



Universidad de Oviedo

ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN INGENIERÍA EN TECNOLOGÍA Y
SERVICIOS DE TELECOMUNICACIÓN

ÁREA DE INGENIERÍA TELEMÁTICA

Detección del trastorno del espectro autista en niños mediante técnicas
de aprendizaje automático

D. Ordoñez Prendes Ángel
TUTOR: D. Corcoba Magaña Victor

FECHA: Julio de 2024

***Nada hubiese tenido sentido sin el
apoyo incondicional de mi familia.***

Os lo debo todo a vosotros.

Índice de Contenido

1. Introducción	1
1.1. Motivaciones	3
1.2. Planteamiento inicial	3
1.3. Objetivos	4
1.3.1. Objetivo general	4
1.3.2. Objetivos específicos	4
2. Planificación	7
3. Estado del arte	10
3.1. Detección del TEA con métodos clínicos	10
3.2. Detección del TEA con técnicas de clasificación	10
3.3. Detección del TEA desde imágenes 'rs-fMRI'	12
3.4. Detección del TEA según rasgos faciales	13
3.5. Detección del TEA sobre datos de 'Eye Tracking'	13
3.6. Detección del TEA sobre señales EGG	14
4. Metodología de trabajo	16
4.1. Hardware	16
4.1.1. Configuración de acelerador gráfico	16
4.1.2. Servicios hardware en la nube	17
4.2. Software	17
4.2.1. Entornos de ejecución	18
4.2.2. Librerías	19
4.3. Metodología de técnicas de 'Deep Learning' sobre métodos analíticos	20
4.3.1. Base de datos y preprocesado	20
4.3.2. Implementación de modelos 'Deep Learning'	24
4.3.2.1. Modelos secuenciales	24
4.3.2.2. Modelos basados en árboles de decisión	26
4.3.3. Evaluación de los modelos	27

4.4.	Metodología de técnicas de 'Deep Learning' sobre datos 'Eye Tracking'.....	28
4.4.1.	Base de datos y preprocesado.....	29
4.4.2.	Implementación de modelos 'Deep Learning'.....	38
4.4.2.1.	Modelos secuenciales.....	38
4.4.2.2.	Modelos LSTM.....	38
4.4.2.3.	Modelos convolucionales.....	39
4.4.2.4.	Modelos híbridos.....	40
4.4.2.5.	Modelos GRU.....	41
4.4.3.	Evaluación de modelos 'Deep Learning'.....	42
5.	Desarrollo	43
5.1.	Desarrollo de técnicas de 'Deep Learning' sobre métodos analíticos.....	43
5.1.1.	Preprocesado de la base de datos.....	43
5.1.2.	Implementación de modelos 'Deep Learning'.....	46
5.1.3.	Evaluación de modelos 'Deep Learning'.....	47
5.1.4.	Generación de gráficas.....	50
5.2.	Desarrollo de técnicas 'Deep Learning' sobre datos 'Eye Tracking'.....	52
5.2.1.	Agrupación de archivos csv.....	52
5.2.2.	Preprocesado de la base de datos.....	54
5.2.3.	Selección de características.....	59
5.2.4.	Implementación de modelos 'Deep Learning'.....	62
5.2.4.1.	Modelo Secuencial.....	62
5.2.4.2.	Modelo LSTM.....	63
5.2.4.3.	Modelo convolucional.....	63
5.2.4.4.	Modelo híbrido.....	64
5.2.4.5.	Modelo GRU.....	65
5.2.5.	Evaluación de modelos 'Deep Learning'.....	66
6.	Análisis de resultados	68
6.1.	Métricas.....	68
6.2.	Gráficas de análisis.....	70
6.3.	Resultados de técnicas 'Deep Learning' sobre métodos analíticos.....	71
6.3.1.	Resultados sin validación cruzada.....	71

6.3.2.	Gráficas de evaluación sin validación cruzada.....	73
6.3.3.	Resultados con validación cruzada.....	86
6.4.	Resultados de técnicas de 'Deep Learning' sobre datos 'Eye Tracking'.....	87
6.4.1.	Resultados con optimizador 'Adam'.....	87
6.4.2.	Resultados con optimizador 'Adamax'.....	91
7.	Conclusiones y trabajos futuros.....	95
7.1.	Métodos analíticos.....	95
7.2.	Métodos sobre 'Eye Tracking'.....	96
7.3.	Conclusiones generales.....	98
8.	Presupuesto.....	100
9.	Manual de instalación.....	104
9.1.	Instalación de CUDA.....	104
9.1.1.	Actualización de los drivers de 'NVIDIA'.....	104
9.1.2.	Instalación de kit de herramientas 'CUDA'.....	105
9.1.3.	Instalación de las herramientas 'cuDNN'.....	105
9.1.4.	Verificación en Python.....	106
9.2.	Entorno Python.....	107
9.3.	Importación de las datasets.....	109
9.4.	Estudio sobre datos analíticos.....	110
9.5.	Estudio sobre datos 'Eye Tracking'.....	111

Índice de Tablas

2.1.	Organización de cada una de las tareas del proyecto con sus plazos correspondiente.....	8
3.1.	Resumen de las distintas técnicas de detección y clasificación del TEA.....	15
4.1.	Tabla de características en el estudio de Vakadkar et al. [6].....	21
4.2.	Conversión numérica del diagnóstico TEA.....	22
4.3.	Conversión numérica de la característica de género.....	22
4.4.	Conversión numérica de la característica de familiares con TEA.....	22
4.5.	Conversión numérica de la característica de presencia de ictericia.....	22
4.6.	Conversión numérica de la característica etnia, primera parte.....	22
4.7.	Conversión numérica de la característica etnia, segunda parte.....	23
4.8.	Arquitectura del modelo secuencial empleado en el primer estudio.....	25
4.9.	Cantidad de participantes del segundo estudio.....	30
4.10.	Características incluidas en la dataset del segundo estudio.....	31
4.11.	Casos de datos adyacentes en el algoritmo de aproximación.....	33
4.12.	Características del segundo estudio con su coeficiente de correlación respecto a la detección del TEA, primera parte.....	35
4.13.	Características del segundo estudio con su coeficiente de correlación respecto a la detección del TEA, segunda parte.....	36
4.14.	Arquitectura del modelo secuencial utilizado en el segundo estudio.....	38
4.15.	Arquitectura del modelo LSTM utilizado en el segundo estudio.....	39
4.16.	Arquitectura del modelo convolucional usado en el segundo estudio.....	40
4.17.	Arquitectura del modelo híbrido usado en el segundo estudio.....	41
4.18.	Arquitectura del modelo GRU usado en el segundo estudio.....	42
5.1.	Identificador correspondiente a la columna en la dataset del segundo estudio.....	44
6.1.	Resultados de precisión y pérdidas de los modelos secuenciales del primer estudio entrenados sin validación cruzada.....	71
6.2.	Tiempo de entrenamiento de los modelos secuenciales del primer estudio.....	72
6.3.	Resultados de precisión en los modelos de árbol de decisión sin validación cruzada.....	73
6.4.	Tiempo de entrenamiento de los modelos de árbol de decisión.....	73

6.5.	Resultados de precisión y pérdidas de los modelos secuenciales del primer estudio entrenados con validación cruzada.....	85
6.6.	Resultados de precisión en los modelos de árbol de decisión sin validación cruzada.....	86
6.7.	Resultados de precisión y error de los modelos del segundo estudio según su grupo de características, con optimizador 'Adam' y con validación cruzada.....	87
6.8.	Mejores resultados de precisión de los modelos del segundo estudio.....	88
6.9.	Pérdidas de los modelos del segundo estudio según su grupo de características.....	89
6.10.	Tiempo de entrenamiento de los modelos del segundo estudio.....	89
6.11.	Uso de memoria en el entrenamiento de los modelos del segundo estudio.....	90
6.12.	Comparativa de resultados de los modelos del segundo estudio.....	91
6.13.	Resultados de los modelos del segundo estudio entrenados con optimizador 'Adamax'.....	91
6.14.	Precisión por cada conjunto de la validación cruzada de los mejores modelos del segundo estudio.....	92
6.15.	Pérdidas por cada conjunto de la validación cruzada de los mejores modelos del segundo estudio.....	93
6.16.	Comparación del modelo secuencial del segundo estudio con optimizadores 'Adam' y 'Adamax'.....	93
6.17.	Comparación del modelo LSTM del segundo estudio con los optimizadores 'Adam' y 'Adamax'.....	94
6.18.	Comparación del modelo convolucional del segundo estudio con los optimizadores 'Adam' y 'Adamax'.....	94
7.1.	Modelo del primer estudio que mejores resultados ha obtenido.....	96
7.2.	Modelo del segundo estudio que mejores resultados ha obtenido.....	96
8.1.	Gastos brutos del proyecto.....	101
8.2.	Costes a añadir sobre el precio bruto del proyecto.....	102
8.3.	Coste total del proyecto.....	103

