Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Mención en Computación

Detección automática de microdespertares en niños con apnea obstructiva del sueño mediante técnicas de *deep learning*

Autor:

Héctor Hugo de la Torre Díaz

Tutores:

Benjamín Sahelices Fernández

Fernando Vaquerizo Villar

Agradecimientos

Me gustaría dedicar unas palabras de agradecimiento a todas las personas que me han acompañado y apoyado durante estos 5 años de trayectoria universitaria.

En primer lugar, a mis tutores Benjamín y Fernando, por su dedicación y ayuda constante para guiar este Trabajo Fin de Grado y proporcionarme los conocimientos necesarios para realizar este proyecto.

A mi familia, por su paciencia, comprensión y apoyo en todo momento.

A mis amigos y compañeros, tanto de Valladolid como del Erasmus, que me han acompañado en muchas aventuras y me han animado a continuar cuando más lo necesitaba.

Por último, a los voluntarios de ESN Valladolid, por hacerme redescubrir mi ciudad natal y ser mi segunda familia. Gracias por tanto.

Resumen

La apnea obstructiva del sueño (AOS) infantil es una enfermedad caracterizada por pausas respiratorias y alteración del sueño que llega a afectar hasta el 5 % de los niños y que tiene consecuencias negativas para su salud y desarrollo. A través de una polisomnografía noctura (PSG), se puede diagnosticar la AOS y medir su severidad. Sin embargo, la PSG es una prueba compleja, costosa y cuya posterior interpretación por un especialista conlleva mucho tiempo. Por ello, se buscan alternativas más rápidas y sencillas que permitan automatizar la interpretación de la PSG.

El objetivo de este Trabajo Fin de Grado (TFG) ha sido evaluar la utilidad de técnicas de *deep learning*, en concreto redes neuronales convolucionales (CNN), aplicadas sobre la señal de un electroencefalograma (EEG) para la la detección automática de microdespertares (*arousals*) en niños con sospecha de AOS. Los *arousals* ayudan a analizar las fases de sueño de los niños y medir el grado de severidad de la AOS. Sin embargo, los estudios previos que han abordado la detección automática de *arousals* se han centrado únicamente en pacientes adultos, lo que, junto con las diferencias de la AOS en la población adulta provoca que los modelos de clasificación no sean fácilmente generalizables.

Para lograr dicho objetivo, se ha utilizado la base de datos *Childhood Adenotonsillectomy Trial* (CHAT), que contiene 1633 registros de EEG válidos procedentes de la PSG de niños con sospecha de AOS. A partir del canal C4-M1 del EEG de estos registros, se han entrenado diferentes arquitecturas de CNN, con diferentes configuraciones de los siguientes hiperparámetros para conseguir el mejor modelo y medir su rendimiento en la detección de *arousals* en cada época de 30 segundos: número de bloques convolucionales, número de segmentos de entrada, balanceo de clases y probabilidad de *dropout*.

El rendimiento superior fue obtenido por el modelo más profundo de 7 bloques convolucionales y tasa de *dropout* de 0.1, procesando los datos para atajar el problema de las clases desbalanceadas e incluyendo la época anterior y la época posterior como contexto temporal, logrando una precisión del 90 %, junto a una sensibilidad de 89 %, especificidad del 90 %, área bajo la curva ROC (AUROC) de 0.9589 y área bajo la curva *precision-recall* (AUPRC) de 0.7487. Estos resultados sugieren que el análisis del EEG mediante técnicas de *deep learning* es de gran utilidad para la detección automática de *arousals* en niños con sospecha de AOS.

Abstract

Pediatric obstructive sleep apnea (OSA) is a disease characterized by breathing pauses and sleep disturbance that affects up to 5 % of children and has negative consequences for their health and development. Through an overnight polysomnography (PSG), OSA can be diagnosed and measure its severity. However, the PSG is a complex, expensive test that is time-consuming to be interpreted by a specialist. Therefore, faster and simpler alternatives are needed to automate the interpretation of the PSG.

The aim of this Bachelor's Thesis has been to evaluate the utility of deep learning techniques, specifically convolutional neural networks (CNN) applied on the signal of an electroencephalogram (EEG) for the automatic detection of arousals in children with suspected OSA. Arousals contribute to study the child's sleep phases and measure the degree of severity of an OSA. However, previous studies that have addressed automatic detection of arousals have focused only on adult patients, which, together with the differences in OSA on adults, it makes the classification models not easily generalizables.

To achieve this goal, we used the Childhood Adenotonsillectomy Trial database (CHAT), containing 1633 valid EEG records from PSGs of children with suspected AOS. Using the C4-M1 channel of the EEG from these records, different CNN architectures were trained, with different settings of the following hyperparameters to get the best model and measure its performance in detecting arousals in each 30 second epoch: number of convolutional blocks, number of segments, class balancing, and probability of dropout.

The best performance was obtained by the deepest CNN model with 7 convolutional layers and a dropout probability of 0.1, processing the data to tackle the problem of unbalanced classes and including the epoch before and epoch after as temporal context, achieving an accuracy of 90 %, along with a sensitivity of 89 %, specificity of 90 %, area under the ROC curve (AUROC) of 0.9589 and area under the precision-recall curve (AUPRC) of 0.7487. These results suggest that analysis of EEG using deep learning techniques is of great utility for the automatic detection of arousals in children with suspected AOS.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

"El hito siguiente en la historia de la IA: aplicar una ingeniería inversa al cerebro humano."

- Michio Kaku [1]

La Inteligencia Artificial (IA) ha evolucionado mucho a día de hoy, de modo que ya no es fácil determinar que imagen es real o ficticia, que artículos han podido ser escritos por un ordenador, o que música ha sido realmente compuesta por un humano. La constante evolución de las nuevas tecnologías, y en concreto de las técnicas de Aprendizaje automático o *Machine Learning*, ha permitido que diversos campos como las finanzas, la medicina, las comunicaciones, o la seguridad experimenten un crecimiento exponencial, desarrollándose a una velocidad nunca antes vista en la historia [2].

Y aún no hemos tocado techo. Más bien, se podría afirmar que la revolución correspondiente a la inteligencia artificial acaba de empezar. Existen diferentes tipos de tecnologías englobadas en ella como son el *Machine Learning/Deep Learning*, Aprendizaje supervisado, no supervisado, semi-supervisado o por refuerzo. Además de disponer de una gran variedad de técnicas: árboles de decisión, máquinas de vectores de soporte, redes neuronales... Año tras año salen nuevos métodos y formas de IA que manifiestan que todavía hay mucho por descubrir y experimentar [3].

Como bien se ha mencionado anteriormente, una de las grandes salidas de la IA es su aplicación al mundo de la medicina. La apnea obstructiva del sueño infantil, un trastorno del sueño caracterizado por la corta interrupción total o parcial del flujo de aire en las vías respiratorias, afecta a alrededor del 2 %-5 % de niños con edades comprendidas entre los 2 y los 6 años [4].

Esta enfermedad es diagnosticada y estudiada a través de una polisomnografía (PSG) nocturna, un estudio del sueño que recoge muchos parámetros fisiológicos que posteriormente deben ser

analizados manualmente por profesionales, lo que conlleva un tiempo y esfuerzo enorme. Además, dada la dificultad de unificar en un manual las características propias de cada sujeto, los resultados de la PSG pueden variar según el profesional que lo analice. Es por esto que desde hace tiempo se recurre a algoritmos de IA para automatizar el proceso de análisis de una PSG [5].

El objetivo de este trabajo es diseñar y evaluar un método automático de detección de micro-despertares (*arousals*) en niños con sospecha de apnea del sueño mediante la aplicación de una arquitectura de *deep learning* sobre un único canal del electroencefalograma (EEG). Para lograr dicho objetivo, se va a hacer uso de las polisomnografías que componen la base de datos *Childhood Adenotonsillectomy Trial* (CHAT) proporcionada por el *National Sleep Research Resource* [6], con registros de más de 1000 niños que padecen apnea del sueño infantil.

Esta memoria está dividida en 7 capítulos, siendo este primer capítulo el introductorio al trabajo realizado donde se expone el objetivo de este proyecto, la motivación subyacente y la estructura de la memoria del proyecto.

En el capítulo 2 de esta memoria se introducen las ideas principales sobre la apnea del sueño, la polisomnografía (PSG), el EEG y los *arousals*, además de la base de datos utilizada. En el capítulo 3 se introducen los métodos teóricos que modelan las redes neuronales, y PyTorch, la biblioteca de aprendizaje automático con la que se han construido las redes neuronales de este trabajo.

En el capítulo 4 se elabora la planificación de este proyecto, desagregado en todas las tareas a realizar y el tiempo estimado de realización, junto con un análisis de riesgos que pueden afectar al objetivo del proyecto y el cumplimiento del plan de planificación.

En el capítulo 5 se expone el tratamiento de los datos realizado, esto es el procesamiento de las señales llevado a cabo, la generación de etiquetas que identifican dónde se han producido *arousals* y la distribución del conjunto de datos para entrenamiento, validación y test. Asimismo, se detalla la arquitectura del modelo óptimo y las métricas utilizadas para evaluar el rendimiento de clasificación.

En el capítulo 6 se explican las variaciones de hiperparámetros objeto de estudio, junto a los resultados obtenidos y su posterior discusión en la detección de *arousals*.

El último capítulo de esta memoria está destinado a las conclusiones recabadas de este proyecto y a la presentación de posibles líneas futuras de investigación.

Capítulo 2

Contexto clínico

En este capítulo se a va a explicar lo fundamental sobre la enfermedad de la apnea del sueño infantil, la PSG o prueba utilizada para diagnosticar trastornos del sueño, y las señales registradas mediante la PSG, en concreto la señal de un EEG, que será la utilizada para entrenar la red neuronal posteriormente. Por otra parte, se introduce la base de datos CHAT utilizada en este TFG y las diferencias de este trabajo respecto a estudios similares.

2.1. Apnea obstructiva del sueño infantil

La apnea obstructiva del sueño (AOS) infantil se define como "un trastorno respiratorio del sueño, caracterizado por una obstrucción parcial prolongada de la vía aérea superior y/u obstrucción intermitente completa que interrumpe la ventilación normal durante el sueño y los patrones normales del mismo" [7]. Entre los síntomas principales están los ronquidos (con pausas intermitentes, habitualmente), alteración del sueño, problemas de comportamiento durante el día, o menos frecuente la somnolencia durante el día. En caso de no ser tratada, la AOS puede originar consecuencias muy negativas para la salud de los niños, como la hipertensión, disfunción cardiaca, inflamación sistémica, retrasos en el desarrollo y deterioro neurocognitivo [4].

La prevalencia de la apnea del sueño infantil es de alrededor del 2 %-5 % en niños con edades comprendidas entre los 2 y los 6 años [4]. Debido a su alta prevalencia, así como a sus efectos negativos en la salud de los niños, el diagnóstico y tratamiento temprano de esta enfermedad es fundamental, dado que se ha demostrado que puede afectar a la calidad de vida del niño, manifestándose en problemas de concentración, irritabilidad, disminución del interés en las actividades diarias o dificultades en la relación con su familia y compañeros [8].

2.2. Polisomnografía (PSG)

La PSG nocturna es un estudio del sueño que registra múltiples señales neurofisiológicas y cardiorespiratorias durante el sueño de los niños. Se suele realizar en un centro especializado, como puede ser un hospital, aunque también es posible realizarla en el domicilio del paciente [9].

La PSG está monitorizada siempre por un técnico especialista del sueño. Durante esta prueba se registran diferentes registros biomédicos, entre los que se encuentra el EEG, la saturación de oxígeno en sangre, el flujo aéreo oronasal, así como el esfuerzo y frecuencia respiratoria, la frecuencia cardíaca vía electrocardiograma (ECG), los movimientos oculares a través de un electrooculograma (EOG), la posición del cuerpo y el electromiograma, entre otros [10].

A partir de la PSG, los técnicos del sueño anotan las fases del sueño y los eventos cardiorespiratorios, con el fin de obtener un diagnóstico para trastornos del sueño como la apnea obstructiva del sueño (AOS), la narcolepsia o las convulsiones nocturnas. Además, se calcula el índice de apnea-hipopnea (IAH) que equivale al número de eventos de apnea e hipopnea por hora de sueño (e/h) del paciente. Este índice se utiliza para determinar la presencia y severidad de la AOS en niños. A partir del IAH, se distinguen 4 grupos de severidad dentro de la enfermedad [7] [11]:

- No AOS: $IAH < 1$ e/h.
- AOS leve: $1 \leq IAH < 5$ e/h.
- AOS moderada: $5 \leq IAH < 10$ e/h.
- AOS grave: $IAH \geq 10$ e/h.

2.2.1. Tipos de polisomnografía

La PSG nocturna sirve para analizar las etapas del sueño, microdespertares del sueño, eventos respiratorios, la actividad de las ondas cerebrales, los ritmos cardíacos y los movimientos musculares. Permite evaluar tanto trastornos respiratorios del sueño, entre los que se encuentra la AOS, así como movimientos anormales durante el sueño [10].

Sin embargo, una PSG de diagnóstico por lo general requiere que el niño duerma al menos una noche en una unidad especializada, fuera de su ambiente habitual y con múltiples sensores por su cuerpo que pueden resultar incómodos. Además, es necesario disponer de personal cualificado que supervise toda la prueba y que registre manualmente los eventos de apnea-hipopnea que se produzcan [12].

Todas estas dificultades han motivado el uso de equipos portátiles como alternativa a la PSG de

diagnóstico estándar [13]. Tanto es así que podemos distinguir, además de la PSG de diagnóstico estándar, otros tres tipos de estudios del sueño según el número y tipo de las señales registradas [14].

PSG portátil completa

La PSG registra un mínimo de siete canales, incluyendo EEG, EOG, electromiograma submentoniano, ECG o frecuencia cardíaca, flujo aéreo, esfuerzo respiratorio y la saturación de oxígeno en sangre. En este caso, no se requiere la supervisión de personal cualificado. Los resultados de la PSG portátil permiten calcular el IAH e identificar las fases del sueño.

Test portátil modificado de apnea del sueño

En este tipo de estudio, se registran señales relacionadas con la respiración como la ventilación (un mínimo de dos señales de movimiento respiratorio o una señal de movimiento respiratorio y el flujo aéreo), además del ECG o ritmo cardíaco, y la saturación de oxígeno en sangre.

Registro continuo de uno o dos parámetros biomédicos

Sólo se registra una o dos señales fisiológicas, siendo tradicionalmente la saturación del oxígeno en sangre una de esas medidas. También es incluido en este tipo de PSG todo equipamiento que no cumple los criterios del test portátil modificado.

2.3. Electroencefalograma

El electroencefalograma (EEG) es una prueba neurofisiológica no invasiva que registra la actividad bioeléctrica del cerebro cuando el paciente está en reposo, vigilia o sueño mediante un equipo especializado. Es una prueba segura y no invasiva, acompañada siempre por un técnico encargado del registro de las señales [15].

Un EEG no solo es útil para diagnosticar trastornos del sueño, sino también para identificar y tratar tumores cerebrales, trastornos convulsivos como la epilepsia, disfunciones cerebrales, encefalitis, daños cerebrales causados por lesiones u otros tipos de trastornos cerebrales [15].

Para registrar el EEG, al paciente se le deben colocar unos electrodos en la cabeza siguiendo el Sistema 10-20, tal y como se muestra en la Figura 2.1.

Por lo general, la mayoría de estudios relacionados con el diagnóstico de enfermedades del sueño y que hacen uso de la PSG, se centran en utilizar un único canal de EEG para simplificar el

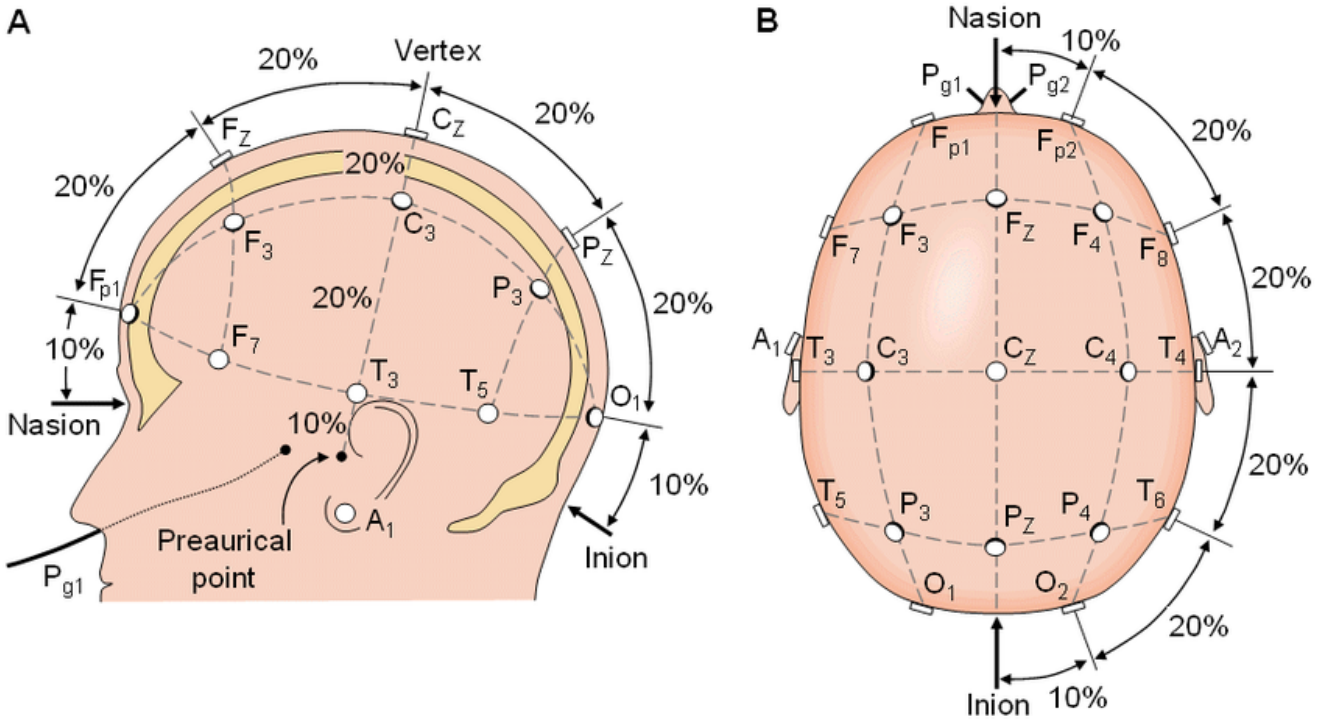


Figura 2.1: Posicionamiento de electrodos en un EEG según el Sistema 10-20

proceso y reducir la incomodidad de la prueba [16]. En este Trabajo Fin de Grado se ha utilizado la derivación C4-M1 del EEG, debido a que es una de las recomendaciones dadas por la Academia Americana de Medicina del Sueño (*American Academy of Sleep Medicine*, AASM) en estudios del sueño [11].

2.3.1. Microdespertares

Los microdespertares o *arousals* son identificados en la fase REM (*Rapid Eye Movement*) y en los niveles 1 (N1), 2 (N2) y 3 (N3) de sueño no-REM cuando hay cambios abruptos en la frecuencia de la señal del EEG, entre los que se encuentran las bandas de frecuencia *alpha* (8-13 Hz), *theta* (4-8 Hz) y frecuencias mayores de 16 Hz (sin contar *spindles*), con una duración mayor a 3 segundos y precedidos de al menos 10 segundos de sueño estable. El reglamento completo para el etiquetado de *arousals* y otros eventos del sueño puede consultarse en el manual de la AASM [11].

2.4. Base de datos CHAT

CHAT (*Childhood Adenotonsillectomy Trial*) [6] es un estudio diseñado para determinar la eficacia de realizar una operación de adenotonsilectomía (*early adenotonsillectomy*, eAT) en niños de entre 5 y 9.9 años que padecen AOS. Es un estudio multicentro, realizado en las siguientes instituciones de los Estados Unidos de América (EE.UU.): *Children's Hospital of Pennsylvania, Philadelphia, PA; Cincinnati Children's Medical Center, Cincinnati, OH; Rainbow Babies and Children's Hospital, Cleveland, OH; Boston Children's Hospital, Boston, MA; Cardinal Glennon Children's Hospital, St. Louis, MI; Montefiore Medical Center, Bronx, NY*. Asimismo, el *National Heart, Lung and Blood Institute* de los EE.UU., NHLBI) participó en calidad de colaborador [17][18].

El estudio se compone de 1639 estudios de sueño válidos realizados en el centro especializado *Brigham and Women´s Sleep Reading Center*. 453 niños fueron seleccionados en total para continuar con el estudio y aleatoriamente recibir o no la operación de eAT. A estos sujetos se les repitió la PSG a los 7 meses para poder analizar su evolución y el impacto del tratamiento.

