alta dimensionalidad. La principal desventaja es que involucra extensos cálculos numéricos [11].

Métrica basada en la Clasificación de Una Clase (*one-class classification-based measure*): la clasificación de una clase consiste en aprender a distinguir si una instancia se ajusta a la descripción de los datos. Este método se basa en la premisa de que todas las instancias normales pertenecen a una clase que puede ser aprendida por un modelo compacto, al que las anomalías no se ajustan. La mayoría de los modelos en esta categoría se inspiran en las SVM (*support vector machines*), los modelos más famosos son SVDD (*support vector data description*) y *one-class* SVM [11].

La principal ventaja de este método es que el usuario no tiene que elegir manualmente una función *kernel* como en los enfoques tradicionales. La principal desventaja es que estos modelos pueden ser ineficaces cuando la clase normal tiene una distribución compleja [11].

Métrica basada en Agrupamiento (*clustering-based measure*): en este método, el objetivo es aprender una representación de los datos de forma que las anomalías se desvén claramente de los *clusters* en el nuevo espacio aprendido. Se basa en la premisa de que las instancias normales tienen una mayor adherencia⁵ a los *clusters* que las

⁵ En este contexto, la **adherencia** es la distancia promedio existente entre una instancia y el resto de las instancias del *cluster*. Cuanto menor sea el promedio, más adherencia tiene la instancia al *cluster*.

anomalías. Típicamente, este enfoque se compone de dos pasos: en el primer paso (*forward pass*) se realiza el agrupamiento y en el siguiente paso (*backward pass*) se usa la asignación de *cluster* como pseudoetiqueta. Este método requiere que el conjunto de entrenamiento contenga únicamente instancias normales [11].

La principal ventaja de este método es que, comparados con los métodos tradicionales, pueden aprender representaciones optimizadas que ayudan a detectar las anomalías con más facilidad que en los datos originales. La principal desventaja es que el proceso de agrupamiento puede estar sesgado si los datos están contaminados por anomalías, ya que la representación de los grupos de normalidad es menos efectiva [11].

Categoría 3. Puntuación de anomalías de Extremo a Extremo (*End-to-end anomaly score learning*)

En esta categoría, el objetivo es aprender a puntuar anomalías usando el aprendizaje de extremo a extremo⁶ (en inglés, *End-to-end o E2E*). En este contexto, puntuar significa usar un número para determinar si una instancia es una anomalía o no.

A diferencia de los métodos dependientes de métricas, en este caso, la propia red neuronal aprenderá a calcular las puntuaciones, en lugar de apoyarse en funciones preexistentes como las métricas basadas en distancia, basadas en agrupación (*clustering*), etc. Existen cuatro enfoques principales dentro de esta categoría: modelos de ranking, dirigidos a priori, softmax y clasificación E2E de una clase.

Modelos de Ranking (*ranking models*): en estos métodos se asume la existencia de una variable ordinal que captura alguna anomalía en los datos. En concreto, se asume que existe una variable observable $y = \{c_1, c_2\}$ siendo $c_1 > c_2$, de modo que $y_x = c_1$ cuando la instancia x es una anomalía e $y_x = c_2$ cuando la instancia es normal. La red E2E aprende a optimizar las puntuaciones de modo que, si una entrada muestra un comportamiento similar al anómalo, se genera una puntuación grande, muy cercana a c_1 , y en caso contrario se genera una puntuación pequeña, muy cercana a c_2 .

⁶ En ocasiones, existen sistemas de aprendizaje que requieren múltiples etapas de procesamiento. El **aprendizaje de extremo a extremo (E2E)** consiste en sustituir esas etapas por una sola red neuronal. Por ejemplo, en el reconocimiento de voz, tradicionalmente, se necesitaban varias etapas de procesamiento. Primero se extraían características del audio usando el algoritmo MFCC, a continuación, se aplicaba un algoritmo de aprendizaje automático para encontrar los fonemas del audio y, por último, se concatenaban los fonemas para obtener palabras y luego las palabras para obtener la transcripción del audio completo. Usando **E2E**, todo ese proceso se sustituye por una red neuronal profunda [32].

La principal ventaja de estos métodos es que, en general, están desacoplados de la definición de la anomalía, porque se basan en el orden ($c_1 > c_2$) existente entre las instancias anómalas y las normales [11].

Por otra parte, la principal desventaja es que es necesario disponer de anomalías etiquetadas, lo cual no siempre es posible. Además, existe la posibilidad de que el modelo no sea capaz de detectar aquellas anomalías que muestren unas características diferentes a las que poseen las muestras etiquetadas [11].

Modelos dirigidos A Priori (*prior-driven models*): este enfoque utiliza una distribución de probabilidad a priori (*prior distribution*⁷) para dirigir el aprendizaje. Este modelo asume que la distribución seleccionada captura la normalidad o anormalidad subyacente de los datos.

La principal ventaja de estos métodos es que proporcionan un *framework* flexible para incorporar diferentes distribuciones de probabilidad y se pueden adaptar diferentes tipos de redes Bayesianas para detectar anomalías (*Bayesian deep learning*) [11].

La principal desventaja es que los modelos son menos efectivos si la distribución de probabilidad no se ajusta bien a los datos reales [11].

Modelos Softmax (*softmax models*): este enfoque se basa en el aprendizaje de la probabilidad de que se produzca un evento en el conjunto de entrenamiento. Desde el punto de vista probabilístico, las anomalías son eventos de baja probabilidad (poco frecuentes), mientras que los datos normales son eventos de alta probabilidad (muy frecuentes). La negativa de la probabilidad del evento se puede definir como la puntuación de la anomalía [11].

Clasificación E2E de Una Clase (*E2E one-class classification*): al igual que el método mencionado en la **Categoría 2.2. Aprendizaje Dependiente de Métricas** (anomaly measure-dependent feature learning), el objetivo es en entrenar un clasificador de una clase que aprenda a distinguir si una instancia se ajusta a la descripción de los datos. A diferencia del método descrito anteriormente, aquí no se usan SVMs; la red aprende directamente a discriminar una de las clases usando el enfoque de las redes generativas antagónicas (GAN).

⁷ “A **prior distribution** represents your belief about the true value of a parameter. It’s your “best guess.” One you’ve done a few observations; you recalculate with new evidence to get the posterior distribution.” [34]

La clave de este método es entrenar dos redes neuronales profundas, la primera será entrenada para distinguir las instancias normales de las anomalías y la segunda red se encargará de generar anomalías sintéticas. En este método hay dos premisas: la primera es que se pueden sintetizar anomalías y la segunda es que la representación de normalidad puede ser aprendida. Las diferencias entre las GAN mencionadas en la [**Categoría 2.1. Aprendizaje de Características Genéricas**](#) (generic normality feature learning) y las de este apartado, son [11]:

- El objetivo de las primeras es aprender a generar instancias normales y usan la diferencia entre el valor original y el generado como puntuación de anomalía.
- El objetivo de estas es aprender a discriminar entre las instancias normales y las anómalas generadas.

Las desventajas de este método son las siguientes [11]:

- Es difícil garantizar que las anomalías generadas representen a las reales.
- El rendimiento del clasificador puede fluctuar drásticamente entre los pasos de entrenamiento debido a que el generador de anomalías puede crear instancias de diferente calidad.
- Este método está limitado a escenarios de aprendizaje semisupervisado.



Capítulo 3. Metodología de Trabajo

3.1. Metodología de Desarrollo

Este proyecto se ha desarrollado usando una metodología iterativa incremental combinada con la metodología **Kanban**.

La principal característica del desarrollo incremental es la planificación del proyecto en diversos bloques temporales (iteraciones), en los que se van añadiendo funcionalidades hasta que se finaliza. En este caso, cada iteración se planificó de forma flexible debido a mis problemas de disponibilidad horaria por motivos laborales y académicos. Al final de cada iteración tenía lugar una reunión para validar el trabajo realizado y fijar objetivos para la siguiente.

El método **Kanban** [14] es un marco de trabajo basado en un tablero donde se representa visualmente el estado de cada tarea del proyecto. El tablero básico tiene un flujo de trabajo de tres estados: por hacer (*to do*), en curso (*in progress*) y terminado (*done*). Sin embargo, dicho flujo se puede modificar para ajustarse a las necesidades del equipo. Para crear el tablero de este proyecto, se usó la herramienta **Trello** y se decidió trabajar con cinco estados (ver *Figura 3.1*), en lugar de tres, para organizar mejor las tareas:

- **To Do:** tareas pendientes de realizar y que no se han incluido en la iteración actual.
- **Sprint To Do:** tareas pendientes de realizar en la iteración actual.
- **In Progress:** tareas que están realizándose actualmente.
- **To Verify:** tareas finalizadas, pero pendientes de revisar.
- **Done:** tareas finalizadas y revisadas.

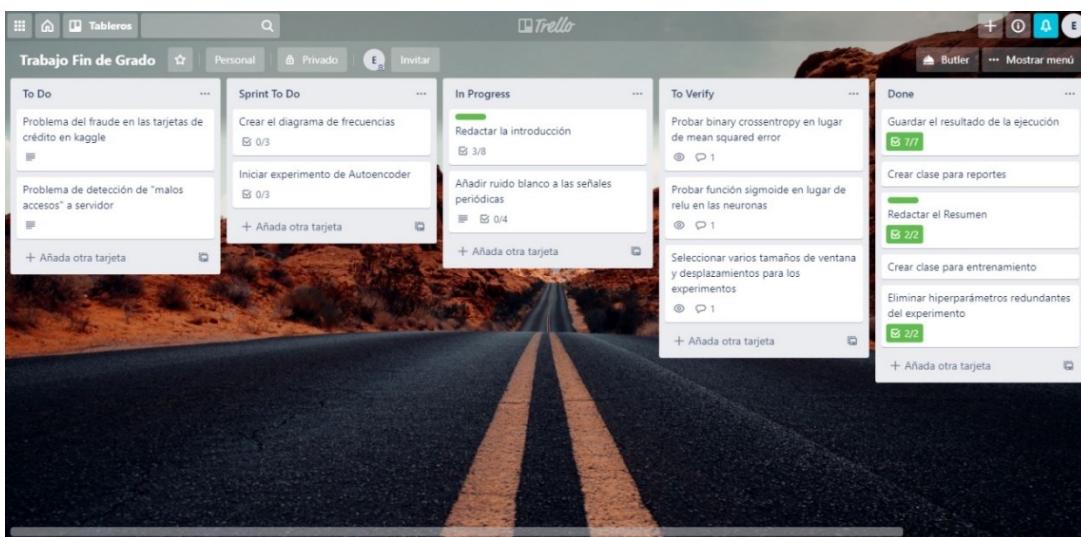


Figura 3.1: Tablero Kanban en Trello.

3.2. Entorno de Desarrollo

En este trabajo de fin de grado se ha usado **Python 3** como lenguaje de programación, ya que dispone de varias herramientas y paquetes relacionados con la inteligencia artificial y la ciencia de datos. El código de Python se ejecutó usando **Jupyter Notebook**, un entorno interactivo que permite desarrollar código de manera dinámica y que se ejecuta como una aplicación cliente-servidor. Los archivos creados en Jupyter se llaman cuadernos y permiten combinar el procesamiento de texto con la ejecución de código. Esta herramienta es particularmente útil porque permite conservar la salida de la última ejecución (texto, gráficos, etc.) aunque se cierre la aplicación y se apague el ordenador. Para escribir las clases y funciones usadas en los cuadernos, se usó **PyCharm** como IDE.

Para desplegar Jupyter en local, se utilizó **Anaconda**. Se trata de una distribución de Python que permite gestionar paquetes y diferentes entornos de ejecución virtuales, es decir, entornos independientes entre sí e independientes del sistema operativo. Esto significa que se pueden instalar varias versiones de Python y varias versiones de librerías sin que interfieran unas con otras.

Se decidió usar el servicio **Google Colaboratory** (también llamado **Google Colab** o simplemente **Colab**) para el entrenamiento de las redes neuronales debido al elevado tiempo que requería entrenarlas en local. Se trata de un servicio gratuito, basado en la nube, que no requiere configuración y que permite ejecutar cuadernos como los de Jupyter haciendo uso de GPUs, lo cual reduce considerablemente el tiempo de ejecución. Colab usa máquinas virtuales para ejecutar los cuadernos; tienen un tiempo máximo de ejecución de 12 horas, aproximadamente, y una vez que se desconectan, se pierden todos los archivos que había en su interior. Para evitar la pérdida accidental de datos (redes

neuronales entrenadas y métricas sobre el proceso de aprendizaje) por desconexión, se conectó Colab con el servicio de alojamiento de archivos de **Google Drive**. Esto también facilita la descarga, ya que en Colab sólo se pueden descargar ficheros de uno en uno.

3.3. Librerías Utilizadas

Las librerías usadas en la elaboración de este estudio se pueden agrupar según su funcionalidad en tres clases: cálculos numéricos, inteligencia artificial y tratamiento y visualización de datos.

Respecto al cálculo numérico, **NumPy**⁸ es actualmente el paquete fundamental para la computación científica en Python. Se trata de una librería que proporciona una gran variedad de rutinas para operaciones rápidas en matrices, incluidas las operaciones matemáticas, lógicas, manipulación de dimensiones, operaciones estadísticas básicas, álgebra lineal básica, etc. Es un proyecto de código abierto creado en el año 2005 y de uso gratuito.

En cuanto a la inteligencia artificial, se usaron **Keras**⁹ y **Scikit-learn**¹⁰ de forma conjunta. **Keras** es una capa de software diseñada para personas, es decir, diseñada para reducir la carga cognitiva a la hora de desarrollar modelos de aprendizaje profundo. Trabaja sobre Tensorflow (una librería de código abierto desarrollada por Google), cuenta con una extensa documentación y guías para desarrolladores. A partir de la versión 2.4.0, se desarrolla dentro del proyecto Tensorflow y deja de tener soporte con Theano. Por otro lado, **Scikit-learn** es una librería de aprendizaje automático de código abierto que soporta tanto el aprendizaje supervisado como el no supervisado. También proporciona varias herramientas para el ajuste de modelos, el preprocesamiento de datos, la selección y evaluación de modelos y otras utilidades.

En cuanto al tratamiento y visualización de datos, se usaron **Matplotlib**¹¹, **Pandas**¹², y **Wave**¹³. **Matplotlib** es una librería de código abierto para crear visualizaciones gráficas estáticas, animadas e interactivas en Python y es compatible con NumPy. **Pandas** es una librería de código abierto para la manipulación y análisis de datos. Es compatible con diferentes formatos, como CSV, texto plano, Microsoft Excel, bases de datos SQL y HDF5. El módulo **Wave** pertenece a la librería estándar de Python y proporciona una interfaz para tratar archivos de sonido en formato WAV.

⁸ <https://numpy.org/>

⁹ <https://keras.io/>

¹⁰ <https://scikit-learn.org/stable/>

¹¹ <https://matplotlib.org/>

¹² <https://pandas.pydata.org/>

¹³ <https://docs.python.org/3/library/wave.html>

3.4. Evaluación de las Redes Neuronales

3.4.1. Matriz de Confusión

En los problemas de clasificación binaria existe un resultado conocido como matriz de confusión, que permite visualizar la calidad de un algoritmo de clasificación al realizar una predicción y asignar a cada muestra una etiqueta de clase (supongamos que son las clases *positiva* y *negativa*). Para ello, se representa en una tabla de 2×2 la cantidad de veces que se han producido los siguientes cuatro casos (**Tabla 3.1**) [15]:

- Las muestras *positivas* son etiquetadas correctamente como *positivas*, lo que se denomina **verdaderos positivos (TP)**
- Las muestras *positivas* son etiquetadas erróneamente como *negativas*, lo que se denomina **falsos negativos (FN)**
- Las muestras *negativas* son etiquetadas erróneamente como *positivas*, lo que se denomina **falsos positivos (FP)**
- Las muestras *negativas* son etiquetadas correctamente como *negativas*, lo que se denomina **verdaderos negativos (TN)**

		Predicción Positiva	Predicción Negativa
Muestra Positiva	Verdaderos Positivos (TP)	Falsos Negativos (FN)	
	Falsos Positivos (FP)	Verdaderos Negativos (TN)	
Muestra Negativa			

Tabla 3.1: Estructura de una matriz de confusión para un problema de clasificación binaria.

Claramente, la suma de TP + FN representa la cantidad total de muestras positivas y TN + FP representa la cantidad total de muestras negativas. Cuanto mayores son los valores en la diagonal principal, mejor funciona el clasificador. La matriz de confusión perfecta es aquella en la que todos los elementos fuera de la diagonal principal están igualados a 0.

La matriz de confusión también se puede usar en problemas de clasificación multiclas, en cuyo caso, los elementos de la diagonal principal representan el número de muestras para las que la etiqueta predicha es igual a la verdadera, mientras que los elementos no pertenecientes a la diagonal principal son los que han sido mal etiquetados por el clasificador.

En la **Figura 3.2** se muestra un ejemplo de matriz de confusión para evaluar la calidad de un clasificador sobre el conjunto de datos de la Flor Iris¹⁴, que contiene tres clases de flor: *setosa*, *versicolor* y *virginica*. La figura muestra la matriz de confusión sin y con normalización (izquierda y derecha, respectivamente). La normalización puede ser interesante para tener una interpretación más visual sobre qué clase se está clasificando mal y qué proporción de errores se produce. En la imagen normalizada, se puede ver que el 62% de las muestras *versicolor* son clasificadas correctamente, mientras que el 38% restante son clasificadas erróneamente como *virginica*. Las otras dos clases tienen una clasificación perfecta.

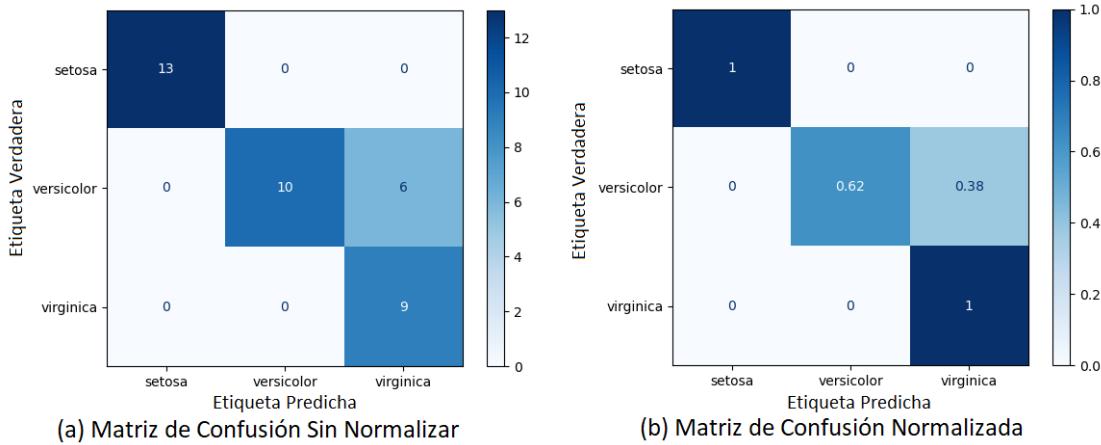


Figura 3.2: Matrices de confusión de un clasificador usando el conjunto de datos Iris. Adaptado de [16].

Tal y como se explica en [15], dado que analizar las cuatro categorías de la matriz (TP, FN, FP, TN) por separado para varios clasificadores llevaría mucho tiempo, existen algunos índices estadísticos capaces de describir la calidad de una predicción, intentando expresar con una sola cifra la estructura de la matriz. Un conjunto de estas funciones actúa en función de la clase, es decir, involucran sólo las dos entradas de la matriz que pertenecen a la misma fila o columna. Estas métricas [ver las ecuaciones en (3.1)] no se pueden considerar totalmente informativas porque sólo utilizan dos categorías de la matriz de confusión.

¹⁴ <http://archive.ics.uci.edu/ml/datasets/Iris>

$$\begin{aligned}
 \text{Sensibilidad, Recall, Tasa de Verdaderos Positivos (TPR)} &= \frac{TP}{TP + FN} \\
 \text{Especificidad, Tasa de Verdaderos Negativos (TNR)} &= \frac{TN}{TN + FP} \\
 \text{Precision, Valor Predictivo Positivo (PPV)} &= \frac{TP}{TP + FP} \\
 \text{Valor Predictivo Negativo (NPV)} &= \frac{TN}{TN + FN} \quad (3.1) \\
 \text{Tasa de Falsos Positivos (FPR), Fallout} &= \frac{FP}{FP + TN} \\
 \text{Tasa de Falsos Negativos (FNR), Tasa de Fallo} &= \frac{FN}{FN + TP} \\
 \text{Tasa de Falsos Descubrimientos (FDR)} &= \frac{FP}{FP + TP}
 \end{aligned}$$

Una métrica ampliamente aceptada, ya que involucra todas las categorías de la matriz, es la exactitud [en inglés, *accuracy*, ecuación (3.2)]. Representa la relación entre las instancias correctamente predichas y todas las instancias del conjunto de datos. Un clasificador perfecto es aquel que tiene *exactitud* = 1, mientras que un clasificador que siempre se equivoca tiene *exactitud* = 0.

$$\text{exactitud} = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.2)$$

Sin embargo, *esta métrica no proporciona resultados fiables cuando el conjunto de datos está muy desequilibrado* [15], ya que proporciona valores altos para modelos que solamente predicen la clase más frecuente.

3.4.2. Críticas hacia el Método de Evaluación Tradicional

En el último año se ha publicado un artículo científico llamado “*Towards a Rigorous Evaluation of Time-series Anomaly Detection*” [17], elaborado por miembros de tres universidades de Corea del Sur: la Universidad Nacional de Ciencia y Tecnología de Seúl, la Universidad Nacional de Seúl y la Universidad Nacional de Kangwon.

En dicho artículo se cuestiona el uso de la métrica F1 en combinación con el protocolo de evaluación de **ajuste de puntos** (en inglés, *points adjustment* o PA) para evaluar el rendimiento de los métodos de detección de anomalías en series temporales (en inglés, *time-series anomaly detection* o TAD). Lo cual es una práctica extendida y es la que se estaba usando para este estudio cuando leí el artículo.

La métrica F1¹⁵ se calcula como la media armónica de dos índices estadísticos de la matriz de confusión: la precisión y el recall [ecuaciones en (3.1)]. Toma valores en el rango [0, 1], siendo el 1 el clasificador perfecto y el 0 el erróneo. La ecuación de F1 se puede ver en (3.3), donde se ha simplificado para mostrarla directamente en función de los términos de la matriz de confusión.

$$\begin{aligned}
 F1 &= 2 \cdot \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}} = 2 \cdot \frac{\frac{TP}{TP + FP} \cdot \frac{TP}{TP + FN}}{\frac{TP}{TP + FP} + \frac{TP}{TP + FN}} \\
 &= \frac{TP}{TP + \frac{1}{2} \cdot (FP + FN)}
 \end{aligned} \tag{3.3}$$

El ajuste de puntos (PA) funciona del siguiente modo: si al menos un punto en un segmento de anomalía contiguo se detecta como anomalía, se considera que todo el segmento se ha detectado correctamente. Este protocolo se usa bajo la premisa de que un solo positivo dentro de un período de anomalía es suficiente para tomar medidas en un entorno de producción.

En el artículo se describen los siguientes problemas:

- a) El ajuste de puntos puede sobreestimar la capacidad de un modelo, porque incrementa el valor de TP, decrementa el de FN, pero no varía el de FP.
- b) Usar la métrica F1 sin ajuste de puntos puede subestimar la capacidad de un modelo.
- c) Se critica que se use el valor de F1 directamente sin que se haya intentado establecer una comparación relativa con un valor base de referencia. Por ejemplo [17], si la precisión de un clasificador binario es 50%, no es muy diferente a hacer una suposición aleatoria, a pesar de que 50 es un número aparentemente grande.

En el artículo se proponen las siguientes soluciones: (I) usar un protocolo de evaluación alternativo a PA, que ellos han denominado PA%K, donde K es un umbral añadido al protocolo PA y que debe ser seleccionado manualmente entre 0 y 100 según conocimiento previo; (II) evaluar el rendimiento de modelos sin entrenar y usar ese valor como referencia base para evaluar los modelos entrenados, de modo que, si el rendimiento del modelo no supera el valor de referencia, la arquitectura debería ser reexaminada.

¹⁵ https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html



Como consecuencia de este artículo, se implementa la proposición (II) como parte de la evaluación de resultados.

Debido al elevado tiempo que se requiere para la ejecución de los experimentos (se explica en el [Capítulo 5](#)), se descarta la proposición (I) porque añadiría un hiperparámetro a la red y eso subiría aún más el tiempo de ejecución. En su lugar, se sustituye la métrica F1 por la descrita en la siguiente sección.

3.4.3. Métrica elegida: Coeficiente de Correlación de Matthews (MCC)

Por los motivos expuestos en las secciones anteriores, la métrica elegida para evaluar los modelos será el ***Coeficiente de Correlación de Matthews***.

Esta es una métrica que no se ve afectada por el desequilibrio de clases, ya que es el único índice de clasificación binaria que genera una puntuación alta sólo si el clasificador fue capaz de predecir correctamente la mayoría de los casos de muestras positivas y la mayoría de los casos de muestras negativas. La métrica se puede definir en términos de la matriz de confusión según la ecuación (3.4); toma valores en el intervalo $[-1, +1]$, siendo -1 el valor correspondiente a un clasificador que siempre se equivoca y $+1$ el clasificador perfecto. El valor 0 se corresponde con un clasificador aleatorio [15].

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \quad (3.4)$$

Un problema potencial de esta métrica es que el resultado se vuelve indefinido cuando toda una fila o columna de la matriz de confusión es cero, es decir, cuando la matriz es una de las siguientes: $\begin{pmatrix} a & 0 \\ b & 0 \end{pmatrix}$, $\begin{pmatrix} a & b \\ 0 & 0 \end{pmatrix}$, $\begin{pmatrix} 0 & 0 \\ b & a \end{pmatrix}$ o $\begin{pmatrix} 0 & b \\ 0 & a \end{pmatrix}$, siendo a y b mayores que 0 . En estos casos, la ecuación toma forma de indeterminación del tipo $0/0$ (cero dividido por cero). En [15] se propone una aproximación matemática para solucionar este problema y hacer que en esos casos MCC valga 0 , pero en la librería de scikit-learn lo solucionan directamente a nivel de programación comprobando si el resultado de la operación es NaN (*Not a Number*) y devolviendo un 0 en ese caso¹⁶.

¹⁶ https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html

3.5. Metodología de Entrenamiento

3.5.1. Preparación de los Datos

El objetivo de las redes neuronales es aprender una función matemática que, dada una entrada de datos, produzca una salida. Por ello, la escala de los datos usados en el entrenamiento es muy importante.

Las variables de entrada pueden tener diferentes unidades (por ejemplo: kilómetros, horas, metros cuadrados, importe monetario, etc.) y diferentes escalas. Las diferencias en las escalas de las variables de entrada pueden aumentar la dificultad del problema que se está modelando. Un ejemplo de ello es que las entradas grandes pueden dar lugar a un modelo que aprenda pesos grandes, haciéndolo susceptible de sufrir un mal rendimiento durante el aprendizaje y tener alta sensibilidad a ciertas variables, lo que se traduce en un mayor error de generalización [18].

Por otro lado, una variable objetivo (la salida de la red) con una gran dispersión de valores, puede originar grandes gradientes durante el aprendizaje, lo cual hace que las ponderaciones de los pesos cambien drásticamente entre un paso del entrenamiento y el siguiente, haciendo que el proceso sea inestable y, en ocasiones, impidiendo la convergencia [18].

Para evitar estos problemas, la práctica más común es ajustar los datos de la red de la siguiente forma [18]:

- Respecto a la entrada, la regla general es escalar cada atributo X para que se ajuste a un rango de valores pequeños, por ejemplo, valores comprendidos en el rango $[0, 1]$ o $[-1, +1]$, o estandarizarlo para que tenga media 0 y varianza 1.
- Respecto a la salida, lo ideal es asegurarse de que la escala de la variable coincide con la escala de la función de activación usada en la capa de salida de la red. Por ejemplo, si la salida es un número real, lo mejor es usar una función de activación lineal.