Índice de Figuras

2.1.	Diagrama de GANT del proyecto, primeros cinco meses.....	9
2.2.	Diagrama de GANT del proyecto, últimos cinco meses.....	9
4.1.	Reparto de la dataset del primer estudio para entrenamiento, test y validación.....	23
4.2.	Validación 'K-Folder' de ejemplo para tres conjuntos.....	28
4.3.	Cálculo del rango intercuartílico.....	34
4.4.	Cálculo de límites superior e inferior.....	35
4.5.	Reparto de las muestras del segundo estudio para entrenamiento y test.....	37
5.1.	Código de importación de la base de datos del primer estudio.....	44
5.2.	Código de cambio de características a formato numérico en el primer estudio.....	45
5.3.	Código de comprobación del preprocesado.....	45
5.4.	Creación de subconjunto de datos para entrenamiento, test y validación del primer estudio.....	46
5.5.	Separación entre las características y sus etiquetas.....	46
5.6.	Código de implementación del modelo secuencial del primer estudio.....	47
5.7.	Código de implementación de los modelos de árbol de decisión en el primer estudio.....	47
5.8.	Código para evaluar los modelos sin validación cruzada en el primer estudio.....	48
5.9.	Código para evaluar los modelos con validación cruzada en el primer estudio.....	49
5.10.	Código para generar matrices de confusión.....	49
5.11.	Código para generar las gráficas de precisión.....	51
5.12.	Código para generar las gráficas de pérdidas en los modelos de árbol de decisión.....	51
5.13.	Código para unir todos los archivos '.csv' en el segundo estudio.....	52
5.14.	Código para agregar la etiqueta a las muestras del segundo estudio.....	53
5.15.	Código para eliminar las características no usadas en el aprendizaje.....	54
5.16.	Código para extraer los valores presente, pasado y futuro para realizar el preprocesamiento de los datos en el segundo estudio.....	54
5.17.	Código para aproximar valores a partir del pasado y futuro.....	55

5.18. Código para almacenar el identificado de la muestra y su característica como temporal.....	55
5.19. Código para almacenar una muestra como valor pendiente a aproximar.....	56
5.20. Código para comprobar la última muestra de un participante.....	56
5.21. Código para extraer el último valor temporal de la lista.....	56
5.22. Código para extraer los valores pendientes de aproximación.....	57
5.23. Código usado para calcular el incremento de aproximación 'delta'.....	57
5.24. Código para aproximar los valores pendientes.....	58
5.25. Código para corregir los valores atípicos de la base de datos.....	59
5.26. Código de extracción de los coeficientes de correlación.....	60
5.27. Código usado para intercalar los participantes 'TEA' con los 'No TEA'.....	60
5.28. Código para crear los conjuntos de datos de entrenamiento y test del segundo estudio.....	61
5.29. Código con las listas de características disponibles para realizar el entrenamiento.....	61
5.30. Función de creación del modelo secuencial del segundo estudio.....	63
5.31. Función de creación del modelo LSTM del segundo estudio.....	63
5.32. Función de creación del modelo convolucional del segundo estudio.....	64
5.33. Función de creación del modelo híbrido LSTM y convolucional del segundo estudio.....	65
5.34. Función de creación del modelo GRU del segundo estudio.....	66
5.35. Función modificada para realizar la validación cruzada del segundo estudio.....	67
6.1. Matriz de confusión del modelo secuencial del primer estudio con optimizador 'Adam'.....	74
6.2. Matriz de confusión del modelo secuencial del primer estudio con optimizador 'Nadam'.....	74
6.3. Matriz de confusión del modelo secuencial del primer estudio con optimizador 'RMSProp'.....	75
6.4. Matriz de confusión del modelo secuencial del primer estudio con optimizador 'Adamax'.....	75
6.5. Matriz de confusión del modelo secuencial del primer estudio con optimizador 'Adagrad'.....	76
6.6. Matriz de confusión del modelo secuencial del primer estudio con optimizador 'Adadelta'.....	76
6.7. Matriz de confusión del modelo 'Random Forest'.....	77
6.8. Matriz de confusión del modelo 'Gradient Boosted'.....	77

6.9.	Gráfica de evolución de precisión en el modelo secuencial del primer estudio con optimizador 'Adam'	78
6.10.	Gráfica de evolución de pérdidas en el modelo secuencial del primer estudio con optimizador 'Adam'	79
6.11.	Gráfica de evolución de precisión en el modelo secuencial del primer estudio con optimizador 'Nadam'	79
6.12.	Gráfica de evolución de pérdidas en el modelo secuencial del primer estudio con optimizador 'Nadam'	80
6.13.	Gráfica de evolución de precisión en el modelo secuencial del primer estudio con optimizador 'Adagrad'	81
6.14.	Gráfica de evolución de pérdidas en el modelo secuencial del primer estudio con optimizador 'Adagrad'	81
6.15.	Gráfica de evolución de precisión en el modelo secuencial del primer estudio con optimizador 'Adadelta'	82
6.16.	Gráfica de evolución de pérdidas en el modelo secuencial del primer estudio con optimizador 'Adadelta'	82
6.17.	Gráfica de la evolución de la precisión en el entrenamiento del modelo 'Gradient Boosted'	83
6.18.	Gráfica de la evolución de la precisión en el entrenamiento del modelo 'Random Forest'	84
6.19.	Gráfica de la evolución de las pérdidas en el entrenamiento del modelo 'Gradient Boosted'	84
6.20.	Gráfica de la evolución de las pérdidas en el entrenamiento del modelo 'Random Forest'	86
9.1.	Código de verificación de uso de 'GPU' por 'Tensorflow'	106
9.2.	Entorno de trabajo 'Google Colaboratory Pro'	107
9.3.	Entorno de trabajo 'Visual Code'	108
9.4.	Entorno de trabajo 'Jupyter Notebook'	109
9.5.	Código para montar 'Google Drive' sobre 'Google Colaboratory'	110
9.6.	Código para usar 'Sklearn' aunque esté deprecado	112
9.7.	Código de instalación de librerías del primer estudio'	111
9.8.	Código de instalación de librerías del segundo estudio'	112



1. Introducción

Las tecnologías basadas en inteligencia artificial han venido para quedarse. En cada vez más ámbitos de la vida cotidiana se están implementando sistemas que hacen uso de estas nuevas técnicas para facilitar tareas hasta ahora impensables. Cada vez es más utilizado en aspectos como en sistemas de recomendación comercial al usuario, detección de spam y phishing, reconocimiento automático de imágenes y voz o asistentes virtuales [1]. La inteligencia artificial tiene sus inicios en una simple cuestión planteada por Alan Turing en 1950. ¿Pueden las máquinas pensar? O, dicho de otra manera, ¿pueden las máquinas pensar del modo en como lo hacen los seres humanos? John McCarthy, pionero en la inteligencia artificial, dio la primera respuesta creando el primer sistema capaz de aprender igual que un ser humano sin usar estrictas reglas de programación.

La inteligencia Artificial tuvo su auge en las décadas de los 50 y los 60, pero debido a las exageradas expectativas y a la limitación de las tecnologías de la época se produjo un estancamiento en los años 70 [1]. No obstante, la tecnología siguió evolucionando hasta la aparición de lo que sería una revolución en la industria, como lo fue el 'Deep Learning'. La evolución de la sociedad demandaba herramientas más precisas capaces de realizar funciones de abstracción más complejas. La limitación del 'Deep Learning' de aquella época era causa de la falta de información o datos. Por esta razón, dichas tecnologías no presentaban una mejora significativa respecto a las tecnologías basadas en 'Machine Learning'. En conclusión, una inteligencia artificial podrá aprender mejor y ser más precisa si podemos mostrarle la mayoría de los escenarios posibles.

Gracias a la incorporación de grandes bases de datos como 'ImageNet' [2] ha sido posible implantar nuevas tecnologías de detección y clasificación de imágenes con precisiones superiores a otros sistemas. Basándose en los



conceptos anteriores se ha demostrado que cuanto mayor sea la base de datos utilizada por una inteligencia artificial mayor será su precisión en sistemas 'Deep Learning', superando incluso a las tecnologías basadas en 'Machine Learning'.

Si hablamos de las aplicaciones del 'Deep Learning' podemos encontrar una cantidad de campos de investigación inmenso. Los trabajos que podían ser repetitivos, laboriosos o aburridos para ser realizados por un ser humano ahora pueden ser realizados por una inteligencia artificial y con mayor efectividad. Podemos encontrar estas tecnologías en diversos ámbitos, tales como aplicaciones tecnológicas industriales y del sector de las telecomunicaciones como aquellas relativas a aplicaciones médicas.

En el campo de la medicina resulta interesante el uso de herramientas basadas en inteligencia. Observando numerosos datos médicos y extrayendo las relaciones existentes entre ellos, se pueden implementar sistemas capaces de realizar predicciones que ayuden a los profesionales. Como se explicará más adelante, uno de los problemas existentes en algunos diagnósticos radica en la subjetividad que tienen en ellos los especialistas. Dotar a estos profesionales con herramientas que les permitan objetivar sus observaciones sería clave para la detección temprana de patologías y para la aplicación de tratamientos que mejoren la calidad de vida de los pacientes.

Un caso interesante y de creciente demanda a la hora de realizar diagnósticos está relacionado con la detección de los trastornos del espectro autista, conocido por sus siglas como TEA. Los casos de este trastorno llevan años en aumento y no se sabe a ciencia cierta cuál es el motivo [3]. Esta cuestión pone en énfasis la necesidad de mejora en los métodos de detección, ya que con ello se conseguiría realizar un diagnóstico más eficaz que permita tomar las medidas oportunas para mejorar la calidad de vida de estas personas.