Base de datos CHAT	
Edad media (SD)	6,92 (1,45)
Distribución por sexo	53 % (F) y 47 % (M)
IMC Medio (% con sobrepeso IMC >25)	18,84 (11,07 %)
IAH Medio (SD)	5,26 (9,1)

Tabla 2.1: Características clínicas y sociodemográficas de los niños que componen CHAT

La base de datos CHAT se compone de 3 subgrupos [17] [18]:

- **Baseline:** Incluye los registros de la PSG inicial de los 453 pacientes seleccionados para continuar con el estudio. Más tarde, este grupo se dividió en dos, con 226 niños que recibirían la operación de eAT, y los 227 restantes no.
- **Follow-up:** Incluye los registros de la PSG repetida 7 meses más tarde. Sin embargo, algunos niños abandonaron el estudio durante este tiempo y solo se incluyen un total de 407 registros, 201 de los niños que fueron operados de eAT y 206 de los niños únicamente en observación.
- **Non-randomized:** Incluye la mayoría de los registros (779) de aquellos niños que se les realizó la PSG pero no fueron seleccionados para el estudio según los criterios de inclusión establecidos en [17] [18].

Para el objetivo principal de este trabajo, la detección de microdespertares o *arousals* a través de redes neuronales, no nos interesa estudiar la efectividad de una eAT, por lo que no se tendrá en cuenta si hubo o no tratamiento a la hora de procesar los registros.

2.5. Estudios anteriores sobre detección de *arousals*

Se han realizado muchos estudios relacionados con el tema principal de este TFG [5]. Por ejemplo, existen estudios donde se emplean redes neuronales convolucionales para anotar eventos del sueño utilizando un único canal de EEG [19], o estudios donde se recurre a otras técnicas de *deep learning*, como las redes neuronales recurrentes para la detección de *arousals* [20], o modelos más complejos que de igual forma hacen uso de la señal EEG para detectar eventos [21].

Sin embargo, todos ellos se centran en la población adulta. A través de la polisomnografía nocturna, los estudios relacionados con la anotación automática de eventos en estudios del sueño se basan en bases de datos formadas por personas adultas mayores de 18 años, con el objetivo de detectar las fases del sueño y diagnosticar trastornos del sueño, entre los que se encuentra la AOS [5] [19] [20] [21]. En estos estudios, no se ha tratado con datos provenientes de niños, cuya arquitectura del sueño y actividad electroencefalográfica presenta diferencias sustanciales con la población adulta [11]. Esto imposibilita extrapolar los estudios anteriores a personas menores de 18 años, es decir, infantiles, siendo necesarios modelos de detección automática de *arousals* específicos para la población pediátrica.

En este proyecto, el nuevo enfoque se basa en estudiar y analizar señales EEG provenientes de PSG de niños, en este caso que padecen de apnea obstructiva del sueño infantil, una enfermedad cuyas cualidades difieren notablemente respecto a su versión adulta. Por tanto, la motivación detrás de este TFG, es que no existen estudios que hagan uso del *deep learning* para detectar los *arousals* en un estudio del sueño infantil.

Capítulo 3

Contexto tecnológico

En este capítulo se van a explicar las herramientas utilizadas para la creación y desarrollo de las diferentes redes neuronales de este trabajo, junto a su marco teórico.

3.1. Redes neuronales

Desde sus orígenes con Alan Turing y John McCarthy en la década de los 50 [22], la IA continúa en constante evolución a día de hoy, desarrollándose nuevas tecnologías de IA como son el aprendizaje automático o *Machine Learning*, o más recientemente el *Deep Learning*, con modelos más profundos y complejos. Es aquí donde se engloban las principales redes neuronales a día de hoy, como las redes neuronales convolucionales (*convolutional neural networks*, CNN) o las redes neuronales recurrentes (*recurrent neural networks*, RNN).

3.1.1. Conceptos básicos

Una red neuronal es un método de aprendizaje automático que busca emular el comportamiento biológico de las neuronas, de modo que el modelo matemático de una red neuronal imita tanto las "estructuras" como el procesamiento de la información que realizan las neuronas biológicas [23].

Tal y como se muestra en la Figura 3.1, el modelo matemático de una neurona consta de pesos asociados a las entradas de datos de la neurona, que se van modificando al entrenar la red neuronal (cuando la neurona está "aprendiendo"), junto a una función de activación que devuelve un valor (por lo general entre 0 y 1) como salida para transmitirlo a las demás neuronas en sus entradas.

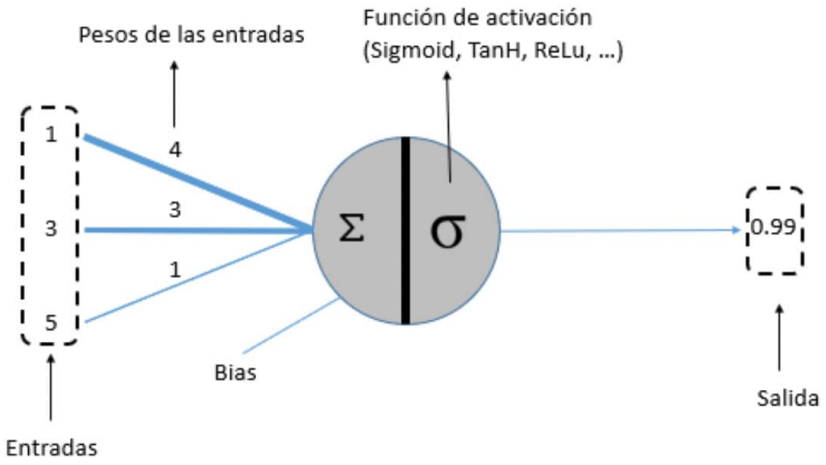


Figura 3.1: Esquema de una neurona artificial [24]

Por tanto, la salida Y de una neurona para la función de activación f queda definida como

$$Y = f(\sum_{i=1}^{n} w_i \cdot x_i + b)$$

donde X es el conjunto de entradas $x_1, x_2, ..., x_n$, w_i el peso asociado a la entrada x_i y b es el sesgo.

Una red neuronal básica se compone de muchas neuronas interconectadas entre sí, organizadas por capas. Como se puede observar en la Figura 3.2, podemos dividir a *grosso modo* una red neuronal en 3 partes: la capa inicial o la capa de entrada de datos, una o más capas ocultas donde se realiza la mayor parte del procesamiento de datos, y una última capa de salida, compuesta de al menos una neurona con el resultado del problema propuesto a la red neuronal.

La combinación de numerosas neuronas implica a su vez calcular miles de funciones de activación, con miles e incluso millones de parámetros variables que se van ajustando a medida que entrenamos la red neuronal. Para ajustar los parámetros, lo más común a la hora de entrenar la red neuronal es minimizar una función de perdida o *loss*, que representa el error acumulado en las predicciones de la red neuronal. A menor error, y por tanto menor valor en la función de perdida, mejor resolverá el problema la red neuronal que estamos entrenando. Más adelante, en la Sección 3.1.3, se detalla el proceso de entrenamiento de una red neuronal.

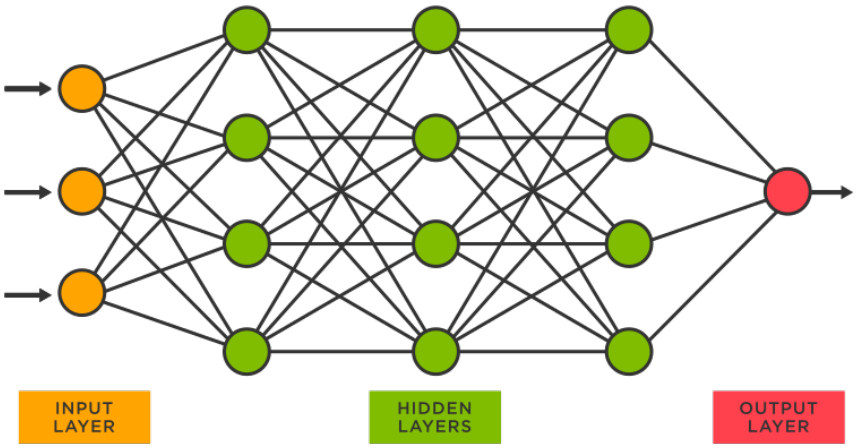


Figura 3.2: Esquema de una Red Neuronal [25]

3.1.2. Funciones de activación

La función de activación, tal y como indica su nombre, es la encargada de "decidir" si una neurona está o no activada. La función de activación, calcula la salida de la neurona mediante la suma ponderada de las variables de entradas multiplicadas por sus pesos correspondientes [26].

Podemos distinguir dos tipos de funciones de activación: lineal y no lineal. Actualmente, dentro del *Deep Learning*, se utilizan funciones de activación no lineales, dado que permiten que los posibles valores que pueda tomar dicha función estén acotados en un rango específico, así como permiten aprender relaciones complejas entre los datos. Las funciones de activación más comunes, y que se manejarán a la hora de diseñar la arquitectura de la red neuronal, son las funciones signo, sigmoide y rectificador lineal (ReLU) [26].

Función signo

La función signo (Figura 3.3a) toma valores de entrada reales y devuelve -1 si la entrada es negativa, o 1 si es positiva. Es una función asimétrica, monótona, discontinua y no diferenciable en $x = 0$.

$$f(x) = \begin{cases} +1 & si \quad x > 0 \\ 0 & si \quad x = 0 \\ -1 & si \quad x < 0 \end{cases}$$

Función sigmoide

La función sigmoide (Figura 3.3b), utilizada principalmente cuando se quieren calcular probabilidades, es una función cuya salida está acotada en el rango (0,1). Es una función no lineal, simétrica, monótona continua y diferenciable.

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

Función ReLU

La función ReLU (Figura 3.3c) no está acotada, devolviendo 0 para valores negativos de entrada, y x para valores positivos de entrada. Es una función no lineal, no acotada (rango en $[0, \infty)$), monótona y diferenciable. Es la función de activación más común en modelos de *Deep Learning*, y la utilizada en este proyecto junto a la función sigmoide.

$$f(x) = max(0, x)$$

(a) Función signo (b) Función sigmoide (c) Función ReLU

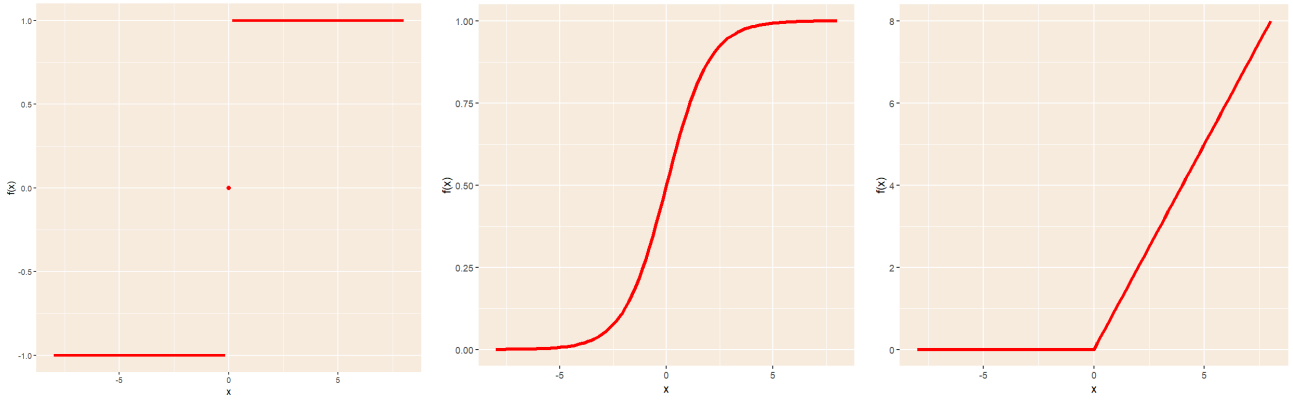


Figura 3.3: Funciones de activación

3.1.3. Entrenamiento

El entrenamiento de una red neuronal puede dividirse en tres etapas que se suceden secuencialmente: propagación de la entrada o *feedforward*, cálculo de la función de pérdida o *loss* y retropropagación del error o *backpropagation* para recalcular los pesos de la red. Además, una vez que se han sucedido las 3 etapas en orden decimos que ha pasado una época, por lo que una red neuronal aprende tras entrenarse durante varias épocas, o lo que es lo mismo, repetir estas tres etapas varias veces [27].

Propagación de la entrada

Durante la propagación de la entrada, se introducen los datos en la capa inicial de la red, y se van calculando las activaciones de las capas sucesivas, donde la salida de una capa sirve como entrada de la capa siguiente [27].

Los datos de entrada pueden agruparse en particiones del mismo tamaño, llamadas lotes o *batch*. De tal forma que la red aprenda más rápido ejecutando las etapas de entrenamiento en paralelo con cada partición o *batch*, considerándose una época cuando todos los *batches* de entrenamiento han pasado por la red.

Función de pérdida

La función de pérdida mide el error o las diferencias entre las predicciones a la salida de la red neuronal y el valor deseado o real, es decir, cuánto difieren las etiquetas originales y las predichas. Es calculada con los resultados ofrecidos por la red (los valores predichos) una vez la etapa *feedforward* ha terminado, y los valores reales de cada instancia.

Las funciones de pérdida son distintas según el problema que estemos tratando. La más común ante un problema de clasificación (y la que se utilizará en este proyecto) es *Cross-Entropy Loss function* l_i:

- Número de clases igual a 2 (clasificación binaria) [28]:

$$l(x_i, y_i) = -w_i[y_n \cdot logx_n + (1 - y_n) \cdot log(1 - x_n)] \tag{3.1}$$

  donde w_i son los pesos correspondientes a la clase i (0 y 1); x_n es la etiqueta original; y_n es la etiqueta predicha y n indica el segmento de datos a considerar.

- Número de clases mayor de dos (clasificación multiclase) [29]:

$$l(x_i, y_i) = \sum_{c=1}^{C} w_c log(\frac{exp(x_{i,c})}{\sum_{j=1}^{C} exp(x_{i,j})}) y_{i,c} \tag{3.2}$$

  donde w_c son los pesos correspondientes a la clase c; $x_{i,c}$ es la etiqueta original del segmento i respecto a la clase c; $y_{i,c}$ es la etiqueta original del segmento i respecto a la clase c y C indica el número total de clases.

Retropropagación del error

Minimizando la función de pérdida, es decir, minimizando el error entre los valores predichos por la red respecto a los valores reales de cada instancia, la red "aprende". Esto se consigue

modificando los pesos de las neuronas, ajustándolos para mejorar las predicciones de la red y, en consecuencia, minimizar la función de pérdida. Esto se conoce como retropropagación del error o *backpropagation*.

Los algoritmos que ajustan estos pesos con el objetivo de minimizar la función de pérdida son conocidos como optimizadores. Los optimizadores principales más utilizados son el descenso estocástico del gradiente (*Stochastic Gradient Descent*, SGD) y la estimación adaptativa del momento (*Adaptive Moment Estimation*, ADAM) [30].

Los optimizadores hacen uso del gradiente, o lo que es lo mismo, un cálculo numérico que marca la dirección en la que ajustar los parámetros de la red para lograr el error mínimo en las predicciones de la red. Asimismo, el nivel de cambio en los pesos es controlado por un hiperparámetro bastante importante denominado tasa de aprendizaje o *learning rate* [31]. Si seleccionamos una tasa de aprendizaje baja, el aprendizaje de la red será muy lento, mientras que si seleccionamos una tasa de aprendizaje alta, puede que nuestra red no consiga aprender (véase la Figura 3.4).

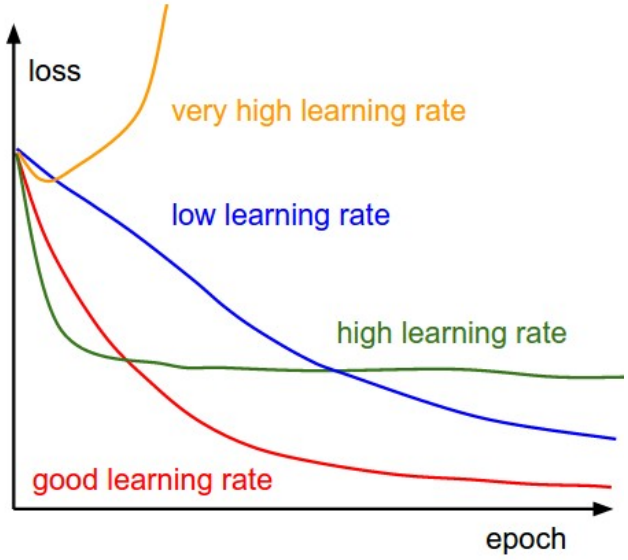


Figura 3.4: Efectos de seleccionar una tasa de aprendizaje no óptima [32]

El descenso del gradiente hace uso de las derivadas de primer orden de las funciones de activación y la regla de la cadena para determinar la dirección en la que se minimiza la función de pérdida. La versión estocástica (SGD) es una adaptación al algoritmo de descenso del gradiente, donde se calcula el gradiente aproximado en lugar del gradiente exacto. Algunas ventajas de la versión estocástica es que los pesos se actualizan con mayor frecuencia y requiere de menos memoria. Por el contrario, no siempre logra encontrar el valor mínimo de *loss* [30].

ADAM hace uso de momentos de primer y segundo orden, adaptando la tasa de aprendizaje en cada época, logrando así un entrenamiento más rápido (converge hacia el valor mínimo de *loss* más rápido) en algunos casos. En cambio, requiere de mayores recursos computacionales, a diferencia de SGD [30]. Para este proyecto, se ha escogido utilizar ADAM como algoritmo de optimización.

3.1.4. Convoluciones

Una convolución es una operación matemática que permite combinar dos funciones f y g, resultando en una nueva función h, también denominada mapa de características (*feature map*), que representa la función g traspuesta e invertida sobre la función f [26]. Matemáticamente, la convolución queda representada como

$$h(t) = (f * g)(t) \approx^{def} \int_{-\infty}^{\infty} f(\tau)g(t-\tau)dr \tag{3.3}$$

donde $g(t-\tau)$ representa la función transpuesta e invertida a $f(\tau)$.

Dentro del procesamiento de señales, una convolución es una operación que aplica un filtro o *kernel* a los datos correspondientes a una señal, derivando en la obtención de una característica. Dicha convolución es extensible a datos tanto de una dimensión (nuestro caso), como datos bidimensionales (procesamiento de imágenes) o multidimensionales [26][33].

Una convolución queda definida por tres parámetros: *kernel*, *padding* y *stride*. En la Figura 3.5 se muestra la aplicación de una convolución (*kernel* en azul) a datos unidimensionales (en naranja) [26].

Parámetro *Kernel*

El parámetro *kernel* define cuál será el tamaño del filtro a aplicar para los datos de entrada de la convolución. Por lo general, se corresponde con una matriz de tamaño personalizable, cuyos valores (también personalizables) de la matriz son los que definen el filtro utilizado en ese momento. En el caso de convoluciones 1D, el *kernel* se corresponde con un vector unidimensional [26].