3.5.2. Elección de Arquitectura de Red

Se conoce como arquitectura a la cantidad de capas y neuronas por capa que componen una red neuronal. Tal y como se describe en [19], no existe una forma analítica de calcular estos hiperparámetros, de modo que suele usarse alguno de los siguientes enfoques para configurarlos:

- **Experimentación:** consiste en probar diferentes arquitecturas sobre un conjunto de prueba. Este enfoque es la base de los siguientes.
- **Intuición:** se usa la experiencia previa en la construcción de modelos para elegir la arquitectura.
- **Tomar ideas:** consiste en buscar artículos sobre problemas similares al que se intenta resolver y probar las arquitecturas que usan en ellos.
- **Búsqueda:** consiste en programar una búsqueda automática para probar diferentes arquitecturas.

Las estrategias de búsqueda más famosas son:

- **Búsqueda de Cuadrícula o Grid Search:** consiste en definir una serie de configuraciones (combinaciones de hiperparámetros) y probar un modelo con cada una.
- **Búsqueda Aleatoria:** se define una serie de hiperparámetros y las configuraciones se seleccionan combinándolos al azar entre ellos.
- **Búsqueda Heurística:** se realiza una búsqueda dirigida entre las distintas configuraciones posibles.
- **Búsqueda Exhaustiva:** consiste en probar todas las combinaciones de hiperparámetros posibles.



Tradicionalmente existe desacuerdo a la hora de describir la arquitectura de una red: hay quien considera que la capa de entrada no se debe contar porque carece de función de activación (tiene solamente los valores de entrada) [19]. De este modo, una red que tuviera una capa de entrada, una capa oculta y una capa de salida se consideraría una red de dos capas.

En este estudio se usará esta convención.

3.5.3. Validación Cruzada (k-Fold Cross-Validation)

Aprender los parámetros de una función de predicción o clasificación y probarlos con los mismos datos es un error de metodología, debido a que el modelo repetiría las etiquetas de las muestras que acaba de ver y tendría una puntuación perfecta, pero no tendríamos evidencia de que fuera capaz de predecir algo útil sobre datos nuevos [20].

Esto se conoce como sobreajuste, explicado anteriormente en la Sección [2.1.2](#), justo antes de la definición de *dropout*. Para evitar esto, la práctica más extendida es dividir el conjunto de datos original en tres partes:

- **Conjunto de Entrenamiento:** contiene las muestras que usará el algoritmo de aprendizaje para aprender los parámetros del modelo.
- **Conjunto de Validación:** contiene las muestras que se usarán, al final de cada iteración de entrenamiento, para evaluar si el proceso de aprendizaje se está llevando a cabo correctamente.
- **Conjunto de Prueba:** contiene las muestras que se usarán para la evaluación final del modelo. Dichas muestras representan datos nuevos, es decir, que no han sido vistos durante el entrenamiento; esto nos permite determinar si el modelo ha conseguido aprender.

Sin embargo, al dividir los datos disponibles en tres conjuntos, se reduce drásticamente el número de muestras que pueden utilizarse para el aprendizaje y los resultados pueden depender en gran medida de una elección particular de conjuntos de entrenamiento y validación [20].

Una solución para este problema es un procedimiento llamado **validación cruzada** (en inglés, *cross-validation*), en el que se divide el conjunto de datos en dos partes: entrenamiento y prueba. A continuación, el conjunto de entrenamiento se divide en K conjuntos más pequeños y se realizan las siguientes acciones con cada uno de ellos [20]:

- se entrena un modelo usando $K - 1$ conjuntos como muestras de entrenamiento
- el modelo resultante se valida con el conjunto que no se usó para entrenar

En este caso, el rendimiento del modelo se calcula como la media aritmética entre los rendimientos de los K modelos entrenados en total. El [Algoritmo 3.1](#) muestra la forma genérica de implementar este método.

Algoritmo 3.1: K-FOLD CROSS-VALIDATION

Input: C – conjunto de datos de entrenamiento
 K – número de subconjuntos que se crearán
 F – función creadora del modelo

Output: M – media aritmética de las evaluaciones de los modelos

```

1: grupos ← dividir el conjunto  $C$  en  $K$  grupos
2: evaluaciones ← crear una lista para guardar las evaluaciones de los modelos
3: FOR EACH grupo IN grupos DO
4:   modelo ← crear un nuevo modelo usando  $F$ 
5:    $C'$  ← unir todos los grupos excepto grupo en un nuevo conjunto
6:   entrenar el modelo usando  $C'$ 
7:   evaluación ← evaluar el modelo usando grupo
8:   almacenar la evaluación del modelo en evaluaciones
9: END FOR
10: RETURN MEAN (evaluaciones)

```

Este procedimiento tiene un mayor coste computacional que el mencionado inicialmente, pero aprovecha mejor los datos disponibles y proporciona una visión global de lo buena que es la arquitectura elegida, ya que los modelos son diferentes entre sí debido a la inicialización aleatoria de los parámetros de cada uno en las distintas iteraciones.

La elección de K es típicamente 5 o 10, ya que se ha demostrado empíricamente que estos valores producen estimaciones de la tasa de error que no sufren ni un sesgo (*bias*) ni una varianza excesivamente elevados, aunque el valor $K = 10$ es el más común en la práctica [21]. En la **Figura 3.3** se muestra un ejemplo de división de un conjunto de datos para $K = 5$.

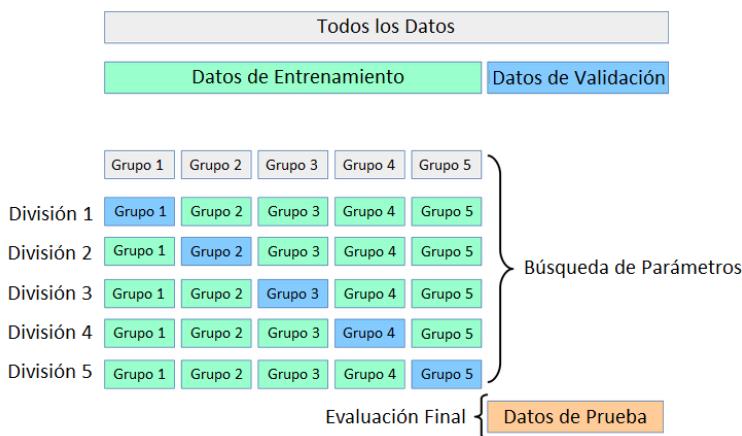


Figura 3.3: Diagrama de división de un conjunto de datos para un k -Fold con $k=5$. Adaptado de [20].

En cuanto a la forma de dividir el conjunto de datos entre entrenamiento y prueba, no existe ningún porcentaje óptimo, pero lo más común es usar una mayor cantidad de datos para entrenamiento. Los porcentajes de división más comunes en los ejemplos encontrados han sido: 80% entrenamiento y 20% pruebas o 67% entrenamiento y 33% pruebas.

3.5.4. Ajuste del Umbral (Threshold Moving)

Muchos algoritmos de aprendizaje automático son capaces de proporcionar una probabilidad (o puntuación) de pertenencia a una clase para un problema de clasificación. Esto proporciona una medida de la certeza o incertidumbre de una predicción, pero algunas tareas de clasificación necesitan convertir ese valor en una etiqueta de clase concreta.

Esto se consigue utilizando un parámetro denominado **umbral** (en inglés, **threshold**), de modo que todos los valores iguales o superiores al umbral se asignan a una clase y el resto de los valores se asignan a la otra. El valor por defecto del umbral es 0.5 para probabilidades o puntuaciones normalizadas en el rango entre 0 y 1. Por ejemplo, en un problema de clasificación binaria con etiquetas de clase 0 y 1, los valores menores que 0.5 se asignan a la clase 0 y los valores mayores o iguales se asignan a la clase 1 [22].

Sin embargo, para aquellos problemas de clasificación en los que existe un gran desequilibrio entre clases, como en el caso de la detección de anomalías, el umbral por defecto puede proporcionar un mal rendimiento. Por lo tanto, un enfoque simple y directo para mejorar el funcionamiento del clasificador es ajustar el umbral utilizado para asignar etiquetas de clase. A esta práctica se la conoce en inglés como **threshold-moving**, **threshold-tuning**, o **thresholding**.

El método **threshold moving** usa el conjunto de entrenamiento original para entrenar un modelo de clasificación y luego modifica el umbral de decisión de modo que las instancias de la clase minoritaria sean más fáciles de predecir correctamente [22].

El proceso consiste en entrenar el modelo usando el conjunto de entrenamiento y hacer predicciones usando el conjunto de prueba. Las predicciones se presentan en forma de probabilidades normalizadas o puntuaciones que se transforman en probabilidades normalizadas. A continuación, se prueban diferentes valores de umbral y se evalúan las etiquetas resultantes (clasificación final) utilizando la métrica de evaluación seleccionada. El umbral que consigue la mejor evaluación se adopta para el modelo cuando se hacen predicciones sobre nuevos datos en el futuro. El Algoritmo 3.2 resume este procedimiento [22].

Algoritmo 3.2: AJUSTE DEL UMBRAL (THRESHOLD MOVING)

Input: M – modelo entrenado con el conjunto de entrenamiento
 C – conjunto de prueba
 F – función con la métrica para evaluar la clasificación

Output: U – umbral escogido para hacer predicciones en nuevos datos

```

1: probabilidades  $\leftarrow$  aplicar el modelo  $M$  al conjunto  $C$ 
2: FOR EACH umbral IN umbrales DO
3:   predicciones  $\leftarrow$  convertir probabilidades en etiquetas usando umbral
4:   evaluación  $\leftarrow$  aplicar  $F$  a predicciones
5:   IF evaluación es la mejor evaluación THEN
6:      $U \leftarrow$  umbral
7:   END IF
8: END FOR
9: RETURN  $U$ 
```

Existen diferentes formas de implementar este algoritmo dependiendo de las circunstancias. Por ejemplo, se podrían probar todos los umbrales posibles entre 0 y 1 usando un incremento constante (por ejemplo 0.001).

En este caso se ha optado por una aproximación más eficiente: en lugar de probar todos los umbrales posibles, se obtendrán a partir de la lista de probabilidades, eliminando los valores duplicados. De este modo, si las probabilidades calculadas para un conjunto de prueba son 0.3, 0.4, 0.3 y 0.7, la lista de umbrales sería 0.3, 0.4 y 0.7.



Capítulo 4. Descripción de los Datos

4.1. Aspectos Generales

Para realizar este estudio, se decidió trabajar con series de datos unidimensionales generadas artificialmente. De este modo, se tiene un control absoluto sobre la cantidad de anomalías existentes, su distribución y se pueden manipular las series para crear variaciones. Las series utilizadas se pueden agrupar en dos categorías: series periódicas y series naturales.

Cada serie se guardó en un fichero de texto en formato **CSV**¹⁷ (del inglés *comma-separated values*) con dos columnas: *data*, que almacena los puntos de la serie y *label*, que indica si el punto es una instancia normal (*label* = 0) o una anomalía (*label* = 1).

4.2. Series Periódicas

En este caso, el objetivo era detectar **anomalías contextuales** en una serie dada. Para ello, se crearon dos series periódicas basadas en la ecuación [\(4.1\)](#) (función senoide), donde *A* es la amplitud de oscilación, *f* es la frecuencia de oscilación, *t* es el instante de tiempo y *φ* (*phi*) es la fase inicial.

$$y(t) = A \sin(2\pi ft + \varphi) \quad (4.1)$$

La primera señal, en adelante “**señal simple**”, se generó aplicando la ecuación anterior con los parámetros $A = 1$, $f = 1/40$, $\varphi = 0$ y t tomando valores en el intervalo semiabierto $[0, 12000]$ con $\Delta t = 1$. Esto da como resultado la señal presentada en la [Figura 4.1](#), que se repite con un periodo $T = 40$ (por simplicidad, se muestran sólo los tres primeros periodos).

La segunda señal, en adelante “**señal combinada**”, se generó realizando operaciones aritméticas con cuatro señales diferentes, evaluadas en el mismo intervalo que la señal simple. En concreto, se utilizó el sistema de ecuaciones descrito en [\(4.2\)](#), lo que da como

¹⁷ https://es.wikipedia.org/wiki/Valores_separados_por_comas

resultado la señal presentada en la **Figura 4.2**, que se repite con un periodo $T = 400$ (por simplicidad, se muestran sólo los dos primeros períodos).

$$\begin{aligned} y_a(t) &= 1.0 \sin\left(2\pi \frac{0.5}{40}t + 1.0\right) & y_b(t) &= 0.5 \sin\left(2\pi \frac{1.2}{40}t + 2.0\right) \\ y_c(t) &= 2.0 \sin\left(2\pi \frac{1.8}{40}t + 0.5\right) & y_d(t) &= 0.5 \sin\left(2\pi \frac{1.0}{40}t + 0.0\right) \end{aligned} \quad (4.2)$$

$$\text{combinada}(t) = y_a(t) \cdot y_b(t) \cdot y_c(t) + y_d(t)$$

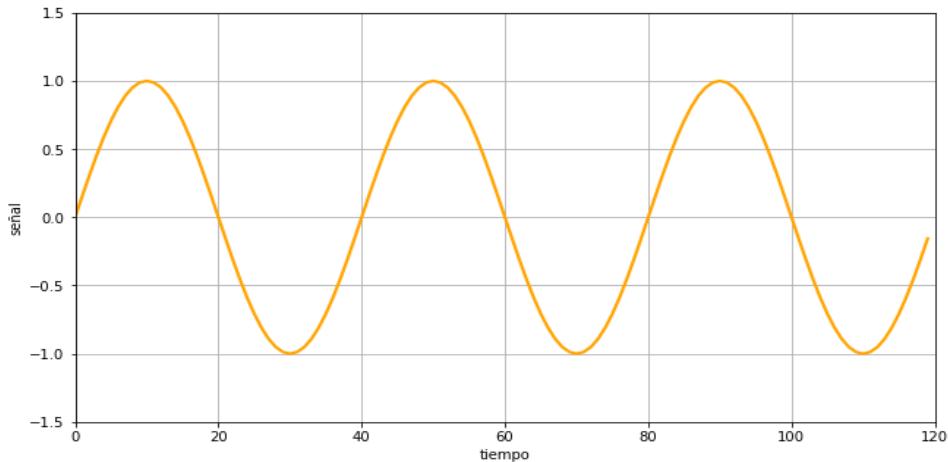


Figura 4.1: Gráfica con los tres primeros períodos de la señal simple ($T = 40$).

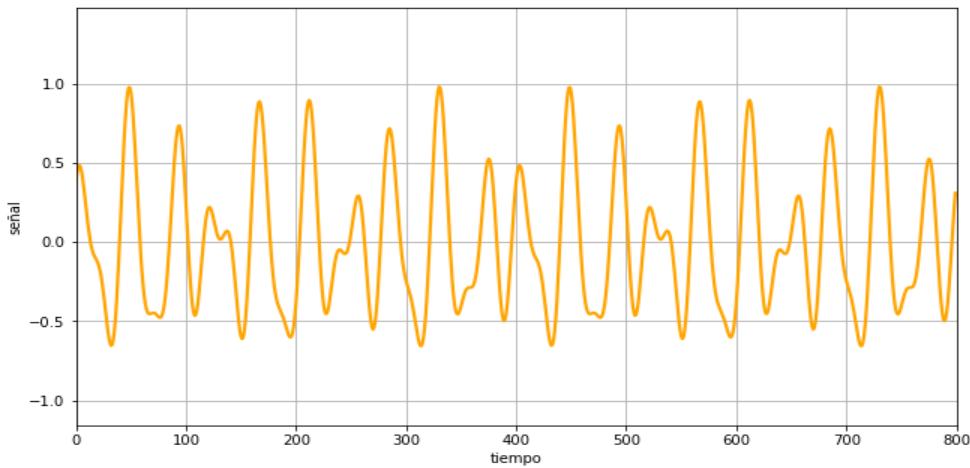


Figura 4.2: Gráfica con los dos primeros períodos de la señal combinada ($T = 400$).

A partir de cada señal, se crearon dos series con un 1.1% y un 2% de anomalías, respectivamente. Se escogieron porcentajes bajos porque las anomalías se caracterizan por ser poco frecuentes. Para crearlas, se seleccionaron posiciones aleatorias de la señal y se alteraron usando la ecuación (4.3), donde: p representa la posición seleccionada; $s'(p)$ representa la anomalía creada en esa posición; $s(p)$ representa el valor de la señal original en esa posición; $\cos(t)$ es el coseno del instante de tiempo usado para crear $s(p)$; ranf_1 y

`ranf2` representan la primera y la segunda ejecución de la función `ranf`, una función que devuelve un número decimal aleatorio en el intervalo [0, 1) y `randint` es una función que, dados un límite inferior y un límite superior, devuelve un número entero en el intervalo [*inferior*, *superior*). En la **Figura 4.3** se muestra un fragmento de la señal simple con 2% de anomalías como ejemplo.

$$s'(p) = \begin{cases} s(p) + \cos(t), & \text{si } ranf_1 \geq 0.5 \text{ y } ranf_2 > 0.4 \\ s(p) + \frac{\cos(t)}{randint(2,4)}, & \text{si } ranf_1 \geq 0.5 \text{ y } ranf_2 \leq 0.4 \\ s(p) - \cos(t), & \text{si } ranf_1 < 0.5 \text{ y } ranf_2 > 0.4 \\ s(p) - \frac{\cos(t)}{randint(4,7)}, & \text{si } ranf_1 < 0.5 \text{ y } ranf_2 \leq 0.4 \end{cases} \quad (4.3)$$

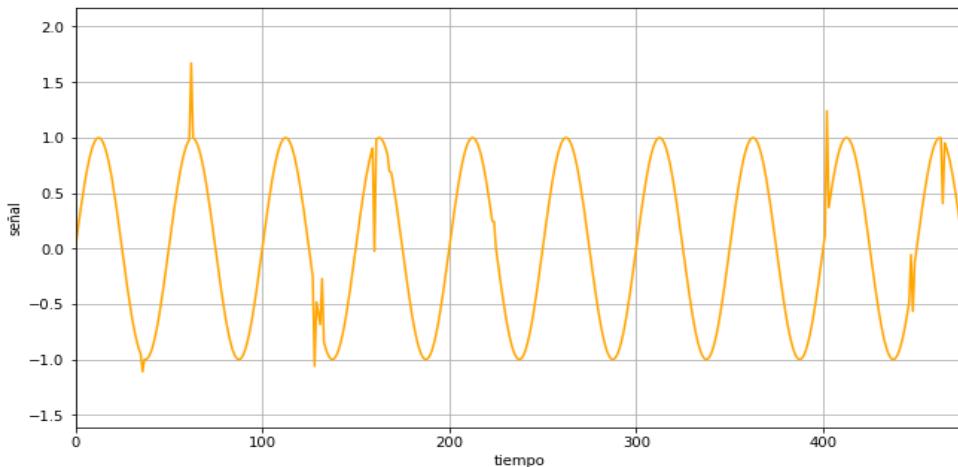


Figura 4.3: Primeros 475 puntos de la señal simple con 2% de anomalías.

Por último, a cada serie se le añadió ruido blanco mediante la ecuación (4.4) para comprobar cómo afecta a las redes neuronales. En la ecuación: *i* representa la posición de un elemento en la serie; $s'(i)$ es el nuevo valor para esa posición; $s(i)$ es el valor original; β (beta) representa el factor de ruido introducido y `uniform` es una función que proporciona elementos aleatorios siguiendo una distribución uniforme en un intervalo dado, es decir, cualquier valor contenido en ese intervalo tiene la misma probabilidad de ser devuelto por la función.

$$s'(i) = s(i) + \beta \cdot uniform(-1, 1) \quad (4.4)$$

Se usaron cinco factores de ruido $\beta = \{0, 0.025, 0.05, 0.1, 0.2\}$ para cada serie (el factor 0 es la serie original sin ruido), lo cual hace un total de 20 series periódicas disponibles para el estudio (2 señales base · 2 porcentajes de anomalías · 5 factores de ruido = 20 series). En la siguiente figura se muestra un fragmento de la señal simple con 2% de anomalías y factor 0.2 de ruido como ejemplo.

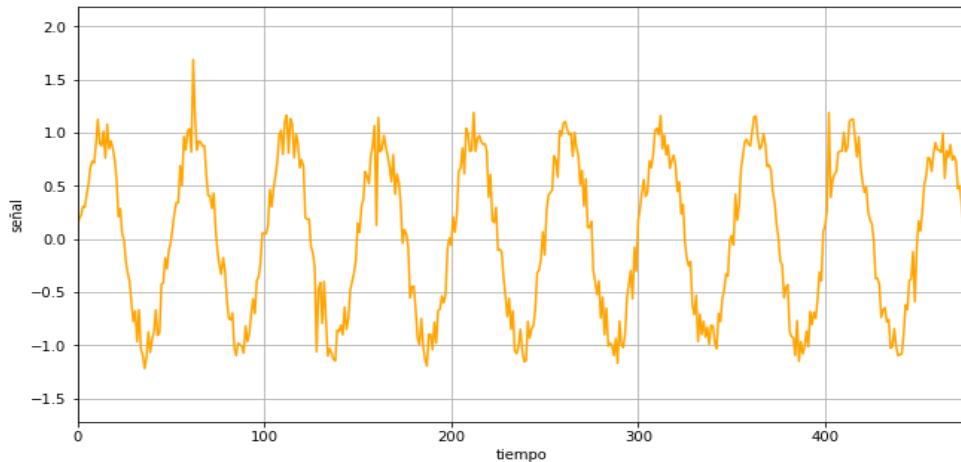


Figura 4.4: Primeros 475 puntos de la señal simple con 2% de anomalías y factor 0.2 de ruido.

4.3. Series Naturales

En este caso, el objetivo era obtener dos señales acústicas y contaminar una con fragmentos de la otra para detectar **anomalías colectivas**. Se extrajeron fragmentos de audio, de 21 segundos, de dos videos de YouTube: en el primero se escuchan grillos cantando en una noche de verano [23] y en el segundo se escuchan los sonidos producidos por una ballena en aguas profundas [24]. Posteriormente, para crear las señales, se extrajeron cien mil muestras de cada archivo de audio y se interpolaron de forma que se ajustaran al rango $[-1, 1]$ para que tuvieran valores comparables.

El resultado final se puede ver en la [Figura 4.5](#) y en la [Figura 4.6](#) para la señal de los grillos y de la ballena respectivamente. Por simplicidad, se muestran únicamente los 200 primeros puntos de cada serie.

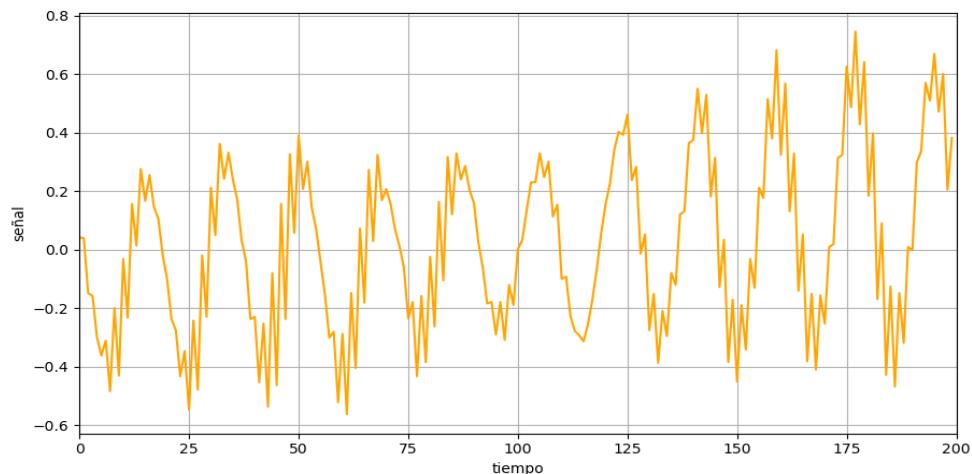


Figura 4.5: 200 primeros elementos de la gráfica de los grillos.

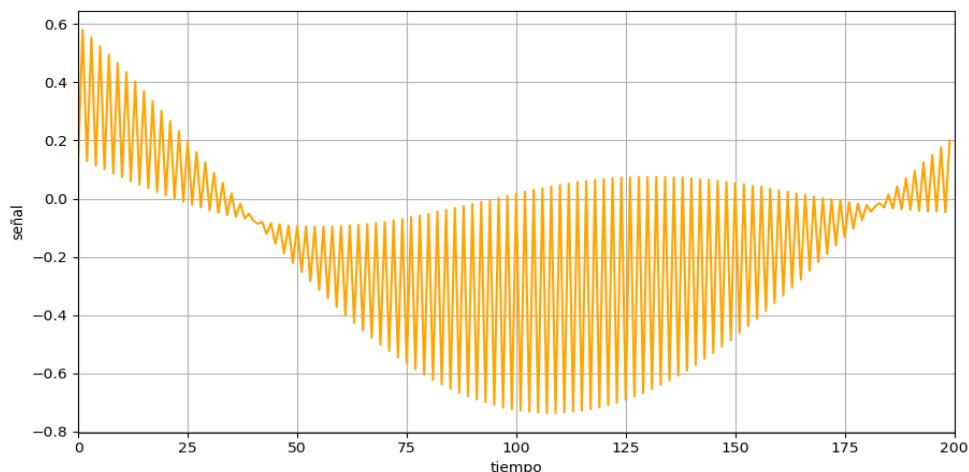


Figura 4.6: 200 primeros elementos de la gráfica de la ballena.

A continuación, se procedió a crear el conjunto de señales para el estudio usando diferente grado de contaminación, es decir, señales que contienen diferentes cantidades de puntos de la otra señal.

Para generar las anomalías colectivas, se seleccionaron segmentos aleatorios de longitud variable (entre 20 y 60 puntos) de una señal y se usaron para sustituir los valores originales de la otra señal en las posiciones equivalentes. Se decidió trabajar con cuatro porcentajes de contaminación (1%, 2%, 3%, 5%) y crear dos grupos de series: el primero, usando la señal de los grillos como señal base y la de ballena como contaminación (en adelante, **señal grillo-ballena**) y el segundo, invirtiendo los roles (en adelante, **señal ballena-grillo**).

En las siguientes figuras se muestran fragmentos de las series contaminadas, en donde el grupo de puntos que forman el segmento anómalo se resalta en color azul. Como se puede ver en la [**Figura 4.7**](#), contaminar la señal de la ballena con segmentos de la señal de grillos casi no provoca distorsión en la señal original, cosa que no ocurre en el caso contrario ([**Figura 4.8**](#)), de modo que sólo se usarán en el estudio las series grillo-ballena.

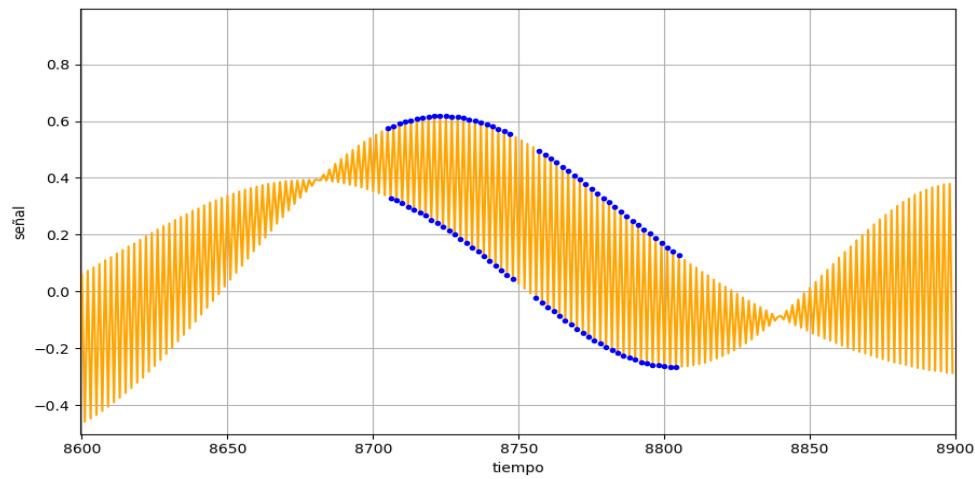


Figura 4.7: Dos segmentos anómalos en la serie ballena-grillo con 3% de contaminación.