Existen múltiples características que pueden ser identificadores clave en una posible presencia de trastornos del espectro autista. Sería interesante conseguir observar estos aspectos mediante tecnologías basadas en inteligencia artificial



para hallar patrones que permitan la detección temprana de estos pacientes. Además de ayudar a la detección del espectro autista, el estudio en este campo concreto puede ayudar a encontrar nuevas vías de detección para otro tipo de trastornos.

1.1. MOTIVACIONES

El campo dedicado a la detección del TEA y sus identificadores de detección es amplio, pero está lejos de conseguir métodos concretos que permitan una detección lo suficientemente precisa para poder hallar diagnósticos. Por ello es importante seguir estudiando y aportando conocimiento, no solo para los métodos de inteligencia artificial, sino para la búsqueda y mejora de nuevas características de medición que persigan el objetivo de crear métodos más robustos.

Si se consigue este cometido, la detección temprana y fiable de los trastornos del espectro autista será una realidad que permitirá aplicar métodos que garanticen a los pacientes una mejor adaptación a la sociedad y, en consecuencia, una mejor calidad de vida.

Además, con las nuevas técnicas existentes basadas en inteligencia artificial, se pueden conseguir nuevos resultados basados en estudios realizados con tecnologías anteriores. Resulta conveniente llevar a cabo este proceso de mejora continua debido a las crecientes mejoras en el mercado.

1.2. PLANTEAMIENTO INICIAL

Entre diversos métodos de detección, un enfoque a destacar son los estudios dedicados a las técnicas 'Eye Tracking', consistentes en identificar ciertas características sobre la detección visual de los sujetos ante ciertos estímulos. Resultaría interesante saber cuáles características provenientes de la visión



pueden resultar útiles para distinguir entre personas con características compatibles con TEA de las neurotípicas. Además, resultaría útil evaluar qué tipo de modelos podría aprender mejor de dichos identificadores.

Por otro lado, analizando ciertos estudios de aplicaciones en 'Machine Learning', existe la oportunidad de implementar nuevas arquitecturas 'Deep Learning' haciendo uso de diferentes optimizadores con las cuales se podrían conseguir nuevos resultados.

1.3. OBJETIVOS

1.3.1. Objetivo general

En términos generales, el objetivo de este proyecto consiste en la búsqueda e investigación de nuevas vías de detección del espectro autista a través de métodos basados en inteligencia artificial. Para llevarlo a cabo se indagará en dos estudios que harán uso de un tipo de información específica que permita comprobar el funcionamiento de distintas arquitecturas basadas en inteligencia artificial.

1.3.2. Objetivos específicos

Dentro de este marco de objetivo general, existirán ciertos hitos u objetivos específicos que se pretenderán cumplir para alcanzar las metas establecidas.

- **Mejora de los métodos basados en 'Machine Learning':** En primer lugar, se pretende implementar modelos basados en aprendizaje profundo que puedan significar en una mejora significativa respecto a los modelos



‘Machine Learning’. Para ello se evaluarán los modelos entre ambas tecnologías, como es el caso del primer estudio con el uso de modelos de árboles de decisión.

- **Evaluación de los modelos usando distintos tipos de optimizadores:** Una vez implementados los modelos, a la hora de compilarlos, se harán uso de diferentes optimizadores. Con ello se evaluará cuál o cuáles de ellos se ajustan mejor a los datos y hacen que el funcionamiento de predicción mejore.
- **Comparación de la evaluación sin validación y con validación en los métodos analíticos:** Para el primer estudio se compararán los resultados con los métodos de evaluación con y sin validación cruzada. Para bases de datos que cuentan con una cantidad reducida de información es conveniente hacer uso de este tipo de evaluaciones. De este modo nos cercioramos del comportamiento real de los modelos en términos de precisión y pérdidas.
- **Aproximación de valores nulos y preprocesamiento:** En las bases de datos que cuentan con valores de medición numéricos es interesante implementar métodos de detección y corrección de valores vacíos o nulos. De este modo se conseguiría no perder información relevante a la hora de entrenar los estudios y así mejorar la efectividad de las detecciones.
- **Estudio de correlación de las características de datos ‘Eye tracking’:** No todas las características de mediciones en un estudio tienen que ser válidas para realizar detecciones. Por ello, resulta conveniente realizar un estudio de correlación comparando cada una de las características con la detección de los casos de estudio.



- **Implementación y comparación de distintas arquitecturas ‘Deep Learning’:** Se pretende hacer uso de arquitecturas de inteligencia artificial que usen distintas tecnologías para sacar conclusiones sobre el tipo de modelos que mejor se pueden ajustar a determinados tipos de datos.
- **Implementación de modelos ‘Deep Learning’ híbridos:** Además de evaluar modelos que hagan uso de una arquitectura particular, se pretende implementar modelos basados en tecnologías híbridas para evaluar su funcionamiento.
- **Uso de herramientas de aceleración gráfica:** Con el fin de mejorar los tiempos de entrenamiento y optimizar el uso de recursos, se pretenderá hacer uso de la tecnología de aceleración gráfica para trabajar con los diferentes casos de estudio.
- **Funcionamiento en distintos entornos de ejecución:** Se comprobará la ejecución de los distintos procesos en entornos y aplicaciones de diferentes características con el fin de evaluar con qué entorno puede ser más conveniente trabajar.



2. Planificación

El proyecto consta de dos estudios diferentes, cada uno específico a una base de datos de cierto tipo. Podemos dividir el proyecto en diferentes fases en las que se incluirían la fase de documentación, la fase de desarrollo, la fase de análisis de resultados y la corrección de errores (Tabla 2.1.).

La fase de documentación inicial consistirá en el estudio previo del campo de investigación. En primer lugar, existirá una fase de aprendizaje dedicada a la adquisición de conocimientos sobre el 'Deep Learning' y sus aplicaciones. Además, se realizará una evaluación del estado del arte del sector. Esta primera fase se realizará en dos meses para asegurar la suficiente recolección de información.

Después de la fase de documentación se iniciará la fase de desarrollo. En esta etapa se procederá a realizar las distintas tareas relacionadas con la búsqueda y análisis de la base de datos, su preprocesado, la implementación de modelos y su evaluación. Se hará distinción sobre ambos estudios en cuanto a plazos, puesto que cada uno tomará un tiempo de desarrollo diferente según su complejidad. En este caso, al primer estudio se le destinarán dos meses y dos semanas de plazo mientras que, al otro estudio tres meses y dos semanas debido a su extensión.

Tras el desarrollo de ambos estudios, se procederá a la fase de evaluación y de extracción de conclusiones. En esta fase se realizará tanto la memoria final como una de corrección de errores. Esta fase contará con un tiempo establecido de dos meses.



Documentación (2 Meses)		
	Aprendizaje	4 semanas
	Estado del arte.	4 semanas
Desarrollo 1er Estudio (2 meses y 2 semanas)		
	Búsqueda de dataset	2 semanas
	Análisis de dataset.	2 semanas
	Preprocesado.	2 semanas
	Implementación de modelos.	2 semanas
	Evaluación de modelos.	2 semanas
Desarrollo 2º Estudio (3 meses y 2 semanas)		
	Búsqueda de dataset	2 semanas
	Análisis de dataset.	2 semanas
	Preprocesado.	4 semanas
	Implementación de modelos.	2 semanas
	Evaluación de modelos.	4 semanas
Evaluación de resultados (2 Meses)		
	Creación de memoria.	6 semanas
	Corrección de errores.	2 semanas
TOTAL: 10 Meses		40 semanas

Tabla 2.1.- Organización de tareas y sus plazos.

Para visualizar de modo más gráfico las tareas con sus respectivos plazos, conviene hacer un diagrama de GANT (Figuras 2.1 y 2.2). En dichas figuras se describen cada una de las tareas citadas en la Tabla. 2.1.

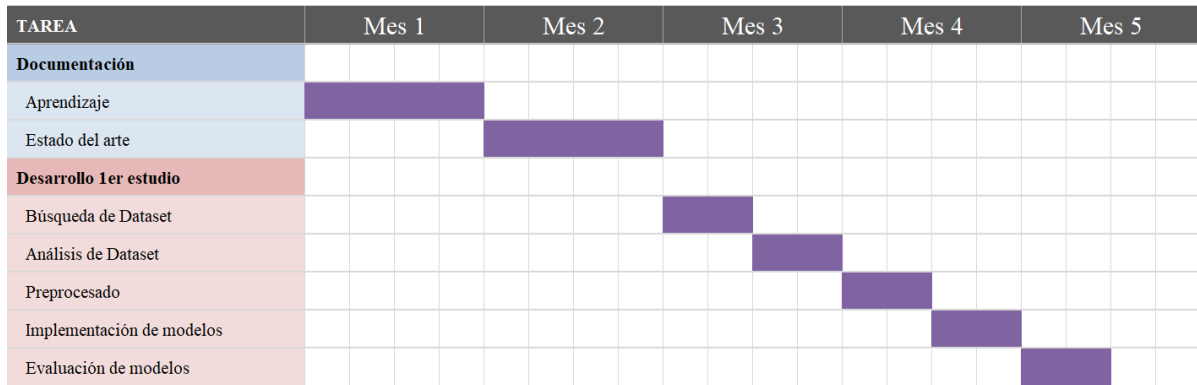


Figura 2.1.- Diagrama de GANT (1)