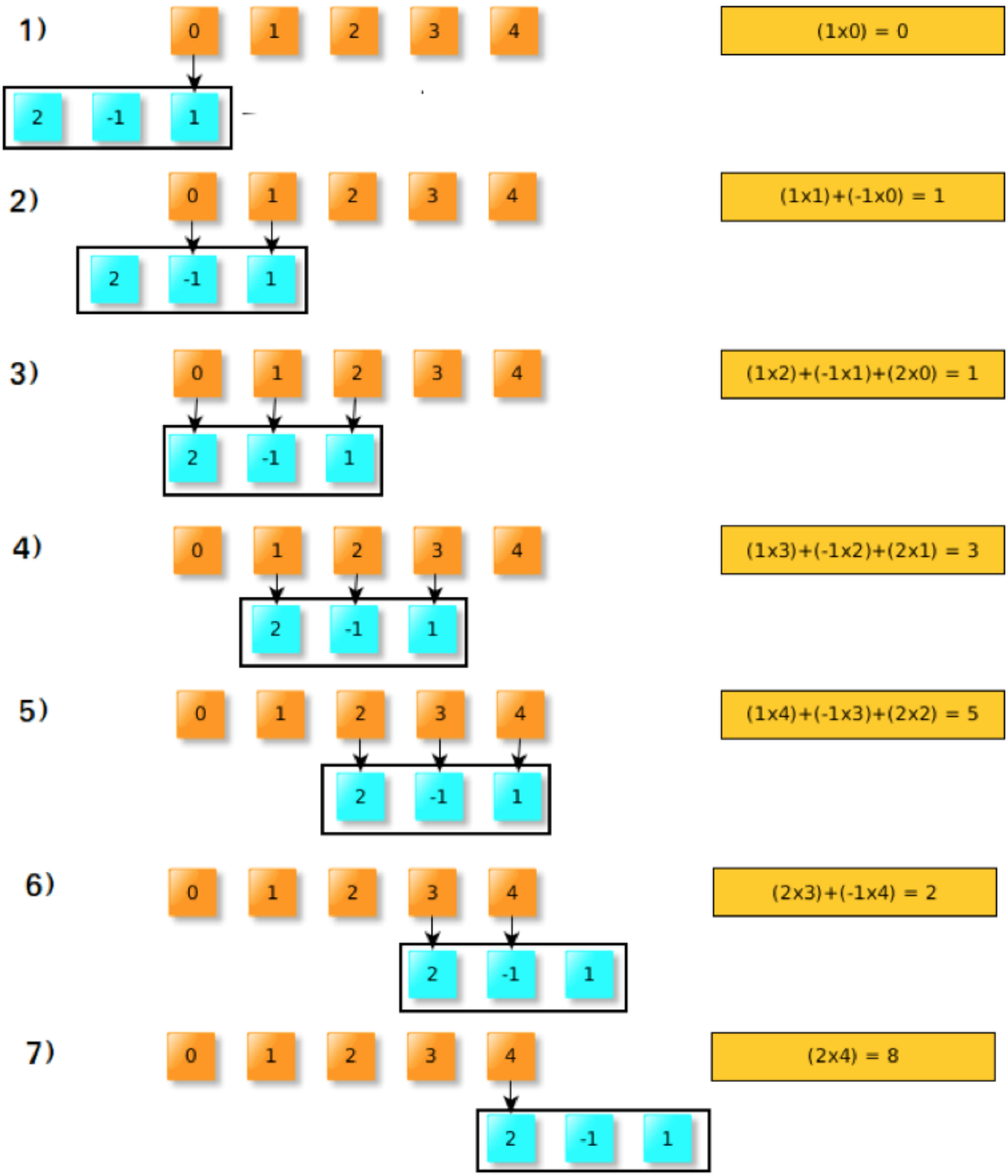


Figura 3.5: Ejemplo de convolución en 1 dimensión [34]

En el caso unidimensional, el kernel es aplicado en orden de izquierda a derecha sobre el vector de datos (véase la Figura 3.5). En el caso multidimensional, tal y como se puede apreciar en la Figura 3.6, el kernel se aplica sobre la matriz de entrada en orden de izquierda a derecha y de arriba a abajo.

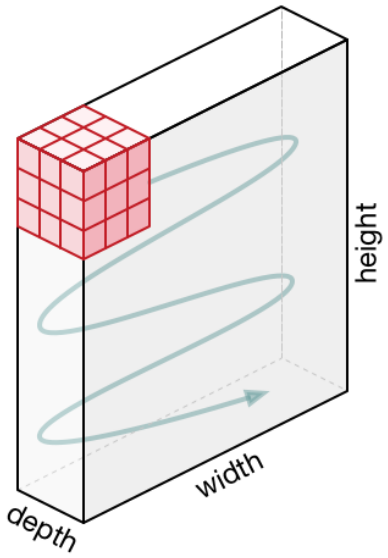


Figura 3.6: Movimiento del kernel para datos multidimensionales

Parámetro *padding*

El parámetro *padding* permite ampliar artificialmente los bordes del vector de datos (o matriz de datos en el caso multidimensional), por lo general, con el valor 0. Es muy común cuando estamos realizando convoluciones en imágenes, de forma que el kernel pueda cubrir una mayor parte de la imagen, aplicando el filtro en los píxeles de los bordes de la imagen [26].

En la Figura 3.7, para una matriz de datos de tamaño 6×6 al que se le va a aplicar un kernel de tamaño 3×3, la aplicación del *padding* sobre dicha matriz resulta en una nueva matriz artificial de tamaño 8×8 ampliada por los bordes con valores 0.

Parámetro *stride*

El parámetro *stride* regula el movimiento del kernel sobre los datos de entrada. En el caso unidimensional, indica cuantos datos debe desplazarse el kernel a la derecha, siendo 1 el valor más habitual de *stride*. Para el caso multidimensional, el parámetro *stride* indica cuántos datos debe moverse el kernel para cada dimensión [26].

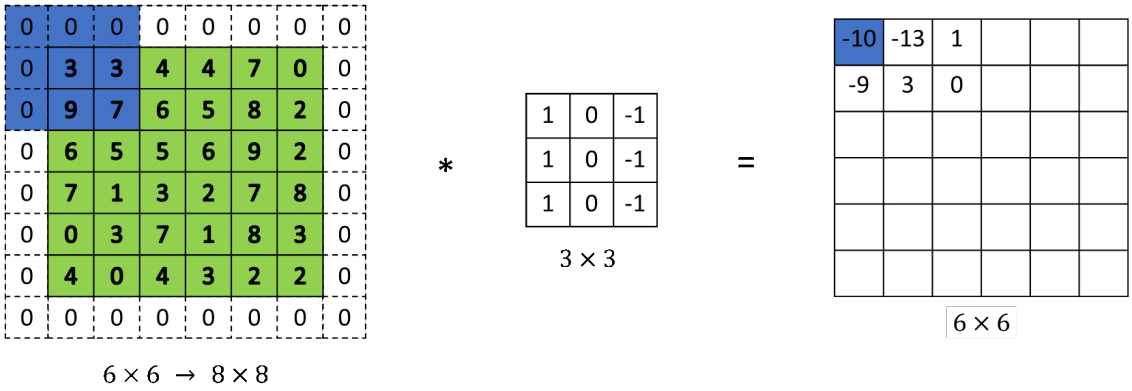


Figura 3.7: *Padding* sobre una matriz de datos

Atendiendo al ejemplo de convolución unidimensional de la Figura 3.5, el *stride* utilizado toma el valor 1, para un kernel de tamaño 1×3.

Dimensión de la salida

La dimensión del resultado de la aplicación de la convolución queda definida de la siguiente forma para el caso unidimensional [35]:

$$O = \frac{I - K + 2P}{S} + 1 \tag{3.4}$$

Siendo:

- **O**: Dimensión del vector de salida tras aplicar la convolución.
- **I**: Dimensión del vector de entrada a la convolución.
- **K**: Tamaño del *kernel*.
- **P**: Incremento de la dimensión al realizar el *padding*.
- **S**: *Stride* utilizado.

A modo de ejemplo, la dimensión resultante de la convolución aplicada al ejemplo de la Figura 3.5 se calcularía de la siguiente manera:

$$O = \frac{5 - 3 + 2 \cdot 2}{1} + 1 = 7$$

Resultando en un nuevo vector de tamaño 1×7.

Para el caso multidimensional, se aplica la misma la fórmula 3.4 para cada dimensión, tomando los valores correspondientes de I, K, P, S en cada dimensión.

3.1.5. Pooling

El término *pooling* denota la operación encargada de reducir el tamaño del mapa de características, así como reducir la carga computacional a la hora de entrenar la red. Esta operación suele realizarse justo después de realizar una operación de convolución [26].

El hecho de reducir la información mediante *pooling* se basa en resaltar las propiedades más importantes del mapa de características. El mapa de características se divide en cuadrículas de tamaño personalizable, siendo las más comunes cuadrículas formadas por 2 y 3 datos (o cuadrículas 2×2 y 3×3 en el caso multidimensional) Dicha reducción o resumen de la información puede realizarse de varias formas, siendo las más comunes [26]:

- *MaxPooling*: Se toma como salida el valor máximo de cada cuadrícula.
- *MinPooling*: Se toma como salida el valor mínimo de cada cuadrícula.
- *AveragePooling*: Se toma como salida el valor medio de cada cuadrícula.

En la Figura 3.8 se ilustra cómo funcionan los tres tipos más comunes de *pooling* para un conjunto de datos unidimensional, con tamaño de *pooling* igual a 3.

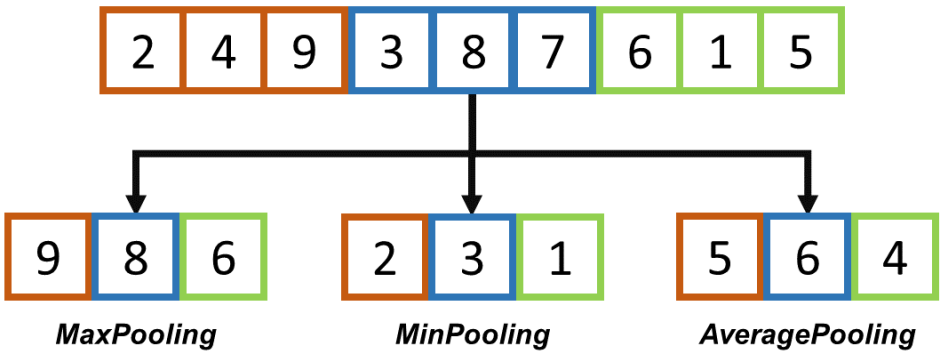


Figura 3.8: Tipos de *pooling* más comunes

Dimensión de la salida

La dimensión del resultado de la aplicación del *pooling* queda definida así [35]:

$$O = \frac{I - P}{S} + 1$$

Siendo:

- **O**: Dimensión del vector de salida tras aplicar el *pooling*.

- **I**: Dimensión del vector de entrada a la convolución.
- **P**: Tamaño del *pooling*.
- **S**: *Stride* utilizado.

A modo de ejemplo, la dimensión resultante del *pooling* aplicado al ejemplo de la Figura 3.8 (vector de tamaño 1×9) se calcularía de la siguiente manera:

$$O = \frac{9-3}{3} + 1 = 3$$

Resultando en un nuevo vector de tamaño 1×3.

3.1.6. Problemas

Las redes neuronales, debido a su complejidad y gran número de parámetros, pueden verse afectadas por contratiempos como la evanescencia o explosión del gradiente o como el sobreajuste o infraajuste, entre otros problemas que influyen negativamente en el entrenamiento y rendimiento final de la red neuronal [36].

Desvanecimiento del gradiente (*Vanishing gradient*)

Las redes neuronales profundas se componen de un gran número de capas. Al realizar *backpropagation*, puede darse el caso de que los gradientes de la función de pérdida comiencen a disminuir rápidamente, provocando que los pesos de las entradas en las primeras capas tomen valores pequeños cercanos a 0. Esto es debido a que el cálculo de las derivadas parciales resulta en valores muy pequeños, es decir, grandes cambios en las entradas de la función de activación resultan en pequeños cambios en la salida [37].

Por tanto, a raíz del problema de la evanescencia del gradiente, nuestro modelo aprenderá muy lentamente, o incluso llegará un momento donde el modelo deje de aprender (por muchas más épocas que destinemos al entrenamiento) si el gradiente llega a 0.

Algunas soluciones serían reducir el número de capas de la red neuronal o usar funciones de activación cuyas derivadas no devuelvan valores muy pequeños, como ReLU, que ha sido la función de activación empleada en este trabajo.

Explosión del gradiente (*Gradient exploding*)

De la misma forma que el problema del desvanecimiento del gradiente, las redes neuronales profundas también pueden padecer de explosión del gradiente. Este problema es justo el contrario

al desvanecimiento del gradiente, donde las derivadas toman valores grandes, haciendo que el gradiente aumente exponencialmente hasta "explotar". Es decir, pequeños cambios en las entradas de la función de activación resultan en grandes cambios en la salida [37]. A consecuencia de esto, nuestro modelo es inestable e incapaz de aprender, con valores muy diferentes (incluso $NaNs$) de la función de pérdida en cada época del entrenamiento.

Una posible solución, común al problema de desvanecimiento del gradiente, es reducir el número de capas de la red neuronal. También puede limitarse el tamaño de los gradientes del modelo para que no "exploten".

Sobreajuste

El sobreajuste u *overfitting* es un problema que puede darse al tratar con cualquier algoritmo de aprendizaje automático. Por lo general, se produce sobreajuste cuando nuestro modelo es demasiado complejo, con alta variabilidad, y el número de datos de entrenamiento no es muy grande [38].

El efecto principal del sobreajuste es que nuestro modelo tiene un desempeño bastante pobre en los conjuntos de datos de validación y test, mientras se obtienen muy buenos resultados con el conjunto de datos de entrenamiento. Es decir, el modelo memoriza los datos de entrenamiento en lugar de aprender de ellos.

Dado que es un problema común en *deep learning*, existen varias técnicas que buscan eliminar o minimizar los efectos del sobreajuste [39]:

- Simplificar el modelo: La principal causa del sobreajuste es la complejidad del modelo, por ello, reducir el número de parámetros, capas, y neuronas por capa es un buen comienzo.
- *Early Stopping*: Consiste en terminar con el entrenamiento cuando la función de pérdida para el conjunto de datos de validación no disminuya durante un número establecido de épocas. Esta técnica va a ser aplicada para todos los modelos que se entrenen en este TFG.
- Técnicas de regularización: La regularización simplifica indirectamente el modelo, por ejemplo, a través de penalizaciones en la función de pérdida (Regularización L1 y L2) o a través de *dropout* [40], donde se eliminan aleatoriamente conexiones entre neuronas (con p la probabilidad de que una neurona sea seleccionada) durante las etapas de entrenamiento y *backpropagation*. La Figura 3.9 ilustra un modelo sin *dropout* y aplicando *dropout* (con $p = 0.5$ para que una conexión neuronal sea eliminada). El *dropout*, además, es una técnica muy común en CNNs, que se ha tenido en cuenta en la etapa de experimentación de este proyecto.

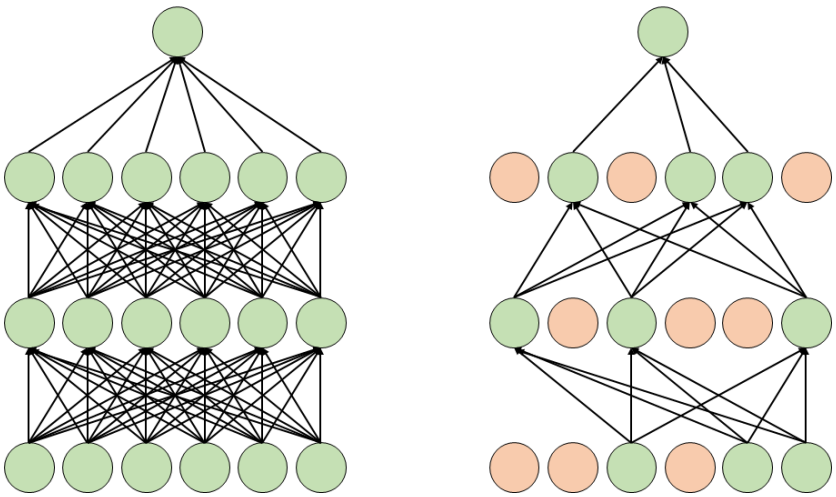


Figura 3.9: Modelo sin *dropout* (izquierda) y modelo con *dropout* $p = 0.5$ (derecha)

Infraajuste

El infraajuste o *underfitting* es el problema complementario al sobreajuste. En este caso, nuestro modelo es demasiado sencillo para nuestros datos.

En este caso, los errores son grandes tanto para el conjunto de datos de entrenamiento como el conjunto de datos de validación y test. Las predicciones no suelen ser buenas en ningún caso.

Las posibles soluciones a aplicar, al contrario que en casos de sobreajuste, es aumentar la complejidad del modelo, aumentar el número de parámetros, capas y neuronas por capa, así como aplicar menos regularización (evitar simplificar indirectamente el modelo).

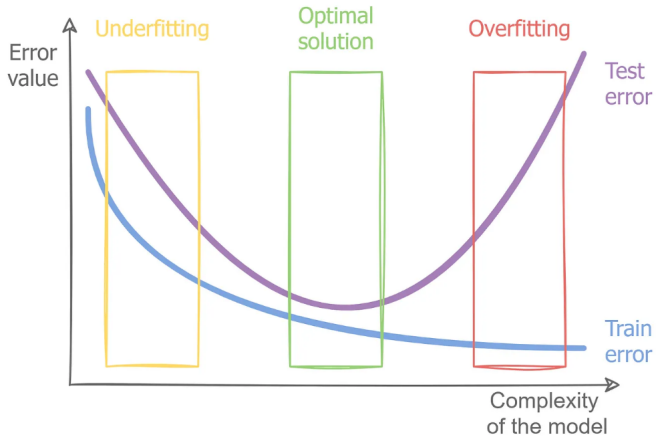


Figura 3.10: Relación directa entre la complejidad y el error de un modelo [38]

No es fácil encontrar el equilibrio entre hacer modelos más complejos (con riesgo de sobreajuste) y modelos más simples (con riesgo de infraajuste). Existe una relación directa entre la complejidad y el error en los modelos de redes convoluciones que puede ayudarnos a dar con el modelo ideal (el que realiza las mejores predicciones), tal y como se muestra en la Figura 3.10. En este TFG, hemos variado el número de capas convolucionales y el valor de *dropout* para maximizar la precisión.

3.2. PyTorch

PyTorch es una biblioteca de aprendizaje automático libre y de código abierto basado en Torch desarrollada por el *Facebook Artificial Intelligence Research* (FAIR). Es utilizada ampliamente para aplicaciones de visión artificial y procesamiento de lenguajes naturales, dado que entre sus características más importantes destacan el aprovechamiento de unidades de procesamiento gráfico (*Graphics Processing Unit*, GPUs) para acelerar la computación o la obtención de gradientes automáticamente, por ejemplo. Dispone de dos interfaces, una para Python y otra para C++. Se ha elegido la primera para este proyecto [41].

Los elementos básicos que forman parte de PyTorch necesarios para la construcción y ejecución de modelos de IA, y más en específico redes neuronales profundas (como las que se diseñarán y entrenarán en este proyecto) son los siguientes:

- Estructura de datos "Tensor": Es una estructura matemática que pertenece a un espacio vectorial tensorial invariante ante cambios de base. Es similar a la estructura de datos `np.array` de la biblioteca NumPy, con la ventaja de que pueden ser alojados en la GPU [42].
- Clase `Dataset`: Es una clase abstracta utilizada para definir nuestro propio *dataset* [43]. Consta mínimo de dos funciones:
  - `__getitem__()`: Accede al elemento i-ésimo del *dataset*.
  - `__len__()`: Muestra el número total de instancias.
- Clase `DataLoader`: Es una clase que permite tanto iterar sobre el *dataset* como mapear el *dataset*. Además, nos permite reordenar aleatoriamente las instancias, cargar los datos por *batches*, mantener o eliminar *batches* incompletos o cargar los datos en paralelo, entre otras más ventajas [43].
- Clase `Module`: Es la clase padre a partir de la cual se define la arquitectura deseada de una red neuronal. Podemos crear nuestra propia red neuronal o utilizar modelos ya entrenados mediante *transfer learning* [44]. Consta mínimo de dos funciones:

- `__init__(self)`: Constructor donde se definen las capas que compondrán el modelo.
- `forward()`: Define como será la propagación o flujo de datos a través de la red neuronal.

Más adelante, en la Sección 5.5, se presenta la arquitectura del modelo base utilizado en este proyecto, donde mediante herencia de la clase `Module` se define nuestra propia red neuronal que será utilizada durante la etapa de experimentación.

- Paquete `cuda`: Proporciona el soporte necesario para tensores CUDA (*Compute Unified Device Architecture*), es decir, tensores que utilizan una GPU para realizar cálculos. Mediante el método `is_available()` podemos comprobar si es posible utilizar tensores CUDA en nuestro sistema.
- Bucle de entrenamiento: El entrenamiento del modelo suele programarse paso a paso en PyTorch, siguiendo la misma estructura. Se selecciona la función de pérdida y optimizador para nuestro modelo, e indicamos a nuestro modelo que "va a ser entrenado". A continuación, se define un bucle ejecutándose durante el número de épocas que queremos entrenar el modelo, cargamos los datos por lotes utilizando un `Dataloader`, movemos los datos a la GPU y establecemos a 0 todos los gradientes.

  Después, se suceden las tres etapas de entrenamiento de la red vistas en 3.1.3, donde alimentamos la red con nuestros datos cargados por lotes (etapa de propagación de la entrada), calculamos la función de pérdida, realizamos *backpropagation* y actualizamos los pesos.