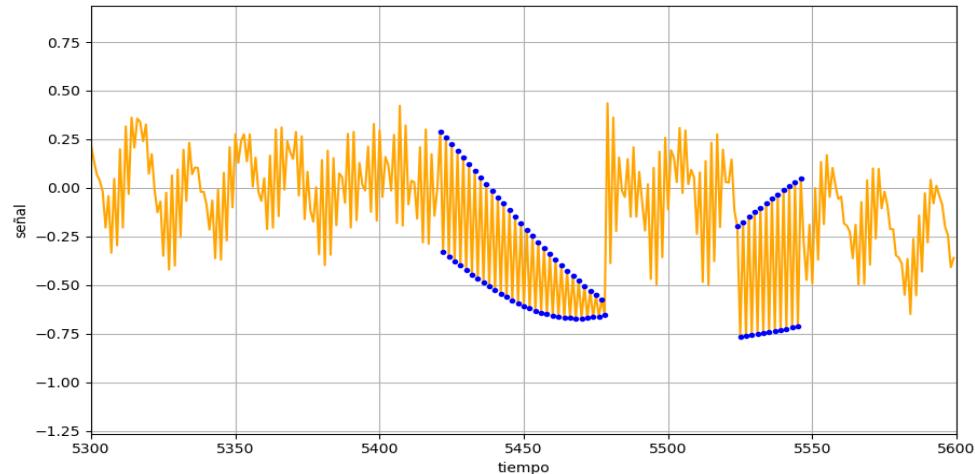


Figura 4.8: Dos segmentos anómalos en la serie grillo-ballena con 5% de contaminación.



Capítulo 5. Desarrollo Experimental

5.1. Planificación Inicial

La planificación inicial para este proyecto consta de cuatro fases y varias tareas, que se pueden ver desglosadas en la **Tabla 5.1**, incluyendo la distribución de las 300 horas que se disponen para su elaboración.

Fases	Horas Estimadas	Tareas
Estudio previo / Análisis	70	Tarea 1.1: Estudio de los conceptos y técnicas de Detección de Anomalías. Tarea 1.2: Estudio de las técnicas de aprendizaje automático para utilizar en Detección de Anomalías. Tarea 1.3: Selección de las series temporales a usar para prueba y evaluación. Tarea 1.4: Familiarización con las herramientas y medios software a utilizar. Tarea 1.5: Definición y Análisis de la solución software al problema.
Diseño / Desarrollo / Implementación	100	Tarea 2.1: Desarrollo de los diferentes módulos del problema, con test de cada uno de ellos. Tarea 2.2: Integración de los módulos para implementar la solución software.
Evaluación / Validación / Prueba	50	Tarea 3.1: Evaluación de la solución software contra las series de datos seleccionadas.
Documentación / Presentación	80	Tarea 4.1: Elaboración de la memoria. Tarea 4.2: Elaboración y ensayo de la presentación.

Tabla 5.1: Planificación inicial del proyecto.

En general, se ha seguido la planificación sin desviaciones significativas. No se ha incluido el tiempo requerido para entrenar las redes neuronales en dicha planificación, debido a que esa es una tarea que se ejecuta en segundo plano, de modo que no se considera tiempo de trabajo efectivo.

5.2. Descripción de Experimentos y Conjunto de Datos

En este trabajo se eligió comparar algunos métodos de la taxonomía de la sección [2.2.2](#) (ver [Figura 5.1](#)). En concreto, se escogió comparar el aprendizaje de la representación de normalidad (explicado en el epígrafe [Categoría 2](#)) con la puntuación de anomalías de extremo a extremo (explicado en el epígrafe [Categoría 3](#)) y, dentro de esas categorías, se escogieron los métodos de *Autoencoders* y *Modelos de Ranking*, respectivamente. En los siguientes apartados se describen los experimentos realizados y sus resultados.

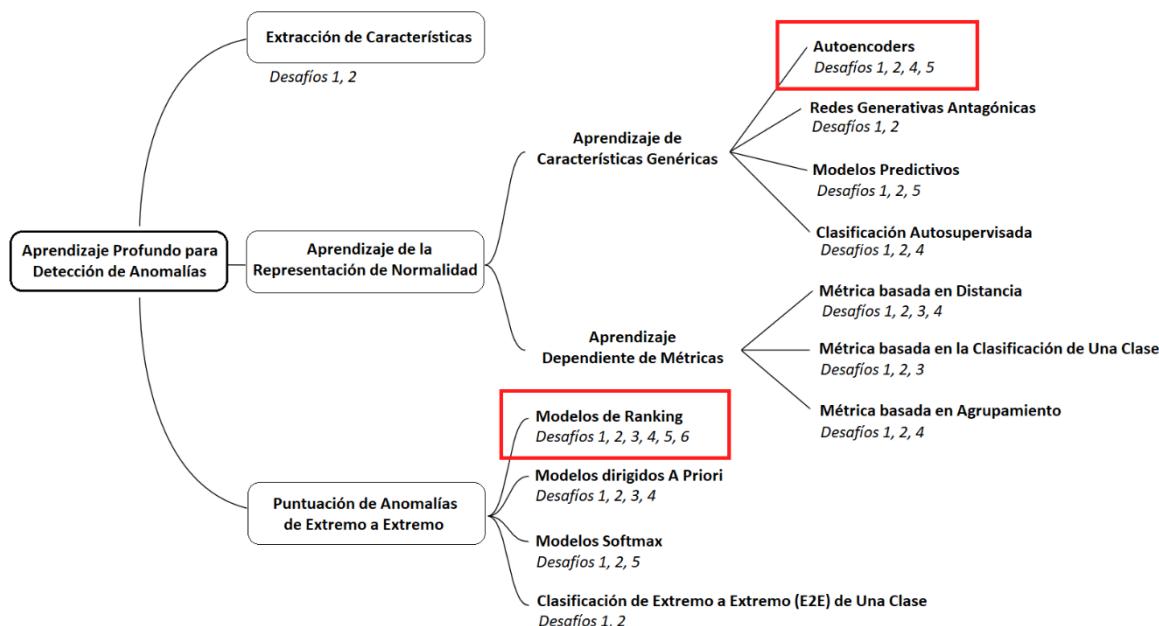


Figura 5.1: Diagrama con los métodos que se comparan en este estudio. Adaptado de [11].

Para trabajar con las series de datos descritas en las secciones [4.2](#) y [4.3](#) es necesario definir un contexto, ya que los puntos que definen las anomalías contextuales y colectivas pertenecen al rango de valores normales que puede tomar la serie, es decir, el punto por sí mismo no se puede considerar anómalo, es su aparición en un determinado contexto lo que lo convertirá en anomalía.

Para ello, puesto que las series tienen una componente temporal, se decidió usar el método de ventana deslizante. De este modo, todos los puntos contenidos dentro de una ventana forman un contexto y cuando dicha ventana se desplace, se define un contexto diferente. En la [Figura 5.2](#) se muestra un diagrama con una ventana de longitud 40 y un desplazamiento de longitud 20 como ejemplo. Dos de los contextos que obtendríamos, en ese caso, estarían formados por los puntos situados entre los instantes t_{20} y t_{60} (resaltado en color azul) y entre los instantes t_{40} y t_{80} (resaltado en color verde). Como

se puede observar, ambas ventanas tienen 20 puntos en común (solapados) entre t_{40} y t_{60} .

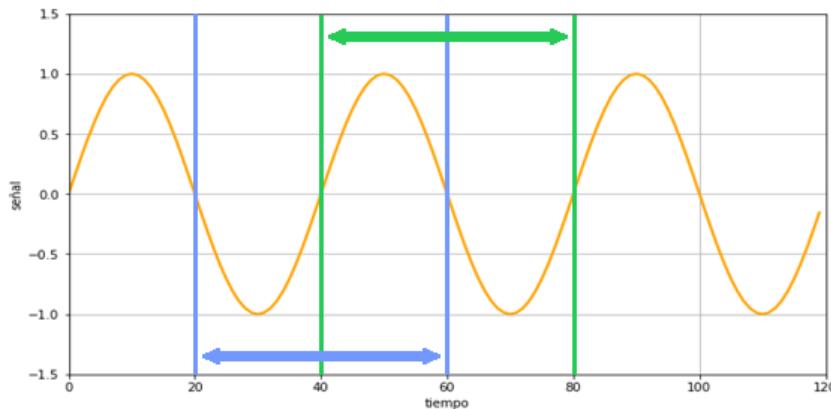


Figura 5.2: Diagrama de ventana deslizante con longitud 40 y desplazamiento 20.

Para el caso de las **series periódicas**, las ventanas se eligieron tomando como referencia la señal simple, usando ventanas superiores, inferiores e iguales al periodo de la señal ($T = 40$). En el caso de la señal combinada, pese a tener un periodo $T = 400$, se decidió usar las mismas ventanas para poder comparar los resultados entre ambas señales.

Para el caso de las **señales naturales**, las ventanas se eligieron tomando como referencia la longitud mínima y máxima de los segmentos de contaminación (20 y 60 puntos, respectivamente).

En cuanto a los desplazamientos, en todos los casos se decidió trabajar con los siguientes tres valores, para probar distintos grados de solapamiento entre ventanas consecutivas (usando el redondeo inferior de la división):

- **Desplazamiento de 1 punto:** implica que las ventanas consecutivas tienen un único punto diferente.
- **Desplazamiento de mitad de ventana:** si, por ejemplo, la ventana tiene longitud 60, implica que las ventanas consecutivas tendrán $60/2 = 30$ puntos diferentes.
- **Desplazamiento igual a la ventana:** implica que las ventanas consecutivas tienen todos sus puntos diferentes.

En lo referente al conjunto de datos, como se explicó en la sección [3.5.3](#), se utilizó la función `train_test_split`¹⁸ de scikit-learn para separar el 20% de muestras para pruebas (80% para entrenamiento) y la función `KFold`¹⁹ para implementar la validación cruzada con $K = 10$ grupos. En ambos casos se deshabilitó la mezcla de datos antes de la división (parámetro `shuffle=False`). Esto lo hice así para que todos los grupos del `KFold` se crearan usando el mismo conjunto de entrenamiento y tuvieran el mismo conjunto de prueba, porque, si se mezclaran, los modelos se probarían con conjuntos distintos y la comparación de resultados de clasificación se vería alterada.

Se construyeron varios conjuntos de datos formando parejas entre las ventanas y los desplazamientos. En total, fueron 15 conjuntos por cada serie (5 ventanas por 3 desplazamientos). Los contextos se etiquetaron como anómalos cuando contenían al menos un punto perteneciente a una anomalía colectiva.



Es importante recalcar que los conjuntos tienen diferentes cantidades de muestras. En la [Tabla 5.2](#) y la [Tabla 5.3](#) se pueden ver las cantidades para cada combinación de ventana y desplazamiento, según el tipo de serie.

Tamaño de Ventana \ Desplazamiento	Un punto	Mitad de ventana	Ventana
5	11995	5998	2399
10	11990	2398	1199
20	11980	1198	599
40	11960	598	299
60	11940	398	199

Tabla 5.2: Cantidad de muestras de los conjuntos de datos (Series Periódicas).

Tamaño de Ventana \ Desplazamiento	Un punto	Mitad de ventana	Ventana
10	99990	19998	9999
20	99980	9998	4999
40	99960	4998	2499
60	99940	3332	1666
90	99910	2221	1111

Tabla 5.3: Cantidad de muestras de los conjuntos de datos (Series Naturales).

¹⁸ https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

¹⁹ https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

5.3. Experimento con Modelo de Ranking

5.3.1. Arquitecturas de Red

Para este experimento se decidió usar la arquitectura de **perceptrón multicapa** (en inglés, *Multilayer Perceptron*, o *MLP*) porque, tal y como se menciona en [19], en el artículo de Richard P. Lippmann “*An introduction to computing with neural nets*”²⁰, se muestra que un MLP con dos capas ocultas es suficiente para resolver problemas no lineales. Aunque no se proporcionan indicaciones sobre la cantidad de neuronas necesarias o el algoritmo de aprendizaje que se debe usar. También se ha demostrado que **los MLP son aproximadores universales**.

Por otro lado, una de las desventajas de este tipo de red es que en la función de coste pueden existir mínimos locales, lo cual implica que no está garantizada la convergencia a un único mínimo y diferentes inicializaciones de pesos pueden producir diferentes resultados de aprendizaje [25].

Se ha escalado la entrada de datos para que se ajuste al rango $[-1, +1]$ y, dado que la salida se interpreta como probabilidad, se ha optado por usar la función **sigmoide** como activación de la salida [ecuación (2.2)]. Para entrenar las redes, se usó la función objetivo **binary crossentropy**, ya que es la más usada en problemas de clasificación, y el optimizador **Adam**, que también se usa con frecuencia en *deep learning* por su eficiencia y bajo requerimiento de memoria [26].

Se decidió trabajar con dos arquitecturas de MLP: la primera con dos capas y la segunda con tres, con el objetivo de comprobar si hay una variación significativa en el rendimiento. Para encontrar la configuración de red más apropiada de cada arquitectura, se planteó una **búsqueda exhaustiva** (definido en la sección 3.5.2) partiendo de configuraciones iniciales escogidas por intuición.



Debido al elevado tiempo de entrenamiento que se requería con los hiperparámetros originales, se optó por realizar algunos recortes en los mismos. Los valores originales se pueden ver en el **Anexo A**.

En el caso del MLP de 2 capas (**Figura 5.3**), se planteó una búsqueda con los siguientes hiperparámetros: longitud de ventana (**W**), desplazamiento de ventana, cantidad de neuronas en la capa oculta (**L**) y ratio de dropout (**D**). Se decidió empezar con 20 neuronas para la capa oculta y un dropout de 0.3.

²⁰ <https://www2.cs.sfu.ca/CourseCentral/414/li/material/refs/Lippmann-ASSP-87.pdf>

En el caso del MLP de 3 capas (*Figura 5.4*), se planteó una búsqueda con los mismos hiperparámetros, añadiendo la cantidad de neuronas en la segunda capa oculta (**M**). Se decidió empezar con 20 neuronas para la primera capa, 10 para la segunda y un dropout de 0.3.

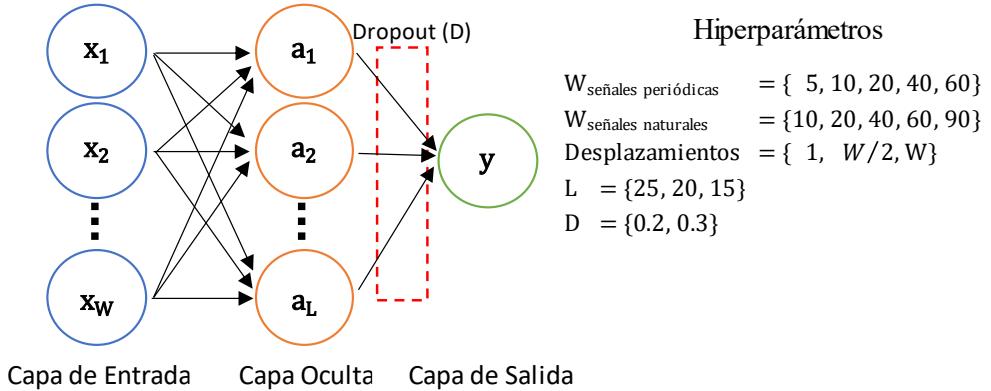


Figura 5.3: Arquitectura e hiperparámetros para el MLP de 2 capas.

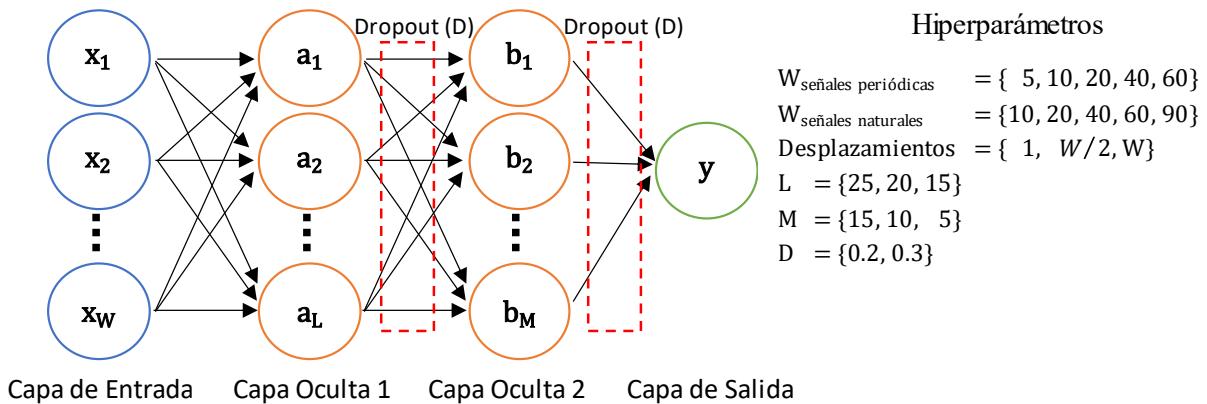
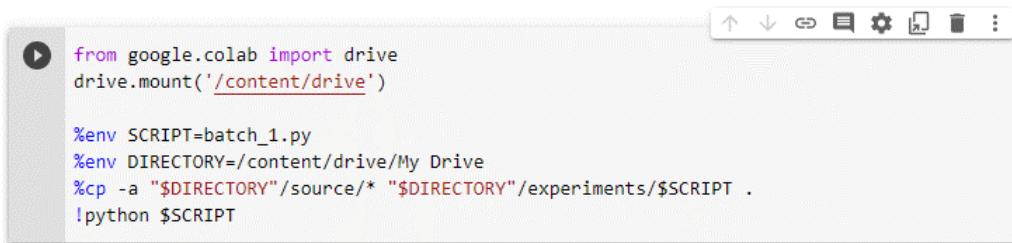


Figura 5.4: Arquitectura e hiperparámetros para el MLP de 3 capas.

5.3.2. Procedimiento

Después de varios intentos para entrenar todos los modelos en un tiempo razonable, se acabaron usando **70 cuadernos de Google Colab** ejecutándose en paralelo.

Cada cuaderno copia desde Google Drive el código fuente y los archivos CSV de las series en su entorno virtual y ejecuta un script que contiene varias configuraciones de red. La clase `ExperimentsCreator` es la encargada de crear los scripts usando un objeto de configuración. A continuación, se pueden ver un ejemplo de cuaderno y script respectivamente. En la **Tabla 5.4** se muestra el tiempo empleado (en la última ejecución) para entrenar todos los modelos según el tipo de arquitectura y el grupo de series.



```

from google.colab import drive
drive.mount('/content/drive')

%env SCRIPT=batch_1.py
%env DIRECTORY=/content/drive/My Drive
%cp -a "$DIRECTORY"/source/* "$DIRECTORY"/experiments/$SCRIPT .
!python $SCRIPT

```

Figura 5.5: Ejemplo de cuaderno de Google Colab.

```

1 from experiments_runners import run_MLP2_experiment
2 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=5, displacement=5, h1_units=25, rate=0.2)
3 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=5, displacement=2, h1_units=25, rate=0.2)
4 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=5, displacement=1, h1_units=25, rate=0.2)
5 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=10, displacement=10, h1_units=25, rate=0.2)
6 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=10, displacement=5, h1_units=25, rate=0.2)
7 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=10, displacement=3, h1_units=25, rate=0.2)
8 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=10, displacement=1, h1_units=25, rate=0.2)
9 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=20, displacement=20, h1_units=25, rate=0.2)
10 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=20, displacement=10, h1_units=25, rate=0.2)
11 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=20, displacement=6, h1_units=25, rate=0.2)
12 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=20, displacement=1, h1_units=25, rate=0.2)
13 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=40, displacement=40, h1_units=25, rate=0.2)
14 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=40, displacement=20, h1_units=25, rate=0.2)
15 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=40, displacement=13, h1_units=25, rate=0.2)
16 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=40, displacement=1, h1_units=25, rate=0.2)
17 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=60, displacement=60, h1_units=25, rate=0.2)
18 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=60, displacement=30, h1_units=25, rate=0.2)
19 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=60, displacement=20, h1_units=25, rate=0.2)
20 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=60, displacement=1, h1_units=25, rate=0.2)
21 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=80, displacement=80, h1_units=25, rate=0.2)
22 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=80, displacement=40, h1_units=25, rate=0.2)
23 run_MLP2_experiment(data_file='senoidal_anomala_1.1porciento_40.csv', prefix='Simple-1-1P', window_length=80, displacement=26, h1_units=25, rate=0.2)

```

Figura 5.6: Ejemplo de script.

	Series Periódicas	Series Naturales
Arquitectura MLP de 2 capas	5 horas, 30 minutos	16 horas, 30 minutos
Arquitectura MLP de 3 capas	18 horas, 43 minutos	42 horas, 37 minutos
Total	24 horas, 13 minutos	59 horas, 7 minutos
		83 horas, 20 minutos

Tabla 5.4: Tiempo empleado para entrenar todos los modelos según la arquitectura de MLP.

Debido a las limitaciones de la versión gratuita de Colab (ver [Anexo B](#)), las 83 horas de entrenamiento tuvieron que repartirse entre 10 días, en períodos de ejecución de 7 horas aproximadamente. Para recopilar los datos del entrenamiento, se optó por usar varios ficheros y directorios que se explican a continuación.

En primer lugar, para cada configuración de red se creó un directorio en cuyo nombre se definen los detalles de la ejecución: serie de datos empleada, cantidad de neuronas en las capas ocultas, ratio de dropout, tamaño de ventana, desplazamiento y función objetivo. En dicho directorio se almacenan los modelos entrenados en ficheros

con formato H5²¹, así como su curva de aprendizaje; además, se guarda el conjunto de prueba que se usó para evaluar los modelos en un fichero con formato NPZ²² y, por último, en un fichero CSV se guarda la matriz de confusión y el coeficiente de correlación de Matthews obtenido al evaluar cada modelo. En la *Figura 5.7* se muestra un ejemplo de fichero CSV para la señal grillo-ballena con un 1% de contaminación.

	A	B	C	D	E	F	G	H
1	ventana	desplazamiento	id_modelo	TP	FN	TN	FP	MCC
2	10		1	0	0,371875	0,628125	0,999441	0,000559 0,57977619
3	10		1	1	0,321875	0,678125	0,99969509	0,00030491 0,54805153
4	10		1	2	0,325	0,675	0,99964427	0,00035573 0,54831116
5	10		1	3	0,315625	0,684375	0,99964427	0,00035573 0,53978131
6	10		1	4	0,246875	0,753125	0,99974591	0,00025409 0,47849397
7	10		1	5	0,334375	0,665625	0,99949182	0,00050818 0,54932161
8	10		1	6	0,290625	0,709375	1	0 0,53601368
9	10		1	7	0,325	0,675	0,99964427	0,00035573 0,54831116
10	10		1	8	0,321875	0,678125	0,99979673	0,00020327 0,5532968
11	10		1	9	0,290625	0,709375	0,99979673	0,00020327 0,52453111

Figura 5.7: Fichero CSV para la señal grillo-ballena 1% con MLP de 2 capas (15 neuronas y dropout 0.2).

A partir de los CSV anteriores, se crearon ficheros para resumir las evaluaciones de los modelos en función de la configuración de red. En estos ficheros se anotó el valor promedio, el mejor y el peor del coeficiente de Matthews, así como qué modelo obtuvo el mejor resultado. Además, se incluyeron la cantidad de neuronas, ratio de dropout, serie de datos con la que se entrenó, longitud de ventana y el desplazamiento. En la *Figura 5.8* se muestra un ejemplo de fichero para las series naturales (se muestran sólo las 10 primeras filas por simplicidad), nótese que la cuarta fila corresponde al resumen de la *Figura 5.7*, en donde el mejor modelo fue el 0.

	A	B	C	D	E	F	G	H	I
1	unidades capa1	ratio de dropout	serie	ventana	desplazamiento	peor MCC	promedio MCC	mejor MCC	mejor modelo
2	15	0,2	GB1	10	10	0,299238042	0,416936026	0,521077422	0
3	15	0,2	GB1	10	5	0,527670479	0,54823147	0,569718387	6
4	15	0,2	GB1	10	1	0,478493968	0,540588853	0,579776194	0
5	15	0,2	GB1	20	20	0,305654945	0,416350586	0,530474035	6
6	15	0,2	GB1	20	10	0,4239872	0,501902155	0,567032319	6
7	15	0,2	GB1	20	1	0,474031191	0,605276811	0,659111838	2
8	15	0,2	GB1	40	40	0,35766365	0,556121685	0,615319849	6
9	15	0,2	GB1	40	20	0,45775864	0,541665039	0,677210802	2
10	15	0,2	GB1	40	1	0,625135995	0,705781659	0,758926475	7

Figura 5.8: Primeras 10 filas del fichero CSV para las señales naturales con MLP de 2 capas (15 neuronas y dropout 0.2).

²¹ https://keras.io/guides/serialization_and_saving/#keras-h5-format

²² <https://numpy.org/doc/stable/reference/generated/numpy.savez.html>

Por último, se creó un fichero en el que se resume cómo de buena es cada configuración para cada grupo de series en promedio. En la *Figura 5.9* se muestra una parte del resumen para el MLP de 2 capas: en las dos primeras columnas podemos ver todas las combinaciones posibles de los parámetros **L** y **D**, en la tercera columna se muestra el coeficiente de Matthews promedio calculado entre todos los modelos que tuvieron los mismos parámetros (sin diferenciar entre señales) y, a continuación, se muestra el valor promedio y el mejor por cada señal (por simplicidad, se muestran sólo los valores de la señal grillo-ballena con un 1% de contaminación).

	A	B	C	D	E
1	unidades capa1	ratio de dropout	promedio MCC general	promedio MCC (GB1)	mejor MCC (GB1)
2	25	0,2	0,605705778	0,579530644	0,817994755
3	25	0,3	0,606305913	0,579894183	0,826801288
4	20	0,2	0,599639952	0,568152153	0,812432477
5	20	0,3	0,598586178	0,562883666	0,820405729
6	15	0,2	0,589042308	0,545693125	0,811285651
7	15	0,3	0,586524147	0,540687021	0,810624965

Figura 5.9: Primeras 5 columnas del fichero CSV que resume el rendimiento del MLP de 2 capas para las Series Naturales.



La elección de la mejor configuración se hace en función del mejor valor de la columna “**promedio MCC general**” de este último fichero.

5.3.3. Resultados para las Series Periódicas

Se han entrenado 72.000 modelos en total (7.200 configuraciones de red):

- **MLP 2 capas:** 20 series · 5 ventanas · 3 desplazamientos · 2 dropout · 3 neuronas · 10 KFold = 18.000 modelos.
- **MLP 3 capas:** 20 series · 5 ventanas · 3 desplazamientos · 2 dropout · 3 neuronas · 3 neuronas · 10 KFold = 54.000 modelos.

En la **Tabla C.1** se puede ver el mejor modelo para cada serie de forma individual. En el caso del MLP de 2 capas, el mejor modelo en general es el que tiene 25 neuronas en la capa oculta y un dropout de 0.3; mientras que el mejor para el MLP de 3 capas es el que tiene 25 neuronas en la primera capa, 15 en la segunda y un dropout de 0.2.

De forma general, tanto para la **señal simple (Figura 5.10)** como para la **señal combinada (Figura 5.11)**, se puede ver que el rendimiento promedio del clasificador aumenta ligeramente cuando se usa la arquitectura de 3 capas y la distancia entre el promedio y el mejor caso se reduce. Esto es lo esperado puesto que la red es más compleja y puede aprender mejor.

En promedio, se detectan mejor las anomalías en la **señal simple** y el rendimiento empeora en las series que tienen un 2% de anomalías respecto a las que tienen un 1.1%. Aun así, alcanza un rendimiento casi perfecto en el mejor caso cuando las señales tienen poco o nada de ruido. Lo cual seguramente se deba a un overfitting debido a la periodicidad de las señales.

En lo referente al ruido, se comprueba que cuanto más ruidosa es la señal, peor es el rendimiento del clasificador, esto se debe a que la forma original de la serie se pierde cada vez más y las anomalías no se aprecian correctamente.