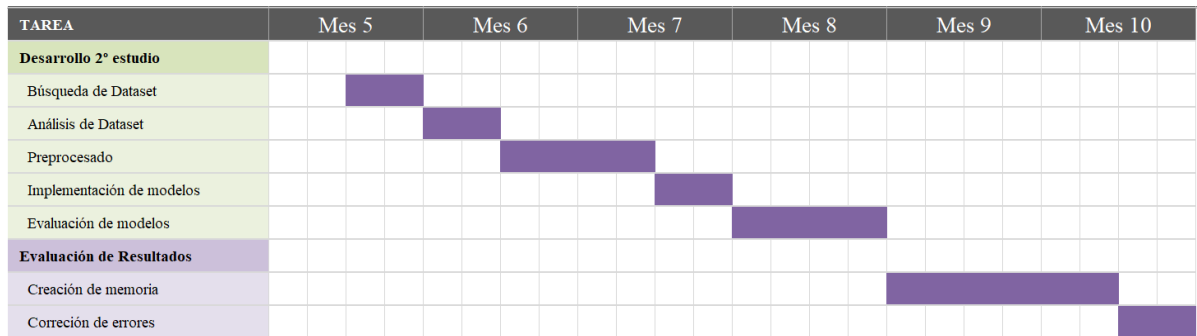


Figura 2.2.- Diagrama de GANT (2)

Se puede observar cómo el segundo estudio llevará más tiempo respecto al primero. Debido a la complejidad de la base de datos y su extenso proceso de preprocesado, es necesario dedicarle más tiempo y más recursos en comparación al desarrollo del primer estudio. Los últimos dos meses del proyecto se destinarán para realizar la memoria, aunque durante todo el proceso se realizarán distintas evaluaciones y anotaciones para poder desarrollar más eficazmente el resto de las tareas.



3. Estado del arte

La detección del espectro autista era una tarea que se había basado mayoritariamente en métodos clínicos. En el contexto actual, se están introduciendo de forma progresiva métodos de detección basados en características objetivas que permiten reducir fallos en los diagnósticos. De este modo, se están implementando diferentes soluciones de detección basadas en inteligencias artificiales capaces de realizar análisis en bases de datos, detectar patrones característicos y extraer predicciones de ellos.

3.1. DETECCIÓN DEL TEA CON MÉTODOS CLÍNICOS

Las primeras aproximaciones de detección del trastorno del espectro autista se basaron en las propias observaciones conductuales del sujeto a estudiar. Los sujetos estudian características conductuales, como la existencia de problemas de comunicación social o la existencia de patrones repetitivos [3]. Esta práctica es la más habitual hoy en día para la detección temprana del espectro autista. Aun así, resulta ser un método sesgado por la subjetividad de los métodos utilizados, como cuestionarios o evaluaciones de profesionales [4].

Gracias a los métodos clínicos se ha podido establecer una base sólida de conocimiento acerca de las características relacionadas con el espectro autista. Con ello se pueden aplicar nuevos métodos de detección basados en características objetivas que permitan una detección eficiente y rápida.

3.2. DETECCIÓN DEL TEA CON TÉCNICAS DE CLASIFICACIÓN

De la búsqueda de mayor objetividad en los métodos clínicos de detección del espectro autista, surge la idea de implementar técnicas de clasificación basadas



en 'Machine Learning'. De este modo se suministrarán las características clínicas obtenidas de los métodos analíticos a una inteligencia artificial. A partir de esa información suministrada, la inteligencia artificial podrá hacer una predicción probabilística para discernir entre presencia o no de TEA.

Un estudio basado en técnicas de clasificación [4] ha probado diversos algoritmos, tales como 'Naive Bayes', 'Support Vector Machine' (SVM), 'Logistic Regression' y 'K-Nearest Neighbors (KNN) aplicados sobre una base de datos de características pertenecientes a 'UC Irvine Machine Learning Repository' [5]. Dicha base de datos está formada por diversas observaciones extraídas de los métodos clínicos tradicionales. Dichas observaciones recogen información sobre el comportamiento del sujeto ante diversos escenarios, tales como la capacidad del sujeto a responder cuando se le llama por su nombre, uso de gesticulaciones o la presencia de empatía en determinadas situaciones. Estas características se contrastan además con técnicas basadas en redes neuronales estándar y convolucionales. Comparando los algoritmos en términos de precisión, se observan mejores resultados con el uso de las redes neuronales convolucionales.

Otro estudio realizado por Vakazkar et al. [6] hace uso de una base de datos consistente en atributos predefinidos por métodos clínicos. Para este estudio se implementan herramientas basadas en 'Machine Learning' usados para predecir con cierta exactitud la presencia de características del TEA. De los resultados de dicho estudio, se ha obtenido un valor de precisión correspondiente al 97,15% en aquellos modelos basados en regresión logística (LR). Dicho modelo se ha comparado con otros como 'Support Vector Machine' (SVM) con una precisión del 93,84%, 'Random Forest Classifier' (RFC) con una precisión del 81,52%, 'Naive Bayes' (NB) con un resultado del 94,79% y con el modelo 'KNN' con una precisión del 90,52%. Todos los modelos descritos anteriormente han obtenido un nivel de precisión inferior respecto al modelo de regresión logística.

En estos estudios se observa que las técnicas de clasificación hacen que los mejores valores de precisión se obtienen con las redes neuronales convolucionales.



3.3. DETECCIÓN DEL TEA DESDE IMÁGENES ‘RS-FMRI’

Se han encontrado ciertas características en las imágenes ‘rs-fMRI’ (Resting estate functional magnetic resonance imaging) que pueden servir como identificador a la hora de detectar el T.E.A. usando inteligencia artificial. El estudio realizado por Heinsfeld et al. [7] propone la aplicación de una inteligencia artificial sobre una extensa base de datos de imágenes rs-fMRI de sujetos con espectro autista llamada ABIDE I. Dicho estudio ha identificado ciertos patrones de conectividad funcional característicos con los sujetos con espectro autista. Más en concreto, se ha observado una anti-correlación de funciones cerebrales correspondientes a las partes interior y posterior del cerebro.

El estudio citado anteriormente ha propuesto el uso de diversos métodos de ‘Deep Learning’ que combina métodos supervisados y no supervisados. Como se ha comentado en las técnicas de detección basadas en clasificación, el objetivo primordial es el encontrar ciertas características para el estudio que estén exentas de subjetividad. Para ello es preferible el uso de métodos no supervisados, evitando así el procedimiento subjetivo de etiquetado de las muestras.

Haciendo uso de métodos ‘Deep Learning’ no supervisado se ha obtenido una precisión del 70%. Comparando los resultados con el uso de métodos de clasificación como SVM y RF, se han obtenido valores de precisión de 65% y 63% respectivamente. Otro estudio basado en métodos de clasificación ‘Leave-One-Out Cross-Validation’ (LOOCV) [8] ha obtenido resultados de precisión cercanos al 89% en sujetos menores de 20 años.

La tendencia actual consiste en diseñar métodos híbridos donde la subjetividad del etiquetado de muestras no tome un papel determinante y se usen métodos no supervisados capaces de detectar patrones característicos.



3.4. DETECCIÓN DEL TEA SEGÚN RASGOS FACIALES

Con la implementación de las redes neuronales convolucionales, se han podido implementar ciertas técnicas de detección capaces de detectar patrones característicos sin la necesidad de supervisión o etiquetado.

El estudio realizado por Beary et al. [9] propone la aplicación de un método de clasificación basado en el modelo de redes convolucionales de 'MobileNet' [10] que permite detectar los sujetos con TEA según sus rasgos faciales. Este método ha resultado ser efectivo, consiguiendo una precisión del 94,6%. Con este método el modelo detecta los patrones faciales característicos de los sujetos con trastorno del espectro autista para realizar predicciones en otras muestras.

Otro estudio realizado por Lu y Perkowski [11] propone el análisis de los fenotipos faciales por medio de una red convolucional basada en técnica de 'transfer-training' sobre el modelo 'VGG16'. De este modo se ha conseguido una precisión del 95%

3.5. DETECCIÓN DEL TEA DESDE DATOS DE 'EYE TRACKING'

Una característica de las personas con trastorno del espectro autista es la presencia de una atención visual atípica, refiriéndose al diverso modo de procesar visualmente ciertos estímulos si se compara una persona TEA con otra neurotípica.

Un estudio realizado por Ahmed et al. [12] ha comparado tres diversas técnicas de 'Machine Learning', 'Deep Learning' y una técnica híbrida entre ambas. Como técnicas de Machine Learning, se ha combinado el método 'LBP (Local Binary Pattern)' (Local Binary Pattern) con 'GLCM (Gray Level Co-occurrence Matrix)' [13] además de técnicas de redes neuronales convolucionales de los modelos 'GoogleNet' [14] y 'ResNet18' [15]. Finalmente se ha combinado dicha red



convolucional con la técnica de clasificación SVM. Ha demostrado ser más precisa la técnica de 'GoogLeNet' + SVM con un 95,5% de precisión.

Además de los estudios dedicados a la implementación de técnicas de machine learning, el estudio realizado por Cilia et al. [16] consigue elaborar una base de datos de sujetos que presentan tanto trastorno del espectro autista como neurotípicos. En ella se recolecta información sobre diversas características relacionadas con la atención visual, tales como movimiento de los ojos en diversos ejes, tamaño de las pupilas, etc. El estudio no aplica métodos de inteligencia artificial para validar el uso de este dataset en la detección del espectro autista.