  El código base correspondiente a todo lo indicado anteriormente en PyTorch es el siguiente:

```
criterion = nn.BCELoss()                              # Funcion de perdida
optimizer = torch.optim.Adam(model.parameters(), lr=LR) # Optimizador
model.train()
for epoch in range(num_epocas):                       # Epocas
    for inputs, labels in dataloader_train:           # Cargamos los datos
        inputs = inputs.to('cuda')          # Movemos los datos a la GPU
        labels = labels.to('cuda')
        optimizer.zero_grad()                         # Gradientes a 0
        # Entrenamiento de la red neuronal
        outputs = model(inputs)                 # Feedforward
        loss = criterion(outputs, labels)       # Calculamos el loss
        loss.backward()                         # Backpropagation
        optimizer.step()                        # Actualizamos los pesos
```

Figura 3.11: Bucle de entrenamiento genérico en PyTorch

- Bucle de validación y test: A la hora de evaluar los resultados sobre el conjunto de datos de validación y test en PyTorch, debemos indicar al modelo que "va a ser evaluado" y que no es necesario calcular los gradientes. A continuación, cargamos los datos, los trasladamos a la GPU, introducimos los datos en la red neuronal para obtener las predicciones (y calculamos el valor de la función de pérdida para el conjunto de datos de validación).

  En PyTorch, la parte correspondiente a evaluar el conjunto de datos de validación se programaría como sigue (dentro del bucle de épocas):

```
model.eval()
with torch.no_grad():                    # No calculamos los gradientes
    for inputs, labels in dataloader_validation: # Cargamos los datos
        inputs = inputs.to('cuda')       # Movemos los datos a la GPU
        labels = labels.to('cuda')

        outputs = model(inputs)          # Obtenemos las predicciones
        loss = criterion(outputs, labels)        # Calculamos el loss
```

Figura 3.12: Bucle de validación genérico en PyTorch

Capítulo 4

Planificación

En este capítulo se van a exponer y explicar las decisiones tomadas en cuanto a la planificación de este proyecto, la metodología seguida y sus posibles riesgos, junto al seguimiento y cumplimento del plan propuesto.

4.1. Metodología de trabajo

La metodología elegida para este proyecto, dadas sus características de Trabajo de Fin de Grado centrado en investigación, es de tipo ágil. Esta meteorología se caracteriza por dar un enfoque iterativo al desarrollo del mismo mediante entregas y cumplimiento de tareas. Esto permite evaluar de forma continua tanto los requisitos, la planificación como los resultados de cada iteración tal que se pueda responder a cualquier imprevisto cuanto antes.

Aproximadamente, cada iteración tiene una duración de una semana, donde al inicio de cada semana se expone vía email a ambos tutores las tareas llevadas a cabo, así como los resultados de la experimentación y avances en la memoria si procede, de forma que se pueda tener una idea global del estado del proyecto.

Además, cuando surjan imprevistos que amenacen con retrasos en la planificación o afecten a la completitud del proyecto, se van a solicitar reuniones (presencial u online según la disponibilidad del alumno y tutores) para abordar dichos imprevistos y solucionarlos rápidamente para retomar el plan de planificación inicial.

4.2. Planificación

El trabajo de fin de grado del Grado de Ingeniería Informática, propio de este proyecto, está regulado en la Universidad de Valladolid y la Escuela de Ingeniería Informática con 12 créditos (*European Credit Transfer System*, ECTS) [45]. Cada ECTS equivale a 25 horas de trabajo, resultando en un total de 300 horas de trabajo para este proyecto. Todas estas horas serán repartidas en diferentes tareas a realizar a lo largo del segundo cuatrimestre del curso 2022-2023.

4.2.1. Diagrama de descomposición del trabajo

Para entender mejor la composición del proyecto, se ha realizado un diagrama de descomposición del trabajo (*Work Breakdown Structure*, WBS) a modo de esquema con las actividades principales que involucra este proyecto (ver figura 4.1).

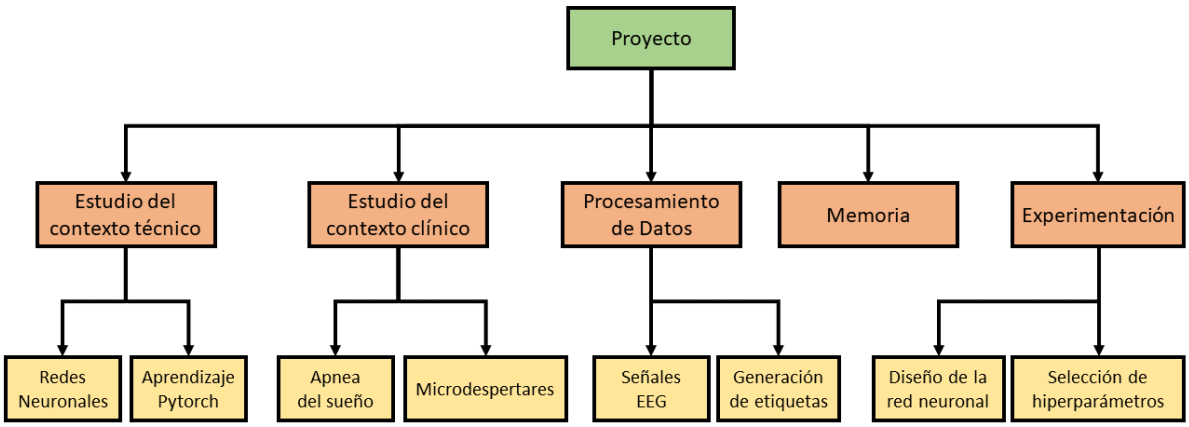


Figura 4.1: Diagrama de descomposición del trabajo (WBS)

4.2.2. Plan de tareas calendarizado

La identificación y planificación de las tareas a desempeñar en este proyecto pueden observarse en la Figura 4.2, ajustándose a los periodos de inicio del segundo cuatrimestre académico y el límite de entrega de actas de Trabajos Fin de Grado en convocatoria ordinaria de la Escuela de Ingeniería Informática [45].

El proyecto comenzará el día 1 de febrero de 2023, una vez superado el primer cuatrimestre, y finalizará el día 22 de junio de 2023, un día antes del límite de entrega de actas en convocatoria ordinaria. Además, se han establecido 3 hitos en los que se deben haber acabado etapas fundamentales del proyecto para lograr finalizarlo en tiempo:

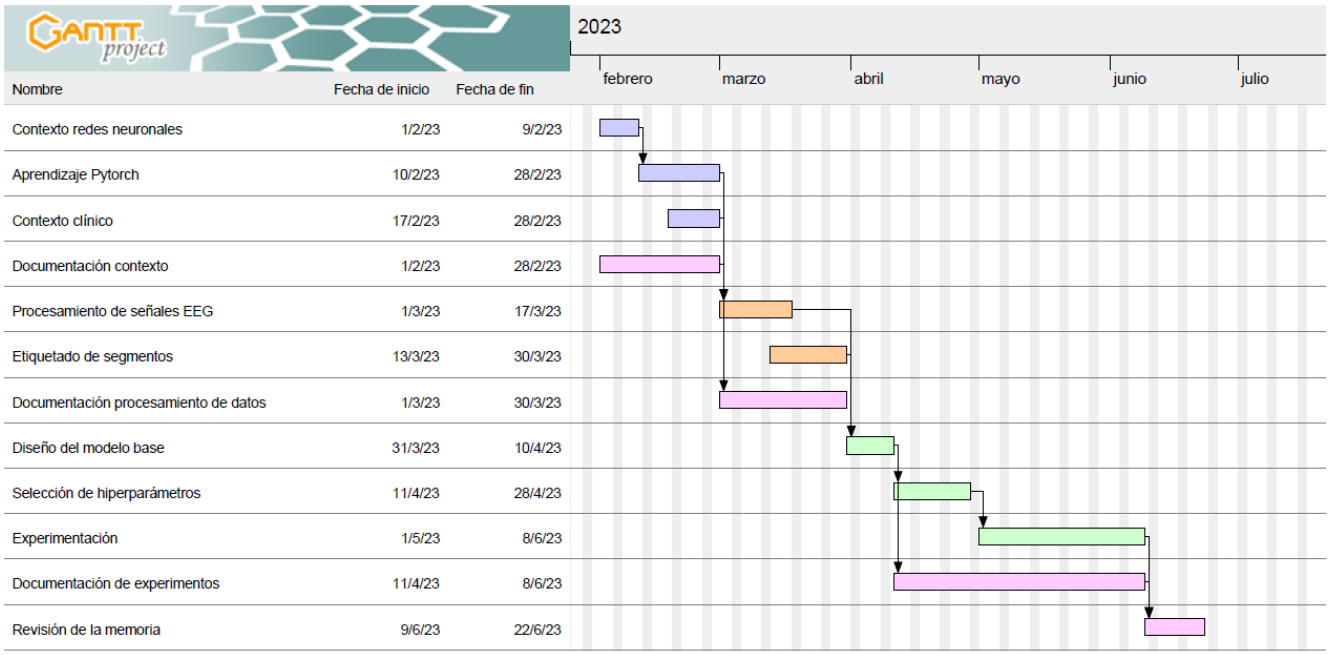


Figura 4.2: Diagrama de Gantt

- **28 de febrero de 2023**: Fecha límite para entender el contexto técnico y clínico del proyecto, además de haber practicado y saber manejar con destreza *PyTorch*.
- **31 de marzo de 2023**: Fecha límite para haber terminado el procesamiento de los datos, es decir, el procesamiento de las señales EEG y la generación de etiquetas para cada segmento.
- **8 de junio de 2023**: Fecha límite para haber terminado con toda la experimentación. Esto incluye diseño y optimización de hiperparámetros del modelo base, resultados y su documentación. Así se asegura también disponer de tiempo suficiente para revisar la memoria.

4.3. Identificación y tratamiento de riesgos

Seguidamente se exponen los riesgos que pueden materializarse durante la ejecución del proyecto, junto a la exposición de estos y las estrategias sugeridas a seguir para eliminar o minimizar los efectos de los riesgos en caso de que ocurran.

A su vez, a cada riesgo le es asignado tanto una probabilidad de ocurrencia con un valor entre 1 y 10, siendo 1 nada probable/sin impacto y 10 muy probable/con consecuencias graves; así como el impacto en el proyecto, con un valor igual entre 1 y 10 siguiendo la misma interpretación anterior. Con estos dos valores, se calcula la exposición al riesgo como el producto de ambas métricas.

Riesgo 1	**Estimación insuficiente de tiempos y mala organización**
Descripción	Este proyecto de investigación es complejo, donde el tiempo dedicado a la experimentación juega un papel fundamental, por lo que una mala planificación podría ocasionar demoras en su entrega.
Probabilidad	3
Impacto	5
Riesgo	15
Estrategia	Identificar y definir claramente todas las tareas del proyecto, junto a una correcta estimación del tiempo necesario para cada tarea. Además, se fijan una serie de hitos a cumplir rigurosamente.

Tabla 4.1: Riesgo 1

Riesgo 2	**Ejecución técnica muy complicada**
Descripción	Puede darse el caso de que este proyecto sea demasiado ambicioso, quedando inconcluso o modificándolo a última hora para reducir la complejidad, simplificar los diseños iniciales y transformar la idea inicial en una más sencilla.
Probabilidad	4
Impacto	9
Riesgo	36
Estrategia	Realizar un estudio de viabilidad del proyecto y las tecnologías a utilizar para su desarrollo.

Tabla 4.2: Riesgo 2

Riesgo 3	**Recursos bibliográficos insuficientes**
Descripción	El diseño y ejecución de este proyecto requiere conocimiento del tratamiento de señales, conceptos generales médicos sobre la apnea del sueño y polisomnografías, así como del estado del arte de proyectos similares.
Probabilidad	2
Impacto	9
Riesgo	18
Estrategia	Búsqueda de referencias en libros especializados y publicaciones de estudios previos de todo lo relacionado al proyecto.

Tabla 4.3: Riesgo 3

Riesgo 4	**Proyectos y asignaturas para este semestre**
Descripción	Simultáneamente a este proyecto, se debe realizar otro TFG para el grado de Estadística y una asignatura sumando un total de 12 ECTS más los 12 ECTS de este proyecto, a diferencia de los 36 ECTS habituales por semestre.
Probabilidad	5
Impacto	7
Riesgo	35
Estrategia	Existe una menor carga lectiva en cuanto a ECTS se refiere, lo que se traduce en un horario más flexible y ligero para completar los proyectos requeridos y aprobar la asignatura restante.

Tabla 4.4: Riesgo 4

Riesgo 5	**Recursos de hardware reducidos**
Descripción	La gran cantidad de datos disponibles para las redes neuronales convoluciones, así como su entrenamiento y optimización de hiperparámetros, requiere de una GPU potente.
Probabilidad	9
Impacto	3
Riesgo	27
Estrategia	Utilización de una GPU potente prestada por la Escuela de Ingeniería Informática de la Universidad de Valladolid para la experimentación.

Tabla 4.5: Riesgo 5

Riesgo 6	**Procesamiento de las señales desacertado**
Descripción	El tratamiento de las señales de polisomnografías, desde la recogida de datos hasta su preparación para la aplicación de redes neuronales, puede influir negativamente en la experimentación si no es el adecuado.
Probabilidad	4
Impacto	7
Riesgo	28
Estrategia	Revisión de la señal procesada y consulta de referencias donde las características del procesado y experimentación sean similares a este proyecto.

Tabla 4.6: Riesgo 6

Riesgo 7	**Generación de etiquetas erróneas**
Descripción	La correcta clasificación y entrenamiento de los modelos de este proyecto depende en gran medida de que las etiquetas correspondientes sean correctas en todo momento.
Probabilidad	1
Impacto	9
Riesgo	18
Estrategia	Comprobación de las etiquetas obtenidas una vez generadas y revisión manual de casos atípicos.

Tabla 4.7: Riesgo 7

Riesgo 8	**Conclusión tardía del proyecto**
Descripción	Tanto por el retraso de las tareas programadas independientemente de la causa, como por el surgimiento de imprevistos fuera del proyecto pero que afectan a su ejecución, existe la posibilidad de no concluir y entregar el proyecto en tiempo
Probabilidad	3
Impacto	7
Riesgo	21
Estrategia	Buena planificación, con hitos fijados para su seguimiento y un compromiso fuerte para cumplir las tareas en tiempo, además de añadir horas de margen para posibles imprevistos.

Tabla 4.8: Riesgo 8

4.4. Presupuesto

Se ha estimado brevemente cual sería el presupuesto de este proyecto en un entorno laboral, teniendo en cuenta el equipamiento necesario, así como el tiempo requerido por una persona (un investigador especializado en inteligencia artificial, y más en concreto, en ingeniería biomédica). Este proyecto, de 300 horas, se ha estimado que debe ser completado en aproximadamente 2 meses (con una jornada laboral completa de 40 horas a la semana).

Dado que se necesitaría disponer de un equipo personal y un servidor que disponga de bastante memoria RAM (alrededor de 64 GB) y al menos una GPU (a partir de 12GB) para el cómputo,

podemos asignar 800€ para la obtención de un portátil, y 2000€ para montar un servidor con las características mencionadas previamente. Dado que la vida media de estos dispositivos hardware es de 5 años, y solo se van a utilizar durante 2 meses aproximadamente, se ha ajustado el coste total del portátil y servidor a su periodo de uso, resultando en 27€ para el portátil y 67€ para el servidor.

Por otra parte, un ingeniero capaz de desarrollar un proyecto de estas características, requiere de un salario bruto alto. En este caso, como ejemplo, se ha fijado un salario bruto de 100€ por hora, resultando en un total de 30000€ para 300 horas.

En total, el presupuesto de este proyecto se cifra en 30094€.

4.5. Cumplimiento del plan

Debido a diversos imprevistos no se pudo disponer del conjunto de datos completo de la base de datos CHAT hasta varias semanas más tarde de lo previsto (en abril en lugar de en marzo). Dicho retraso se ha producido en las fases tempranas de la planificación, avivando el hecho de que el riesgo 8 detallado en la Tabla 4.8 se hiciese realidad. Como consecuencia de dicho retraso, tuvo que reajustarse la planificación, teniendo en cuenta el tiempo disponible, para completar este TFG a tiempo para la convocatoria extraordinaria.

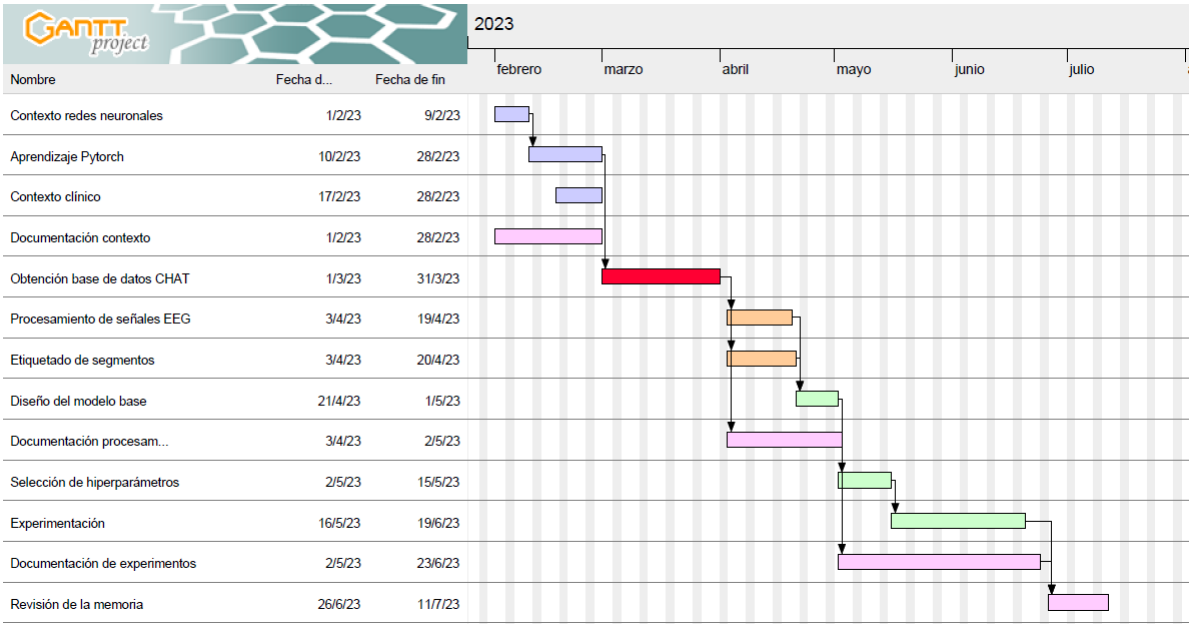


Figura 4.3: Diagrama de Gantt ajustado a la nueva planificación

Una vez se reajustaron los tiempos e hitos del plan calendarizado para semanas más tarde, dejando margen para completar y revisar todo el proyecto antes de la fecha límite de la convocatoria extraordinaria, todas las demás tareas se terminaron con éxito. En la Figura 4.3 puede verse el reajuste realizado a la planificación, teniendo como nueva fecha límite el 11 de julio de 2023.

Capítulo 5

Metodología

En este capítulo se va a explicar todo el tratamiento del conjunto de datos realizado, desde la extracción de los datos en bruto desde CHAT, el procesamiento de las señales y su división en segmentos de 30 segundos, el etiquetado de segmentos que indican si se ha producido o no un *arousal* en dicho segmento, y la posterior división del conjunto de datos en los conjuntos de datos utilizados para el entrenamiento, la validación y test. También se detalla la arquitectura del modelo óptimo y las métricas de rendimiento utilizadas para evaluar los resultados de la experimentación.

5.1. Procesamiento de las señales

Esta etapa de procesamiento inicial de las señales ya ha sido realizada en el Grupo de Ingeniería Biomédica (GIB) de la Universidad de Valladolid, donde se ha trabajado con la misma base de datos para otros estudios con diferentes objetivos al de este proyecto.

El tamaño de los datos originales disponibles en la base de datos CHAT [6] es de alrededor de 220 GB en archivos .mat (señales de EEG) y archivos .xml (anotaciones de los eventos producidos durante la PSG), consiguiendo reducir dicho tamaño a 55 GB en archivos .p después de procesar toda la información.

En este estudio, hemos utilizado la derivación EEG C4-M1, que es uno de los canales EEG recomendados por la AASM para la clasificación del sueño [11]. Los datos, originalmente adquiridos con frecuencias de muestreo (fs) de 200 a 500 Hz, se remuestrearon a una tasa de muestreo común de 125 Hz para homogeneizar su frecuencia, así como para reducir el coste computacional. A continuación, la nueva señal obtenida fue dividida en segmentos de 30 segundos, siendo la nueva dimensión de cada segmento $125Hz \cdot 30s = 3750$.