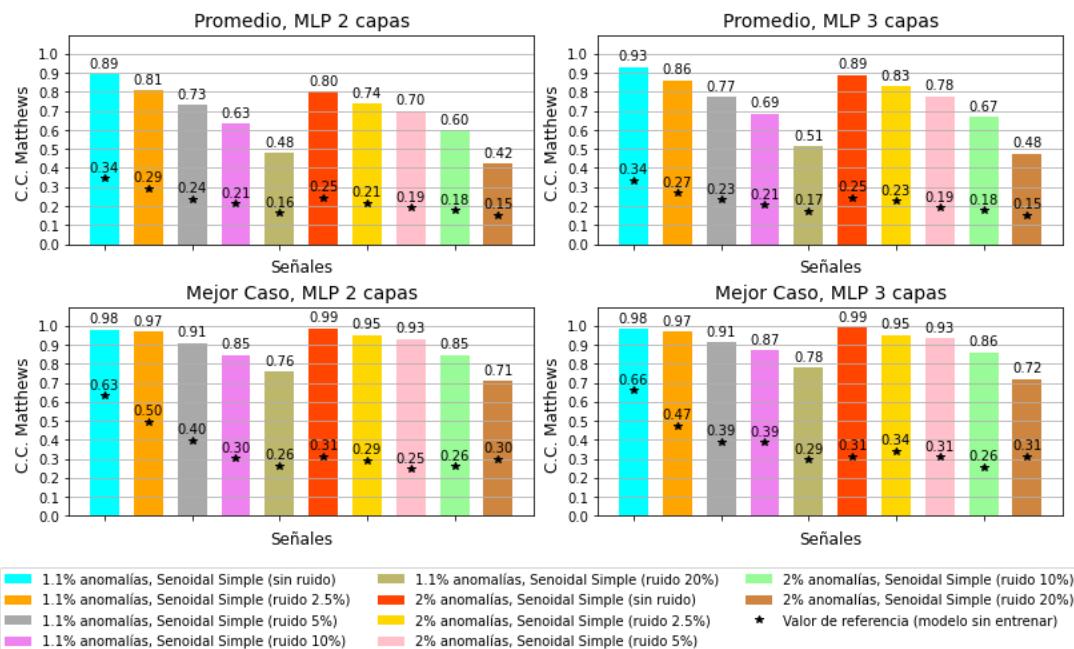
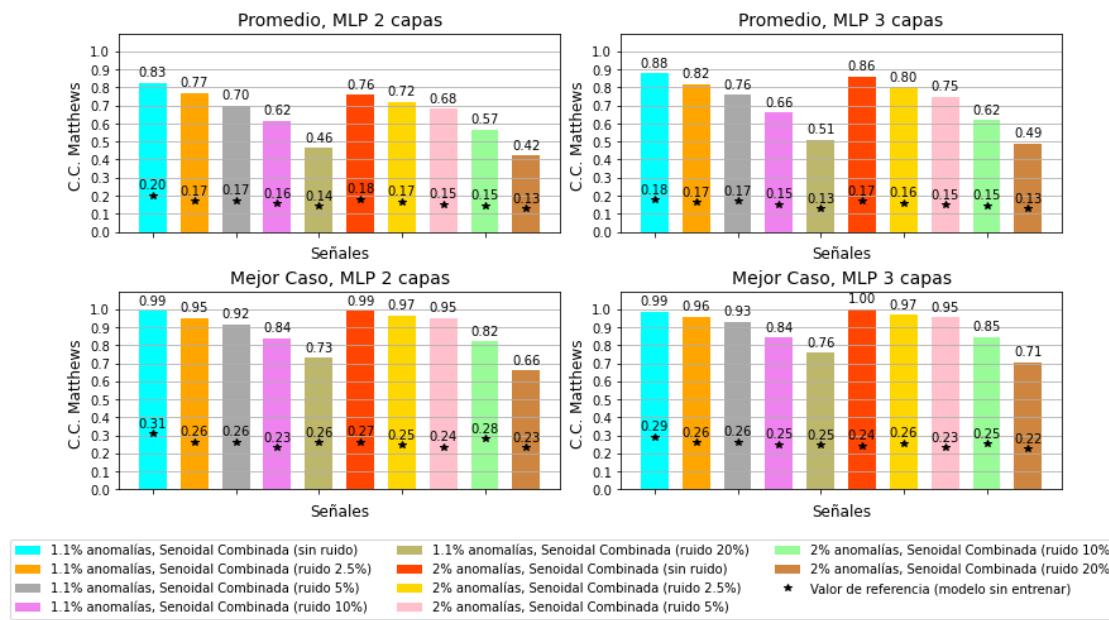
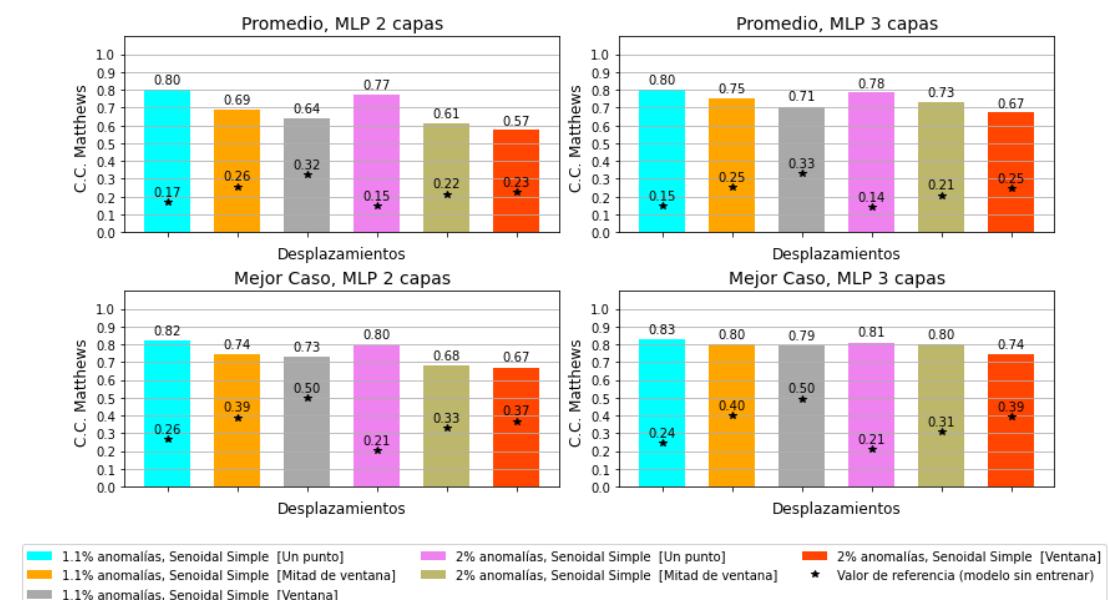


Figura 5.10: Rendimiento general del MLP para la señal periódica simple.

**Figura 5.11:** Rendimiento general del MLP para la señal periódica combinada.

En lo referente al desplazamiento de ventana: en promedio, el rendimiento del clasificador disminuye cuanto mayor es el desplazamiento, de modo que el desplazamiento de un punto es el mejor tanto para la **señal simple** (**Figura 5.12**) como para la **señal combinada** (**Figura 5.13**) independientemente de la arquitectura de red. Esto se debe a que la red tiene más ejemplos para entrenar cuando el desplazamiento es pequeño, pudiendo aprender la forma normal de la serie con mayor precisión. Esto se puede comprobar en la **Tabla 5.2**, donde se puede ver que los conjuntos con desplazamiento 1 son los que más muestras tienen.

**Figura 5.12:** Rendimiento MLP según desplazamiento de ventana para la señal periódica simple.

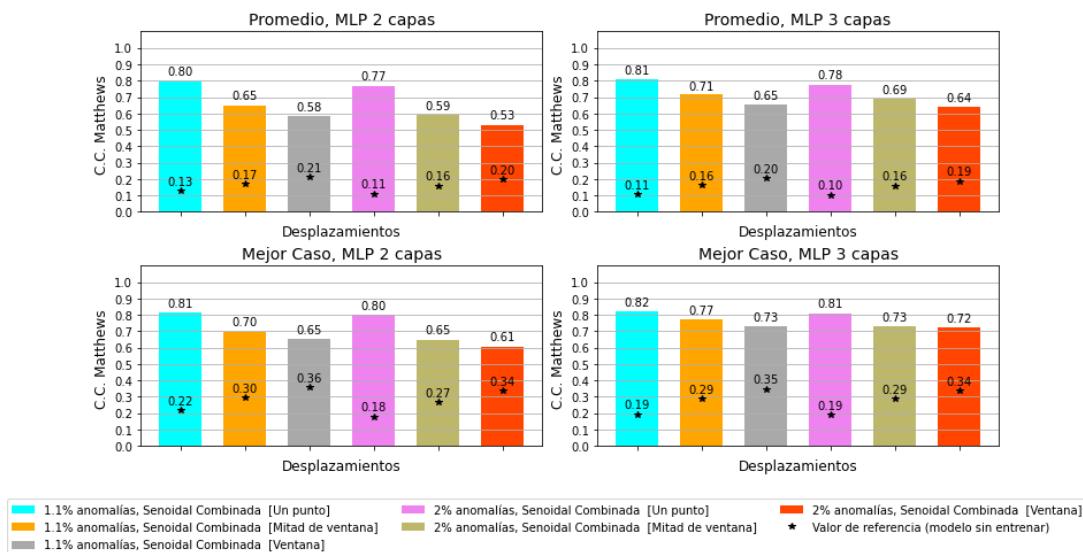


Figura 5.13: Rendimiento MLP según desplazamiento de ventana para la señal periódica combinada.

En lo referente al tamaño de ventana, el mejor rendimiento se alcanza siempre con la ventana de longitud 5 y decrece conforme aumenta el tamaño de ventana, tanto para la *señal simple* ([Figura 5.14](#) y [Figura 5.15](#)) como para la *señal combinada* ([Figura 5.16](#) y [Figura 5.17](#)) independientemente de la arquitectura de red. De nuevo, se debe a que la red tiene más ejemplos para entrenar cuando la ventana es pequeña. En la [Tabla 5.2](#) se puede observar que la cantidad de muestras disminuye según crece la ventana.

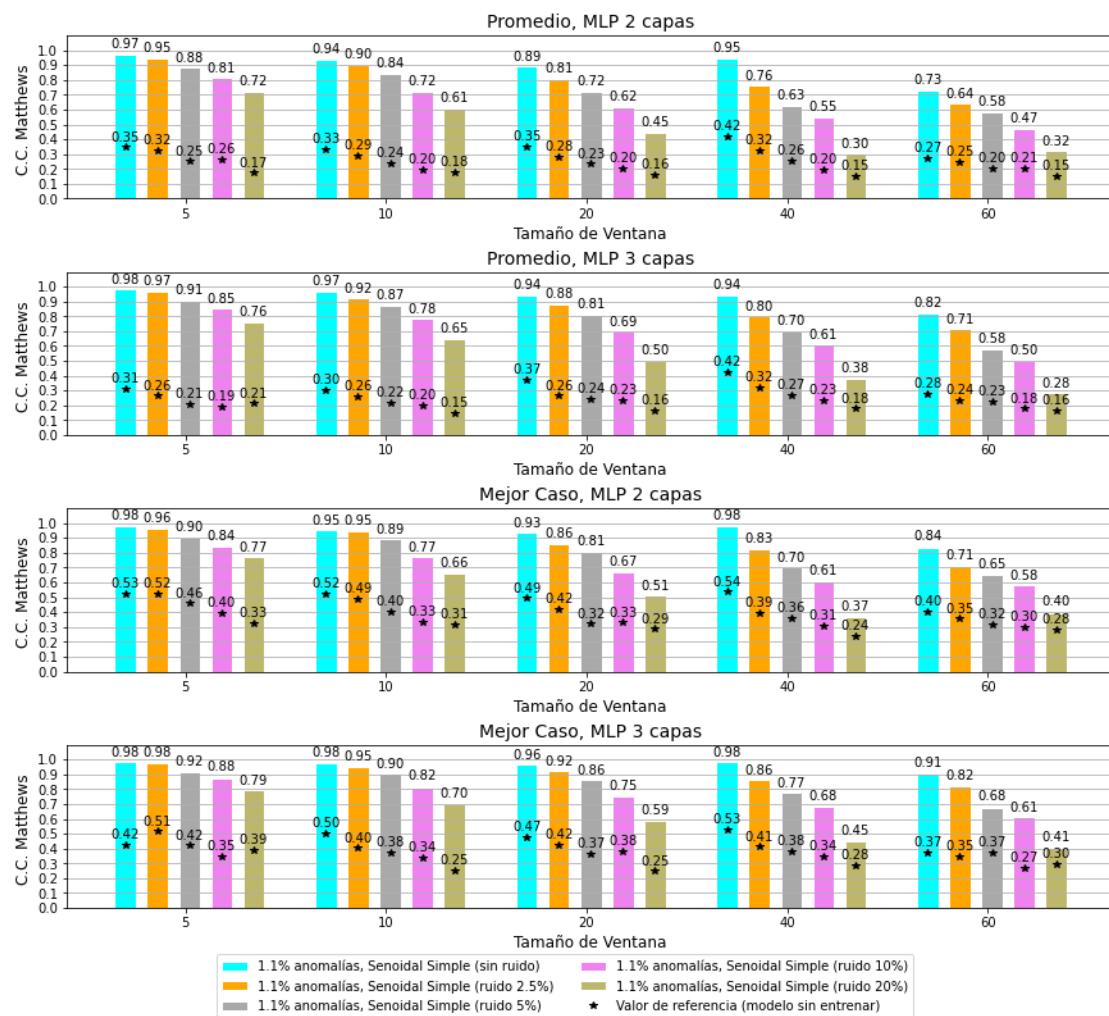


Figura 5.14: Rendimiento MLP según tamaño de ventana para la señal periódica simple con 1.1% de anomalías.

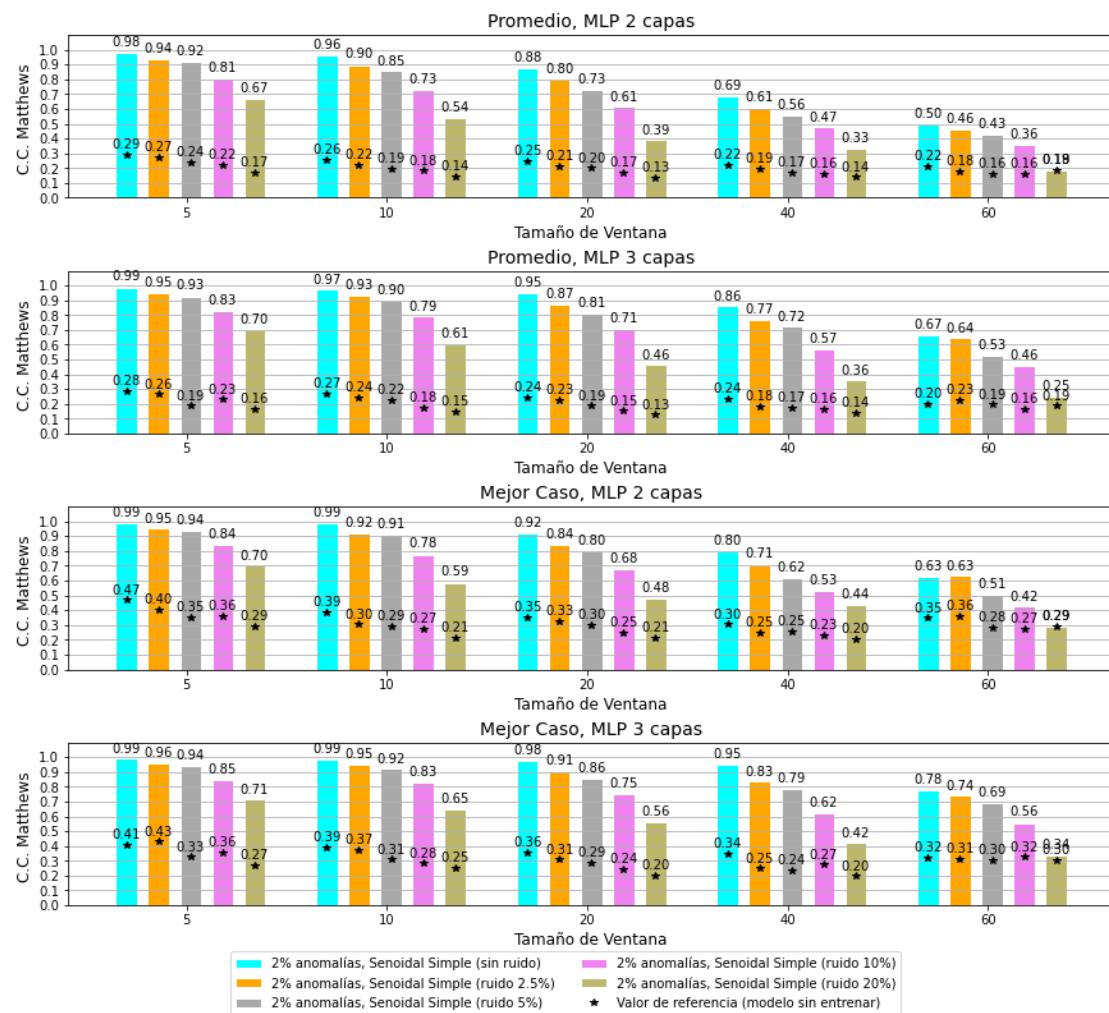


Figura 5.15: Rendimiento MLP según tamaño de ventana para la señal periódica simple con 2% de anomalías.

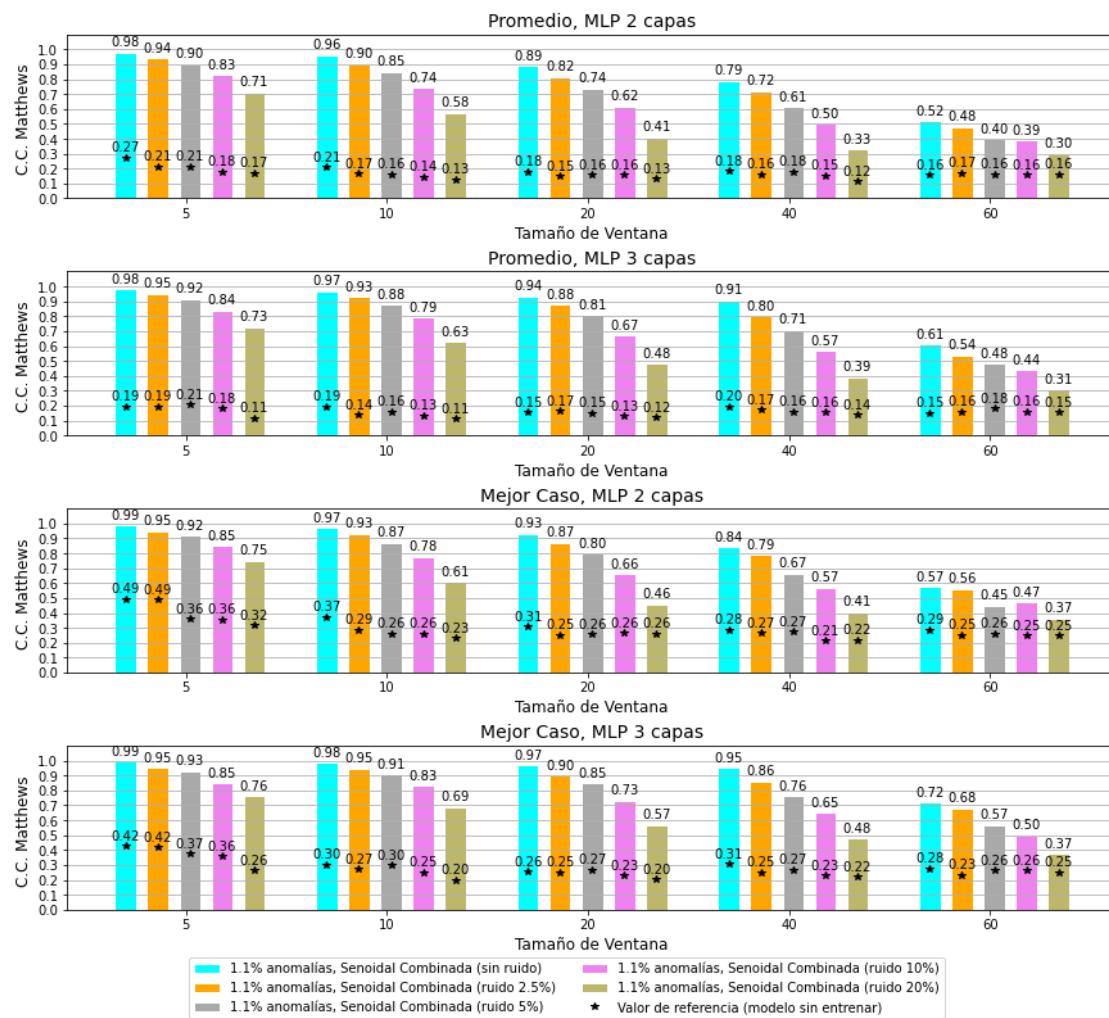


Figura 5.16: Rendimiento MLP según tamaño de ventana para la señal periódica combinada con 1.1% de anomalías.



Figura 5.17: Rendimiento MLP según tamaño de ventana para la señal periódica combinada con 2% de anomalías.

5.3.4. Resultados para las Series Naturales

Al realizar el análisis tras la primera ejecución del experimento, se observó un resultado inesperado al ver la gráfica del MLP de 3 capas: el rendimiento del clasificador disminuye según aumenta el porcentaje de contaminación hasta llegar al 5%, donde los resultados son mejores que para el 3%. En la [Figura 5.18](#) se puede ver la gráfica de ese caso²³, el modelo seleccionado fue el que tiene 25 neuronas en la primera capa, 5 en la segunda y un dropout de 0.2.

Al revisar la cantidad de segmentos de contaminación en cada serie y su longitud, se descubrió que las series con 2%, 3% y 5% de impureza tienen segmentos que superan el tamaño máximo que se usó en el script de creación de las señales (longitud 60), siendo

²³ La gráfica del MLP de 2 capas no está disponible porque se paró el experimento para analizar esta fluctuación.

la serie de 3% la que tiene los segmentos más grandes: uno de longitud 74, otro de 80 y otro de 111. En la **Figura 5.19** se puede ver el diagrama de frecuencia de este análisis, donde se han enmarcado en un recuadro rojo los segmentos que superan la longitud máxima.

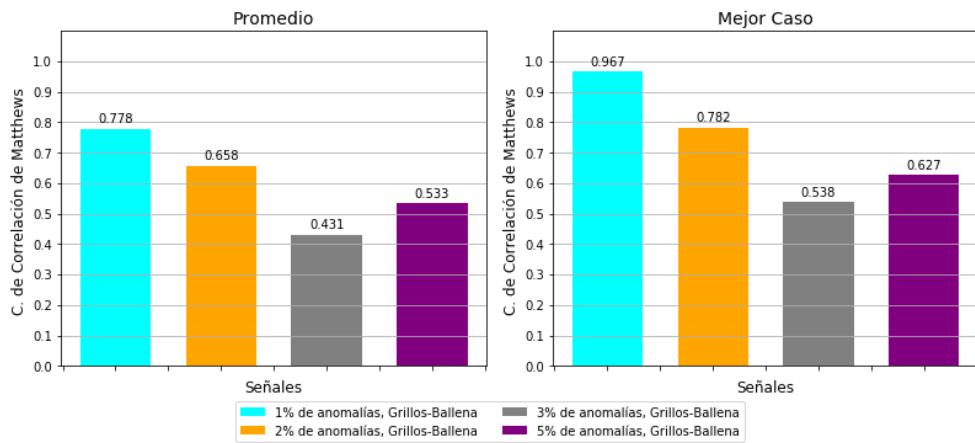


Figura 5.18: Gráfica de la primera ejecución para MLP de 3 capas.

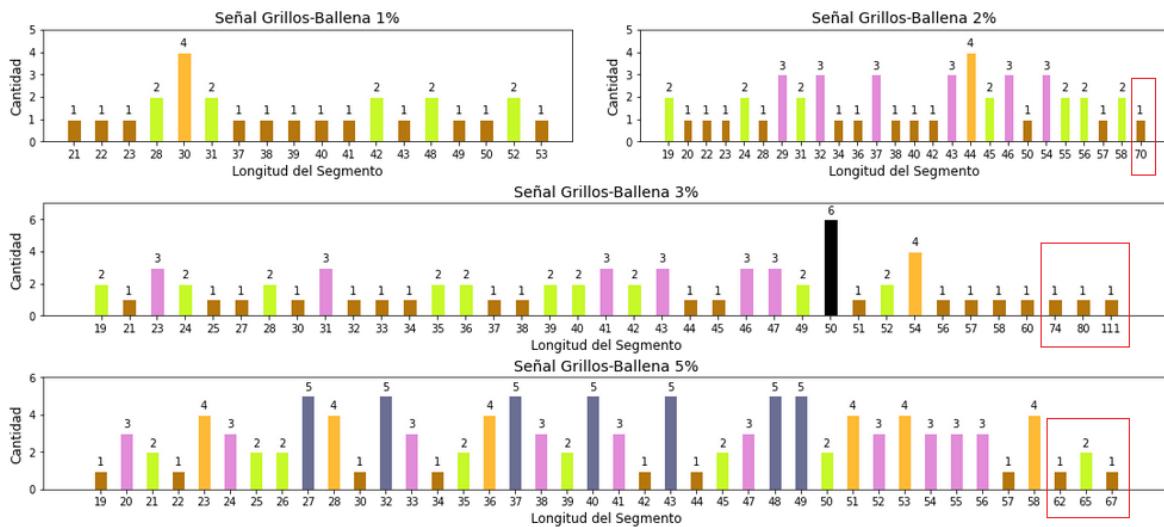


Figura 5.19: Frecuencia de la longitud de los segmentos en las señales originales.

Este error se debe a que en el script de creación se escogía aleatoriamente la posición de inicio del segmento y en ocasiones se solapaban unos con otros formando segmentos más largos. Para corregirlo, se forzó que los segmentos tuvieran una separación mínima aleatoria comprendida entre la longitud mínima y máxima de los mismos. Sin embargo, tras repetir la ejecución con las nuevas series, se sigue observando el mismo comportamiento y además ha empeorado el resultado para la serie del 1%. Como el nuevo diagrama de frecuencia es correcto, se decide continuar con el experimento añadiendo dos series más para ver si existe alguna tendencia: contaminación de 7% y de 10% (diagrama completo en **Figura 5.20**).

Tras añadir los nuevos porcentajes, se han entrenado 21.600 modelos en total (2.160 configuraciones de red):

- **MLP 2 capas:** 6 series · 5 ventanas · 3 desplazamientos · 2 dropout · 3 neuronas
· 10 KFold = 5.400 modelos.
- **MLP 3 capas:** 6 series · 5 ventanas · 3 desplazamientos · 2 dropout · 3 neuronas
· 3 neuronas · 10 KFold = 16.200 modelos.

El mejor modelo para el MLP de 2 capas, en general, es el que tiene 25 neuronas en la capa oculta y un dropout de 0.3; mientras que el mejor para el MLP de 3 capas es el que tiene 25 neuronas en la primera capa, 10 en la segunda y un dropout de 0.2. En la **Tabla C.3** se puede ver el mejor modelo para cada serie de forma individual.