3.6. DETECCIÓN DEL TEA DESDE SEÑALES EGG

Además de las características encontradas del trastorno del espectro autista en las imágenes cerebrales, se han realizado estudios para buscar ciertos patrones característicos en la actividad cerebral por medio de las señales electroencefalográficas o también llamadas señales EGG. Este método de detección resulta ser menos intrusivo para algunas personas con trastorno autista, muy para tener en cuenta debido a que suelen ser personas que presentan una alta hipersensibilidad a los estímulos externos.

En este campo existen estudios que detectan ciertos patrones de las señales EGG desde modelos de inteligencia artificial, como el caso del estudio de Radhakrishnan et al. [17] en el cual se comparan distintas redes neuronales de convolución como 'AlexNet', 'VGGNet' o 'DenseNet' entre otras. Además, se han comparado distintos métodos de ResNet como 'ResNet50', 'ResNet101' y 'ResNet152'. Se ha comprobado que el modelo que consigue una mejor precisión es el 'ResNet50' con una precisión del 81%.



Otro estudio realizado por Olaniyan et al. [18] implementa un método de clasificación basado en ‘Machine Learning’ que tiene en cuenta, entre otras variables, la medición de las señales EGG, los puntos de referencia de los sensores y las actividades que están realizando en el momento de tomar las muestras. Para realizar la clasificación y detección del espectro autista se han implementado modelos ‘Generic Algorithm’ (GA) [19]

Todos los métodos de detección del trastorno del espectro autista citados en este apartado se pueden ver de un modo resumido en la Tabla 3.1. Se observa que para abordar el estudio existen diversas características susceptibles de investigación. Aquellas que resultan menos invasivas de realizar y con resultados prometedores son aquellas características extraídas por ‘Eye Tracking’ y por señales ‘EGG’

Métodos de Detección	Resumen
Métodos clínicos	Método tradicional de detección basado en las observaciones y pruebas médicas.
Técnicas de clasificación sobre métodos clínicos	Aplicación de métodos ‘Machine Learning’ y ‘Deep Learning’ para la clasificación en los métodos clínicos.
Imágenes rs-fMRI	Detección basada en ‘Deep Learning’ sobre imágenes cerebrales de los pacientes.
‘Eye Tracking’	Aplicación de arquitecturas basadas en ‘Deep Learning’ para detectar características relacionadas con el TEA en el modo que los pacientes captan los estímulos a través de la visión.
Señales E.G.G.	Análisis de las señales electromagnéticas del cerebro para detectar patrones compatibles con el TEA

Tabla 3.1.- Resumen de técnicas de detección del TEA.



4. Metodología de Trabajo

4.1. HARDWARE

Para realizar estudios relacionados con inteligencia artificial es necesario contar con ciertas características específicas en cuanto a hardware se refiere. Se trabajará con bases de datos muy extensas que deberán ser procesados para así abstraer la información que permita clasificar las muestras. Es esencial por ello el uso de aceleradores gráficos (también conocidos como GPU's) debido a su eficiencia a la hora de entrenar los modelos.

En el caso de este estudio, se hará uso de dos ordenadores con tarjetas gráficas dedicadas de NVIDIA y un entorno virtual creado en 'Google Colaboratory' que permitirá hacer uso de numerosos tipos de aceleradores gráficos.

4.1.1. Configuración de acelerador gráfico

Al configurar adecuadamente el entorno de aceleración gráfica se deben de considerar diversos aspectos cuando se trabaja con inteligencia artificial.

- **Actualización de los drivers de NVIDIA:** Es importante asegurarse de que los dispositivos dispongan de la actualización de drivers más reciente para su correcto funcionamiento. Dichos drivers se pueden actualizar desde la página oficial de NVIDIA.
- **Instalación de las herramientas 'cuDNN':** Estas librerías, pertenecientes a la compañía NVIDIA, permiten entrenar modelos de redes neuronales utilizando la aceleración GPU propia de los dispositivos de dicha compañía.



- **Uso de Tensorflow en entorno GPU:** Es importante cerciorarnos de que Tensorflow esté utilizando el entorno de aceleración GPU para entrenar los modelos. En el caso de Tensorflow, existen diversas herramientas que nos permiten validar si el programa reconoce la existencia de dispositivos GPU. De este modo nos aseguramos de que estén todas las herramientas correctamente instaladas.

4.1.2. Servicios hardware en la nube

Para acelerar el entrenamiento de los modelos y hacer pruebas en diferentes entornos de aceleración se ha optado por utilizar el servicio de ‘Google Colaboratory’. Se trata de un servicio perteneciente a la empresa Google que permite crear, modificar y ejecutar programas en entornos virtuales de la nube. Se trata de una herramienta muy interesante en el campo de la inteligencia artificial ya que se puede trabajar en distintos entornos de aceleración gráfica. El servicio de pago ‘Google Colaboratory Pro’ añade numerosas ventajas en el estudio al tener la posibilidad de acceder a dispositivos de aceleración gráfica más potentes, permitir la ejecución de más de un entorno virtual en el mismo tiempo y minimizar las interrupciones de conexión constantes de la versión gratuita.

4.2. SOFTWARE

Por software nos referiremos como las herramientas y aplicaciones usadas para realizar el estudio. Haremos distinción entre los entornos de ejecución en base Python y las diferentes librerías utilizadas para procesar los datos y entrenar los modelos.



4.2.1. Entornos de Ejecución

En este apartado englobaremos las distintas aplicaciones y servicios utilizados para la realización de este estudio.

- **Anaconda Navigator:** Se trata de una interfaz que contiene diferentes entornos y herramientas para trabajar en numerosos campos, especialmente en aquellos de análisis de datos y ‘Machine Learning’. En esta interfaz se pueden configurar varios entornos que pueden adaptarse a diferentes características del estudio. En este caso interesa configurar un entorno que tenga acceso a las herramientas de aceleración por GPU.
- **Jupyter Notebook:** Para realizar un estudio es imprescindible seguir un orden en cuanto a los diferentes procesos que se han de seguir. Para ello, resulta muy cómodo hacer uso de los ‘Jupyter Notebooks’, un tipo de archivo que permite combinar las diferentes partes de código en subíndices con la posibilidad de hacer comentarios. Así se dota al estudio de una apariencia más gráfica y entendible al comprender los elementos correspondientes al estudio.
- **Visual Studio Code:** Se trata de un programa perteneciente a Visual Studio con interfaz sencilla que permite trabajar con numerosos tipos de lenguajes de programación. Se puede configurar para que trabaje con entornos Python de Anaconda y resulta ser una herramienta útil y dinámica para trabajar. Esta aplicación admite la lectura y edición de archivos Jupyter, lo cual facilita la organización de las diferentes etapas del estudio.
- **Google Colaboratory:** Con el fin de agilizar el proceso de entrenamientos de los modelos se hará uso del entorno virtual de ‘Google Colaboratory’, en el cual tenemos la posibilidad de importar los archivos ‘Jupyter Notebook’ para su ejecución en distintos entornos de aceleración gráfica.



4.2.2.Librerías

Dentro de los entornos de ejecución, deberemos de utilizar diversas librerías relacionadas con el procesamiento de las bases de datos y la creación de los modelos.

- **Tensorflow:** Será la librería principal para utilizar. En ella disponemos de todas las herramientas necesarias para crear, compilar y entrenar los diferentes modelos de 'Deep Learning' [20]. Se instalará la versión más reciente que permita la detección automática de los dispositivos de aceleración GPU, teniendo en cuenta la compatibilidad con otras librerías.
- **Sklearn:** Dicha librería cuenta con numerosas herramientas para el análisis de bases de datos. En este estudio se harán uso de dichas herramientas para implementar las funciones de separación de muestras, validación cruzada y generación de gráficas como la matriz de confusión o la curva de precisión.
- **Numpy:** Se trata de una herramienta esencial en el ámbito de la ciencia de datos. Junto con la librería de Pandas (extensión de numpy) se utilizarán en la etapa de preprocesamiento para crear los subconjuntos de datos y realizar las modificaciones pertinentes en ellos.
- **Pandas:** Para la manipulación de datos se utilizará Pandas. Englobará diferentes funciones relacionadas con la agrupación y modificación de conjuntos de datos para la etapa de preprocesamiento.
- **Csv:** Para la lectura de las bases de datos en formato '.csv', así como para realizar el guardado de las bases de datos preprocesadas, hará uso de esta herramienta.



- **Matplotlib:** Utilizada a la hora de generar los gráficos correspondientes a las distintas métricas del estudio del modelo.

4.3. METODOLOGÍA DE TÉCNICAS DE DEEP LEARNING SOBRE MÉTODOS ANALÍTICOS

Los métodos basados en pruebas analíticas resultan ser los más subjetivos en la detección de cualquier trastorno. La clasificación y aprobación de un profesional es determinante a la hora de entrenar un modelo capaz de realizar ciertas detecciones. El estudio realizado por Vakadkar et al. [6] implementa métodos de clasificación basados en ‘Machine Learning’ en estudios clínicos realizados por profesionales según distintas características para el entrenamiento de los modelos.