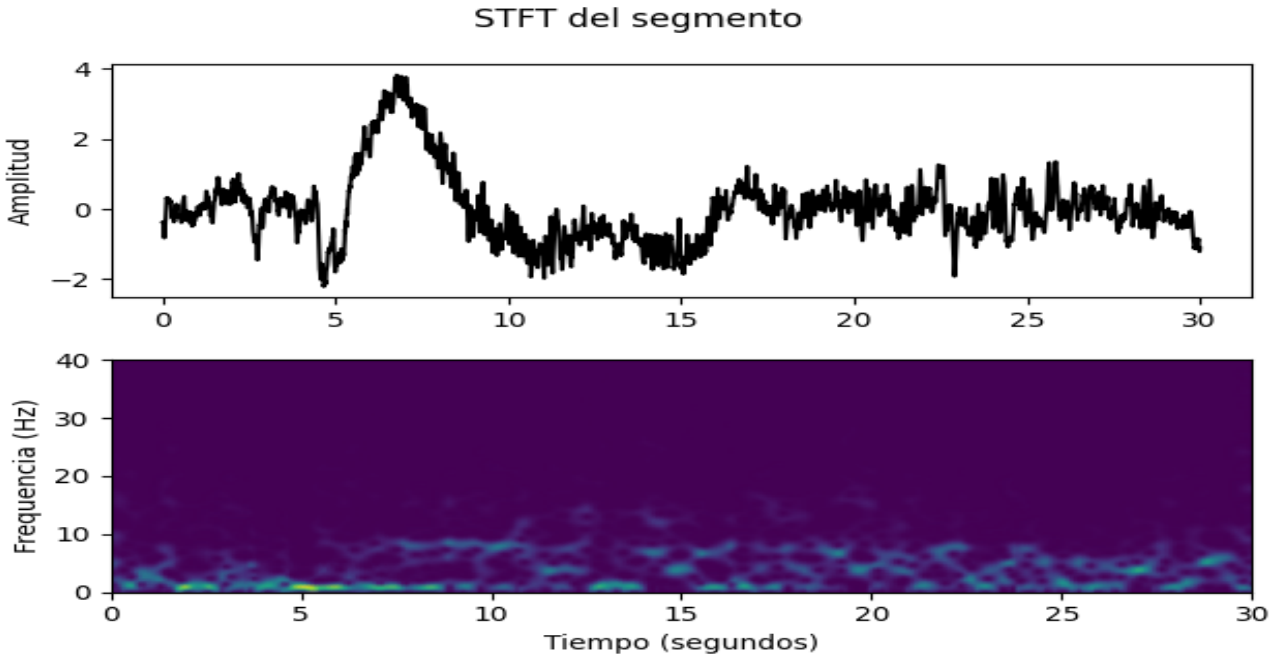


Figura 5.1: Segmento de la señal EEG procesada donde se ha producido un *arousal*

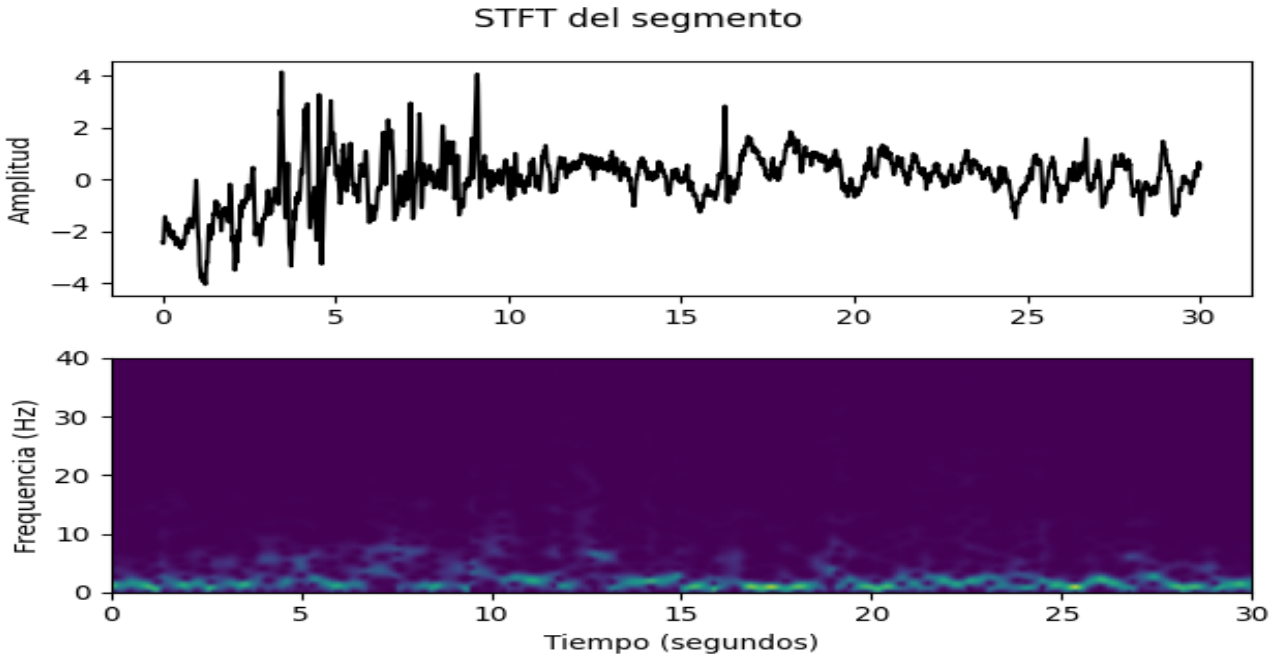


Figura 5.2: Segmento de la señal EEG procesada donde no hay *arousals*

A través de la Transformada de Fourier de Tiempo Corto (*Short-time Fourier transform*, SFTF), en las Figuras 5.1 y 5.2 se muestra un segmento donde se ha producido un *arousal* y un segmento donde no hay *arousals*, respectivamente. Tal y como se detalló anteriormente en la sección 2.3.1, se ve que en la Figura 5.1 se observan cambios abruptos en el EEG con contenido frecuencia en la banda alfa y en frecuencias superiores, permitiendo así identificar un *arousal* en dicho segmento.

5.2. Etiquetado de *arousals*

Cada estudio del sueño de la base de datos CHAT tiene asociado un archivo .xml, donde se recogen las anotaciones por cada uno de los centros participantes de todos los eventos que se han producido durante la PSG utilizando las reglas de la AASM [11], y que posteriormente se reevaluaron, puntuaron y confirmaron de manera centralizada. Dichas anotaciones incluyen fases del sueño, *arousals*, movimientos, etc. Además, en dicho archivo .xml, se indican en segundos la duración del evento y el momento en que dicho evento se ha producido.

Mediante la creación de un *script* propio en *Python*, se procesa el archivo .xml, localizando aquellos eventos anotados como *arousals*. Teniendo en cuenta que la señal del canal C4 del EEG ha sido dividida en segmentos de 30 segundos de duración, se han generado las etiquetas de igual forma, dividiendo la duración total de las anotaciones de la Polisomnografía (PSG) en segmentos de 30 segundos, de forma que coincidan con la señal procesada previamente, indicando con un 0 si no hay ningún *arousal* en esos 30 segundos, y con un 1 si hay algún *arousal* en esos 30 segundos.

Dado que estamos realizando únicamente una clasificación binaria (segmentos con o sin *arousal*), muy a menudo se da el caso de que dada la partición en segmentos de 30 segundos, un *arousal* se sitúe entre dos o más segmentos (en los límites). Todos esos segmentos serán etiquetados con *arousal* (1), incluso aunque el *arousal* se corresponda con los primeros milisegundos de dicho segmento. Todas las etiquetas se guardan en un *array* que se corresponde con la señal de dicho estudio del sueño, de igual tamaño (o con mismo número de segmentos) que la señal procesada.

5.3. Inconsistencias

En la etapa de procesamiento de señales, se han localizado 2 estudios del sueño cuyas señales eran inválidas, es decir, tomaban valores absurdos (véase la Figura 5.3). Esto puede deberse a que no se estuviera registrando ninguna señal, quizás porque los electrodos no han sido colocados correctamente en el niño.

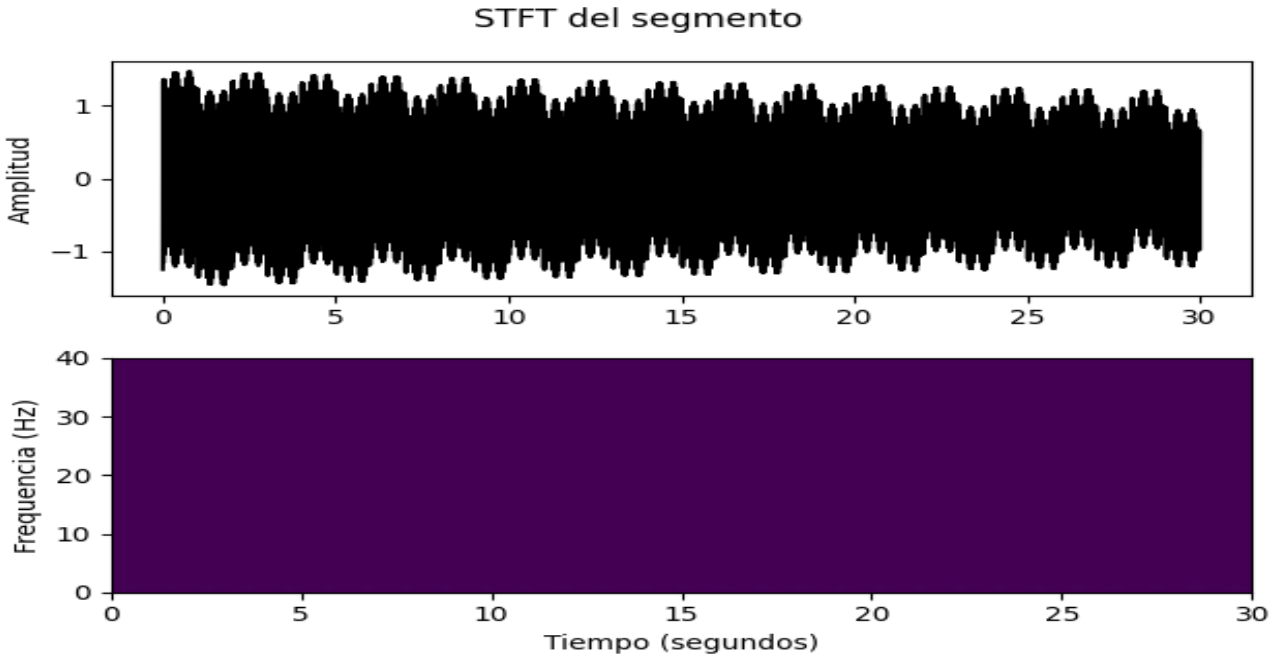


Figura 5.3: Segmento de señal EEG procesada no válida

Además, durante el etiquetado automático de *arousals*, se han descubierto inconsistencias en los registros de 4 estudios del sueño. En este caso, en el resultado de la PSG, aparecían eventos etiquetados como *arousals*, que supuestamente finalizaban tiempo más tarde de terminar el registro y evaluación del estudio del sueño. Por ejemplo, un estudio del sueño con duración total de 1000 segundos, un *arousal* se produce en el instante 998, y dura 10 segundos. Carece de sentido que se diga que el *arousal* ha finalizado en el instante 1008, cuando la grabación de la PSG ha terminado en el instante 1000.

Dado que se disponen de suficientes datos, se ha procedido a eliminar estas 6 señales (es decir, se han invalidado el 0,37 % de los 1639 estudios del sueño iniciales) del conjunto de datos procesado, 1 del subgrupo *baseline*, 1 del subgrupo *followup* y 4 del subgrupo *nonrandomized*. En total, disponemos de 1633 señales EEG válidas con sus respectivas etiquetas para cada segmento.

5.4. Distribución en entrenamiento, validación y test

Este nuevo conjunto de datos ya procesado de la base de datos CHAT contiene en total 1.633 estudios del sueño con registros de EEG válidos: 452 del subgrupo de *baseline*, 406 del subgrupo de *followup* y 775 del subgrupo non-*randomized*.

Los datos de los tres subgrupos se han dividido aleatoriamente en tres conjuntos: entrenamiento (50 %, 815 registros), utilizado para entrenar los modelos de *deep learning*, conjunto de validación (25 %, 409 registros), utilizado para ajustar la regularización y monitorizar la convergencia de los modelos, y conjunto de prueba (25 %, 409 registros), utilizado para evaluar el rendimiento de las arquitecturas de *deep learning*.

Las tablas 5.1 (Entrenamiento), 5.2 (Validación) y 5.3 (Test) muestran los datos clínicos y sociodemográficos de la población bajo estudio para cada conjunto de datos.

Conjunto de datos de **entrenamiento**	
Edad media (SD)	6,99 (1,47)
Distribución por sexo	51,88 % (F) y 48,12 % (M)
IMC Medio (% con sobrepeso IMC >25)	18,77 (10,28 %)
IAH Medio (SD)	5,37 (8,72)

Tabla 5.1: Características clínicas y sociodemográficas de los niños que forman el conjunto de datos de entrenamiento

Conjunto de datos de **validación**	
Edad media (SD)	6,92 (1,43)
Distribución por sexo	53,25 % (F) y 46,75 % (M)
IMC Medio (% con sobrepeso IMC >25)	19 (11,36 %)
IAH Medio (SD)	4,7 (8,25)

Tabla 5.2: Características clínicas y sociodemográficas de los niños que forman el conjunto de datos de validación

Conjunto de datos de **test**	
Edad media (SD)	6,79 (1,43)
Distribución por sexo	55,2 % (F) y 44,8 % (M)
IMC Medio (% con sobrepeso IMC >25)	18,86 (12,34 %)
IAH Medio (SD)	5,61 (10,56)

Tabla 5.3: Características clínicas y sociodemográficas de los niños que forman el conjunto de datos de test

5.5. Arquitectura del modelo óptimo

Se ha decidido tratar un problema de clasificación binaria, donde el modelo predice si en un segmento de 30 segundos dado se ha producido o no un *arousal*. Por ello, se ha optado por una CNN en una dimensión dadas las características de las señales que representan los segmentos de 30 segundos obtenidos a través del procesamiento del canal C4 referenciado a M1 del EEG.

La arquitectura del modelo óptimo puede verse en la Figura 5.4, que consta de una arquitectura formada por diversos bloques convolucionales en una dimensión (después de optimizar el número de bloques, se han escogido 7 bloques convolucionales), seguidas de una capa *fully-conected* que recoge los resultados de aplicar las convoluciones, más una última capa de salida con una única neurona de salida indicando la probabilidad de que dicho segmento contenga un *arousal*.

Como función de activación, tanto en los 7 bloques convolucionales como en la penúltima capa *fully conected* se ha utilizado ReLU. En la capa de salida se ha utilizado la función de activación sigmoide, de forma que la salida de dicha neurona indique la probabilidad de que dicho segmento contenga o no un *arousal*. Si la salida es mayor a 0.5 ($p > 0.5$), se predice que en dicho segmento se ha producido un *arousal*. En caso contrario ($p < 0.5$), se predice que ese segmento no contiene *arousals*.

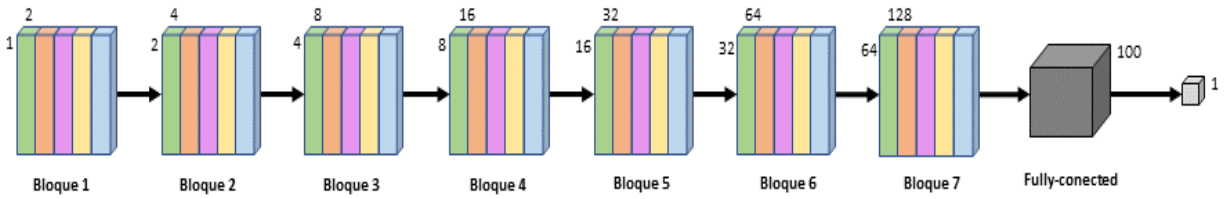


Figura 5.4: Arquitectura de la red neuronal convolucional óptima

5.5.1. Bloque convolucional

Cada bloque convolucional de una dimensión está formada por 5 componentes aplicados secuencialmente como sigue:

1. Una operación de convolución en una dimensión, que extrae las características (*feature maps*) de los datos de entrada.
2. *Batch normalization*, que normaliza las características extraídas por batches.
3. Una función de activación ReLU que determina los *feature maps* relevantes.
4. Una operación de *max pooling* que reduce la dimensión de los datos a la mitad.

5. Una operación o capa de *dropout*, que se encarga de minimizar el *overfitting*.

En la Figura 5.5 se aprecia el diseño común de los 7 bloques convolucionales en una dimensión utilizados en el modelo óptimo. Se ha establecido un tamaño de *kernel* de 32 para todas las convoluciones. El número de canales o filtros de cada operación de convolución se va incrementando en cada capa, desde un primer canal de entrada en el primer bloque convolucional, hasta los 128 canales de salida en el séptimo y último bloque convolucional. Además, se ha fijado un *max pooling* de tamaño 2 y *dropout* igual a 0.1.

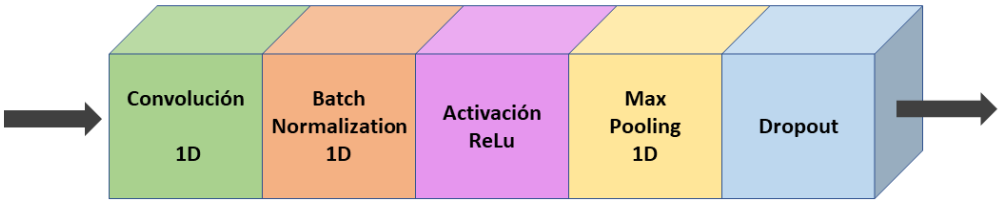


Figura 5.5: Elementos que forman un bloque convolucional 1D en el modelo óptimo

5.5.2. Código en *PyTorch*

A continuación, se presenta el código necesario para el diseño y creación de esta red convolucional (modelo óptimo) en *PyTorch*:

```
class model(nn.Module):
    def __init__(self):
        super().__init__()

        self.layer_1 = nn.Sequential(
          nn.Conv1d(1, 2, kernel_size=32, bias=False),
          nn.BatchNorm1d(2),
          nn.ReLU(),
          nn.MaxPool1d(kernel_size = 2),
          nn.Dropout(p = 0.1)
        )

        self.layer_2 = nn.Sequential(
          nn.Conv1d(2, 4, kernel_size=32, bias=False),
          nn.BatchNorm1d(4),
          nn.ReLU(),
          nn.MaxPool1d(kernel_size = 2),
          nn.Dropout(p = 0.1) )
```

```
        self.layer_3 = nn.Sequential(
          nn.Conv1d(4, 8, kernel_size=32, bias=False),
          nn.BatchNorm1d(8),
          nn.ReLU(),
          nn.MaxPool1d(kernel_size = 2),
          nn.Dropout(p = 0.1)
        )

        self.layer_4 = nn.Sequential(
          nn.Conv1d(8, 16, kernel_size=32, bias=False),
          nn.BatchNorm1d(16),
          nn.ReLU(),
          nn.MaxPool1d(kernel_size = 2),
          nn.Dropout(p = 0.1)
        )

        self.layer_5 = nn.Sequential(
          nn.Conv1d(16, 32, kernel_size=32, bias=False),
          nn.BatchNorm1d(32),
          nn.ReLU(),
          nn.MaxPool1d(kernel_size = 2),
          nn.Dropout(p = 0.1)
        )

        self.layer_6 = nn.Sequential(
          nn.Conv1d(32, 64, kernel_size=32, bias=False),
          nn.BatchNorm1d(64),
          nn.ReLU(),
          nn.MaxPool1d(kernel_size = 2),
          nn.Dropout(p = 0.1)
        )

        self.layer_7 = nn.Sequential(
          nn.Conv1d(64, 128, kernel_size=32, bias=False),
          nn.BatchNorm1d(128),
          nn.ReLU(),
          nn.MaxPool1d(kernel_size = 2),
          nn.Dropout(p = 0.1)
        )

        self.layer_fc = nn.Sequential(
          nn.Linear(11032, 100),
          nn.ReLU() )
```

```
        self.layer_out = nn.Sequential(
          nn.Linear(100, 1),
          nn.Sigmoid()
        )

    def forward(self, inputs):
        # Redimensionamos los datos de entrada para realizar las convoluciones
        inputs = inputs.reshape(x.shape[0], 1, x.shape[1])
        # 7 bloques convolucionales
        x = self.layer_1(inputs)
        x = self.layer_2(x)
        x = self.layer_3(x)
        x = self.layer_4(x)
        x = self.layer_5(x)
        x = self.layer_6(x)
        x = self.layer_7(x)
        # Flattening
        x = x.reshape(x.shape[0], x.shape[1]*x.shape[2])
        # Capa Fully-conected
        x = self.layer_fc(x)
        x = self.layer_out(x)
        return x

model = model()
```

Figura 5.6: Arquitectura del modelo óptimo en PyTorch

5.5.3. Número de parámetros

En la Tabla 5.4 pueden verse el número de parámetros para cada elemento de las capas que componen la red neuronal convolucional en una dimensión, así como sus dimensiones de salida, es decir, la dimensión que tienen los datos al pasar al siguiente elemento u capa.