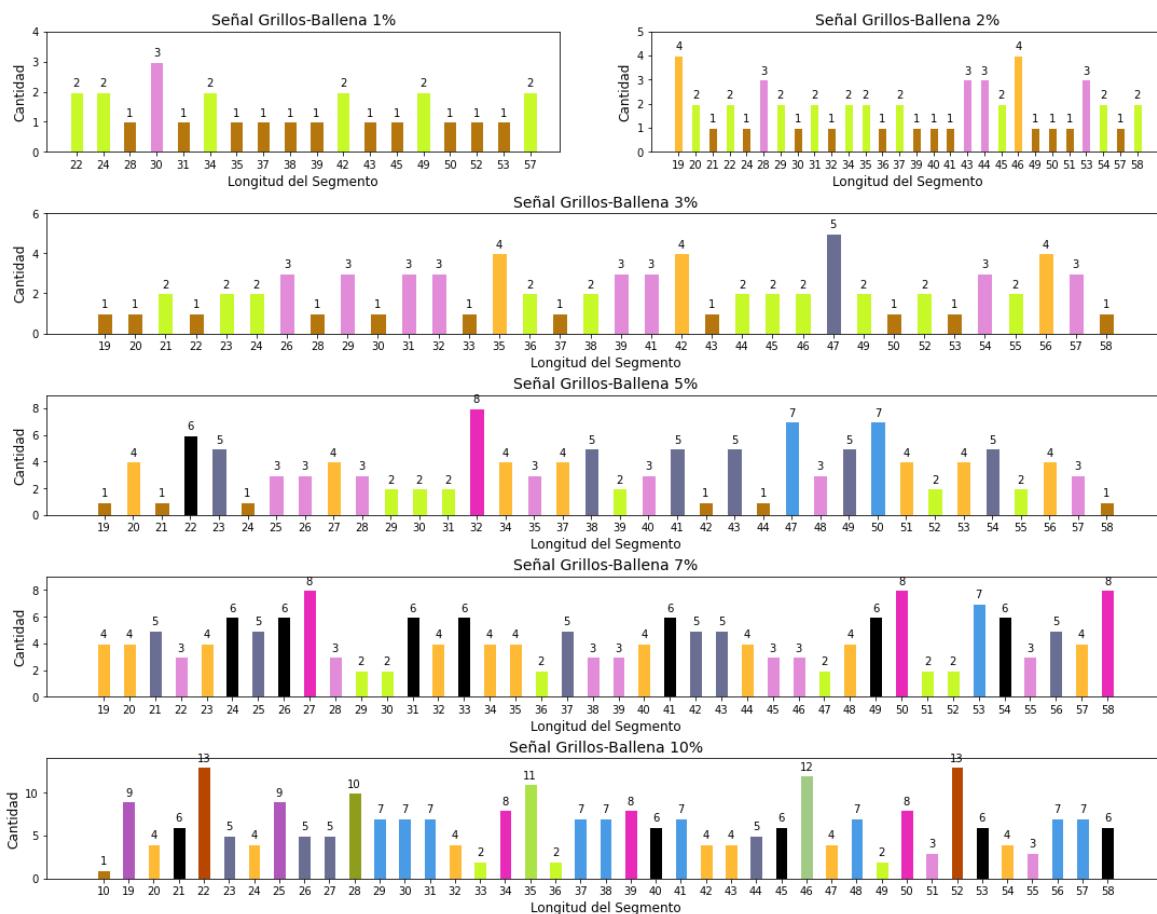


Figura 5.20: Frecuencia de la longitud de los segmentos en las señales nuevas.

De forma general, al igual que ocurre con las otras series, el rendimiento promedio del clasificador aumenta ligeramente cuando se usa la arquitectura de 3 capas y la distancia entre el promedio y el mejor caso se reduce (**Figura 5.21**).

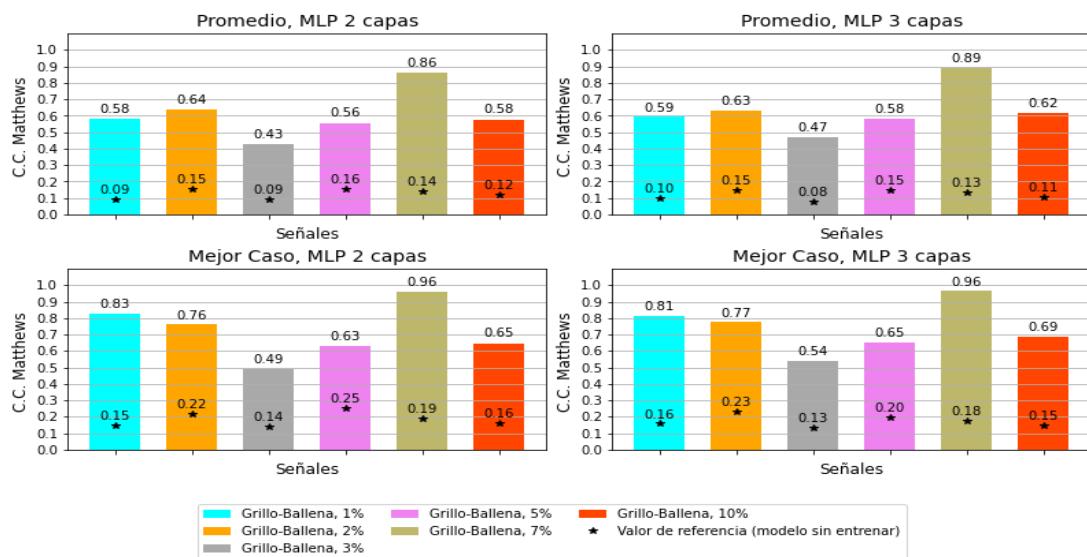


Figura 5.21: Rendimiento general del MLP para las señales grillo-ballena.

El rendimiento del clasificador parece ser algo errático: a diferencia de las series periódicas, donde la gráfica de rendimiento muestra una tendencia decreciente según aumenta el porcentaje de anomalías, aquí no se aprecia que exista ninguna regla, excepto que siempre se obtiene el peor rendimiento con el 3% y el mejor con el 7%.

Tras analizar cómo están repartidas las anomalías en cada serie (**Figura 5.22**) y junto al análisis de frecuencia de segmentos (**Figura 5.20**), concluyo que la bajada de rendimiento en el 3% se debe a que hay tramos donde hay pocos segmentos y además su longitud es pequeña (p. ej. entre los instantes [10.000, 20.000] y [80.000, 90.000]). El repunte en los casos del 5% y 7% se debe a lo contrario: los segmentos están bien repartidos y hay mayor cantidad de cada longitud. En el caso del 10% vuelve a bajar porque hay demasiados segmentos y el clasificador no puede discriminar entre las clases.

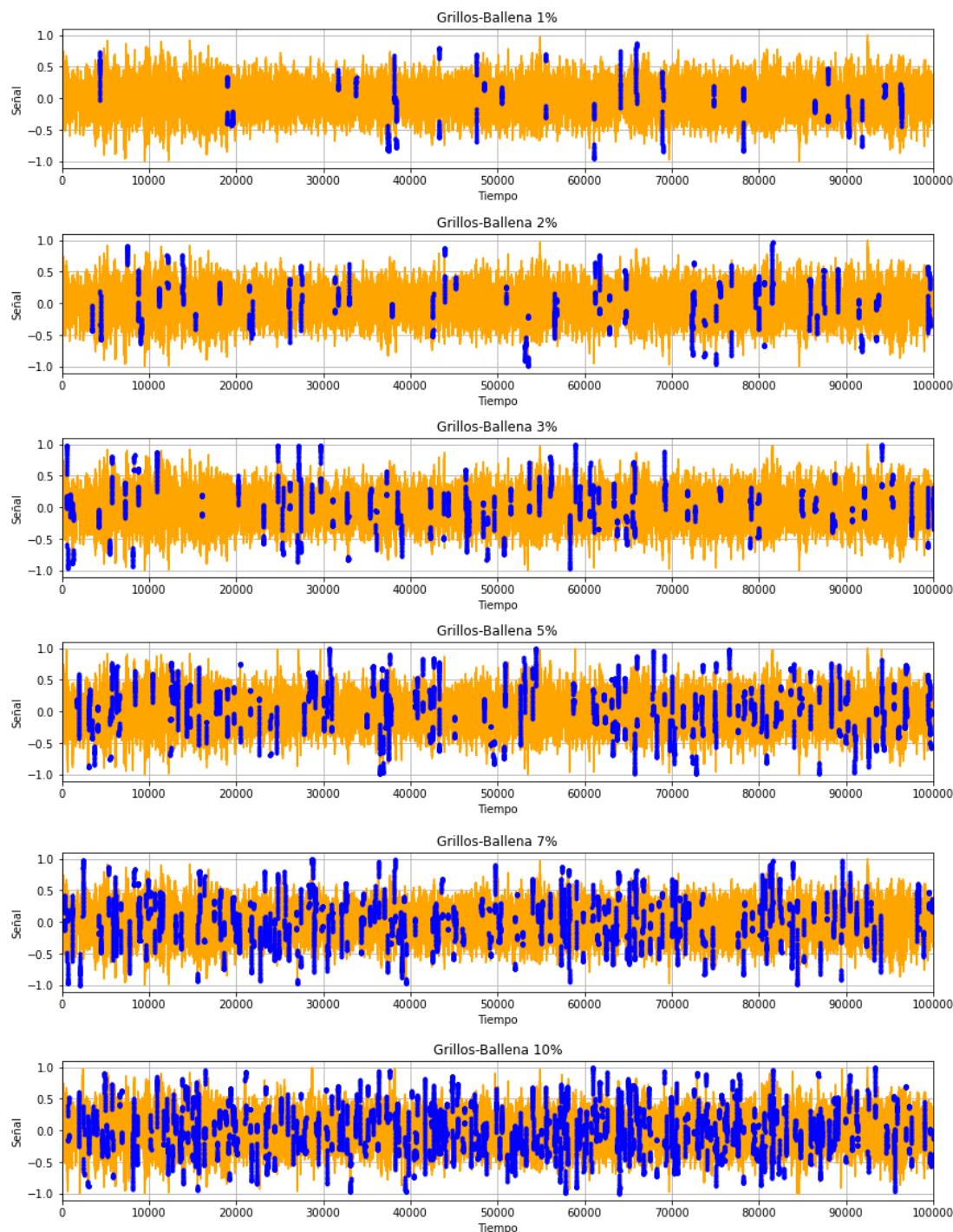


Figura 5.22: Series naturales con los segmentos de contaminación resaltados (azul).

La distribución de las anomalías tiene más relevancia con estas series, cosa que no ocurre con las series periódicas, debido a que la señal no se repite, es decir, el clasificador no puede aprender la forma de la señal con la misma eficiencia porque esta tiene más variaciones que las series periódicas.

Continuando con el análisis, en lo referente al desplazamiento de ventana ([Figura 5.23](#)), ocurre lo mismo que con las series periódicas: el mejor es el de un punto, independientemente de la arquitectura de red, y el rendimiento disminuye según crece. También se debe a la cantidad de muestras del conjunto de datos, en la [Tabla 5.3](#) se puede ver que los conjuntos con desplazamiento 1 son los que más muestras tienen.

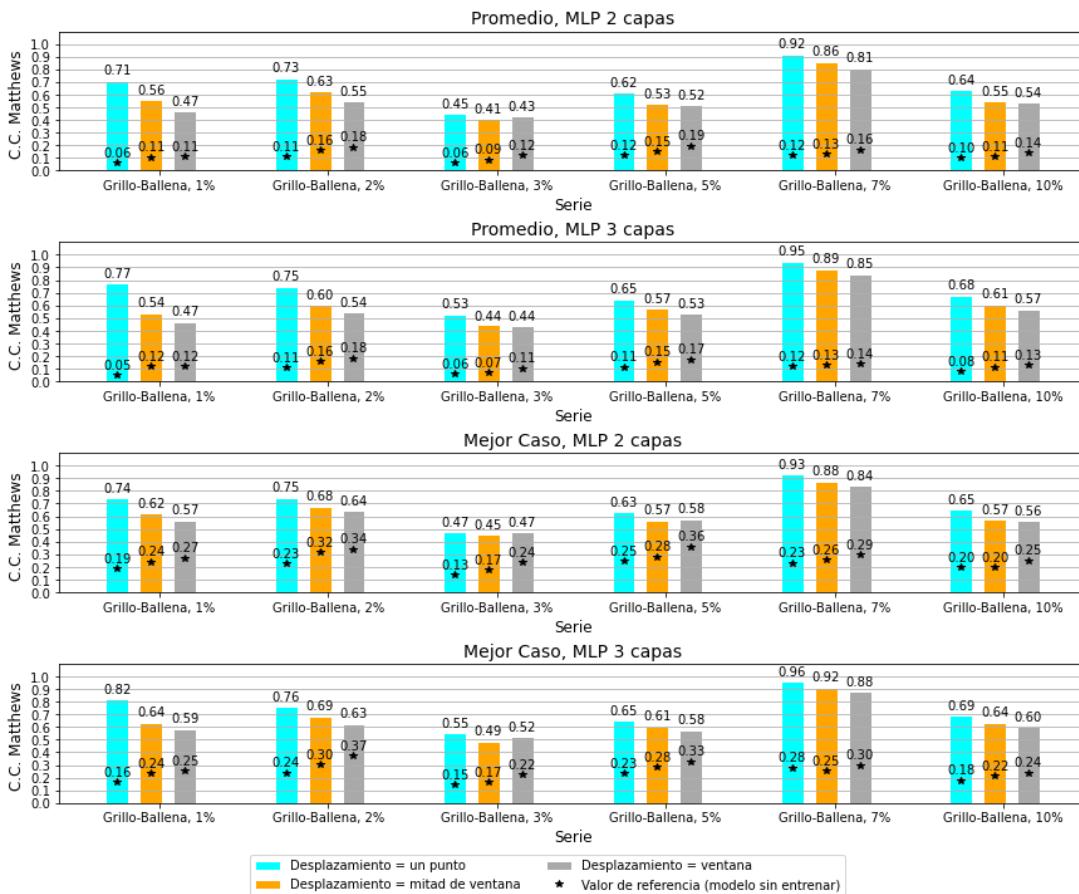


Figura 5.23: Rendimiento MLP según desplazamiento de ventana para las señales grillo-ballena.

En lo referente al tamaño de ventana ([Figura 5.24](#)), los mejores rendimientos se alcanzan en general cuando la ventana tiene valor 10, 20 o 40, excepto en el caso del 7% de contaminación, donde se alcanzan rendimientos superiores en la ventana de 90. Lo más probable es que esto se deba a que el 60% de los segmentos en todas las series tienen una longitud inferior a 43 (ver percentiles en [Tabla 5.5](#)).

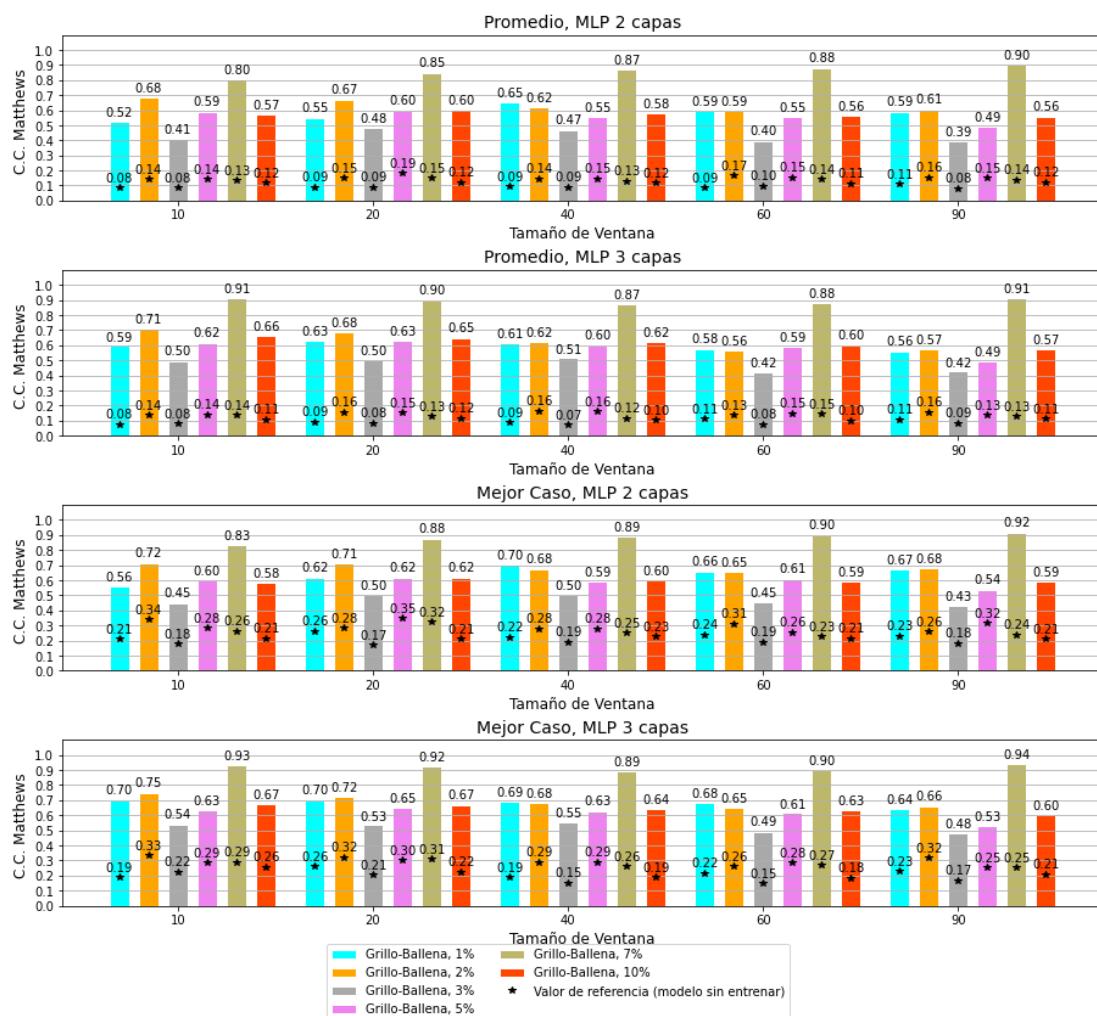


Figura 5.24: Rendimiento MLP según tamaño de ventana para las señales grillo-ballena.

Series \ Percentiles	P.10%	P.25%	P.50%	P.60%	P.75%	P.90%
Grillos-Ballena 1%	24	30	37,5	42	48	52,5
Grilos-Ballena 2%	20,1	28,75	38	43	46	53
Grilos-Ballena 3%	24	31	41	43,8	48,5	55,7
Grilos-Ballena 5%	23	29	39	43	50	54
Grilos-Ballena 7%	23	27	39	43	50	55
Grilos-Ballena 10%	22	28	38	41	48	54

Tabla 5.5: Percentiles de la longitud de los segmentos.

5.4. Experimento con Autoencoder

5.4.1. Arquitecturas de Red

Para este experimento se decidió comparar un autoencoder básico (AE) con uno variacional (VAE). Según se describe en [27]:

- El autoencoder básico (AE) es el tipo más simple de autoencoder: la capa de entrada y la de salida tienen la misma dimensión, mientras que las capas intermedias tienen una dimensión menor (cuello de botella o **bottleneck**).

La red aprende dos funciones: (I) convertir la entrada en una representación comprimida (o latente) de la misma; (II) convertir dicha representación en la entrada original.

- El autoencoder variacional (VAE) es un autoencoder que ha sido regularizado durante el proceso de entrenamiento para evitar el overfitting y asegurarse de que la representación latente tiene propiedades adecuadas para generar datos. Por lo tanto, los VAE son modelos generativos.

La red aprende varias funciones: (I) al igual que el AE básico, aprende a convertir la entrada en una representación latente; (II) sobre esa representación, aprende una distribución de probabilidad normal; (III) por último, aprende a generar la entrada de la red a partir de esa distribución.

La función objetivo en los VAE debe tener dos términos: el primero sirve para forzar que los datos generados sean iguales a la entrada de la red y, el segundo, es un término de regularización que sirve para hacer que la distribución que se aprende se aproxime a una distribución normal estándar. Este término de regularización se expresa como la **divergencia de Kullback-Leibler** (*KL divergence*) entre la distribución devuelta y una gaussiana estándar.

Al igual que con el perceptrón, en este experimento se ha escalado la entrada de datos para que se ajuste al rango $[-1, +1]$ y se ha usado el optimizador Adam para el entrenamiento. También se ha usado una **búsqueda exhaustiva** para encontrar la configuración de red más apropiada de cada arquitectura.

Dado que la salida de la red tiene que ser igual a la entrada, se ha optado por usar la función **tangente hiperbólica** como activación de la salida [ecuación (2.3)]. Para entrenar las redes, se usó la función objetivo **mean squared error**.

Los hiperparámetros que se plantean para ambos autoencoders son iguales a los usados en el experimento del perceptrón: longitud de ventana (W), desplazamiento de ventana, cantidad de neuronas en la capa intermedia (I) y cantidad de neuronas en la capa del *bottleneck* (B). En la [Figura 5.25](#) y [Figura 5.26](#) se pueden ver los diagramas de arquitectura (los hiperparámetros sólo se incluyen en la primera figura porque son los mismos).

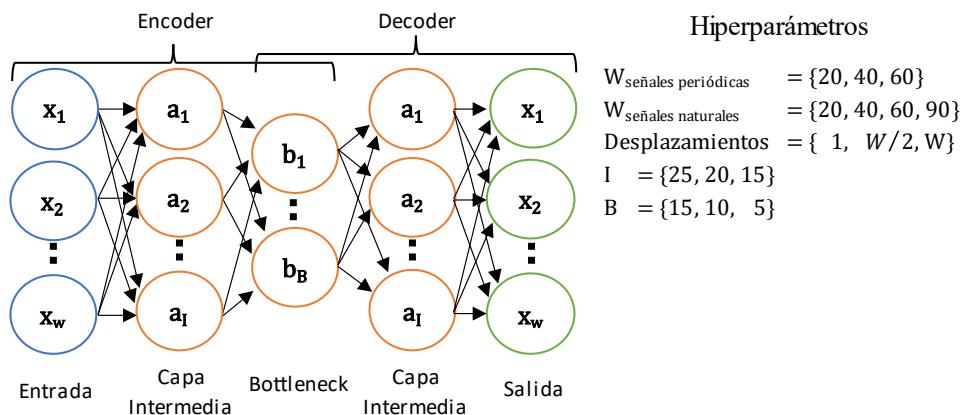


Figura 5.25: Arquitectura e hiperparámetros para el Autoencoder Básico.

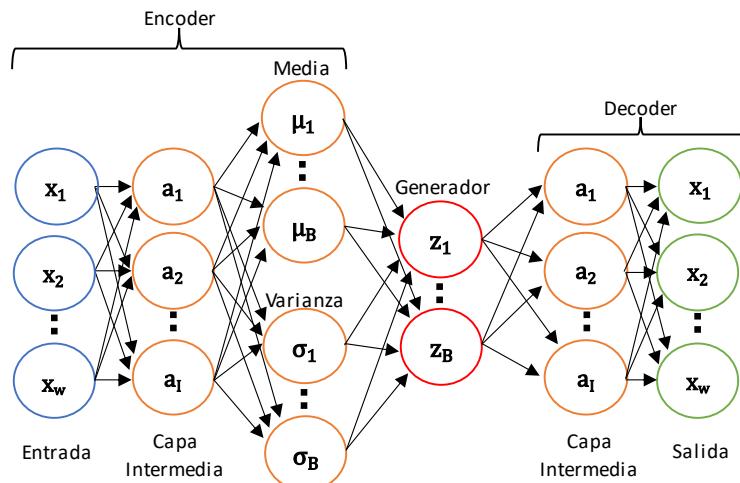


Figura 5.26: Arquitectura para el Autoencoder Variacional.

Se ha decidido eliminar las ventanas de longitud 5 y 10 de este experimento porque en esos casos no existiría *bottleneck* en el autoencoder y pasaría a ser un *overcomplete* autoencoder [28] ([Figura 5.27](#)). El problema de que ocurra esto, es que existe un camino directo entre la entrada y la salida, es decir, la entrada puede atravesar la red sin cambiar y no habría representación intermedia, lo cual hace que la red no sea útil. Para evitar este problema, se puede añadir una función de regularización a la red, pero se ha descartado esa opción para evitar mezclar configuraciones con y sin *bottleneck* en el mismo experimento.

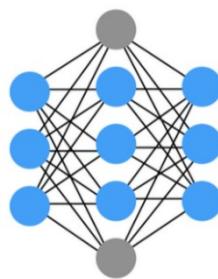


Figura 5.27: Ejemplo de un overcomplete autoencoder. Fuente [28].

Por otro lado, para el caso de la ventana de longitud 20, hay dos valores de capa intermedia (25 y 20 neuronas) que expanden o conservan la dimensionalidad de la entrada, respectivamente, pero en este caso no se considera que el autoencoder sea *overcomplete* porque siempre existe *bottleneck*.

También se decidió no tratar el problema del overfitting en el autoencoder básico porque el variacional ya lo hace.

5.4.2. Procedimiento

El procedimiento seguido es igual al descrito para el perceptrón en la sección [5.3.2](#), lo único que cambia son los campos de los ficheros CSV porque los hiperparámetros son diferentes. En la [Tabla 5.6](#) se muestra el tiempo empleado para entrenar todos los modelos según el tipo de arquitectura y el grupo de series. El tiempo de entrenamiento tuvo que repartirse entre 3 días, en periodos de ejecución de 7 horas aproximadamente, por las restricciones de Google Colab ya mencionadas anteriormente (ver [Anexo B](#)).

	Series Periódicas	Series Naturales
Arquitectura AE	2 horas, 48 minutos	8 horas, 47 minutos
Arquitectura VAE	2 horas, 20 minutos	9 horas, 2 minutos
Total	5 horas, 8 minutos	17 horas, 49 minutos
		22 horas, 57 minutos

Tabla 5.6: Tiempo empleado para entrenar todos los modelos según la arquitectura de Autoencoder.

Respecto a la fase de entrenamiento, existen dos enfoques a la hora de entrenar autoencoders:

1. **Entrenar sólo con datos normales:** en este caso, se eliminan las muestras anómalas del conjunto de entrenamiento antes de empezar a entrenar. Requiere que todos los datos estén etiquetados.
2. **Entrenar con todos los datos:** en este caso, no se filtra el conjunto de entrenamiento. Se supone que la red puede aprender la forma normal de los datos porque hay pocas anomalías.

En ambos casos, el conjunto de prueba debe estar etiquetado para poder validar si la red ha aprendido. Para este experimento se usó el enfoque 1, aprovechando que se dispone de todos los datos etiquetados.

Respecto al cálculo del umbral (*threshold*) del clasificador, según se describe en [29], parece que lo normal es calcular el error de reconstrucción de todas las muestras en el conjunto de prueba y tomar un percentil alto como umbral. En este caso, se ha usado el rango [80, 99] para buscar dicho percentil.

En este experimento se han entrenado 22.680 modelos (2.268 configuraciones de red) por cada tipo de autoencoder, 45.360 modelos en total:

- **Series periódicas:** 20 series · 3 ventanas · 3 desplazamientos · 3 neuronas intermedias · 3 dimensiones latentes · 10 KFold = 16.200 modelos.
- **Series naturales:** 6 series · 4 ventanas · 3 desplazamientos · 3 neuronas intermedias · 3 dimensiones latentes · 10 KFold = 6.480 modelos.

5.4.3. Resultados para las Series Periódicas

En la Tabla C.2 se puede ver el mejor modelo para cada serie de forma individual. En el caso del autoencoder básico, el mejor modelo en general es el que tiene 25 neuronas en la capa intermedia y 10 neuronas en el bottleneck; mientras que el mejor para el autoencoder variacional es el que tiene 25 neuronas en la capa intermedia y 15 neuronas en el bottleneck.

De forma general, tanto para la *señal simple* (*Figura 5.28*) como para la *señal combinada* (*Figura 5.29*), se puede ver que el rendimiento promedio del clasificador disminuye considerablemente en el autoencoder variacional respecto al básico. Esto seguramente se deba al overfitting del básico. De todos modos, el rendimiento del variacional es inferior al esperado, lo cual sugiere que la arquitectura es demasiado simple para poder aprender la distribución de probabilidad adecuadamente.

Se observa también que, en promedio, se detectan mejor las anomalías en la *señal simple* y el rendimiento empeora en las series que tienen un 2% de anomalías respecto a las que tienen un 1.1%.

En cuanto al ruido, resulta llamativo lo poco que afecta al clasificador, sobre todo si lo comparamos con los resultados del MLP en el experimento anterior. Parece que el VAE se ve menos afectado que el AE básico, aunque el rendimiento sea menor.