Además de los modelos de ‘Machine Learning’ aplicados, pueden implementarse otras técnicas basadas en ‘Deep Learning’ que podrían aportar nuevas conclusiones.

En este estudio se aplicarán distintas técnicas basadas en modelos secuenciales y modelos basados en árboles de decisión, tales como ‘Gradient Boosted’ y ‘Random Forest’ para validar su comportamiento en este caso de estudio.

4.3.1. Base de datos y preprocesado

En la base de datos del estudio inicial de donde partimos [19] se utilizan características extraídas de métodos analíticos realizados por profesionales. Estas características se refieren a observaciones presentes o no en cada individuo estudiado tal como se describe en la Tabla 4.1.



Dataset variable	Description
A1	Child responding to you calling his/her name
A2	Ease of getting eye contact from child
A3	Child pointing to objects he/she wants
A4	Child pointing to draw your attention to his/her interests
A5	If the child shows pretense
A6	Ease of child to follow where you point/look
A7	If the child wants to comfort someone who is upset
A8	Child's first words
A9	If the child uses basic gestures
A10	If the child daydreams/stares at nothing

Tabla 4.1.- Tabla de características del estudio realizado [6]

Además de estas características se tienen en cuenta otros aspectos tales como la edad, el género, la etnia o la existencia de familiares con TEA. Se tratan de características categóricas que precisan de un preprocesado antes de introducirlos a los modelos basados en inteligencia artificial. En primer lugar, es necesario convertir las variables de formato texto en un formato legible para el modelo. Para ello se optará por asignarles un valor numérico determinado para cada grupo de características, como se observa en la Tabla 4.2. para la clasificación, en la Tabla 4.3. para el género, la Tabla 4.4. para la existencia o no de familiares con TEA o para las Tablas 4.5 y 4.6 que denota la presencia o no de ictericia.

Diagnóstico TEA	Valor numérico
Sí	1
No	0

Tabla 4.2.- Conversión numérica de característica de diagnóstico TEA



Género	Valor numérico
Femenino	1
Masculino	0

Tabla 4.3.- Conversión numérica de característica de género

Miembros familiares con TEA	Valor numérico
Sí	1
No	0

Tabla 4.4.- Conversión numérica de característica de familiares con TEA

Participante con Ictericia (Jaundice)	Valor numérico
Si	1
No	0

Tabla 4.5.- Conversión numérica de característica de presencia de ictericia

El propio estudio indica que la característica referida a 'QChat-10-Score' debe de ser eliminada antes de utilizar la base de datos en los modelos debido a que genera sobreajuste a la hora de entrenar los modelos. Además, se eliminarán las columnas inservibles para el entrenamiento como son las características 'Número de caso' o 'Persona que completó el test'

Etnia	Valor numérico
White European	1
Hispanic	2
Black	3
Asian	4
South asian	5

Tabla 4.6.- Conversión numérica de característica de etnia (1)



Etnia	Valor numérico
Native Indian	6
Others	7
Latino	8
Mixed	9
Pacífica	10
Middle Eastern	11

Tabla 4.7.- Conversión numérica de característica de etnia (2)

Realizando la conversión numérica de las características ya será posible el entrenamiento de los modelos. Se crearán tres subconjuntos de datos para entrenamiento, test y validación. El subconjunto de entrenamiento corresponderá al 80% de todos los datos de la dataset mientras que el 20% restante irá destinado para el subconjunto de test y validación. Dentro del conjunto de prueba y validación se destinará el 80% para crear el subconjunto de test y el 20% restante para el subconjunto de validación. Esta división de los datos se puede observar gráficamente en la Figura 4.1.

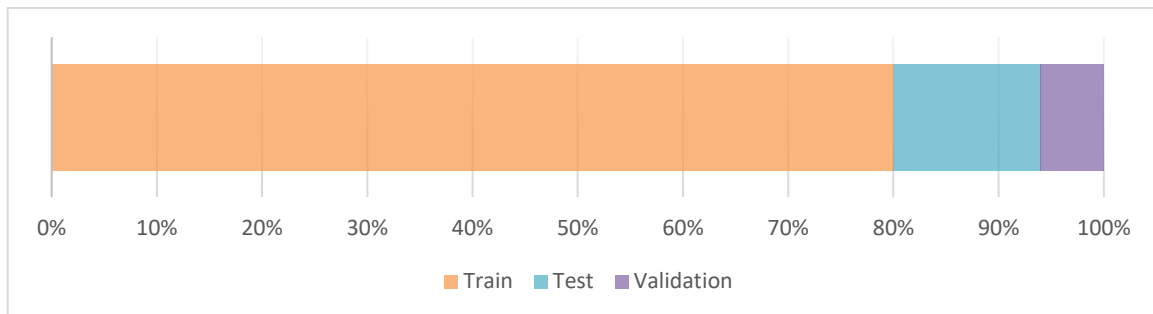


Figura 4.1.- Reparto de la dataset para entrenamiento, test y validación.

El modelo usará el subconjunto correspondiente para entrenamiento y tendrá en cuenta el subconjunto de validación para medir la evolución del entrenamiento. Antes de implementar dichos modelos, es necesario realizar una separación de las características a estudiar y la catalogación de cada una de las muestras.



Es importante cerciorarse de que ninguna característica de las muestras presente algún dato inválido o nulo. Para ello se comprobará en cada una de las muestras su valor para utilizarla o no dependiendo de su formato.

4.3.2. Implementación de modelos ‘Deep Learning’

Para evaluar las características de esta dataset se tendrán en cuenta distintos tipos de arquitecturas. De este modo se conseguirá evaluar y comparar su comportamiento para determinar cuál se podría ajustar mejor. Además, se considerará el uso de varios optimizadores al compilar el modelo para ver su comportamiento en el entrenamiento.

4.3.2.1. Modelos secuenciales

Uno de los modelos a evaluar consistirá en una arquitectura basada en un modelo secuencial. Seguirá la estructura descrita en la Tabla 4.8. donde se puede observar que la arquitectura del modelo está compuesta por una capa de entrada definida por quince unidades, correspondiente al número de características de la base de datos, y una capa de salida de dos unidades, correspondientes a los dos tipos de categorías ‘TEA’ o ‘No TEA’.

En las capas ocultas se configurarán dos capas densas de diez unidades que, entre medias, se incluirá una capa de ‘dropout’ al 10%. Dicha capa permitirá que el modelo no se sobreajuste al desconectar cierta porción de neuronas entre ambas capas densas.



Capa	Tamaño (unidades)
Densa (Input)	15
Densa_1	10
Dropout	10
Densa_2	10
Densa (Output)	2

Tabla 4.8.- Sumario de arquitectura del modelo secuencial.

A esta arquitectura de modelo se le aplicarán diferentes tipos de optimizador para evaluar su comportamiento.

- **Optimizador ‘Adam’:** Se considera el optimizador más equilibrado en su uso para los modelos secuenciales ya que resulta ser muy eficiente en términos computacionales y requiere poco espacio de memoria para su uso.
- **Optimizador ‘Nadam’:** Se trata de un optimizador basado en el algoritmo Adam al cual se le implementa la técnica de optimización ‘Nesterov’s Momentum’. Dicha técnica consigue acelerar la convergencia de gradiente descendiente y puede ser conveniente para la mejora en algunos modelos.
- **Optimizador ‘RMSProp’:** En vez de dar uso de la técnica de ‘Nesterov’s Momentum’, el optimizador RMSProp hace uso de ‘Plain Momentum’. Dicho optimizador hace uso del valor medio de los gradientes para calcular las varianzas que ajustan los parámetros del modelo a la hora de realizar el entrenamiento.
- **Optimizador ‘Adamax’:** Este tipo de optimizador se trata de una variante del optimizador ‘Adam’ que suele ser utilizado en estudios con datos variantes en el tiempo. Está basado en un método de optimización denominado ‘Norma infinita’ capaz de ajustar el aprendizaje a las características de los datos.



- **Optimizador ‘Adagrad’:** El optimizador ‘Adagrad’ se adapta a la frecuencia de actualización de los parámetros de entrenamiento. Es utilizado para datos basados con variación temporal, ya que estudia las actualizaciones que existen entre las muestras.
- **Optimizador ‘Adadelta’:** Extensión del optimizador ‘Adagrad’ que hace uso de ventanas de gradiente en vez de acumular todos los gradientes generados durante el entrenamiento. Así, el modelo que use este optimizador puede seguir aprendiendo sin importar el número de actualizaciones realizadas.

Se entrenará y evaluará el modelo descrito antes comprobando los seis optimizadores antes descritos. Dicha evaluación se realizará con y sin validación cruzada. Para este caso concreto se procederá a realizar el entrenamiento con treinta ciclos o épocas teniendo en cuenta la cantidad de información y la complejidad de los datos del estudio.

4.3.2.2. Modelos basados en árboles de decisión

Para comparar los modelos basados en ‘Deep Learning’, se usarán otros métodos basados en ‘Machine Learning’ como son los modelos basados en árboles de decisión. Estas arquitecturas sencillas suelen obtener buenos resultados a la hora de realizar tareas de clasificación, como es el caso de este estudio.

Tensorflow cuenta con una librería llamada ‘Tensorflow Decision Forest’ que contiene numerosas herramientas para aplicar estas arquitecturas de un modo sencillo. En este caso se hará uso de los modelos ‘Gradient Boosted’ y ‘Random Forest’.