5.6. Métricas de evaluación de rendimiento

Existen diversas formas de medir el rendimiento de la red neuronal a la hora de afrontar un problema de clasificación. Algunas de estas métricas, como la función de pérdida o *loss*, son vitales durante el proceso de entrenamiento, porque permiten evaluar si la red neuronal está aprendiendo, si es inestable, si hay algún caso de sobreajuste u otros problemas que pueden surgir.

Capa		Dimensión de salida	Nº de parámetros
Bloque 1	Convolución 1D	2, 11219	64
	Norm. Batch 1D	2, 11219	4
	Pooling	2, 5609	0
	Dropout	2, 5609	0
Bloque 2	Convolución 1D	4, 5578	256
	Norm. Batch 1D	4, 5578	8
	Pooling	4, 2789	0
	Dropout	4, 2789	0
Bloque 3	Convolución 1D	8, 2758	1024
	Norm. Batch 1D	8, 2758	16
	Pooling	8, 1379	0
	Dropout	8, 1379	0
Bloque 4	Convolución 1D	16, 1348	4096
	Norm. Batch 1D	16, 1348	32
	Pooling	16, 674	0
	Dropout	16, 674	0
Bloque 5	Convolución 1D	32, 643	16384
	Norm. Batch 1D	32, 643	64
	Pooling	32, 321	0
	Dropout	32, 321	0
Bloque 6	Convolución 1D	64, 290	65536
	Norm. Batch 1D	64, 290	128
	Pooling	64, 145	0
	Dropout	64, 145	0
Bloque 7	Convolución 1D	128, 114	262144
	Norm. Batch 1D	128, 114	256
	Pooling	128, 57	0
	Dropout	128, 57	0
Fully-conected	Lineal	100	729700
Capa de salida	Lineal	1	101

Tabla 5.4: Dimensiones y número de parámetros en cada capa del modelo óptimo

Otras métricas de evaluación utilizadas para valorar la capacidad de clasificación de la red neuronal están basadas en el número de segmentos clasificados correctamente e incorrectamente, calculadas directamente a partir de la matriz de confusión. Para este proyecto, en el que se va a tratar un problema de clasificación binaria de *arousals*, a raíz de los resultados de los diferentes modelos de redes neuronales, se obtendrán matrices de confusión binaria compuestas por los siguientes elementos [46]:

- Verdaderos positivos (VP): Número de segmentos con *arousals* que han sido clasificados correctamente por la red neuronal.
- Falsos positivos (FP): Número de segmentos sin *arousals* que han sido clasificados erróneamente como segmentos con *arousals* por la red neuronal.
- Verdaderos negativos (VN): Número de segmentos sin *arousals* que han sido clasificados correctamente por la red neuronal.
- Falsos negativos (FN): Número de segmentos con *arousals* que han sido clasificados erróneamente como segmentos sin *arousals* por la red neuronal.

En la Figura 5.7 se ilustra la matriz de confusión binaria y los elementos que la componen, adaptada al problema de clasificación de segmentos con y sin *arousals* de este proyecto.

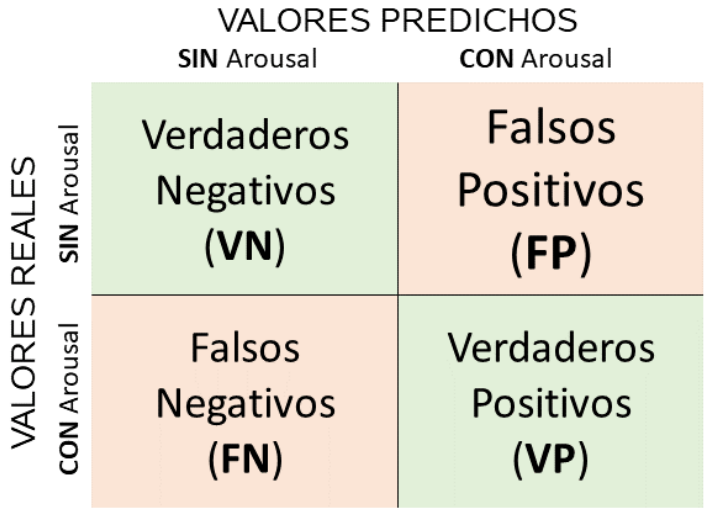


Figura 5.7: Matriz de confusión binaria genérica adaptada a este proyecto

Por tanto, basándose en los componentes de la matriz de confusión binaria, se van a utilizar las siguientes métricas de evaluación:

- Sensibilidad (*Recall*): Proporción de segmentos con *arousals* clasificados con *arousal*.

$$Sensibilidad = \frac{VP}{VP + FN} \cdot 100$$

- Especificidad: Proporción de segmentos sin *arousals* clasificados sin *arousal*.

$$Especificidad = \frac{VN}{VN + FP} \cdot 100$$

- Precisión (*Accuracy*): Proporción de segmentos clasificados correctamente.

$$Precisión = \frac{VP + VN}{VP + VN + FP + FN} \cdot 100$$

- Área bajo la curva ROC (AUROC): La curva ROC representa gráficamente la sensibilidad frente a la especificidad obtenidas al ir variando el umbral o valor de probabilidad límite para clasificar un segmento con o sin *arousal*. El área bajo la curva ROC permite evaluar el rendimiento general del clasificador. Los posibles valores del AUROC están comprendidos entre 0.5 (capacidad discriminatoria nula) y 1 (capacidad discriminatoria perfecta) [47].
- Área bajo la curva *precision-recall* (AUPRC): La curva *precision-recall* representa gráficamente la precisión frente a la sensibilidad obtenidas al ir variando el umbral o valor de probabilidad límite para clasificar un segmento con o sin *arousal*. Al igual que AUROC, permite evaluar el rendimiento general del clasificador, siendo más informativa cuando tratamos con clases desbalanceadas [48].

Capítulo 6

Resultados

Todos los resultados obtenidos durante la experimentación se presentan en este capítulo. Además, se comparan los resultados del modelo base con los generados por las variaciones de valores de los hiperparámetros objeto de estudio.

6.1. Modelos

Se han entrenado un total de 126 modelos, correspondientes a las combinaciones formadas por distintos valores de los siguientes hiperparámetros:

- Número de bloques convolucionales: Se ha experimentado con redes neuronales con 3, 5, y 7 bloques convolucionales 1D.
- Número de segmentos de entrada: Se ha experimentado con datos formados por un único segmento con tamaño de entrada de 3750, datos formados por 3 segmentos con tamaño de entrada $3750 * 3 = 11250$ (la concatenación del segmento anterior [i-1], segmento actual [i] y segmento posterior [i+1]) y datos formados por 5 segmentos con tamaño de entrada $3750 * 5 = 18750$ (la concatenación resultante de los segmentos [i-2], [i-1], [i], [i+1] y [i+2]). Estas variaciones de tamaño han causado que para todos aquellos modelos donde se utilice 1 segmento como dato de entrada con 7 bloques convolucionales, los dos últimos bloques convolucionales tengan un tamaño menor de kernel igual a 16 para evitar quedarnos sin datos al reducir la dimensión.
- Balanceo de clases: Dado que en el conjunto de datos procesado aproximadamente 9 de cada 10 segmentos son segmentos sin *arousals*, esta desproporción puede afectar al rendimiento

final del clasificador. Se han entrenado modelos donde las clases están desbalanceadas y modelos con las clases balanceadas en los conjuntos de datos de entrenamiento y validación, realizando un submuestreo de entre los segmentos sin *arousal*, resultando en conjuntos de datos con muchas menos observaciones pero mismo número de segmentos con y sin *arousal*.

- Probabilidad de *dropout*: Se ha experimentado un total de 7 valores distintos de *dropout*, desde un valor de 0 (no se realiza la operación de *dropout*) hasta un valor de 0.3, con un paso de 0.05.

En la Tabla 6.1 se muestra un resumen de los hiperparámetros y sus posibles valores que conforman el conjunto de los 126 modelos entrenados.

Hiperparámetros	Espacio de búsqueda
Nº de convoluciones	7, 5 y 3
Nº de segmentos de entrada	1 (3750), 3 (11250) y 5 (18750)
Proporción de cada clase	Balanceada y desbalanceada
Dropout	0, 0.05, 0.10, 0.15, 0.20, 0.25 y 0.30

Tabla 6.1: Valores de hiperparámetros empleados en la experimentación

6.2. Experimentación

Se van a analizar los resultados del modelo óptimo y el proceso de selección del mejor modelo, a través de una comparativa de los resultados obtenidos al variar uno u otro valor de los hiperparámetros a estudiar.

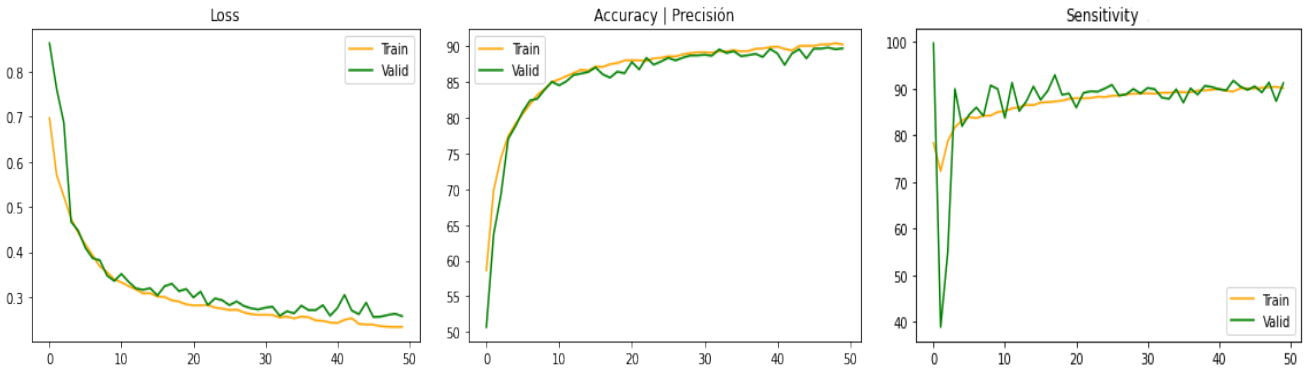
6.2.1. Resultados del modelo óptimo

En el modelo óptimo, cuya arquitectura ha sido explicada previamente en la Sección 5.5, se ha entrenado una red neuronal convolucional formada por 7 bloques convoluciones, 3 segmentos de entrada, clases balanceadas y probabilidad de *dropout* de 0.1. Además se ha utilizado un tamaño de *batch* de 4000 segmentos, función de perdida *CrossEntropyLoss* y optimizador ADAM con un *learning rate* inicial de 0.001. Por último, se ha realizado un submuestreo en los conjuntos de entrenamiento y validación, de tal forma que haya el mismo número se segmentos con *arousal* que sin *arousal*. En el conjunto de datos test, no se ha realizado ningún submuestreo, donde aproximadamente un 90 % de los segmentos que componen dicho *dataset* no contienen ningún *arousal*.

La red neuronal ha sido entrenada durante 50 épocas, con posibilidad de *early stopping*. Después de 50 épocas, la función de pérdida tanto en el conjunto de entrenamiento como en el conjunto de validación disminuía muy lentamente, estabilizándose alrededor de los mismos valores, razón por la cual el entrenamiento no ha continuado durante más épocas. Se ha mantenido el modelo cuyo valor de la función de pérdida es menor para el conjunto de validación.

En la Figura 6.1a (izquierda), se muestra la evolución de la función de pérdida para el conjunto de entrenamiento (naranja) y conjunto de validación (verde). En ambos conjuntos de entrenamiento la función de pérdida disminuye a la par, rápidamente durante las 10 primeras épocas, y más lentamente durante las épocas sucesivas. No hay indicios de sobreajuste analizando el decrecimiento de la función de pérdida.

Para cada época, durante el entrenamiento se calculaban y guardaban la precisión y sensibilidad del modelo para ambos conjuntos de datos. En la Figura 6.1b y la Figura 6.1c, se muestra la evolución de la precisión y sensibilidad de clasificación en cada época de nuestro modelo óptimo respectivamente para entrenamiento (naranja) y validación (verde). De forma pareja a la evolución de la función de pérdida, durante las 10 primeras épocas la precisión aumenta considerablemente. En cambio, la sensibilidad ha sido alta desde el inicio, mejorando ligeramente en cada época.

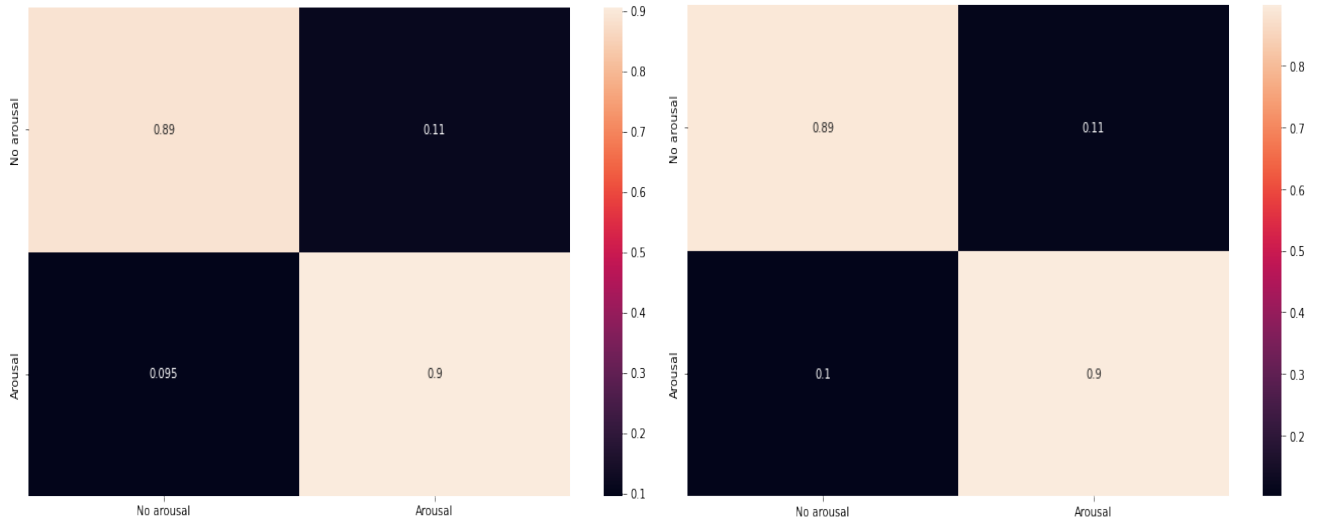


(a) Función de pérdida (b) Precisión (c) Sensibilidad

Figura 6.1: Evolución del entrenamiento del modelo base

Este modelo es el correspondiente a la época número 45 cuyo valor de función de pérdida para el conjunto de validación ha sido de 0.2537. En la Figura 6.2a (izquierda), se presenta la matriz de confusión binara para el *dataset* de validación (las filas se corresponden con los valores reales, y las columnas con los valores predichos, véase la Figura 5.7). Se han clasificado correctamente alrededor del 90 % de los segmentos, de la misma forma que la especificidad y sensibilidad son del 89 % y 90 %, respectivamente.

Los resultados obtenidos con el modelo base son similares para el conjunto de datos test, donde, a diferencia de validación, no se ha realizado submuestreo y las clases están desbalanceadas. En la Figura 6.2b (derecha), se presenta la matriz de confusión binaria para el dataset *test*, con prácticamente la misma precisión, sensibilidad y especificidad de alrededor del 89 %-90 %, pero teniendo en cuenta que en la gran mayoría de los segmentos que conforman este dataset no hay *arousals*, a diferencia de validación. Además, los valores AUROC y AUPRC de este modelo son 0.9589 y 0.7487, respectivamente.



(a) Validación (b) Test

Figura 6.2: Resultados del modelo óptimo en validación y test

6.2.2. Resultados por hiperparámetros

En los siguientes apartados se mostrarán los resultados obtenidos de AUROC y AUPRC para las variaciones de los hiperparámetros objeto de estudio vistos previamente en comparación a los resultados del modelo óptimo. De esta forma, se busca analizar gráficamente que configuración de hiperparámetros ha proporcionado los mejores resultados. En el Apéndice A se encuentran los resultados completos de cada uno de los 126 modelos entrenados.

Consideraciones a tener en cuenta

En aquellos modelos con 7 bloques convolucionales donde los datos de entrada están formados por un único segmento (en lugar de 3 o 5), ha sido necesario reducir el tamaño del *kernel* de los dos últimos bloques convoluciones de 32 a 16, para no quedarnos sin dimensión.

Además, para todos los 42 modelos cuyos datos de entrada son 5 segmentos (la concatenación de los segmentos [i-2] hasta [i+2]), se ha reducido el tamaño de *batch* de 4000 a 2500, para disponer de suficiente memoria en la GPU.

Número de bloques convolucionales

A continuación, se ha estudiado el impacto en AUROC y AUPRC de utilizar diferente número de bloques convolucionales (para 3 segmentos con clases balanceadas) según el *dropout* que forma parte del propio bloque convolucional.

En la Figura 6.3a (izquierda) y la Figura 6.3b (derecha), se muestran el AUROC y AUPRC, respectivamente, para 3 bloques convolucionales (naranja), 5 bloques convolucionales (azul) y 7 bloques convolucionales (verde), según la información proporcionada en las Tablas 6.2a y 6.2b.

A la vista de ambos gráficos, queda claro que utilizar un mayor número de bloques convolucionales en los modelos mejora los resultados obtenidos. El valor máximo, tanto en el caso del AUROC como del AUPRC, se da con las configuraciones de 7 bloques convolucionales y *dropout* de 0.1. Asimismo, e independientemente del número de convoluciones, un *dropout* de alrededor de 0.1 parece que proporciona resultados ligeramente superiores que los obtenidos con otros valores de *dropout*, siendo esta diferencia más clara en los valores de AUPRC (Figura 6.3b).

(a) AUROC

Dropout	**7 conv.**	**5 conv.**	**3 conv.**
0	0.945	0.9454	0.908
0.05	0.9545	0.9398	0.9074
0.10	**0.9589**	0.9468	0.9148
0.15	0.9499	0.9365	0.7796
0.20	0.9383	0.925	0.7313
0.25	0.9421	0.9149	0.9043
0.30	0.9022	0.9149	0.9053

(b) AUPRC

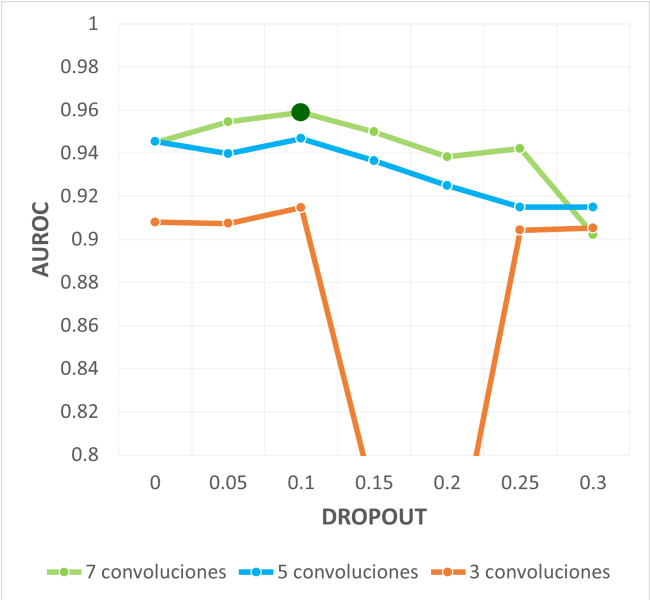
Dropout	**7 conv.**	**5 conv.**	**3 conv.**
0	0.6947	0.6948	0.5871
0.05	0.7269	0.6811	0.5606
0.10	**0.7487**	0.7039	0.5862
0.15	0.7136	0.6717	0.2495
0.20	0.6792	0.6412	0.2223
0.25	0.695	0.6257	0.5526
0.30	0.5445	0.6103	0.5536

Tabla 6.2: Resultados obtenidos según el número de convoluciones vs. *dropout*

Número de segmentos de entrada

Para estudiar las posibles diferencias entre utilizar diferentes tamaños de segmentos, se confrontan los valores AUROC y AUPRC obtenidos al variar este hiperparámetro según el *dropout*

(a) AUROC



(b) AUPRC

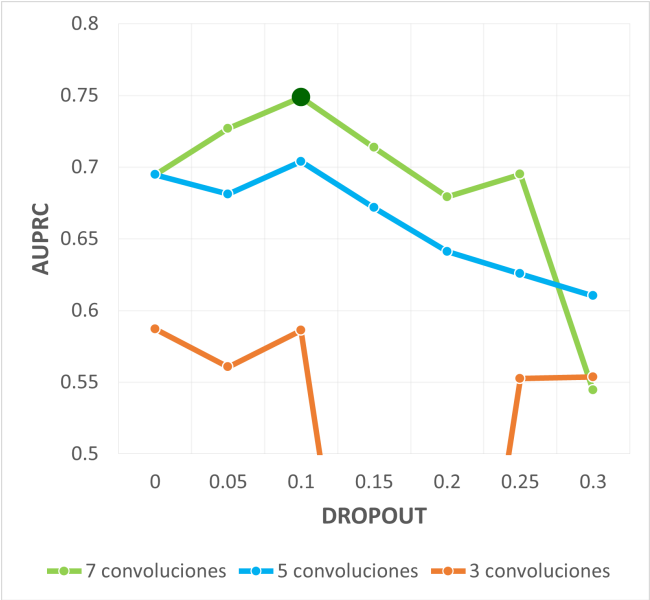


Figura 6.3: Resultados desagregados por número de convoluciones

utilizado en el modelo en las figuras que se presentan a continuación, para modelos con 7 bloques convolucionales y clases balanceadas.