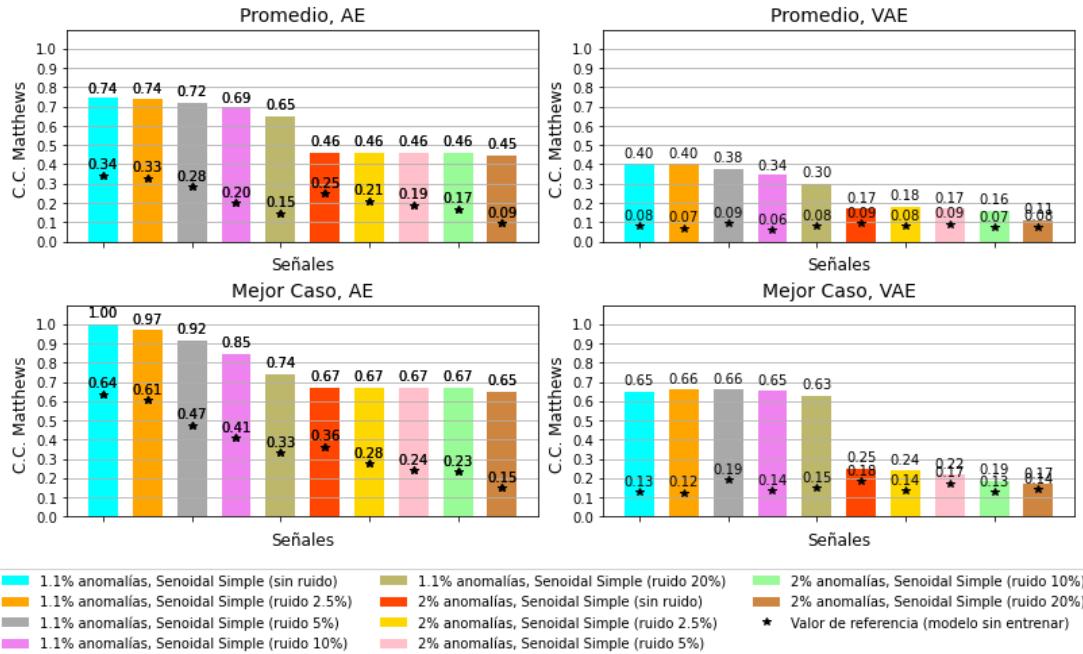


Figura 5.28: Rendimiento general del autoencoder para la señal periódica simple.

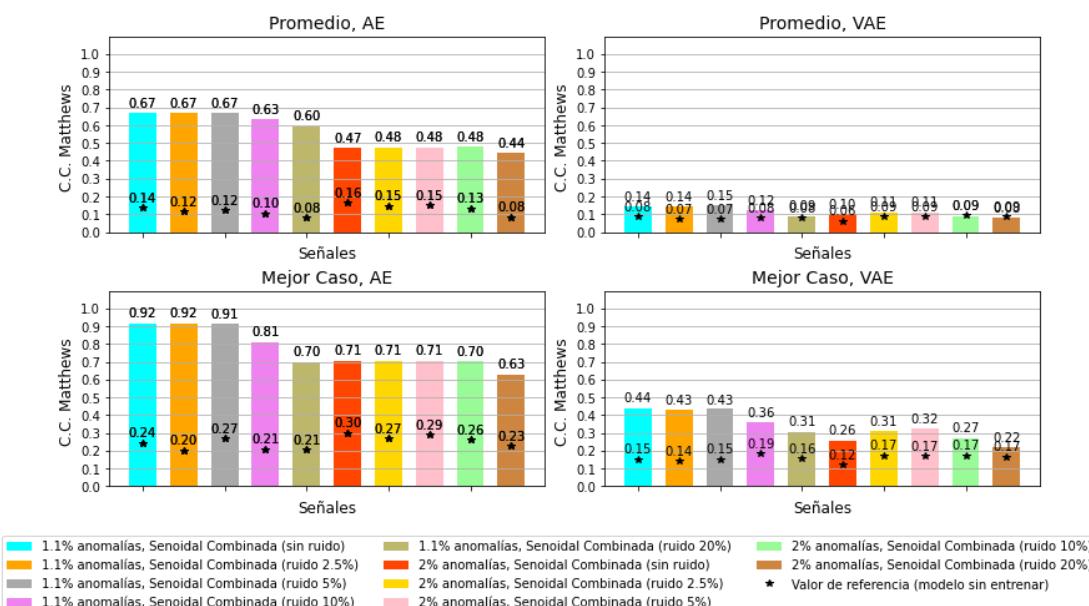


Figura 5.29: Rendimiento general del autoencoder para la señal periódica combinada.

En lo referente al desplazamiento de ventana: en el caso del AE básico (*Figura 5.30*) se observa que, a diferencia del MLP, no afecta en absoluto al rendimiento del clasificador, seguramente por el overfitting. En el caso del VAE, el rendimiento es un tanto errático, lo cual refuerza la teoría de la simpleza de la arquitectura.

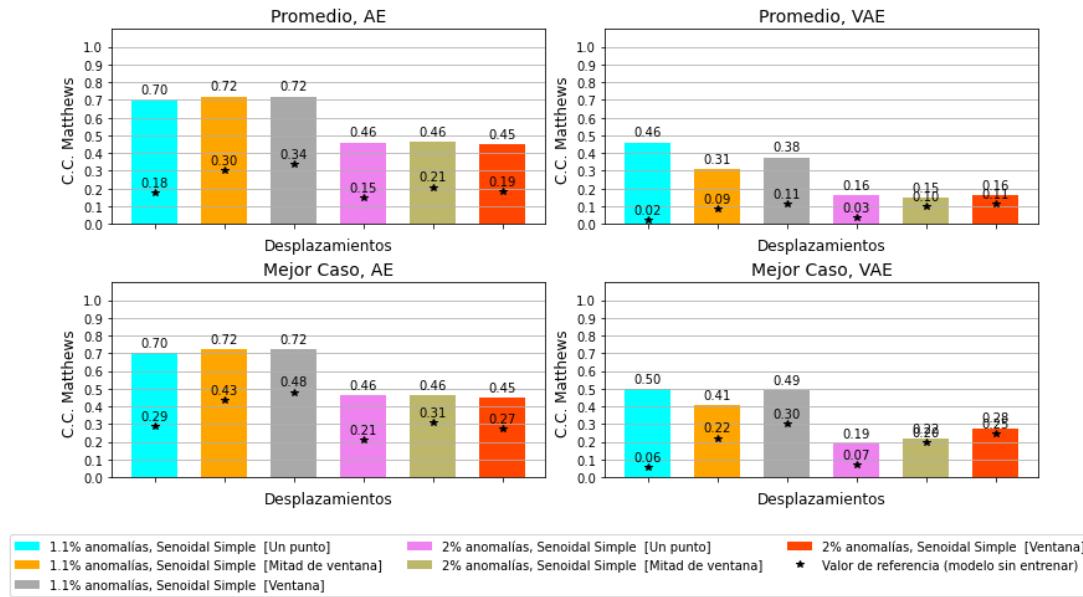


Figura 5.30: Rendimiento autoencoder según desplazamiento de ventana para la señal periódica simple.

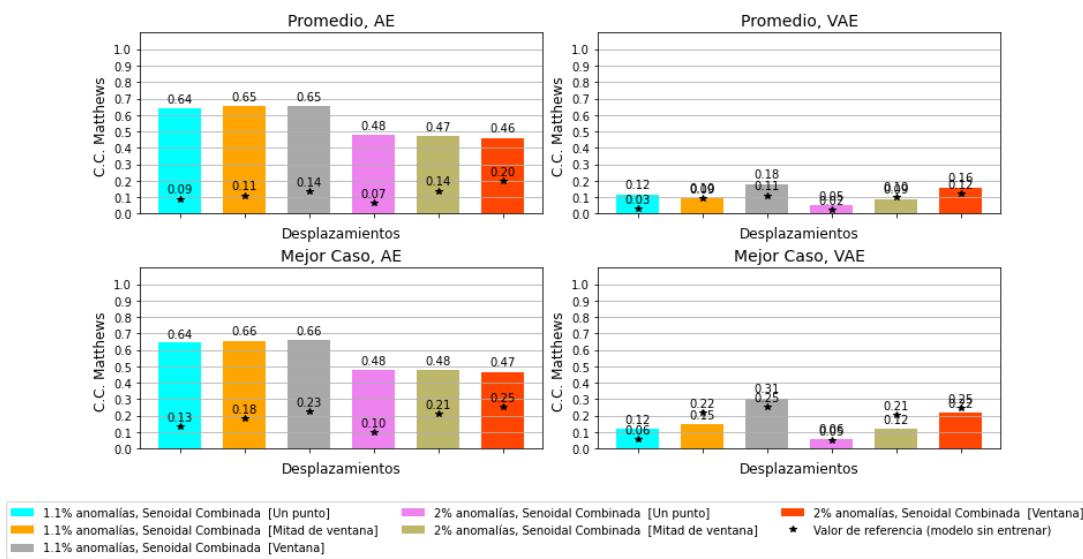


Figura 5.31: Rendimiento autoencoder según desplazamiento de ventana para la señal periódica combinada.

En lo referente al tamaño de ventana, el mejor rendimiento se alcanza siempre con la ventana de longitud 20 para el AE básico y decrece conforme aumenta el tamaño de ventana, tanto para la *señal simple* (*Figura 5.32* y *Figura 5.33*) como para la *señal combinada* (*Figura 5.34* y *Figura 5.35*). De nuevo, se debe al overfitting. Por otro lado, el VAE sigue comportándose de forma errática.

El ruido parece afectar menos al rendimiento cuando las ventanas son grandes, pero no le encuentro una justificación.

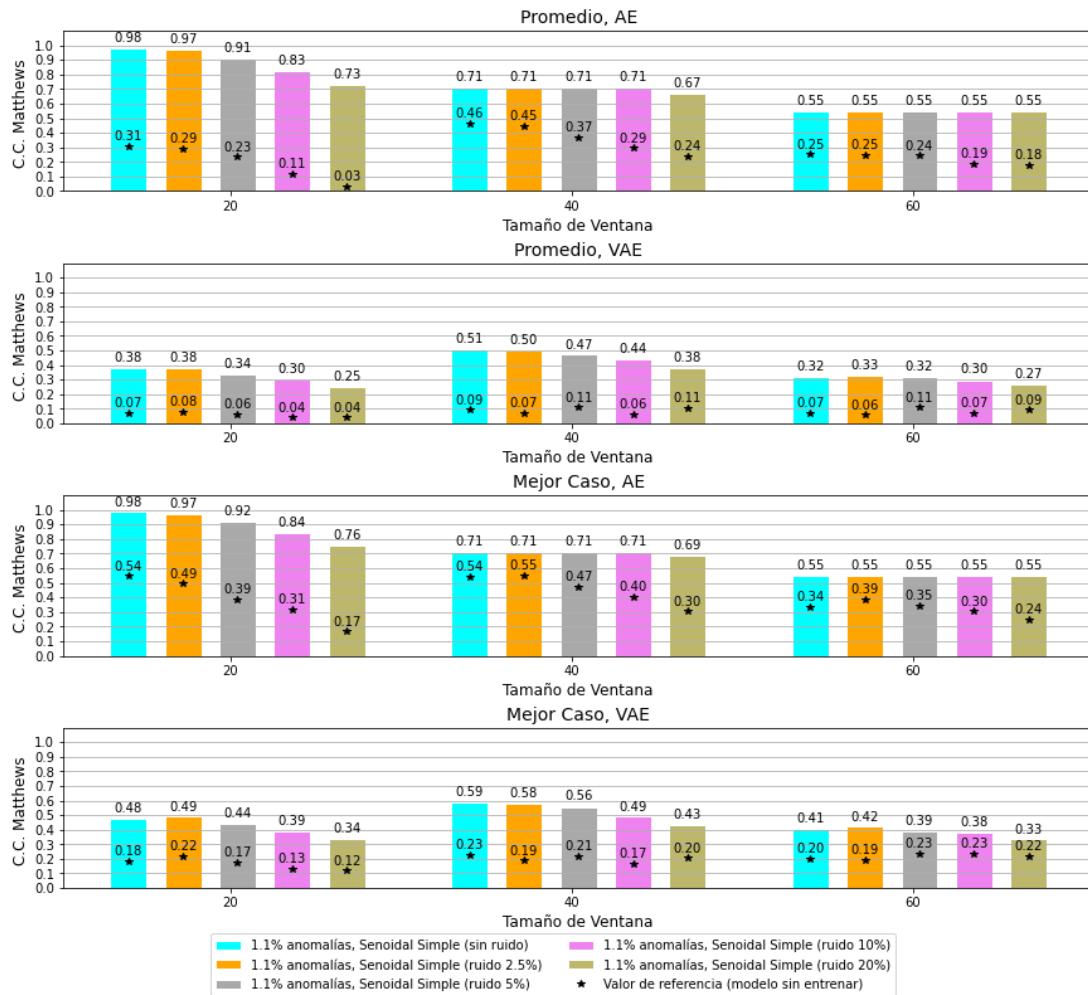


Figura 5.32: Rendimiento autoencoder según tamaño de ventana para la señal periódica simple con 1.1% de anomalías.

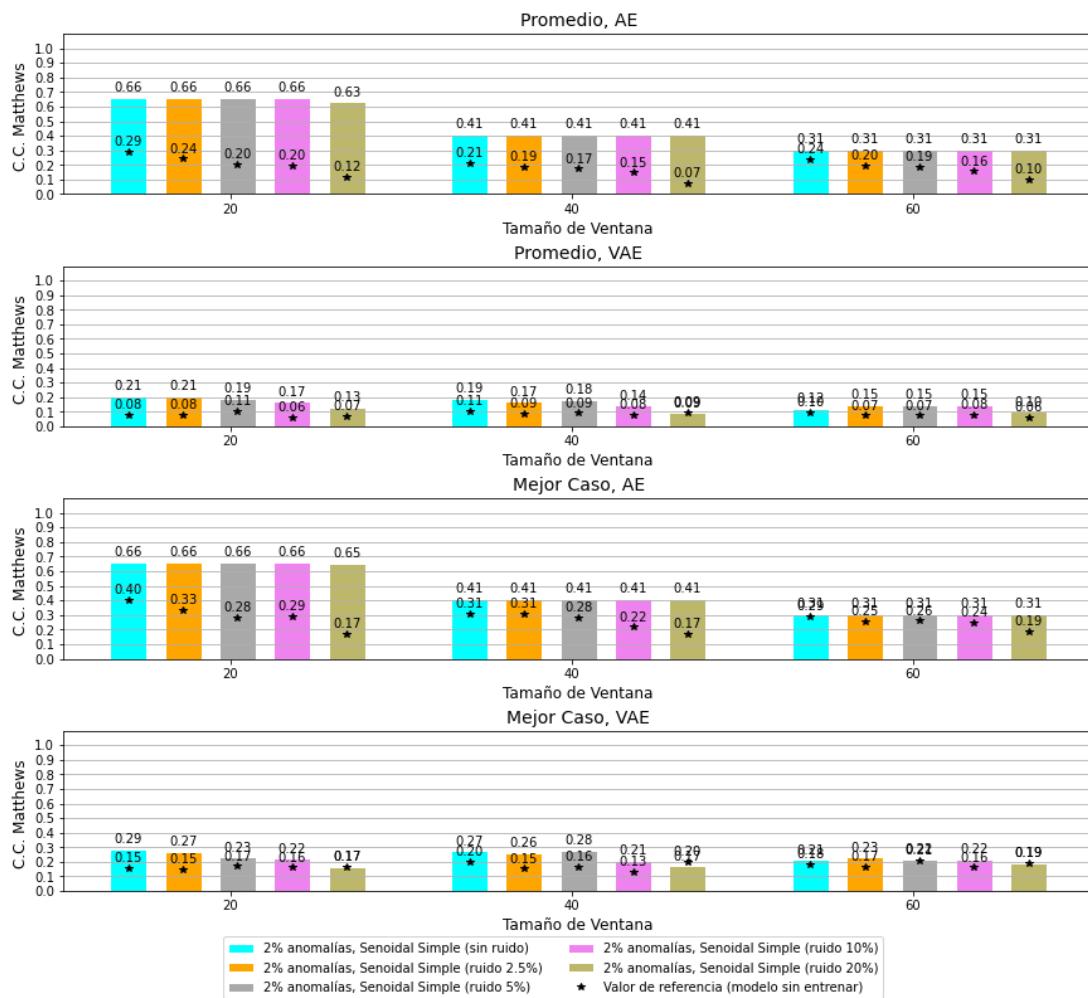


Figura 5.33: Rendimiento autoencoder según tamaño de ventana para la señal periódica simple con 2% de anomalías.

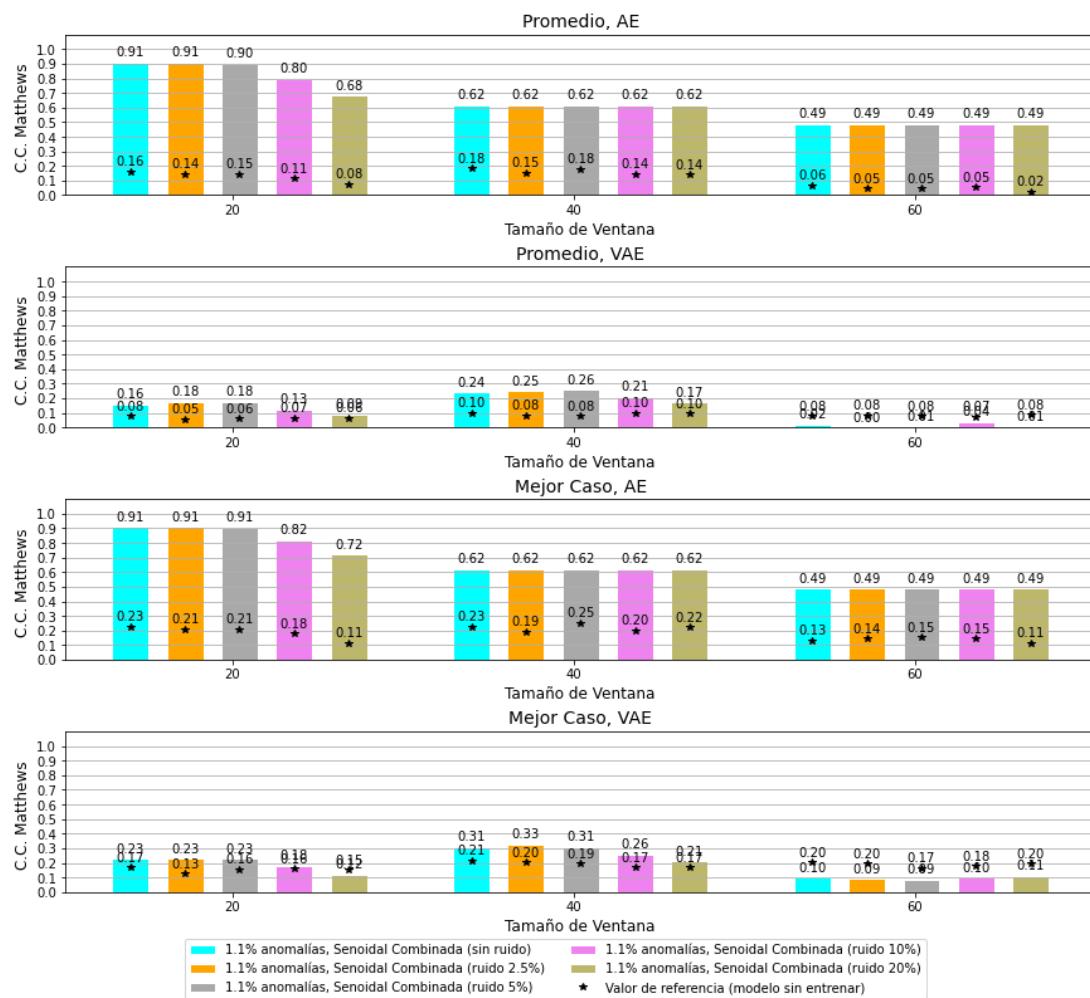


Figura 5.34: Rendimiento autoencoder según tamaño de ventana para la señal periódica combinada con 1.1% de anomalías.

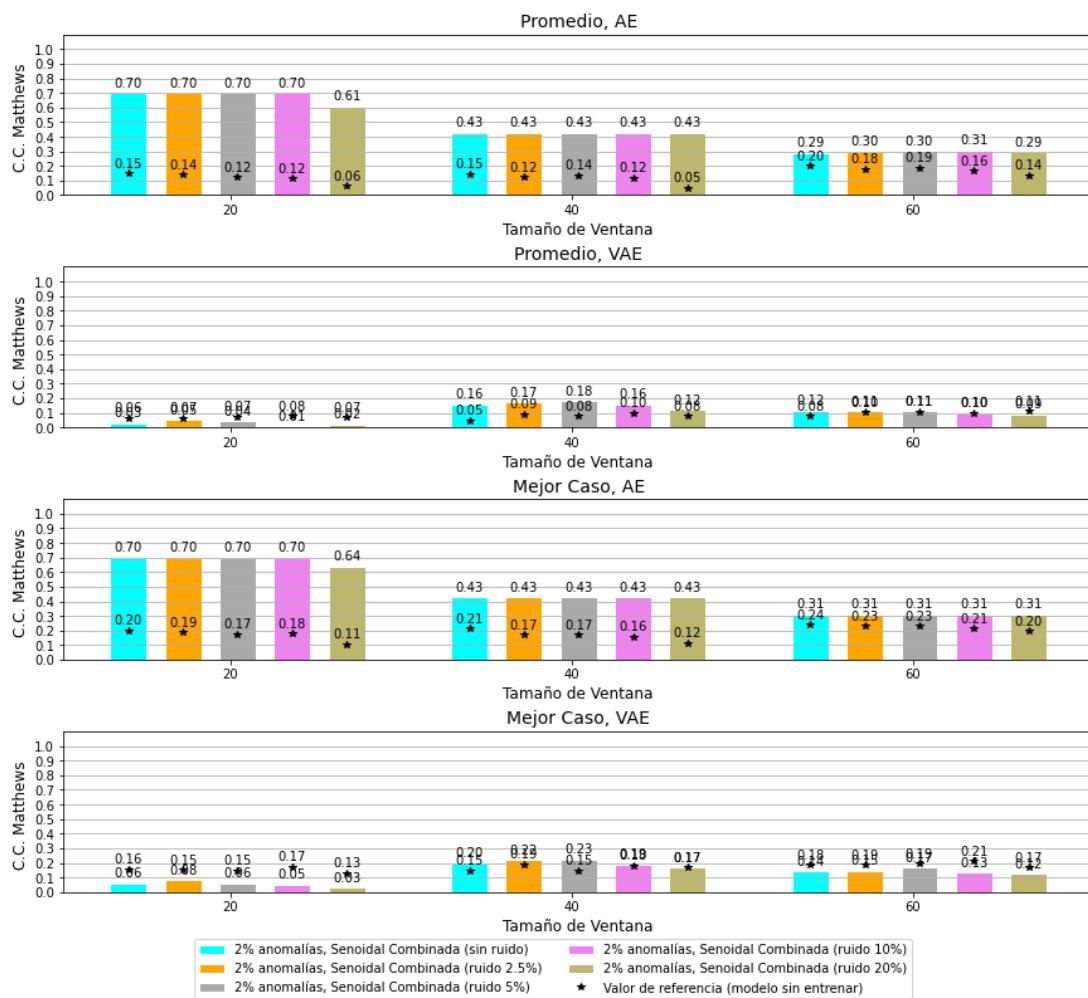


Figura 5.35: Rendimiento autoencoder según tamaño de ventana para la señal periódica combinada con 2% de anomalías.

5.4.4. Resultados para las Series Naturales

En la **Tabla C.4** se puede ver el mejor modelo para cada serie de forma individual. En el caso del autoencoder básico, el mejor modelo en general es el que tiene 20 neuronas en la capa intermedia y 10 neuronas en el bottleneck; mientras que el mejor para el autoencoder variacional es el que tiene 25 neuronas en la capa intermedia y 15 neuronas en el bottleneck.

Al igual que ocurre con las series periódicas, el rendimiento del AE básico supera al variacional en estas series. Y, al igual que ocurre con el MLP, el rendimiento resulta errático debido a que estas señales no tienen una forma tan bien marcada como las periódicas.

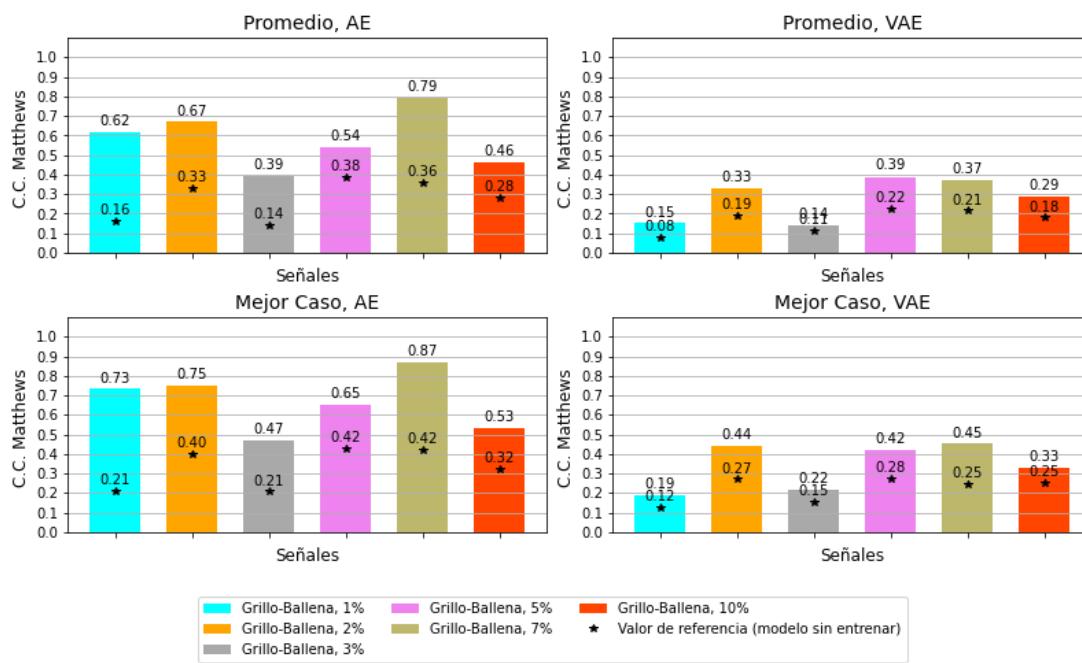


Figura 5.36: Rendimiento general del autoencoder para las señales grillo-ballena.

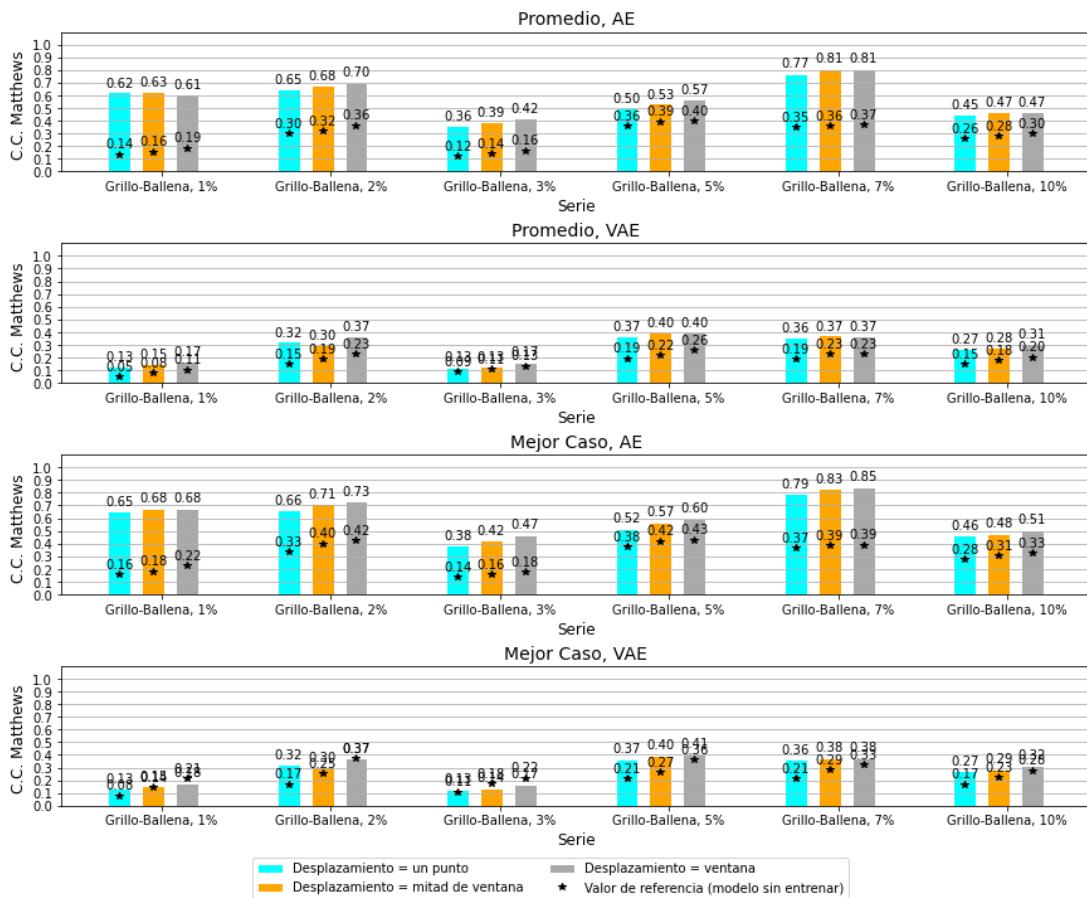


Figura 5.37: Rendimiento autoencoder según desplazamiento de ventana para las señales grillo-ballena.