- **Modelo Gradient Boosted:** Este modelo de árbol de decisión genera numerosos árboles de decisión de forma secuencial. Cada árbol generado tiene en cuenta los errores de los árboles generados anteriormente para así ir paulatinamente reduciendo el error de predicción del modelo.
- **Modelo Random Forest:** El modelo 'Random Forest' genera sucesivos árboles de decisión basados en muestras aleatorias de la base de datos de entrenamiento. Esta arquitectura de modelo es muy robusta en cuanto a sobreajuste del entrenamiento.

Como con el modelo secuencial, se evaluarán ambos modelos con y sin validación cruzada.

4.3.3. Evaluación de los modelos 'Deep Learning'

La evaluación de los modelos anteriormente descritos se realizará, en primer lugar, sin validación cruzada y luego realizando su validación por medio del método 'K-Folder'.

El método de validación 'K-Folder' consiste en el uso de un número determinado de particiones dentro de la dataset denominado 'k' o 'Folders', correspondiente al número de subconjuntos realizados en toda la dataset. En este caso concreto se realizará un estudio de validación cruzada con diez particiones ($k = 10$ conjuntos).

En cada iteración se seleccionará un folder como test, un folder de diez en este caso, y el resto se utilizará para entrenar el modelo. Este proceso se realizará para cada una de las particiones, cambiando en cada iteración el folder destinado a test y los restantes adjudicados para entrenamiento.

Un ejemplo del funcionamiento de este método de validación se puede ver en la Figura 4.2. En este ejemplo se hace uso de tres conjuntos de datos para simplificar el entendimiento.

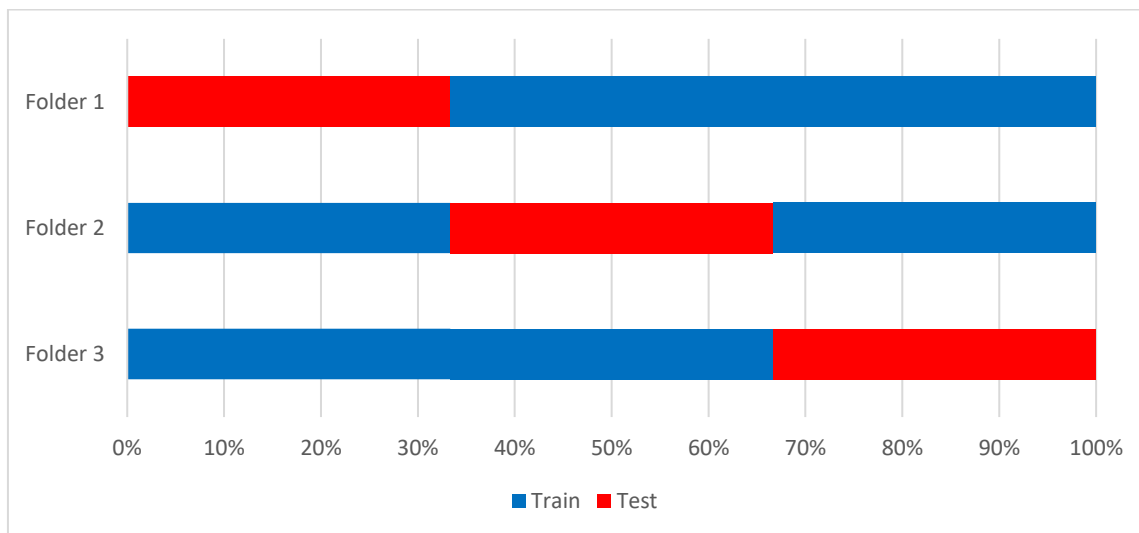


Figura 4.2.- Ejemplo de validación K-Folder con tres conjuntos en un dataset.

En el caso de los modelos secuenciales, estos se entrenarán con un número de treinta etapas unas diez veces, iterando en cada una de ellas los subconjuntos de datos correspondientes a entrenamiento y test. Se compararán los resultados con el entrenamiento sin validación, consistente en el entrenamiento de toda la base de datos sin particiones en treinta etapas.

4.4. METODOLOGÍA DE TÉCNICAS DE DEEP LEARNING SOBRE DATOS 'EYE TRACKING'

A la hora de implementar herramientas capaces de detectar cualquier tipo de trastornos es importante fijarse en características objetivas comunes de los pacientes para asegurar cierta precisión en la detección. Para ampliar el estudio sobre las técnicas de detección del TEA, se ha optado por utilizar un caso en el que trabaje con características objetivas y tangibles como se ha comentado anteriormente. Usando la base de datos del estudio realizado por Cilia et al. [16] se pretende implementar modelos basados en 'Deep Learning' para trabajar con



datos de 'Eye Tracking', confiriendo así un valor añadido al trabajo citado anteriormente.

A partir de los datos extraídos de la base de datos anterior, se realizará un procesamiento para que se puedan aplicar modelos de aprendizaje en ellos capaces de abstraer las características de cada muestra y su evolución temporal. Para ello se aplicarán arquitecturas de distintos modelos para evaluar cuales se amoldan mejor a las características del estudio. Además de ello, se comprobará qué uso de determinadas características pueden resultar en una mejora en cuanto a precisión de detección. Para ello se realizará un estudio de correlación para hallar qué características tienen mayor relación con la detección.

En primer lugar, se realizará un estudio con el optimizador 'Adam' para evaluar que arquitecturas funcionan mejor con la dataset. Finalmente se comprobarán los modelos con el uso del optimizador 'Adamax' que suele ser más eficaz para datos variantes en el tiempo.

4.4.1. Base de datos y preprocesado

El estudio en cuestión cuenta con múltiples archivos en formato '.csv' que conforman los datos de las mediciones 'Eye Tracking' de todos los participantes. Además de este conjunto de datos, existe otro archivo correspondiente a los metadatos de la dataset, identificando a cada uno de los participantes y su clasificación como neurotípico o participante con TEA.

La base de datos recoge múltiples características de diversos formatos. Podemos distinguir dos tipos de datos en función de la información que contienen, además de la identificación del participante.



- **Identificación del participante:** Corresponde al número de identificación del participante. Cada una de las muestras tiene dicha etiqueta para cotejarla con los metadatos del participante.
- **Datos de información:** Este tipo de información recoge el entorno y las características de procedimiento a la hora de tomar las medidas. En este apartado contamos con diferentes características como el tipo de estímulo que se le expone al participante (información del video reproducido, tipo de estímulo, duración del video, etc.). Este tipo de información se considerará como no relevante para el aprendizaje del modelo.
- **Datos de ‘Eye Tracking’:** Se refiere a la información perteneciente a las propias medidas del seguimiento de mirada de los participantes. En estos datos obtendremos la información efectiva para el estudio de los modelos de aprendizaje. En esta dataset se obtendrán características referidas al diámetro de las pupilas, posición del ojo, punto de mirada, etc.

Categoría	N.º de participantes
Neurotípico	30
T.E.A.	29

Tabla 4.9.- Cantidad de participantes por categoría.

Como se puede observar en la Tabla 4.9. la cantidad de participantes del estudio puede considerarse balanceada, siendo prácticamente el mismo número de participantes de un caso como del otro. Eso permitirá que el aprendizaje del modelo no esté desajustado y pueda aprender a categorizar entre ambos casos de un modo más preciso.

En primer lugar, hay que importar los datos de los múltiples archivos en la misma variable para su posterior procesado. Además, se debe identificar cada participante que identifique a la muestra y agregarle la etiqueta de clasificación.



Este proceso se lleva a cabo extrayendo el valor de la 'Id' del participante de cada una de las muestras para cotejarlo con los metadatos.

Una vez realizado este proceso, obtendremos la base de datos ya estructurada para efectuar el preprocesado y seleccionar de ella las características pertinentes para el aprendizaje. En la Tabla 4.10. Podemos observar alguna de estas características de la base de datos. Se hace distinción entre aquellas características que solo son informativas y aquellas otras susceptibles a ser usadas.

Información	'Eye Tracking'
Tiempo de Registro	Diámetro de pupilas
Hora	Puntos de visión
Ensayo	Área de Interés
Estímulo	Vectores de mirada
Inicio de Ensayo	Direcciones de desplazamiento
Fin de Ensayo	Tamaños de pupila
Color	Posiciones de ojos
Ratio de Tracking	-
Anotaciones	-
Contenido	-

Tabla 4.10.- Características de la dataset.

Para el aprendizaje se utilizarán aquellas características propias del proceso de registro de visión. Las características correspondientes a información sobre la toma de muestras se obviarán.

A partir de este punto, la base de datos se basará en información en formato numérico que representará información de posición de los diferentes ejes de posición del ojo y de visión, así como el diámetro o tamaño de las pupilas. La medición de estos parámetros es susceptible de presentar ciertos valores no



válidos o erróneos. Para ello es importante realizar un proceso de análisis y corrección para corregir estos errores puntuales que puedan afectar al aprendizaje.

Para este proceso se implementará un algoritmo capaz de leer cada una de las características en su evolución a lo largo del tiempo para así detectar valores nulos y modificarlos según los valores de las muestras adyacentes. Este algoritmo leerá el valor presente, su valor pasado y el futuro. Según los valores que detecte, realizará una acción u otra para corregir los valores de la base de datos. Este proceso será independiente entre participantes para no realizar aproximaciones erróneas.