En la Figura 6.4a (izquierda) y la Figura 6.4b (derecha), se comparan el AUROC y AUPRC, respectivamente, para datos de entrada formados por 1 segmento (naranja), 3 segmentos (azul) y 5 segmentos (verde), realizadas según los datos de las Tablas 6.3a y 6.3b.

A la vista de las dos figuras, parece evidente que concatenar a un segmento el inmediatamente anterior y posterior a ese mismo segmento contribuye a mejorar la calidad de clasificación de la red neuronal. Sin embargo, fijándonos en el AUROC (Figura 6.4a), no hay una diferencia clara entre utilizar 3 y 5 segmentos, como sí se aprecia mejor a la hora de evaluar el AUPRC (Figura 6.4b), que utilizar 3 segmentos ofrece mejores resultados (excepto en los casos donde no se realiza *dropout* o este valor es muy alto, como 0.3). En conclusión, la mejor opción sería mantener como datos de entrada 3 segmentos, donde, una vez más, los máximos se localizan en los modelos cuyo *dropout* es igual a 0.1 tanto según el valor de AUROC como de AUPRC.

(a) AUROC

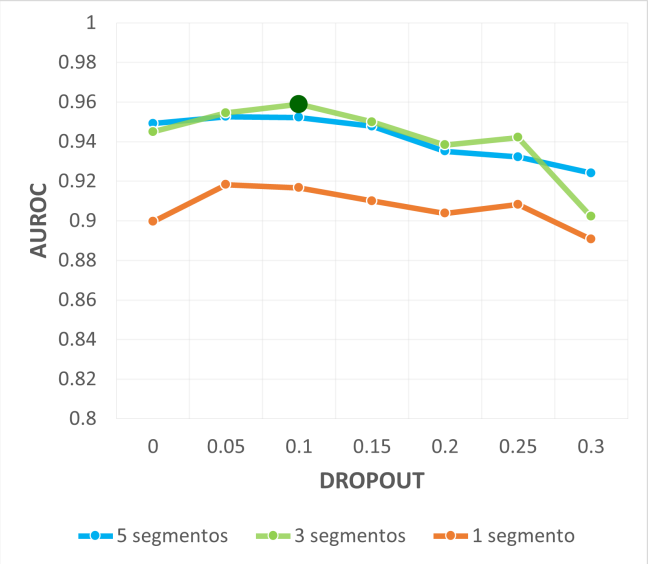
Dropout	5 segm.	3 segm.	1 segm.
0	0.9491	0.945	0.8996
0.05	0.9526	0.9545	0.9183
0.10	0.9522	**0.9589**	0.9167
0.15	0.9477	0.9499	0.91
0.20	0.9351	0.9383	0.9038
0.25	0.9322	0.9421	0.9082
0.30	0.9241	0.9022	0.8907

(b) AUPRC

Dropout	5 segm.	3 segm.	1 segm.
0	0.7142	0.6947	0.5474
0.05	0.7171	0.7269	0.5992
0.10	0.7096	**0.7487**	0.5922
0.15	0.6978	0.7136	0.5657
0.20	0.637	0.6792	0.545
0.25	0.6464	0.695	0.5668
0.30	0.63	0.5445	0.5104

Tabla 6.3: Resultados obtenidos según el número de segmentos vs. *dropout*

(a) AUROC



(b) AUPRC

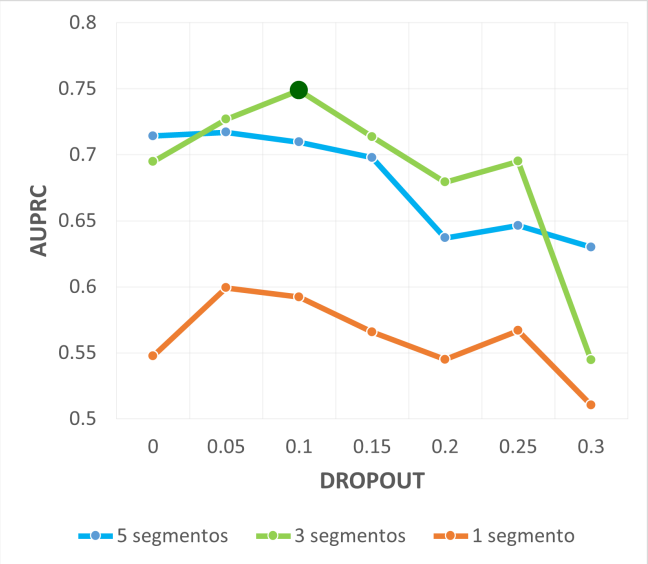


Figura 6.4: Resultados desagregados por número de segmentos

Balanceo de clases

Ahora, se va a comparar el hecho de balancear las clases, es decir, submuestrear el conjunto de datos en los grupos de entrenamiento y validación para que éste posea el mismo número de segmentos etiquetados con *arousal* como segmentos etiquetados sin *arousal*, respecto a no balancear las clases (dejar la proporción de clases tal cual está, aproximadamente 90 %-92 % sin

arousal frente al 8 %-10 % con *arousal*). Los datos corresponden a modelos con 3 segmentos de entrada y *dropout* de 0.1, relacionados con el número de convoluciones del modelo.

En la Figura 6.5a (izquierda) y la Figura 6.5b (derecha), se analizan el AUROC y AUPRC, respectivamente, para datos de entrada cuyos conjuntos de datos están desbalanceados (naranja) y conjuntos de datos que han sido balanceados (azul), según la reorganización de los resultados en las Tablas 6.4a y 6.4b.

(a) AUROC

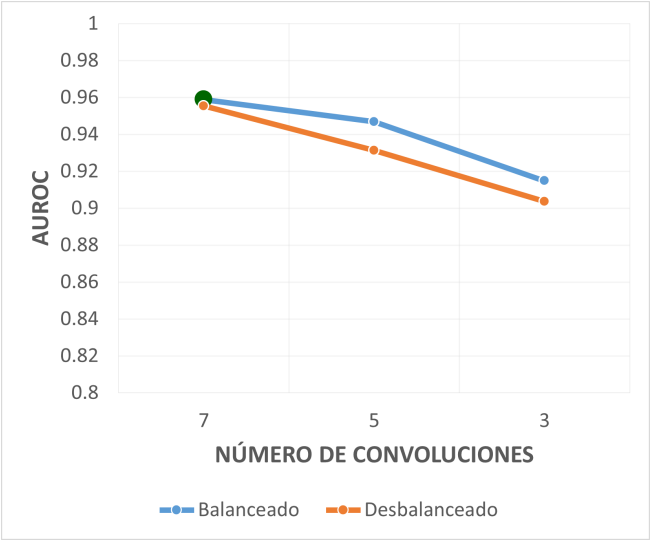
Nº conv.	Balanceado	Desbalanceado
7	**0.9589**	0.9554
5	0.9468	0.9313
3	0.9148	0.9037

(b) AUPRC

Nº conv.	Balanceado	Desbalanceado
7	**0.7487**	0.7482
5	0.7039	0.6742
3	0.5862	0.5903

Tabla 6.4: Resultados obtenidos según la proporción de clases vs. número de convoluciones

(a) AUROC



(b) AUPRC

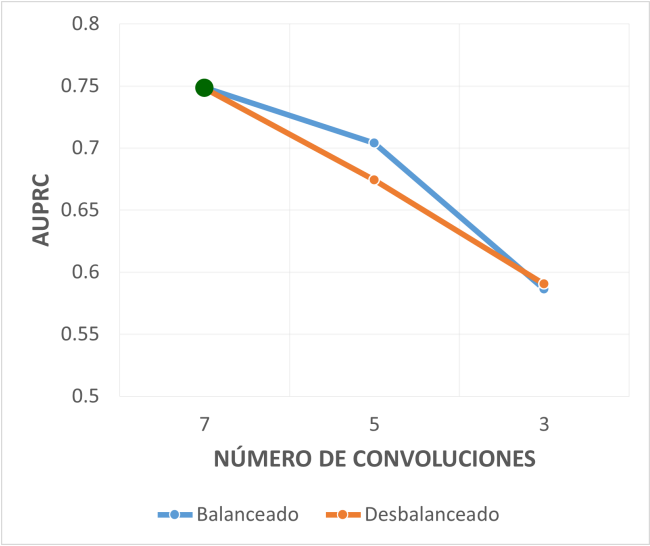


Figura 6.5: Resultados desagregados por clases balanceadas y desbalanceadas

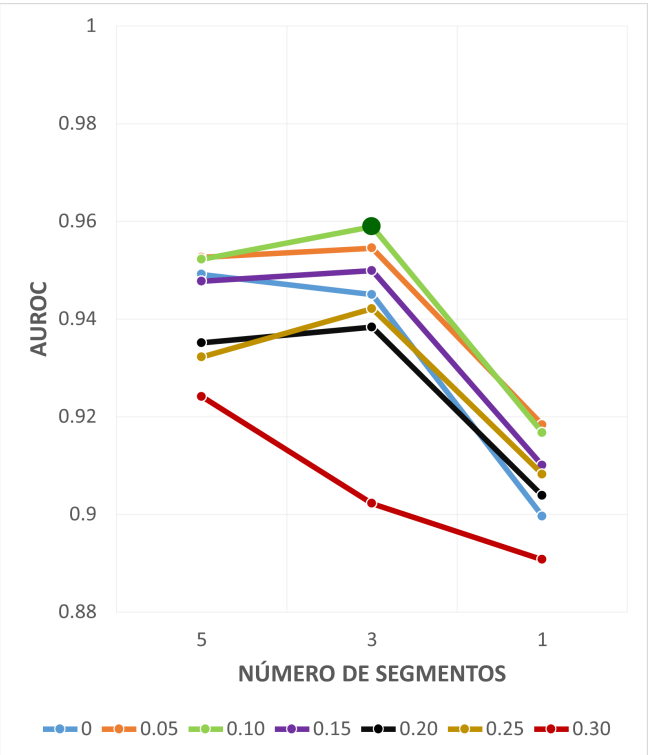
Para 7 convoluciones, los modelos más complejos y con mayor número de parámetros, los resultados tanto de AUROC y AUPRC son similares, quizás levemente mejores en el caso de usar clases balanceadas. Para un menor número de convoluciones, si se aprecia que subsanar el problema

de las clases desbalanceadas contribuye a mejorar las predicciones realizadas por el modelo. Por otra parte, parecen mayores las diferencias obtenidas en los resultados al variar el número de convoluciones que al balancear los *datasets*. El valor máximo obtenido, cuya configuración es más clara atendiendo al AUROC, se corresponde con 7 bloques convolucionales y clases balanceadas.

Dropout

Por último, pero no por ello menos importante, se analiza el efecto de variar el *dropout* al variar el número de segmentos. Anteriormente, al estudiar la variación del número de segmentos en los resultados, ya se intuía que valores de *dropout* pueden ser los más óptimos.

(a) AUROC



(b) AUPRC

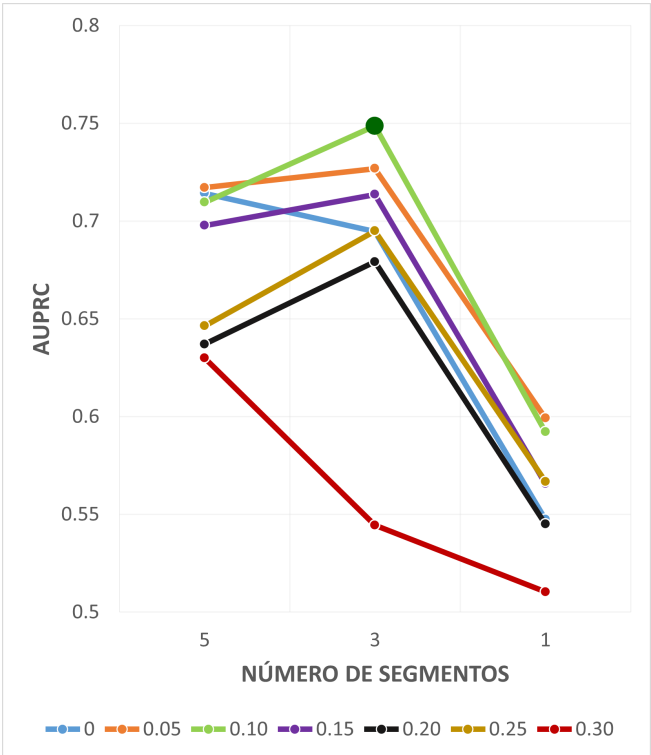


Figura 6.6: Resultados desagregados según el parámetro p de *dropout* utilizado

En la Figura 6.6a (izquierda) y la Figura 6.6b (derecha), se analizan el AUROC y AUPRC, respectivamente, para diferentes valores de *dropout*, desde 0 hasta 0.30 con un paso de 0.05. En la leyenda de ambas figuras se indica cuál es el color asociado a cada valor de *dropout*. Estas figuras se han realizado con los resultados expuestos previamente en las Tablas 6.3a y 6.3b.

De ambas figuras destaca el hecho de que utilizar un *dropout* alto de 0.3 es perjudicial, con resultados notablemente peores a otros valores de *dropout*. Además, destaca el hecho de que no aplicar *dropout* (valor igual a 0) no afecta demasiado al rendimiento de la red neuronal. Eso sí, los mejores resultados, en el caso de 3 segmentos (donde mejor se manifiestan las diferencias entre valores de *dropout*), se corresponden a *dropouts* de 0.1 (valor máximo en ambas figuras), seguidos de los valores inmediatamente cercanos a 0.1 como son 0.05 y 0.15 (en ese orden).

6.3. Discusión

Una vez se han expuesto todos los resultados obtenidos de este TFG, se van a analizar dichos resultados, determinando cuál es el modelo óptimo de entre los 126 modelos entrenados y a su vez, analizando el rendimiento de la clasificación para este modelo óptimo.

6.3.1. Configuración óptima de hiperparámetros

Las redes neuronales cuanto más profundas proporcionan mejores resultados, es decir, a mayor número de convoluciones, mayores valores de AUROC y AUPRC se obtienen. También, valores medios de *dropout*, alrededor de 0.1, son los que contribuyen a maximizar las métricas de evaluación elegidas para este TFG, así como puede ser preferible no realizar *dropout* a escoger valores elevados de *dropout* como puede ser 0.3.

En cuanto a la preparación de los datos, queda claro que balancear ambas clases mediante un submuestreo de la clase dominante (igualar la proporción de segmentos sin *arousal* a la de segmentos con *arousal*, a aproximadamente un 50 %-50 %) ayuda a mejorar ligeramente los resultados. Además, se destaca el hecho de que es preferible concatenar a un segmento sólo el inmediatamente anterior y posterior (segmento formado por el [i-1], [i] y [i+1]), resultando en datos de entrada de 3 segmentos, a otras variaciones como 5 segmentos o no realizar la concatenación.

En conclusión, analizando los resultados para estos 4 hiperparámetros, el modelo óptimo escogido de entre los 126 modelos es el que dispone de la siguiente configuración: 7 bloques convolucionales, *dropout* igual a 0.1, 3 segmentos de entrada y clases que han sido balanceadas. El AUROC y AUPRC para éste modelo es de 0.9589 y 0.7487, respectivamente.

6.3.2. Modelo óptimo

Después del análisis de los resultados por hiperparámetros, y la selección del modelo óptimo, se procede a analizar a mayores la precisión, sensibilidad y especificidad obtenidas en este caso.

El entrenamiento ha sido bastante rápido, gracias al uso de un tamaño de *batch* grande de 4000, poder disponer de una GPU y el uso de técnicas para evitar sobreajuste como *early stopping*, que justo no ha sido necesario en este caso. Atendiendo a los resultados obtenidos para el conjunto de datos test (véase la Figura 6.2b), aproximadamente 1 de cada 10 segmentos es clasificado erróneamente, o similarmente una precisión alta de alrededor del 90 %. De la misma forma, no hay diferencias entre los valores de sensibilidad (89 %) y especificidad obtenidos (90 %), gracias al balanceo de clases durante el entrenamiento que ha influido a la hora de realizar una clasificación uniforme, es decir, que la red neuronal no tienda por inercia a clasificar la mayoría de segmentos sin *arousal* (en el caso de que la proporción de segmentos sin *arousal* fuese la original, del 90 % aproximadamente), sino que realmente aprenda a diferenciar los *arousals* correctamente. Por otra parte, los resultados obtenidos en validación (clases balanceadas) son muy similares a los obtenidos en test (clases no balanceadas), lo que confirma una vez más que la clasificación es uniforme.

Además, sabiendo de antemano que se han utilizado 3 segmentos de entrada, la etiqueta que indica si en dicha instancia se ha producido o no un segmento, hace referencia a la parte central de 30 segundos (segmento [i]). Es decir, una posible razón por la que la red neuronal no proporciona una mayor precisión, es porque en una instancia, se haya producido un *arousal* en los extremos del segmento (el segmento [i-1] o [i+1]), mientras que en la parte central no (segmento [i]), o viceversa. A su vez, sigue resultando útil realizar esta concatenación, porque pudiera darse el caso de que se haya producido un *arousal* en el segmento central de 30 segundos, pero debido a la partición realizada, este *arousal* inicie justo en el último milisegundo del segmento, continuando en el siguiente segmento de 30 segundos. Dicho segmento, va a ser etiquetado con *arousal*, a pesar de que prácticamente en los 30 segundos no se pueda apreciar claramente el *arousal*. Es por esto que añadir al segmento los 30 segundos anteriores y posteriores ayuda a la red neuronal a mejorar su aprendizaje (así en 90 segundos en lugar de 30 segundos, el *arousal* se podría detectar mejor si éste se encuentra en los extremos).

Sería interesante, a partir de éste modelo óptimo, estudiar arquitecturas más profundas o más novedosas, para analizar si las métricas de evaluación (precisión, sensibilidad, AUROC...) obtenidas son aún mayores, y por ende contribuyen a mejorar la clasificación. Otra posible opción de mejora podría ser con un enfoque distinto, donde en lugar de detectar si en un segmento se ha producido o no un *arousal*, proporcionar la señal EEG procesada a la red neuronal, y que ésta detecte los instantes en los que se ha producido y su duración, lo cual es más complejo pero más preciso que lo realizado en este proyecto.

6.3.3. Limitaciones

En este trabajo únicamente se ha usado una CNN que solo detecta si se ha producido un *arousal*, pero no su comienzo y duración en la señal EEG, dando una idea aproximada de dónde se localiza el *arousal* según intervalos de 30 segundos. Esto nos da margen a desarrollar modelos que sean más precisos en la detección de *arousals* en señales EEG.

Por otra parte, el uso de una única base de datos que, aunque sea grande, limita la generalización del modelo óptimo propuesto. De la misma forma, dicha base contiene datos de pacientes de 5 a 10 años, cuando el rango de edad que abarca la AOS infantil va desde el nacimiento (0 años) hasta los 13 años.

Por último, la arquitectura CNN es vista como una caja negra. Es decir, no se puede saber exactamente en que características o patrones se ha fijado para detectar *arousals*, lo que dificulta el análisis de aquellos casos donde no se han detectado arousals o la detección es incorrecta.