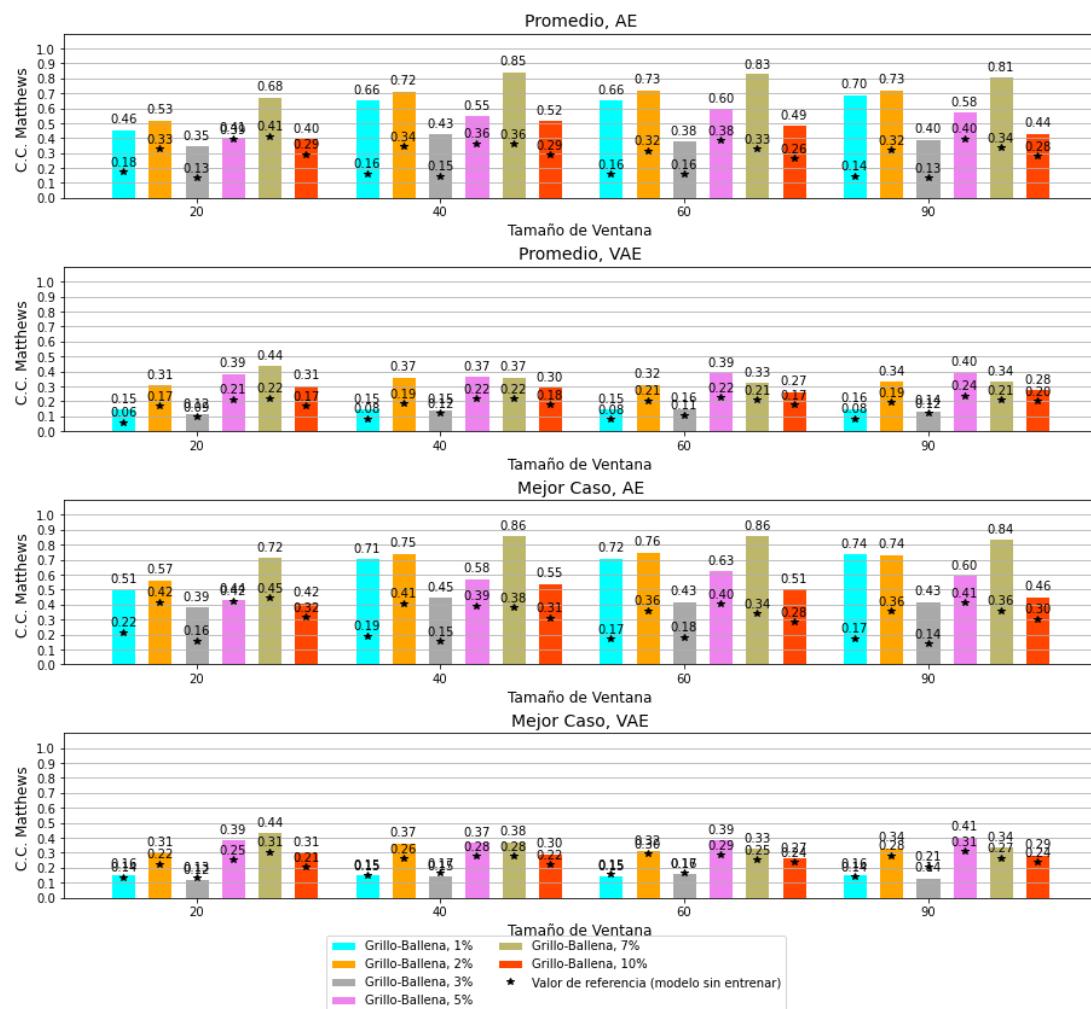


Figura 5.38: Rendimiento autoencoder según tamaño de ventana para las señales grillo-ballena.

5.5. Estructura del Código Fuente

En la [**Figura 5.39**](#) se puede ver la estructura del código fuente de este trabajo. En los directorios “*datos_series_naturales*” y “*datos_series_periodicas*” se encuentran los ficheros CSV de las propias series de datos, así como los cuadernos de jupyter que se usaron para crearlas. En el caso de las series naturales, además, se encuentran los archivos de sonido originales y los cuadernos para crear la gráfica de la [**Figura 5.19**](#), la [**Figura 5.20**](#) y la [**Figura 5.22**](#).

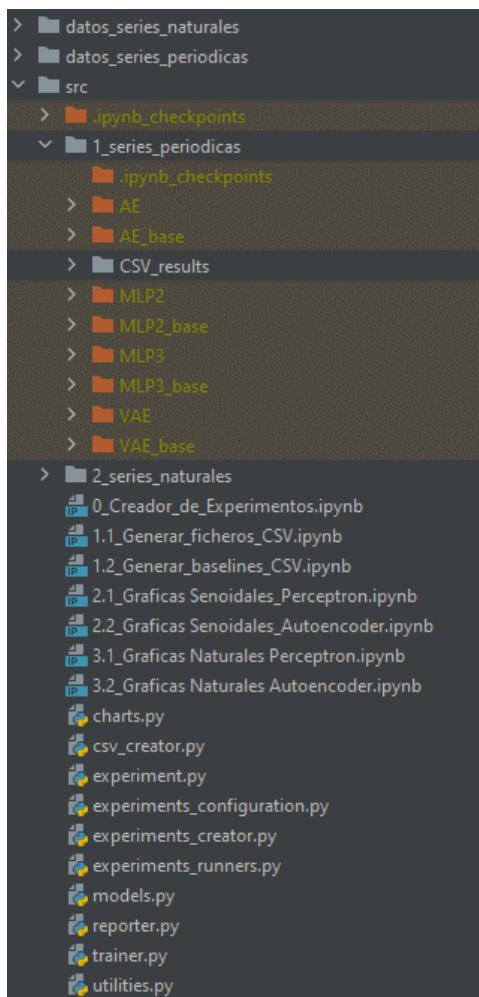


Figura 5.39: Estructura del código fuente del proyecto.

En el directorio “*src*” se encuentran los cuadernos y los ficheros de Python que se usaron para realizar el estudio, así como el directorio raíz de cada grupo de series, “*1_series_periodicas*” y “*2_series_naturales*”, que es donde se almacenan los modelos entrenados en cada experimento y los ficheros CSV que recopilan las evaluaciones de los modelos.

Esos directorios raíz contienen, a su vez, dos directorios por cada arquitectura de red: uno donde se almacenan los modelos entrenados y otro donde se almacenan los modelos sin entrenar, que sirven para calcular el valor de referencia de las gráficas. Todos esos ficheros están excluidos del sistema de control de versiones y del indexado del IDE debido a que son demasiados datos para ser gestionados por ambos programas. Un ejemplo de estos directorios son “*AE*” y “*AE_base*”, para los modelos entrenados y sin entrenar del autoencoder básico, respectivamente.

A continuación, se explica brevemente qué contiene cada fichero en “*src*”:

- **0_Creador_de_Experimentos.ipynb**: cuaderno que crea los scripts (ejemplo en Figura 5.6) para ejecutar los experimentos en Google Colab.
- **1.1_Generar_ficheros_CSV.ipynb**: cuaderno que recopila los datos de los modelos entrenados en ficheros CSV. El cuaderno que recopila los datos de los modelos sin entrenar es **1.2_Generar_baselines_CSV.ipynb**.
- **2.1_Graficas_Senoidales_Perceptron.ipynb**: cuaderno que crea las gráficas para el análisis de las series periódicas con la arquitectura del perceptrón. El cuaderno **2.2_Graficas_Senoidales_Autoencoder.ipynb** hace lo mismo, pero para la arquitectura del autoencoder.
- **3.1_Graficas_Naturales_Perceptron.ipynb**: cuaderno que crea las gráficas para el análisis de las series naturales con la arquitectura del perceptrón. El cuaderno **3.2_Graficas_Naturales_Autoencoder.ipynb** hace lo mismo, pero para la arquitectura del autoencoder.
- **charts.py**: fichero que contiene las funciones y clases empleadas para crear las gráficas de los experimentos.
- **csv_creator.py**: fichero que contiene las funciones y clases empleadas para crear y leer los ficheros CSV.
- **experiment.py**: fichero que contiene las clases empleadas para ejecutar los experimentos. En total son tres: *MLPExperiment*, *AutoencoderExperiment* y *AbstractExperiment*. Esta última contiene el código principal para interpolar las series, crear y guardar los modelos.

- **experiments_configuration.py**: fichero que contiene las funciones que definen los hiperparámetros de las redes. También hay cuatro clases de configuración, que se usan para obtener los nombres de los archivos necesarios para la recopilación de datos y la combinación de hiperparámetros correspondiente. Estas son: *Configuration* (abstracta), *MLP2Configuration*, *MLP3Configuration* y *AutoencoderConfiguration*.
- **experiments_creator.py**: fichero que contiene la clase que crea los scripts para Google Colab.
- **experiments_runners.py**: fichero que contiene las funciones que puede ejecutar el script de Google Colab. Son funciones que encapsulan la creación del objeto del experimento y validan que existe la ruta de almacenamiento en Google Drive.
- **models.py**: fichero que contiene las funciones que crean los modelos de redes neuronales. Son las siguientes: *create_MLP2_model*, *create_MLP3_model*, *create_AE_model* y *create_VAE_model*. También contiene las clases necesarias para crear la arquitectura del autoencoder variacional.
- **reporter.py**: fichero que contiene las clases que evalúan los modelos con el conjunto de prueba. Calculan el umbral para la predicción de clases, la matriz de confusión y el coeficiente de correlación de Matthews. Son tres clases: *MLPReporter*, *AutoencoderReporter* y *AbstractReporter*.
- **trainer.py**: fichero que contiene la clase *KFoldTrainer*, que es la encargada de entrenar los modelos usando la validación cruzada.
- **utilities.py**: fichero que contiene algunas funciones de utilidad.

[Página intencionadamente dejada en blanco]



Capítulo 6. Conclusiones y Trabajo Futuro

6.1. Conclusiones del Estudio

Este trabajo ha abordado el tema de la detección de anomalías en series de datos usando redes neuronales, centrándose en analizar el rendimiento de diferentes modelos y arquitecturas para determinar cuál es la mejor.

Se ha estudiado con la amplitud necesaria el problema de la detección de anomalías y se ha adquirido conocimiento sobre el entrenamiento y la evaluación de redes neuronales. Con esto, se han elaborado dos capítulos que pueden utilizarse como introducción para personas desconocedoras del tema y que deseen o precisen adquirir dichos conocimientos.

Se ha experimentado con dos tipos de series temporales sobre las que detectar anomalías: naturales y periódicas (estas últimas, creadas artificialmente). En ambos tipos de series se añadieron diferentes cantidades de anomalías y de ruido para observar cómo varía el rendimiento del clasificador. A la luz de los resultados descritos en el capítulo anterior, se plantean las siguientes conclusiones:

- Para las series naturales: no se aprecia una gran diferencia entre los resultados obtenidos con la arquitectura del autoencoder y la del perceptrón. La falta de una tendencia clara en los resultados se puede deber a: **(I)** la forma en que se distribuyen los segmentos anómalos; **(II)** que se hizo una asunción insuficiente al suponer que estas series tienen una estructura bien definida y diferenciable después de interpolarlas al rango $[-1, 1]$.
- Para las series periódicas: la arquitectura del perceptrón se ve muy afectada por el ruido, cosa que no ocurre con el autoencoder, debido al comportamiento de este como compresor. Seguramente un autoencoder con una arquitectura más compleja podría superar el rendimiento máximo del perceptrón. Lamentablemente, eso no se ha podido probar debido al coste temporal que supone repetir todos los experimentos, lo cual superaba la carga de trabajo del proyecto.

6.2. Conclusiones Personales

Este trabajo ha supuesto un refuerzo de los conocimientos teóricos sobre redes neuronales que ya tenía y un aprendizaje de lo que conlleva trabajar en un proyecto que haga uso de inteligencia artificial. Además:

- Me ha permitido aprender cómo es el trabajo experimental en un proyecto de investigación, aunque dichos proyectos tienen una extensión mayor a las 300 horas dispuestas para los trabajos de fin de grado.
- He adquirido conocimientos sobre el lenguaje Python y otras librerías usadas en el proyecto; sobre la implementación de redes neuronales usando Keras, incluyendo la implementación propia (en inglés, *custom implementation*) de un modelo, extendiendo clases de la librería; y sobre la herramienta Trello, para gestionar el flujo de trabajo del proyecto.
- He elaborado mi propia metodología para trabajar con *Google Colab*, minimizando la pérdida de datos y tiempo por desconexiones del servicio.

Me parece importante destacar una de las cosas que no interioricé suficientemente durante la realización del grado, pero que he aprendido con la realización de este trabajo, y es la importancia de estimar cuánto tiempo va a tardar, aproximadamente, la ejecución de una búsqueda. Recuerdo haber sufrido bastante angustia durante la primera ejecución del experimento del perceptrón al no haber hecho la estimación de tiempo y no saber que tardaría casi 96 horas en terminar.

Aparte de eso, también me llevo conocimiento sobre como trabajar con grandes cantidades de datos, ya que en total se han probado 13.896 configuraciones de red (se han entrenado 138.960 modelos) y he tenido que buscar la forma de: (I) acelerar la ejecución de los experimentos y descargar todos los datos; (II) resumir y encontrar la mejor configuración de propósito general entre todas ellas.

6.3. Trabajo Futuro

Como trabajo futuro, se podrían realizar las siguientes mejoras y ampliaciones:

- a) Buscar series que, tras interpolar sus datos, tengan una mayor diferencia entre sí que la de los grillos y la ballena, para realizar los experimentos de contaminación.
- b) Investigar más sobre la búsqueda heurística y probar si puede encontrar los mismos resultados que la búsqueda exhaustiva en menos tiempo.
- c) Probar si una arquitectura de autoencoder variacional más compleja puede mejorar los resultados obtenidos en este estudio.
- d) Durante la fase de entrenamiento del autoencoder, se escogió entrenar solamente con los datos etiquetados como normales. Sería interesante añadir otro experimento entrenando con todos los datos y comparar resultados.
- e) Podría ser interesante combinar varios modelos, incluso de diferentes arquitecturas, para ver cómo afecta al rendimiento. Esto se conoce como, **ensamblado, ensamblaje²⁴** o **ensemble** en inglés.

²⁴ <https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/>

[Página intencionadamente dejada en blanco]

Anexo A. Recorte de hiperparámetros MLP

En la Figura A.1 y Figura A.2 se pueden ver los hiperparámetros originales para el experimento del perceptrón multicapa (MLP) y en la Tabla A.1 y la Tabla A.2 se puede ver la cantidad de muestras de los conjuntos de datos.

Los hiperparámetros comunes a ambas arquitecturas son los siguientes:

- **Longitud de ventana:** 6 para las series periódicas (5, 10, 20, 40, 60, 80), 7 para las series naturales (5, 10, 20, 40, 60, 90, 120).
- **Desplazamiento de ventana** (4 valores): 1, $\frac{ventana}{3}$, $\frac{ventana}{2}$, ventana. Cuando la ventana es 5, se omite $\frac{ventana}{3}$ porque coincide con 1.
- **Cantidad de neuronas en la primera capa oculta** (3 valores): 25, 20, 15.
- **Ratio de dropout** (3 valores): 0.2, 0.3, 0.4.

Para el MLP de 3 capas se usan los siguientes hiperparámetros adicionales:

- **Cantidad de neuronas en la segunda capa oculta** (3 valores): 15, 10, 5.

Teniendo en cuenta que se usa un KFold de 10, que hay 20 series periódicas y 6 series naturales, se entrenaron 223.920 modelos en total (22.392 configuraciones):

- **MLP 2 capas, series periódicas:** $20 \text{ series} \cdot ((5 \text{ ventanas} \cdot 4 \text{ desplazamientos}) + (1 \text{ ventana} \cdot 3 \text{ desplazamientos})) \cdot 3 \text{ dropout} \cdot 3 \text{ neuronas} \cdot 10 \text{ KFold} = 41.400$ modelos.
- **MLP 3 capas, series periódicas:** $41.400 \cdot 3 \text{ neuronas} = 124.200$ modelos.
- **MLP 2 capas, series naturales:** $6 \text{ series} \cdot ((6 \text{ ventanas} \cdot 4 \text{ desplazamientos}) + (1 \text{ ventana} \cdot 3 \text{ desplazamientos})) \cdot 3 \text{ dropout} \cdot 3 \text{ neuronas} \cdot 10 \text{ KFold} = 14.580$ modelos.
- **MLP 3 capas, series naturales:** $14.580 \cdot 3 \text{ neuronas} = 43.740$ modelos.

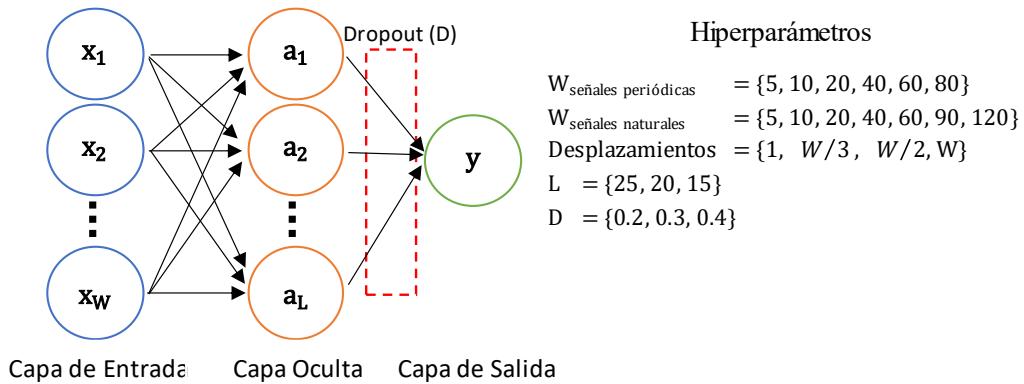


Figura A.1: Arquitectura e hiperparámetros iniciales para el MLP de 2 capas.

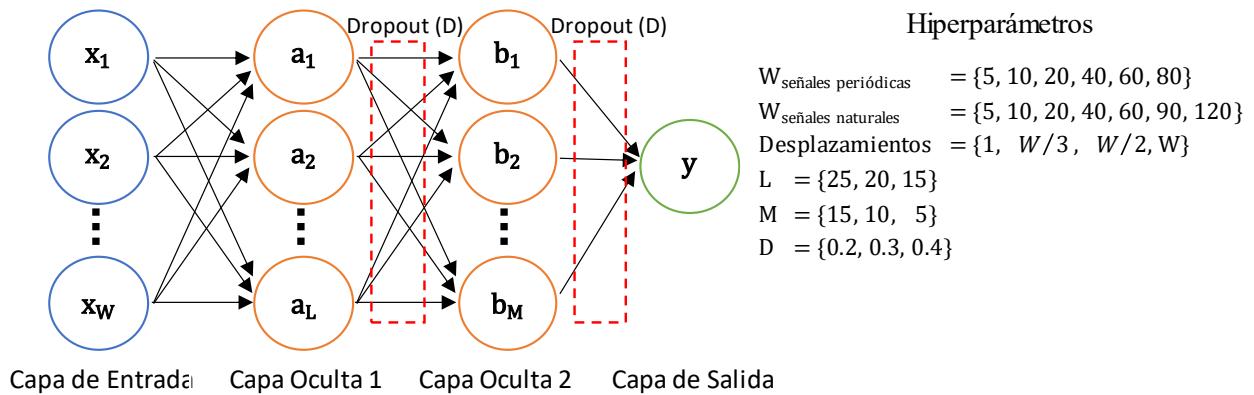


Figura A.2: Arquitectura e hiperparámetros iniciales para el MLP de 3 capas.

Desplazamiento	Un punto	Un tercio de ventana	Mitad de ventana	Ventana
Tamaño de Ventana				
5	11995	11995	5998	2399
10	11990	3997	2398	1199
20	11980	1997	1198	599
40	11960	920	598	299
60	11940	597	398	199
80	11920	459	298	149

Tabla A.1: Cantidad de muestras de los conjuntos de datos para las Series Periódicas.

Tamaño de Ventana \ Desplazamiento	Un punto	Un tercio de ventana	Mitad de ventana	Ventana
5	99995	99995	49998	19999
10	99990	33330	19998	9999
20	99980	16664	9998	4999
40	99960	7690	4998	2499
60	99940	4997	3332	1666
90	99910	3331	2221	1111
120	99880	2497	1665	833

Tabla A.2: Cantidad de muestras de los conjuntos de datos para las Series Naturales.

Debido a que la ejecución completa del experimento dura aproximadamente 96 horas ([Tabla A.3](#)), usando 70 instancias de Google Colab, se plantearon los siguientes recortes, con los que se redujo el tiempo de ejecución en un 13.54% (los nuevos tiempos están en la [Tabla 5.4](#) del capítulo 5):

- **Dropout:** se usan los valores 0.2 y 0.3 porque son los valores más frecuentes en las tablas de mejores modelos ([Tabla A.4](#) y [Tabla A.5](#)). Las frecuencias de cada valor, excluyendo la fila “general” son: 51 veces para 0.2; 27 veces para 0.3 y 26 veces para 0.4.
- **Desplazamiento de ventana:** se elimina el valor $\frac{ventana}{3}$ porque ya se estaba ignorando cuando la ventana tenía longitud 5, debido a que el redondeo inferior de la división da 1, que es un valor que ya se usa como hiperparámetro.
- **Longitud de ventana para series periódicas:** se elimina la ventana de 80 porque, como se ve en la [Figura A.3](#), es la que peores resultados da.
- **Longitud de ventana para series naturales:** se eliminan las ventanas de 5 y 120 porque los segmentos anómalos tienen longitudes entre 20 y 60 puntos, de modo que la ventana de 5 es muy pequeña y la de 120 muy grande en comparación. Aunque en la [Figura A.4](#) no se observa que la de 120 de resultados muy diferentes al resto.

	Series Periódicas	Series Naturales
Arquitectura MLP de 2 capas	13 horas	19 horas
Arquitectura MLP de 3 capas	15 horas	48 horas, 40 minutos
Total	28 horas	67 horas, 40 minutos
		95 horas, 40 minutos

Tabla A.3: Tiempo empleado para entrenar todos los modelos según la arquitectura inicial.

		Mejor MLP de 2 capas			Mejor MLP de 3 capas			
		Capa 1	Dropout	MCC	Capa 1	Capa 2	Dropout	MCC
Promedio	1.1% anomalías, Senoidal Simple	25	0,4	0,83	20	15	0,2	0,84
	1.1% anomalías, Senoidal Simple (ruido 2.5%)	25	0,4	0,71	25	15	0,2	0,74
	1.1% anomalías, Senoidal Simple (ruido 5%)	25	0,4	0,61	25	15	0,4	0,63
	1.1% anomalías, Senoidal Simple (ruido 10%)	25	0,4	0,51	25	15	0,2	0,55
	1.1% anomalías, Senoidal Simple (ruido 20%)	25	0,2	0,37	25	15	0,2	0,40
	2% anomalías, Senoidal Simple	25	0,3	0,63	20	15	0,2	0,77
	2% anomalías, Senoidal Simple (ruido 2.5%)	25	0,2	0,57	25	15	0,3	0,70
	2% anomalías, Senoidal Simple (ruido 5%)	25	0,3	0,53	25	15	0,3	0,64
	2% anomalías, Senoidal Simple (ruido 10%)	25	0,4	0,42	25	10	0,2	0,51
	2% anomalías, Senoidal Simple (ruido 20%)	25	0,2	0,24	25	15	0,4	0,31
	1.1% anomalías, Senoidal Combinada	25	0,4	0,75	25	15	0,2	0,81
	1.1% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,4	0,70	25	15	0,2	0,76
	1.1% anomalías, Senoidal Combinada (ruido 5%)	25	0,4	0,61	20	15	0,4	0,67
	1.1% anomalías, Senoidal Combinada (ruido 10%)	25	0,3	0,47	25	15	0,4	0,55
	1.1% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,35	25	15	0,2	0,39
	2% anomalías, Senoidal Combinada	25	0,4	0,61	25	15	0,2	0,74
	2% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,3	0,56	25	15	0,2	0,69
	2% anomalías, Senoidal Combinada (ruido 5%)	25	0,3	0,51	25	15	0,4	0,63
	2% anomalías, Senoidal Combinada (ruido 10%)	25	0,3	0,38	25	15	0,3	0,46
	2% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,26	25	15	0,2	0,32
Mejor Caso	1.1% anomalías, Senoidal Simple	25	0,3	0,98	15	15	0,3	0,99
	1.1% anomalías, Senoidal Simple (ruido 2.5%)	25	0,2	0,97	20	15	0,4	0,97
	1.1% anomalías, Senoidal Simple (ruido 5%)	25	0,2	0,91	25	10	0,3	0,92
	1.1% anomalías, Senoidal Simple (ruido 10%)	25	0,2	0,85	25	15	0,2	0,88
	1.1% anomalías, Senoidal Simple (ruido 20%)	25	0,3	0,76	25	10	0,3	0,79
	2% anomalías, Senoidal Simple	15	0,4	0,99	25	5	0,4	0,99
	2% anomalías, Senoidal Simple (ruido 2.5%)	25	0,4	0,95	25	15	0,4	0,95
	2% anomalías, Senoidal Simple (ruido 5%)	25	0,2	0,93	25	5	0,2	0,94
	2% anomalías, Senoidal Simple (ruido 10%)	25	0,2	0,85	25	15	0,2	0,86
	2% anomalías, Senoidal Simple (ruido 20%)	25	0,3	0,71	25	15	0,2	0,72
	1.1% anomalías, Senoidal Combinada	15	0,3	0,99	15	5	0,3	0,99
	1.1% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,4	0,95	25	15	0,3	0,96
	1.1% anomalías, Senoidal Combinada (ruido 5%)	25	0,4	0,93	25	15	0,2	0,94
	1.1% anomalías, Senoidal Combinada (ruido 10%)	25	0,2	0,84	20	15	0,2	0,84
	1.1% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,74	25	15	0,2	0,77
	2% anomalías, Senoidal Combinada	25	0,4	0,99	20	5	0,2	1,00
	2% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,3	0,97	20	15	0,2	0,97
	2% anomalías, Senoidal Combinada (ruido 5%)	25	0,2	0,95	25	15	0,2	0,95
	2% anomalías, Senoidal Combinada (ruido 10%)	25	0,2	0,82	25	15	0,2	0,85
	2% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,66	25	10	0,2	0,71
Mejor Modelo en General		25	0,4	0,53	25	15	0,2	0,60

Tabla A.4: mejores MLP para series periódicas.

		Mejor MLP de 2 capas			Mejor MLP de 3 capas			
		Capa 1	Dropout	MCC	Capa 1	Capa 2	Dropout	MCC
Promedio	Grillo-Ballena, 1%	25	0,3	0,55	25	10	0,2	0,56
	Grillo-Ballena, 2%	25	0,4	0,62	25	5	0,2	0,63
	Grillo-Ballena, 3%	25	0,2	0,40	25	15	0,2	0,46
	Grillo-Ballena, 5%	25	0,2	0,53	25	5	0,3	0,57
	Grillo-Ballena, 7%	25	0,3	0,84	25	15	0,2	0,89
	Grillo-Ballena, 10%	25	0,3	0,55	25	15	0,2	0,61
Mejor Caso	Grillo-Ballena, 1%	25	0,4	0,86	20	15	0,4	0,85
	Grillo-Ballena, 2%	20	0,4	0,77	25	15	0,3	0,78
	Grillo-Ballena, 3%	20	0,3	0,49	25	10	0,2	0,54
	Grillo-Ballena, 5%	25	0,3	0,63	25	5	0,2	0,65
	Grillo-Ballena, 7%	25	0,3	0,97	25	10	0,2	0,97
	Grillo-Ballena, 10%	25	0,2	0,65	25	15	0,2	0,69
Mejor Modelo en General		25	0,3	0,58	25	10	0,2	0,62

Tabla A.5: Mejores MLP para series naturales.

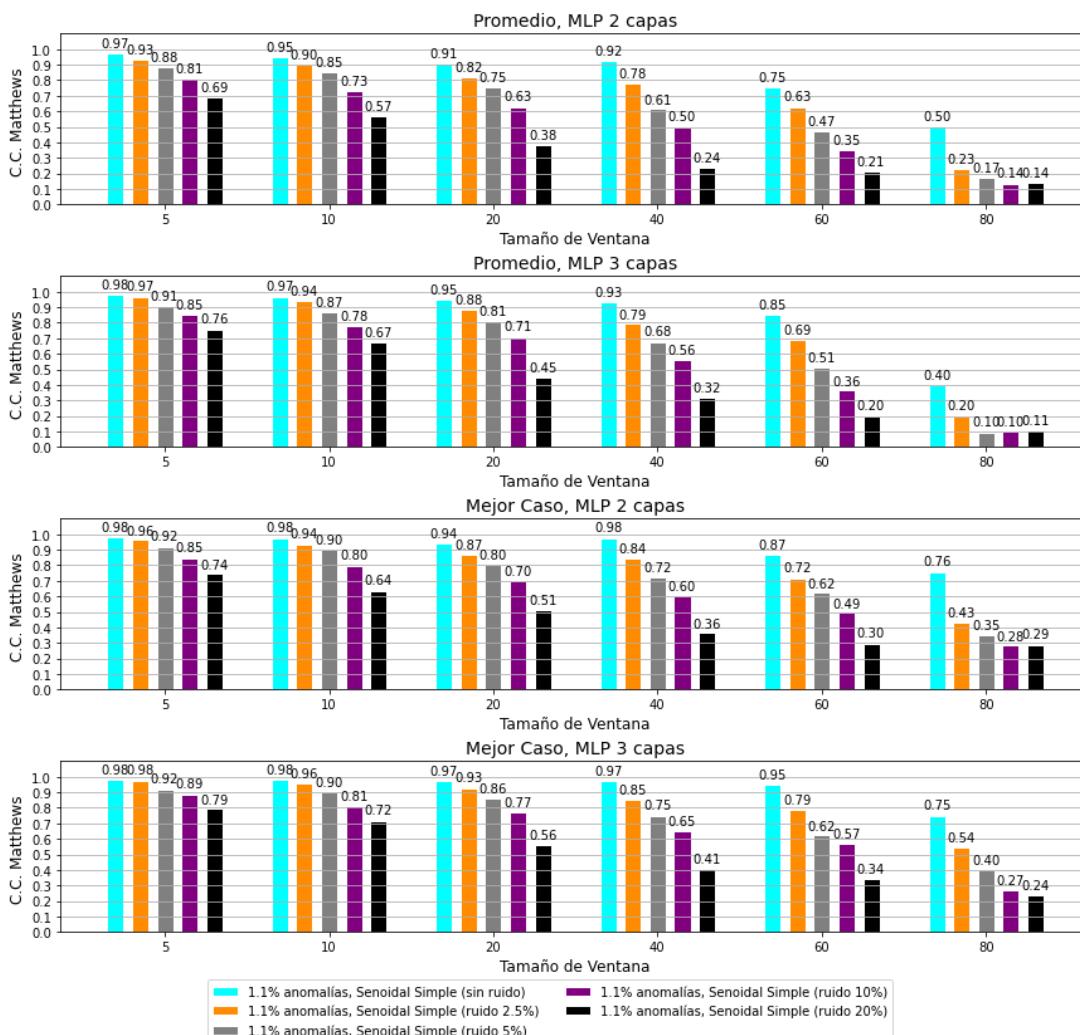


Figura A.3: Rendimiento en función del tamaño de ventana para la señal periódica simple con 1.1% de anomalías.

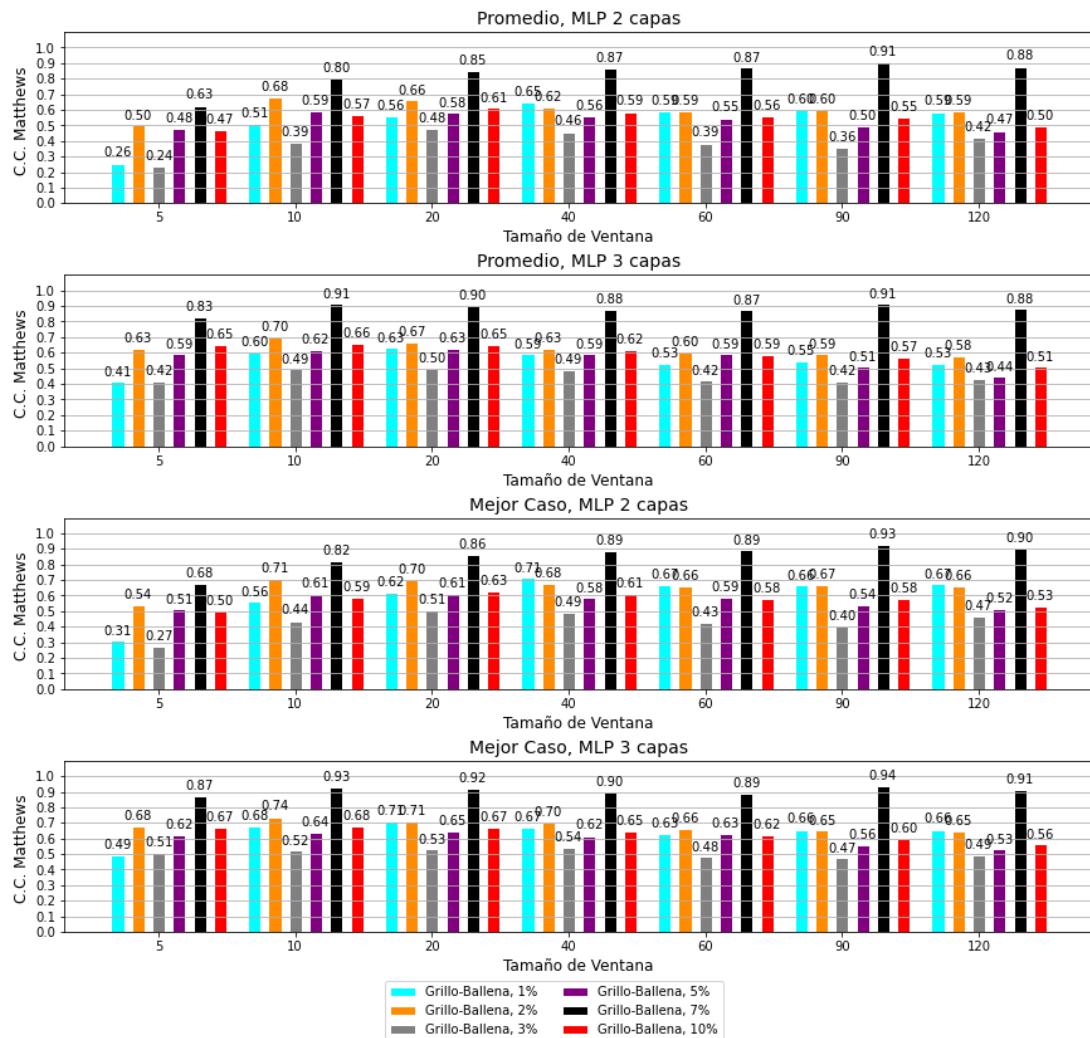


Figura A.4: Rendimiento en función del tamaño de ventana para las señales grillo-ballena.

Anexo B. Limitaciones de Google Colab

Los recursos que ofrece Google Colab varían con el tiempo para adaptarse a las fluctuaciones de demanda por parte de los usuarios, de modo que no están garantizados ni son ilimitados.

Según la propia web del servicio²⁵, las máquinas virtuales tienen una vida útil máxima de 12 horas y, además, se desconectan cuando permanecen inactivas durante cierto tiempo. Estos límites temporales varían en función del uso que hagan los usuarios, de modo que quienes realizan cálculos de larga duración o hayan usado más recursos en los últimos días, tienen más probabilidades de encontrar restricciones más severas (entre ellas, no disponer de GPUs).

En la práctica, las sesiones más largas que he podido usar en este trabajo han sido de 7 horas y 30 minutos (a veces sin poder usar GPUs). También he hecho las siguientes comprobaciones (resumen de un post de mi blog²⁶):

- Cuando hay mala cobertura Wifi, es posible que el servidor interprete que te has desconectado y detenga la máquina virtual, de modo que es importante conectarse a internet usando un cable.
- Cuando una máquina se detiene, los archivos guardados en ella se pierden, así que lo mejor es mover todos los archivos creados a Google Drive tan pronto como sea posible.
- Cada cuenta de Google puede tener como máximo 5 máquinas virtuales (cuadernos) ejecutándose simultáneamente, aunque este límite disminuye con el tiempo, por ejemplo: encadenando varias ejecuciones simultáneas de los 5 cuadernos.
- Si el código genera gráficas, estas sólo se muestran cuando la ejecución de la celda correspondiente termina. Si una celda tarda 7 horas en ejecutarse, significa que no se verán las gráficas hasta que pase ese tiempo, o peor aún, es posible que se detenga la máquina virtual porque se han acumulado tantas gráficas que se ha consumido toda la memoria RAM disponible. Para evitar este inconveniente, lo mejor es guardar las gráficas como imagen y cerrarlas.

²⁵ <https://research.google.com/colaboratory/faq.html#resource-limits>

²⁶ <https://medium.com/the-agile-crafter/17-days-running-google-colab-903ae0359919>

Anexo C. Tablas de Mejores Modelos

		Mejor MLP de 2 capas			Mejor MLP de 3 capas			
		Capa 1	Dropout	MCC	Capa 1	Capa 2	Dropout	MCC
Promedio	1.1% anomalías, Senoidal Simple	25	0,2	0,90	25	15	0,2	0,93
	1.1% anomalías, Senoidal Simple (ruido 2.5%)	25	0,2	0,81	25	15	0,2	0,86
	1.1% anomalías, Senoidal Simple (ruido 5%)	25	0,3	0,73	25	15	0,2	0,77
	1.1% anomalías, Senoidal Simple (ruido 10%)	25	0,2	0,63	25	15	0,2	0,69
	1.1% anomalías, Senoidal Simple (ruido 20%)	25	0,2	0,48	25	15	0,2	0,51
	2% anomalías, Senoidal Simple	25	0,3	0,80	25	15	0,3	0,89
	2% anomalías, Senoidal Simple (ruido 2.5%)	25	0,2	0,75	25	15	0,2	0,83
	2% anomalías, Senoidal Simple (ruido 5%)	25	0,2	0,70	25	15	0,3	0,78
	2% anomalías, Senoidal Simple (ruido 10%)	25	0,2	0,60	25	15	0,2	0,67
	2% anomalías, Senoidal Simple (ruido 20%)	25	0,3	0,42	25	10	0,2	0,48
	1.1% anomalías, Senoidal Combinada	25	0,2	0,83	25	15	0,2	0,88
	1.1% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,2	0,77	25	15	0,3	0,82
	1.1% anomalías, Senoidal Combinada (ruido 5%)	25	0,3	0,70	25	15	0,2	0,76
	1.1% anomalías, Senoidal Combinada (ruido 10%)	25	0,2	0,62	25	10	0,2	0,66
	1.1% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,47	25	15	0,3	0,51
	2% anomalías, Senoidal Combinada	25	0,3	0,76	25	15	0,2	0,86
	2% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,3	0,72	25	15	0,3	0,81
	2% anomalías, Senoidal Combinada (ruido 5%)	25	0,3	0,68	25	15	0,3	0,75
	2% anomalías, Senoidal Combinada (ruido 10%)	25	0,3	0,57	25	15	0,3	0,62
	2% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,43	25	15	0,2	0,49
Mejor Caso	1.1% anomalías, Senoidal Simple	25	0,2	0,98	15	10	0,3	0,99
	1.1% anomalías, Senoidal Simple (ruido 2.5%)	25	0,3	0,97	25	10	0,3	0,97
	1.1% anomalías, Senoidal Simple (ruido 5%)	25	0,2	0,91	20	10	0,2	0,92
	1.1% anomalías, Senoidal Simple (ruido 10%)	20	0,2	0,86	25	15	0,2	0,87
	1.1% anomalías, Senoidal Simple (ruido 20%)	25	0,2	0,76	25	15	0,2	0,78
	2% anomalías, Senoidal Simple	20	0,3	0,99	15	10	0,3	0,99
	2% anomalías, Senoidal Simple (ruido 2.5%)	25	0,3	0,95	25	10	0,3	0,95
	2% anomalías, Senoidal Simple (ruido 5%)	25	0,3	0,93	25	10	0,3	0,93
	2% anomalías, Senoidal Simple (ruido 10%)	20	0,2	0,85	25	15	0,2	0,86
	2% anomalías, Senoidal Simple (ruido 20%)	25	0,3	0,71	25	10	0,2	0,72
	1.1% anomalías, Senoidal Combinada	20	0,3	0,99	15	15	0,3	1,00
	1.1% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,3	0,95	25	15	0,3	0,96
	1.1% anomalías, Senoidal Combinada (ruido 5%)	25	0,2	0,92	25	10	0,3	0,94
	1.1% anomalías, Senoidal Combinada (ruido 10%)	25	0,2	0,84	25	10	0,2	0,84
	1.1% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,73	25	15	0,2	0,76
	2% anomalías, Senoidal Combinada	25	0,2	0,99	25	15	0,2	1,00
	2% anomalías, Senoidal Combinada (ruido 2.5%)	25	0,3	0,97	15	10	0,3	0,97
	2% anomalías, Senoidal Combinada (ruido 5%)	25	0,2	0,95	25	10	0,2	0,95
	2% anomalías, Senoidal Combinada (ruido 10%)	25	0,2	0,82	25	15	0,2	0,85
	2% anomalías, Senoidal Combinada (ruido 20%)	25	0,2	0,66	25	15	0,3	0,71
Mejor Modelo en General		25	0,3	0,67	25	15	0,2	0,73

Tabla C.1: Mejores modelos MLP para las Series Periódicas.

		Autoencoder Básico			Autoencoder Variacional		
		Capa Intermedia	Bottleneck	MCC	Capa Intermedia	Bottleneck	MCC
Promedio	1.1% anomalías, Senoidal Simple	20	15	0,75	15	5	0,44
	1.1% anomalías, Senoidal Simple (ruido 2.5%)	15	5	0,74	15	5	0,43
	1.1% anomalías, Senoidal Simple (ruido 5%)	15	10	0,72	15	5	0,40
	1.1% anomalías, Senoidal Simple (ruido 10%)	15	5	0,70	20	5	0,35
	1.1% anomalías, Senoidal Simple (ruido 20%)	15	5	0,66	25	15	0,30
	2% anomalías, Senoidal Simple	25	15	0,46	15	5	0,20
	2% anomalías, Senoidal Simple (ruido 2.5%)	25	15	0,46	15	5	0,20
	2% anomalías, Senoidal Simple (ruido 5%)	25	15	0,46	15	5	0,18
	2% anomalías, Senoidal Simple (ruido 10%)	25	15	0,46	15	5	0,17
	2% anomalías, Senoidal Simple (ruido 20%)	20	5	0,45	15	15	0,12
	1.1% anomalías, Senoidal Combinada	25	15	0,67	15	5	0,16
	1.1% anomalías, Senoidal Combinada (ruido 2.5%)	25	15	0,67	15	5	0,16
	1.1% anomalías, Senoidal Combinada (ruido 5%)	20	10	0,67	25	5	0,16
	1.1% anomalías, Senoidal Combinada (ruido 10%)	15	10	0,64	25	5	0,13
	1.1% anomalías, Senoidal Combinada (ruido 20%)	25	5	0,60	15	15	0,10
	2% anomalías, Senoidal Combinada	25	15	0,48	25	5	0,11
	2% anomalías, Senoidal Combinada (ruido 2.5%)	25	15	0,48	25	5	0,12
	2% anomalías, Senoidal Combinada (ruido 5%)	25	15	0,48	20	5	0,12
	2% anomalías, Senoidal Combinada (ruido 10%)	25	10	0,48	15	5	0,10
	2% anomalías, Senoidal Combinada (ruido 20%)	25	10	0,44	20	5	0,08
Mejor Caso	1.1% anomalías, Senoidal Simple	25	15	1,00	15	5	0,69
	1.1% anomalías, Senoidal Simple (ruido 2.5%)	15	5	0,97	25	5	0,70
	1.1% anomalías, Senoidal Simple (ruido 5%)	20	15	0,93	25	5	0,68
	1.1% anomalías, Senoidal Simple (ruido 10%)	20	5	0,85	25	5	0,69
	1.1% anomalías, Senoidal Simple (ruido 20%)	20	10	0,76	25	5	0,64
	2% anomalías, Senoidal Simple	25	15	0,67	15	10	0,25
	2% anomalías, Senoidal Simple (ruido 2.5%)	25	15	0,67	25	10	0,24
	2% anomalías, Senoidal Simple (ruido 5%)	25	15	0,67	15	5	0,25
	2% anomalías, Senoidal Simple (ruido 10%)	25	15	0,67	15	5	0,22
	2% anomalías, Senoidal Simple (ruido 20%)	15	5	0,66	15	5	0,20
	1.1% anomalías, Senoidal Combinada	25	15	0,92	20	5	0,50
	1.1% anomalías, Senoidal Combinada (ruido 2.5%)	25	15	0,92	15	5	0,48
	1.1% anomalías, Senoidal Combinada (ruido 5%)	25	15	0,92	15	5	0,47
	1.1% anomalías, Senoidal Combinada (ruido 10%)	20	5	0,85	25	5	0,44
	1.1% anomalías, Senoidal Combinada (ruido 20%)	20	5	0,73	20	5	0,36
	2% anomalías, Senoidal Combinada	25	15	0,71	20	15	0,34
	2% anomalías, Senoidal Combinada (ruido 2.5%)	25	15	0,71	25	5	0,31
	2% anomalías, Senoidal Combinada (ruido 5%)	25	15	0,71	25	10	0,34
	2% anomalías, Senoidal Combinada (ruido 10%)	25	5	0,71	15	5	0,32
	2% anomalías, Senoidal Combinada (ruido 20%)	20	5	0,66	20	5	0,25
Mejor Modelo en General		25	10	0,57	25	15	0,22

Tabla C.2: Mejores modelos autoencoder para las Series Periódicas.

		Mejor MLP de 2 capas			Mejor MLP de 3 capas			
		Capa 1	Dropout	MCC	Capa 1	Capa 2	Dropout	MCC
Promedio	Grillo-Ballena, 1%	25	0,3	0,58	25	5	0,2	0,60
	Grillo-Ballena, 2%	25	0,3	0,64	25	10	0,2	0,63
	Grillo-Ballena, 3%	25	0,3	0,43	25	15	0,2	0,47
	Grillo-Ballena, 5%	25	0,3	0,56	25	5	0,3	0,59
	Grillo-Ballena, 7%	25	0,2	0,86	25	15	0,2	0,90
	Grillo-Ballena, 10%	25	0,3	0,58	25	15	0,3	0,62
Mejor Caso	Grillo-Ballena, 1%	25	0,3	0,83	25	15	0,2	0,82
	Grillo-Ballena, 2%	25	0,3	0,76	25	15	0,3	0,78
	Grillo-Ballena, 3%	20	0,2	0,49	25	5	0,2	0,54
	Grillo-Ballena, 5%	25	0,3	0,63	25	15	0,2	0,65
	Grillo-Ballena, 7%	25	0,2	0,96	25	10	0,2	0,96
	Grillo-Ballena, 10%	25	0,3	0,65	25	10	0,2	0,69
Mejor Modelo en General		25	0,3	0,61	25	10	0,2	0,63

Tabla C.3: Mejores modelos MLP para las Series Naturales.

		Autoencoder Básico			Autoencoder Variacional		
		Capa Intermedia	Bottleneck	MCC	Capa Intermedia	Bottleneck	MCC
Promedio	Grillo-Ballena, 1%	15	10	0,62	20	10	0,15
	Grillo-Ballena, 2%	20	10	0,67	25	5	0,33
	Grillo-Ballena, 3%	25	10	0,39	20	15	0,14
	Grillo-Ballena, 5%	15	5	0,57	15	10	0,39
	Grillo-Ballena, 7%	20	15	0,79	15	5	0,37
	Grillo-Ballena, 10%	25	10	0,46	25	10	0,29
Mejor Caso	Grillo-Ballena, 1%	25	15	0,76	25	15	0,19
	Grillo-Ballena, 2%	20	15	0,80	25	15	0,44
	Grillo-Ballena, 3%	15	5	0,51	25	15	0,22
	Grillo-Ballena, 5%	20	5	0,66	25	15	0,42
	Grillo-Ballena, 7%	20	15	0,90	25	15	0,45
	Grillo-Ballena, 10%	20	15	0,54	25	5	0,33
Mejor Modelo en General		20	10	0,58	25	15	0,28

Tabla C.4: Mejores modelos autoencoder para las Series Naturales.

Bibliografía y Fuentes de Información

- [1] R. Chalapathy y S. Chawla, «Deep Learning for Anomaly Detection: A Survey,» *arXiv:1901.03407v2*, 2019.
- [2] V. Chandola, A. Banerjee y V. Kumar, «Anomaly Detection: A Survey,» *ACM Computing Surveys*, vol. 41, pp. 1-72, 2009.
- [3] C. Guerra Artal, «Redes neuronales 1: El perceptrón,» [En línea]. Available: <https://nbviewer.jupyter.org/url/cayetanoguerra.github.io/ia/nbpy/redneuronal1.ipynb>. [Último acceso: 16 Oct. 2020].
- [4] C. Santana Vega, «¿Qué es una Red Neuronal? Parte 1 : La Neurona | DotCSV,» 19 Mar. 2018. [En línea]. Available: <https://www.youtube.com/watch?v=MRlv2lwFTPg&t>. [Último acceso: 16 Oct. 2020].
- [5] Desconocido. [En línea]. Available: <https://sites.google.com/site/mayinteligenciarificial/unidad-4-redes-neuronales>. [Último acceso: 16 Oct. 2020].
- [6] C. Santana Vega, «¿Qué es el Descenso del Gradiente? Algoritmo de Inteligencia Artificial | DotCSV,» 04 Feb. 2018. [En línea]. Available: https://www.youtube.com/watch?v=A6FiCDoz8_4. [Último acceso: 17 Oct. 2020].
- [7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting.,» *Journal of Machine Learning Research*, nº 15, pp. 1929-1958, 2014.
- [8] J. Brownlee, «Machine Learning Mastery, How to Reduce Overfitting With Dropout Regularization in Keras,» 05 Dic. 2018. [En línea]. Available: <https://machinelearningmastery.com/how-to-reduce-overfitting-with-dropout-regularization-in-keras/>. [Último acceso: 18 Oct. 2020].
- [9] F. Chollet, «Keras API reference,» [En línea]. Available: https://keras.io/api/layers/regularization_layers/dropout/. [Último acceso: 18 Oct. 2020].
- [10] PyTorch Project, «PyTorch Documentation,» [En línea]. Available: <https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html>. [Último acceso: 18 Oct. 2020].
- [11] G. Pang, C. Shen, L. Cao y A. van den Hengel, «Deep Learning for Anomaly Detection: A Review,» *arXiv:2007.02500v2*, 2020.
- [12] N. Hubens, «Towards Data Science, Deep inside: Autoencoders,» 25 Feb. 2018. [En línea]. Available: <https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f>. [Último acceso: 03 Nov. 2020].

- [13] J. Rocca, «Understanding Generative Adversarial Networks (GANs),» 07 Ene. 2019. [En línea]. Available: <https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29>. [Último acceso: 04 Nov. 2020].
- [14] D. Radigan, «Atlassian Agile Coach,» [En línea]. Available: <https://www.atlassian.com/es/agile/kanban>. [Último acceso: 18 Oct. 2020].
- [15] D. Chicco y G. Jurman, «The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation,» 02 Enero 2020. [En línea]. Available: <https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12864-019-6413-7>. [Último acceso: 26 Enero 2021].
- [16] F. Pedregosa y et al., «Scikit Learn, Confusion matrix,» [En línea]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html. [Último acceso: 26 Enero 2021].
- [17] S. Kim, K. Choi, H.-S. Choi, B. Lee y S. Yoon, «Towards a Rigorous Evaluation of Time-series Anomaly Detection,» *arXiv:2109.05257v2*, 2022.
- [18] J. Brownlee, «Machine Learning Mastery, How to use Data Scaling Improve Deep Learning Model Stability and Performance,» 04 Feb 2019. [En línea]. Available: <https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/>. [Último acceso: 10 Mar 2021].
- [19] J. Brownlee, «Machine Learning Mastery, How to Configure the Number of Layers and Nodes in a Neural Network,» 27 Jul 2018. [En línea]. Available: <https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/>. [Último acceso: 10 Mar 2021].
- [20] F. Pedregosa y et al., «Scikit Learn, Cross Validation,» [En línea]. Available: https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation. [Último acceso: 29 Enero 2021].
- [21] J. Brownlee, «Machine Learning Mastery, A Gentle Introduction to k-fold Cross-Validation,» 03 Agosto 2018. [En línea]. Available: <https://machinelearningmastery.com/k-fold-cross-validation/>. [Último acceso: 29 Enero 2021].
- [22] J. Brownlee, «Machine Learning Mastery, A Gentle Introduction to Threshold-Moving for Imbalanced Classification,» 10 Febrero 2020. [En línea]. Available: <https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/>. [Último acceso: 15 Julio 2020].
- [23] Desconocido, «Acerting Art, Sonidos de la Naturaleza para Dormir y Relajarse, Grillos Noche de Verano,» 06 Abr. 2014. [En línea]. Available: <https://www.youtube.com/watch?v=eKmRkS1os7k>. [Último acceso: 08 Jul. 2020].

- [24] J. Lewis, «8 Hours of Whale Sounds Deep Underwater for Sleep and Relaxation,» 01 Nov 2015. [En línea]. Available: <https://www.youtube.com/watch?v=nDqP7kcr-sc>. [Último acceso: 08 Jul. 2020].
- [25] F. Pedregosa y et al., «Scikit Learn, Neural network models (supervised),» [En línea]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html. [Último acceso: 11 Mar 2021].
- [26] J. Brownlee, «Machine Learning Mastery, Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,» 03 Jul 2017. [En línea]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>. [Último acceso: 17 Mar 2021].
- [27] J. Rocca, «Understanding Variational Autoencoders (VAEs),» 24 Sep 2019. [En línea]. Available: <https://towardsdatascience.com/understanding-variational-autoencoders-vae-f70510919f73>. [Último acceso: 12 Mar 2022].
- [28] S. Chaudhary, «Towards Data Science, A High-Level Guide to Autoencoders,» 22 Ago 2019. [En línea]. Available: <https://towardsdatascience.com/a-high-level-guide-to-autoencoders-b103ccd45924>. [Último acceso: 13 Mar 2022].
- [29] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano y L. Benini, «Anomaly Detection using Autoencoders in High Performance,» *arXiv:1811.05269v1*, 2018.
- [30] F. Pedregosa y et al., «Scikit Learn, Matthews Correlation Coefficient,» [En línea]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html#matthews-ccorcoef. [Último acceso: 26 Enero 2021].
- [31] J. Zürn, 24 Feb 2019. [En línea]. Available: <https://jannik-zuern.medium.com/but-what-is-an-autoencoder-26ec3386a2af>. [Último acceso: 12 Mar 2022].
- [32] A. Yan-Tak Ng, «Coursera, What is End-to-end Deep Learning?,» deeplearning.ai, [En línea]. Available: <https://www.coursera.org/lecture/machine-learning-projects/what-is-end-to-end-deep-learning-kOKIk>. [Último acceso: 27 Oct. 2020].
- [33] S. Glen, «Statistics How To, Dimensionality & High Dimensional Data,» 10 Oct. 2016. [En línea]. Available: <https://www.statisticshowto.com/dimensionality/>. [Último acceso: 22 Oct. 2020].
- [34] S. Glen, «Statistics How To, Prior Distribution: Simple Definition,» 30 Jul. 2020. [En línea]. Available: <https://www.statisticshowto.com/prior-distribution/>. [Último acceso: 30 Oct. 2020].