Capítulo 7

Conclusiones y líneas futuras

7.1. Conclusiones

En este Trabajo Fin de Grado se han estudiado diferentes modelos de redes neuronales, basados en variaciones de hiperparámetros, tratando con un problema de clasificación binaria con el objetivo de predecir en que tramos de la señal de un EEG se han producido microdespertares.

Se ha comprobado que el uso de redes neuronales convolucionales profundas, con parámetros de *dropout* modestos ($p = 0.1$), contribuye a mejorar el rendimiento de la red neuronal. Además, aportar cierto contexto temporal a los datos (concatenar a un segmento [i] el segmento anterior [i-1] y posterior [i+1]) también mejora los resultados obtenidos. Por otra parte, como solución al problema de clases desbalanceadas (segmentos sin *arousal* son el 90 %-92 % de los datos disponibles frente a los 8 %-10 % de segmentos con *arousals*), se ha comprobado que realizar submuestreo de los segmentos sin *arousals* también ayuda a que la red neuronal aprenda mejor los patrones ocultos y realice una clasificación más uniforme.

Por tanto, una vez se ha seleccionado el modelo con la configuración óptima de hiperparámetros, el rendimiento obtenido por la red neuronal es alto: precisión y especificidad del 90 %, sensibilidad del 90 %, AUROC del 0.9589 y AUPRC del 0.7487, logrando que la aplicación de dicho modelo a señales EEG sea útil para acelerar y simplificar el diagnóstico de la enfermedad así como el estudio de su severidad. También utilizar *early stopping* ayuda a evitar el sobreajuste, aunque esto se ha visto mejor en otros modelos que no eran óptimos.

7.2. Líneas futuras

Los resultados obtenidos en este proyecto se basan en polisomnografías nocturnas de niños de 5 a 10 años de edad disponibles en la base de datos CHAT. Por una parte, sería útil explorar la aplicación del modelo óptimo aquí propuesto a PSG de niños menores de 5 años, así como niños de 10 a 13 años, para cubrir el segmento poblacional completo sobre el que se define la AOS infantil. Además, el grupo de edad comprendido desde los 13 hasta los 18 años, también denominados adolescentes, cuenta con características propias que, en el caso de que padezcan apnea obstructiva del sueño, no esté claro si encasillarlos en la AOS infantil o la AOS adulta. Como primera línea futura, sería estudiar la AOS en la adolescencia comprobando si los resultados aquí obtenidos sobre la AOS infantil son extrapolables a este segmento de población. En este sentido, sería interesante validar el modelo (u obtener uno nuevo) utilizando múltiples bases de datos que permitan obtener un método de detección automática de *arousals* para niños de 0 a 18 años de edad.

Otra posible línea futura, es explorar nuevas arquitecturas más novedosas para lograr aumentar el rendimiento del diagnóstico obtenido en este TFG, como los *transformers*. Además, sería interesante estudiar el impacto de utilizar espectrogramas como datos de entrada para el modelo, es decir, una representación del tiempo-frecuencia de la señal EEG.

En este proyecto, la señal de un electroencefalograma ha sido dividida en segmentos de 30 segundos, donde el modelo detectaba si en cada segmento se ha producido o no un *arousal*. Se puede cambiar el enfoque, y hacer uso del *deep learning* para detectar el instante en el que un *arousal* se ha producido, así como la duración del mismo. Esto simplificaría bastante el análisis de la señal EEG, como a su vez resulta más complejo de realizar. Haciendo uso de redes *U-Net* o *transformers*, podría lograrse este objetivo.

Una última línea futura propuesta, sería el utilizar técnicas de *explainable artificial intelligence* para solventar la problemática de la caja negra que poseen las redes neuronales. Así, se podrían identificar las características concretas del EEG en las que se fija el modelo de *deep learning* para detectar *arousals*, así como analizar aquellos segmentos que no han sido clasificados correctamente y poder mejorar la calidad de la clasificación.

Siglas

AASM *American Academy of Sleep Medicine*. 6, 35, 37

ADAM *Adaptive Moment Estimation*. 14, 15, 48

AOS Apnea obstructiva del sueño. 3, 4, 7, 8, 58, 60, III, V

AUPRC Área bajo la curva *precision-recall*. 46, 50–56, 59, 67–70, III, V

AUROC Área bajo la curva ROC. 46, 50–57, 59, 67–70, III, V

CHAT *Childhood Adenotonsillectomy Trial*. 2, 3, 7, 33, 35, 37, 38, 60, III, V, VII, XIII

CNN *Convolutional neural networks*. 9, 21, 40, 58, III, V

CUDA *Compute Unified Device Architecture*. 24

eAT *Early Adenotonsillectomy*. 7, 8

ECG Electrocardiograma. 4, 5

ECTS European Credit Transfer System. 28, 31

EE.UU. Estados Unidos de América. 7

EEG Electroencefalograma. 2–6, 8, 29, 35–38, 40, 57–60, III, V, XI

EOG Electrooculograma. 4, 5

FAIR Facebook Artificial Intelligence Research. 23

FN Falsos negativos. 45, 46

FP Falsos positivos. 45, 46

GIB Grupo de Ingeniería Biomédica. 35

GPU *Graphics Processing Unit*. 23–25, 31, 32, 51, 57

IA Inteligencia Artificial. 1, 2, 9, 23

IAH Índice de apnea-hipopnea. 4, 5, 7, 39

IMC Índice de masa corporal. 7, 39

NHLBI *National Heart, Lung and Blood Institute*. 7

OSA *Obstructive Sleep Apnea*. V

PSG Polisomnografía. 1–5, 7, 8, 35, 37, 38, 60, III, V, VII

ReLU Rectificador lineal. 11, 12, 20, 40

REM *Rapid Eye Movement*. 6

RNN *Recurrent neural networks*. 9

ROC *Receiver Operating Characteristic*. 46, 61, III, V

SD Desviación típica. 7, 39

SFTF *Short-time Fourier transform*. 37

SGD *Stochastic Gradient Descent*. 14, 15

TFG Trabajo Fin de Grado. 3, 8, 21, 23, 31, 33, 56, 60, III

VN Verdaderos negativos. 45, 46

VP Verdaderos positivos. 45, 46

WBS *Work Breakdown Structure*. 28, XI

Bibliografía

[1] Kaku, M. *La Física del Futuro*. DEBOLSILLO, 2012.

[2] Citysem. *La evolución de la inteligencia artificial y sus diferentes usos*. 2023. URL: https://citysem.es/evolucion-inteligencia-artificial/#:~:text=La%5C%20inteligencia%5C%20artificial%5C%20ha%5C%20avanzado,y%5C%20la%5C%20seguridad%5C%2C%5C%20entre%5C%20otras. Último acceso: 07-06-2023.

[3] Canorea, E. *Machine Learning para los negocios del futuro (y del presente)*. 2023. URL: https://www.plainconcepts.com/es/futuro-machine-learning/#Panorama_actual_Machine_Learning. Último acceso: 20-04-2023.

[4] Marcus, C. L. et al. «Diagnosis and Management of Childhood Obstructive Sleep Apnea Syndrome». En: *Pediatrics* 130.3 (2012), págs. 576-584.

[5] Fiorillo, L. et al. «Automated sleep scoring: A review of the latest approaches». En: *Sleep Med Rev* 48 (2019).

[6] Resource, National Sleep Research. *Childhood Adenotonsillectomy Trial*. 2012. URL: https://sleepdata.org/datasets/chat. Último acceso: 28-04-2023.

[7] Luz Alonso-Álvarez, M. et al. «Documento de consenso del síndrome de apneas-hipopneas durante el sueño en niños (versión completa)». En: *Archivos de Bronconeumología* 47 (2011), págs. 2-18.

[8] Capdevila, O. y Gozal, D. «Consecuencias neurobiológicas del síndrome de apnea del sueño infantil». En: *Revista de Neurología* (2008).

[9] MedlinePlus. *Polisomnografía*. URL: https://medlineplus.gov/spanish/ency/article/003932.htm#:~:text=Una%5C%20polisomnograf%5C%C3%5C%ADa%5C%20registra%5C%3A,movimientos%5C%20musculares%5C%20durante%5C%20el%5C%20sue%5C%C3%5C%B1o.. Último acceso: 19-05-2023.

[10] García de Gurtubay, I. «Estudios diagnósticos en patología del sueño». En: *Anales del Sistema Sanitario de Navarra* 30 (2007), págs. 37-51.

[11] Iber, C. y Sleep Medicine, American Academy of. *The AASM Manual for the Scoring of Sleep and Associated Events: Rules, Terminology and Technical Specifications*. American Academy of Sleep Medicine, 2007.

[12] Tan, H. L., Kheirandish-Gozal, L. y Gozal, D. «Pediatric home sleep apnea testing: Slowly getting there!» En: *Chest* 148.6 (2015), págs. 1382-1395.

[13] Marcus, C. L. et al. «Diagnosis and management of childhood obstructive sleep apnea syndrome». En: *Pediatrics* 130.3 (2012), e714-55.

[14] «Practice parameters for the use of portable recording in the assessment of obstructive sleep apnea. Standards of Practice Committee of the American Sleep Disorders Association». En: *Sleep* 17.4 (1994), págs. 372-377.

[15] Iranzo de Riquer, A. *Electroencefalograma*. 2022. URL: `https://www.clinicbarcelona.org/asistencia/pruebas-y-procedimientos/electroencefalograma`. Último acceso: 01-06-2023.

[16] Hassan, A. R. y Subasi, A. «A Decision Support System for Automated Identification of Sleep Stages from Single-Channel EEG Signals». En: *Know.-Based Syst.* 128.C (2017), págs. 115-124.

[17] Marcus, Carole L et al. «A randomized trial of adenotonsillectomy for childhood sleep apnea». En: *N. Engl. J. Med.* 368.25 (2013), págs. 2366-2376.

[18] Redline, Susan et al. «The Childhood Adenotonsillectomy Trial (CHAT): rationale, design, and challenges of a randomized controlled trial evaluating a standard surgical procedure in a pediatric population». En: *Sleep* 34.11 (2011), págs. 1509-1517.

[19] Sors, A. et al. «A convolutional neural network for sleep stage scoring from raw single-channel EEG». En: *Biomedical Signal Processing and Control* 42 (2018), págs. 107-114.

[20] Li, Ao et al. «A Deep Learning-based Algorithm for Detection of Cortical Arousal During Sleep». En: *Sleep* 43 (2020).

[21] Chambon, Stanislas et al. «DOSED: A deep learning approach to detect multiple sleep micro-events in EEG signal». En: *Journal of Neuroscience Methods* 321 (2019).

[22] Turing, A. M. «Computing Machinery and Intelligence». En: *Mind* LIX.236 (1950), págs. 433-460.

[23] Bishop, C. M. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[24] García-Olalla Olivera, O. *Redes Neuronales artificiales: Qué son y cómo se entrenan*. URL: `https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i`. Último acceso: 27-03-2023.

[25] TIBCO. *¿Qué es una red neuronal?* URL: `https://www.tibco.com/es/reference-center/what-is-a-neural-network`. Último acceso: 27-03-2023.

[26] Goodfellow, Ian, Bengio, Yoshua y Courville, Aaron. *Deep learning*. MIT press, 2016.

[27] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015.

[28] PyTorch. *Binary Cross-Entropy Loss*. 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html. Último acceso: 08-06-2023.

[29] PyTorch. *Cross-Entropy Loss*. 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html. Último acceso: 08-06-2023.

[30] Doshi, S. *Various Optimization Algorithms For Training Neural Network*. URL: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6. Último acceso: 24-06-2023.

[31] cs321n. *CNN for Visual Recognition*. URL: https://cs231n.github.io/neural-networks-3/. Último acceso: 24-06-2023.

[32] Rakhecha, A. *Understanding Learning Rate*. URL: https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de. Último acceso: 24-06-2023.

[33] Saha, S. *A Comprehensive Guide to Convolutional Neural Networks*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Último acceso: 12-06-2023.

[34] Science, Data. *Convolución*. URL: https://datascience.eu/es/matematica-y-estadistica/convolucion/. Último acceso: 16-06-2023.

[35] Mallick, S. y Natak, S. *Number of Parameters and Tensor Sizes in a Convolutional Neural Network (CNN)*. 2018. URL: https://learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/. Último acceso: 08-06-2023.

[36] Ivanov, S. *37 Reasons why your Neural Network is not working*. URL: https://blog.slavv.com/37-reasons-why-your-neural-network-is-not-working-4020854bd607. Último acceso: 22-06-2023.

[37] Pykes, K. *The Vanishing/Exploding Gradient Problem in Deep Neural Networks*. URL: https://towardsdatascience.com/the-vanishing-exploding-gradient-problem-in-deep-neural-networks-191358470c11. Último acceso: 22-06-2023.

[38] Nikolaiev, D. *Overfitting and Underfitting Principles*. URL: https://towardsdatascience.com/overfitting-and-underfitting-principles-ea8964d9c45c. Último acceso: 22-06-2023.

[39] Science, Open Data. *Classic Regularization Techniques in Neural Networks*. URL: https://odsc.medium.com/classic-regularization-techniques-in-neural-networks-68bccee03764. Último acceso: 22-06-2023.

[40] Srivastava, N. et al. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». En: *Journal of Machine Learning Research* 15.56 (2014), págs. 1929-1958.

[41] *PyTorch*. 2023. URL: https://pytorch.org. Último acceso: 20-06-2023.

[42] *PyTorch Tensor*. 2023. URL: https://pytorch.org/docs/stable/tensors.html. Último acceso: 20-06-2023.

[43] *PyTorch Datasets and Dataloaders*. 2023. URL: https://pytorch.org/docs/stable/data.html#torch.utils.data.Dataset. Último acceso: 20-06-2023.

[44] *PyTorch Module*. 2023. URL: https://pytorch.org/docs/stable/generated/torch.nn.Module.html. Último acceso: 20-06-2023.

[45] Ingeniería Informática de Valladolid, Escuela de. *Trabajos Fin de Grado*. URL: https://www.inf.uva.es/trabajos-fin-de-grado/. Último acceso: 28-05-2023.

[46] Zapata Guzmán, A. A. *Dominando la Matriz de Confusión: La guía completa para entender el rendimiento de nuestros modelos de Machine Learning*. 2023. URL: https://www.linkedin.com/pulse/dominando-la-matriz-de-confusi%5C%C3%5C%B3n-gu%5C%C3%5C%ADa-completa-para-el-zapata-guzman/?originalSubdomain=es. Último acceso: 20-06-2023.

[47] Zweig, M. H. y Campbell, G. «Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine». En: *Clinical Chemistry* 39.4 (1993), págs. 561-577.

[48] Saito, T. y Rehmsmeier, M. «The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets». En: *PLoS One* 10.3 (2015).

Apéndice A

Resultados completos

A.1. Modelos para 1 segmento

(a) 7 convoluciones

Dropout	AUROC	AUPRC
0	0.8996	0.5474
0.05	0.9183	0.5992
0.10	0.9167	0.5922
0.15	0.91	0.5657
0.20	0.9038	0.545
0.25	0.9082	0.5668
0.30	0.8907	0.5104

(b) 5 convoluciones

Dropout	AUROC	AUPRC
0	0.8956	0.5194
0.05	0.9196	0.5968
0.10	0.9201	0.5967
0.15	0.8999	0.5293
0.20	0.7254	0.1813
0.25	0.7283	0.1815
0.30	0.8998	0.5317

(c) 3 convoluciones

Dropout	AUROC	AUPRC
0	0.876	0.4344
0.05	0.8909	0.5046
0.10	0.883	0.4444
0.15	0.8399	0.2543
0.20	0.8679	0.4276
0.25	0.7472	0.2079
0.30	0.8974	0.5167

Tabla A.1: Resultados de los modelos para 1 segmento con clases balanceadas

(a) 7 convoluciones

Dropout	AUROC	AUPRC
0	0.8952	0.5572
0.05	0.9075	0.5873
0.10	0.9176	0.6113
0.15	0.9103	0.585
0.20	0.8903	0.5161
0.25	0.8931	0.5372
0.30	0.6988	0.1736

(b) 5 convoluciones

Dropout	AUROC	AUPRC
0	0.895	0.5625
0.05	0.9167	0.6014
0.10	0.917	0.5898
0.15	0.9149	0.6052
0.20	0.8383	0.3629
0.25	0.709	0.206
0.30	0.8992	0.5381

(c) 3 convoluciones

Dropout	AUROC	AUPRC
0	0.8814	0.4808
0.05	0.8734	0.4462
0.10	0.8847	0.518
0.15	0.7641	0.2448
0.20	0.8938	0.5224
0.25	0.7556	0.2232
0.30	0.7807	0.2562

Tabla A.2: Resultados de los modelos para 1 segmento con clases desbalanceadas

A.2. Modelos para 3 segmentos

(a) 7 convoluciones

Dropout	AUROC	AUPRC
0	0.945	0.6947
0.05	0.9545	0.7269
0.10	0.9589	0.7487
0.15	0.9499	0.7136
0.20	0.9383	0.6792
0.25	0.9421	0.695
0.30	0.9022	0.5445

(b) 5 convoluciones

Dropout	AUROC	AUPRC
0	0.9454	0.6948
0.05	0.9398	0.6811
0.10	0.9468	0.7039
0.15	0.9365	0.6717
0.20	0.925	0.6412
0.25	0.9149	0.6257
0.30	0.9149	0.6103

(c) 3 convoluciones

Dropout	AUROC	AUPRC
0	0.908	0.5871
0.05	0.9074	0.5606
0.10	0.9148	0.5862
0.15	0.7796	0.2495
0.20	0.7313	0.2223
0.25	0.9043	0.5526
0.30	0.9053	0.5536

Tabla A.3: Resultados de los modelos para 3 segmentos con clases balanceadas

(a) 7 convoluciones

Dropout	AUROC	AUPRC
0	0.9478	0.7213
0.05	0.9518	0.7395
0.10	0.9554	0.7482
0.15	0.9511	0.7344
0.20	0.925	0.6185
0.25	0.9474	0.7136
0.30	0.9333	0.6792

(b) 5 convoluciones

Dropout	AUROC	AUPRC
0	0.9359	0.6877
0.05	0.9366	0.6858
0.10	0.9313	0.6742
0.15	0.9302	0.6721
0.20	0.9081	0.6086
0.25	0.9366	0.6787
0.30	0.9035	0.5259

(c) 3 convoluciones

Dropout	AUROC	AUPRC
0	0.8907	0.5377
0.05	0.91	0.5929
0.10	0.9037	0.5903
0.15	0.912	0.6101
0.20	0.8845	0.4384
0.25	0.8942	0.5501
0.30	0.895	0.5497

Tabla A.4: Resultados de los modelos para 3 segmentos con clases desbalanceadas

A.3. Modelos para 5 segmentos

(a) 7 convoluciones

Dropout	AUROC	AUPRC
0	0.9491	0.7142
0.05	0.9526	0.7171
0.10	0.9522	0.7096
0.15	0.9477	0.6978
0.20	0.9351	0.637
0.25	0.9322	0.6464
0.30	0.9241	0.63

(b) 5 convoluciones

Dropout	AUROC	AUPRC
0	0.9285	0.6529
0.05	0.9468	0.691
0.10	0.9179	0.4343
0.15	0.9311	0.6564
0.20	0.9238	0.6322
0.25	0.9084	0.5588
0.30	0.9022	0.5511

(c) 3 convoluciones

Dropout	AUROC	AUPRC
0	0.8966	0.5393
0.05	0.8952	0.5306
0.10	0.8468	0.4207
0.15	0.8404	0.4288
0.20	0.9001	0.5442
0.25	0.8837	0.5049
0.30	0.8892	0.5004

Tabla A.5: Resultados de los modelos para 5 segmentos con clases balanceadas

(a) 7 convoluciones

Dropout	AUROC	AUPRC
0	0.9358	0.6917
0.05	0.9465	0.7239
0.10	0.9457	0.7131
0.15	0.9224	0.6409
0.20	0.9347	0.6885
0.25	0.9093	0.6186
0.30	0.9261	0.6656

(b) 5 convoluciones

Dropout	AUROC	AUPRC
0	0.9263	0.6673
0.05	0.9206	0.6415
0.10	0.9005	0.5757
0.15	0.9209	0.6301
0.20	0.9286	0.6636
0.25	0.9007	0.5964
0.30	0.9094	0.5909

(c) 3 convoluciones

Dropout	AUROC	AUPRC
0	0.8906	0.5437
0.05	0.8874	0.5328
0.10	0.8978	0.5555
0.15	0.8872	0.5376
0.20	0.8908	0.5479
0.25	0.6876	0.1926
0.30	0.786	0.2944

Tabla A.6: Resultados de los modelos para 5 segmentos con clases desbalanceadas