- **Valor 'n':** Referido al valor presente de la muestra, referido al número de fila de la base de datos.
- **Valor 'n - 1':** Valor pasado de la muestra, referido al valor de la fila anterior de la muestra analizada.
- **Valor 'n + 1':** Se refiere al valor futuro de la muestra, correspondiendo a la fila siguiente de la muestra en análisis.

Además de estas variables, el algoritmo hará uso de dos listas de información referente a valores temporales y pendientes.

- **Lista de valores temporales:** Lista en donde se almacenarán los índices de los valores de una muestra no nula cuando se detecte que sus valores de futuro son nulos. Se usará cuando se encuentre un valor futuro no nulo que permita realizar la aproximación de los datos pendientes de aproximar.
- **Lista de valores pendientes:** Referida al almacenamiento de los índices, tanto de muestra como de característica, que contenga un valor nulo que no se pueda aproximar por falta de información.



Dichas variables se tomarán para cada una de las características o columnas de la dataset y se almacenarán en las distintas listas según convenga. Una vez almacenadas dichas variables se hará una comparación para decidir qué acción tomar.

- **Caso 1: Variable 'n' con valor distinto a 'NaN'.** No se realizaría ninguna acción.
- **Caso 2: Variable 'n' con valor 'NaN'.** En este caso se aproximaría el valor presente con el valor medio del valor pasado y del valor futuro.
- **Caso 3: Variable 'n+1' con valor 'NaN'.** Se almacena el valor de la variable presente en la lista de valores temporales.
- **Caso 4: Variable 'n+1' con valor distinto a 'NaN'.** Se aproximaría el valor presente y todos los pasados nulos a partir del valor futuro. Se pueden dar dos casos para llevar a cabo la aproximación.
 - **Existencia de valor temporal almacenado:** Se aproximaría el valor presente y los valores pendientes de forma lineal, de modo que cada valor aproximado configure una evolución según el valor temporal y el valor futuro no nulo.
 - **No existe ningún valor temporal almacenado:** Este sería el caso de muestras que desde el inicio no hayan tenido ningún valor. Se actualizarán todos los valores pendientes de aproximación al valor futuro.
- **Caso 5: Todas las variables con valor 'NaN'.** Se almacenará el índice de la muestra y de la característica (fila y columna) como dato pendiente.



La Tabla 4.11 refleja de un modo esquemático los casos posibles a la hora de analizar los datos. En dicha tabla se representan los valores presentes, pasados y futuros del análisis de la base de datos.

	N - 1	N	N + 1
CASO 1	Value	Value	Value
CASO 2	Value	NaN	Value
CASO 3	Value	Value	NaN
CASO 4	NaN	NaN	Value
CASO 5	NaN	NaN	NaN

Tabla 4.11.- Distintos casos del algoritmo de aproximación.

En los casos que detecte que el valor futuro de una muestra pertenece a otro participante, se reiniciarán las listas y se aproximarán los datos pendientes del participante en estudio según el último valor. Si dicho valor es nulo, los valores presentes se actualizarán al valor medio obtenido de todas las muestras. Así nos aseguramos de que ninguna muestra presente valor nulo al aprender los modelos.

Finalmente es recomendable eliminar los valores atípicos que puedan presentar los datos. Dichos valores atípicos se sustituirán por los valores máximos definidos por el rango intercuartílico.

En primer lugar, se calcularán los cuartiles pertenecientes a la base de datos para así calcular el rango intercuartílico del modo en el que se describe en la Figura 4.3. Una vez obtenido el rango se establecerá un límite superior y un límite inferior calculado como se describe en la Figura 4.4. Si se detecte un valor mayor que el límite superior, este se cambiará directamente a dicho límite. En el caso que un valor sea menor al límite inferior, se sustituirá por dicho límite inferior.



$$IQR = Q_3 - Q_1$$

$IQR \equiv$ Rango Intercuartílico

$Q_1 \equiv$ Primer Cuartil (25%) , $Q_3 \equiv$ Tercer Cuartil (75%)

Figura 4.3.- Cálculo del rango intercuartílico.

$$\text{Límite Superior} = Q_3 + 1,5 * IQR$$

$$\text{Límite Inferior} = Q_1 - 1,5 * IQR$$

Figura 4.4.- Cálculo de los límites superior e inferior.

Una vez realizado este procesado, la base de datos contará con veintiséis características válidas para el estudio. Antes de usarlas para el estudio, es conveniente realizar un estudio de correlación de cada una de ellas respecto a la categoría del participante. De los valores obtenidos se hará un ranking con las características que mayores valores de correlación presenten en valor absoluto como se puede observar en las Tablas 4.12. y 4.13.



N.º	Características	Coefficiente de Correlación
1	Gaze Vector Right Y	-0,240
2	Gaze Vector Right Z	0,229
3	Gaze Vector Left Y	-0,226
4	Gaze Vector Left Z	0,223
5	Point of Regard Right X [px]	-0,215
6	Point of Regard Left X [px]	-0,209
7	Point of Regard Left Y [px]	-0,205
8	Point of Regard Right Y [px]	-0,195
9	Pupil Diameter Right [mm]	0,167
10	Pupil Diameter Left [mm]	0,155
11	Eye Position Right X [mm]	-0,152
12	Pupil Position Left X [px]	-0,137
13	Eye Position Left Y [mm]	-0,133
14	Eye Position Right Y [mm]	-0,118
15	Eye Position Right Z [mm]	-0,111
16	Eye Position Left Z [mm]	-0,099

Tabla 4.12.- Características y su correlación respecto la categoría (1).

17	Pupil Position Right Y [px]	-0,091
18	Pupil Position Left Y [px]	-0,078
19	Pupil Position Right X [px]	-0,067
20	Pupil Size Left Y [px]	-0,038
21	Pupil Size Left X [px]	-0,038
22	Pupil Size Right Y [px]	-0,034
23	Pupil Size Right X [px]	-0,034
24	Eye Position Left X [mm]	-0,014
25	Gaze Vector Right X	0
26	Gaze Vector Left X	0

Tabla 4.13.- Características y su de correlación respecto la categoría (2).



Se ha observado que las características 'Gaze Vector Right X' y 'Gaze Vector Left X' han obtenido un valor de coeficiente de correlación nulo. Eso significa que dichas características no son válidas a la hora de clasificar las muestras. Una vez analizado qué características son válidas para el aprendizaje, se realizarán diversas pruebas usando diferentes grupos de características.

- **Grupo de 24 características:** Se tendrán en cuenta todas las características menos aquellas que han obtenido un valor de coeficiente de correlación nulo.
- **Grupo de 16 Características:** Este conjunto considerará las dieciséis primeras características con mayor valor absoluto de coeficiente de correlación.
- **Grupo de 8 Características:** En este caso se tendrán en cuenta las ocho primeras características con mayor coeficiente de correlación.

Con cada grupo de características se entrenarán todos los modelos para comparar su comportamiento a la hora de realizar predicciones.

Antes de utilizar las características seleccionadas al modelo hay que asegurarse de que estén bien balanceadas. Eso significa crear un subconjunto de la dataset destinado al entrenamiento con el mismo número de participantes, tanto neurotípicos como personas con TEA. Para ello se intercalarán cada uno de los datos de cada participante de modo que se asegure este balance a la hora de la creación de los subconjuntos de datos.

Hay que tener en cuenta además que en este caso el orden de las muestras es importante. Cada muestra no se puede tratar como un dato independiente y es importante que el modelo analice la evolución de las características en el orden en el que se han medido. Para ello se evitará cualquier proceso que genere aleatoriedad en el orden de las muestras.

La base de datos se dividirá en dos conjuntos de datos destinados al entrenamiento, con un 80% del total, y test con el 20% restante como se puede observar en la Figura 4.5. En este caso no se destinará una parte específica de la dataset a validación por la poca cantidad de participantes en el estudio y por el uso de validación cruzada para realizar las evaluaciones.

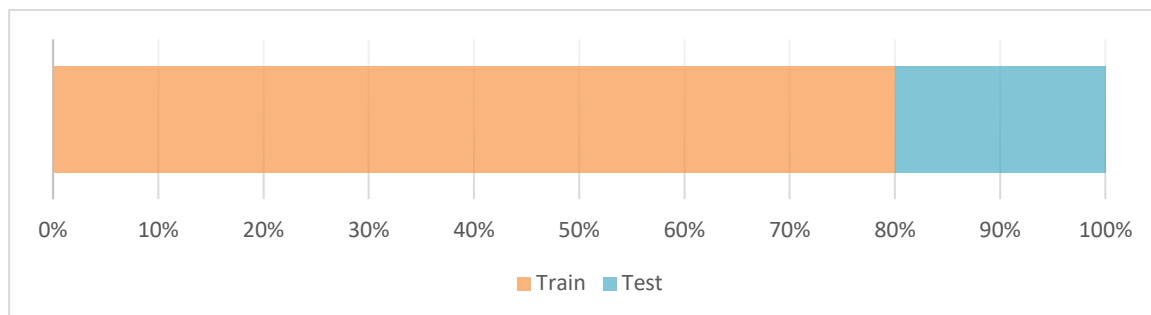


Figura 4.5.- Reparto de muestras para la fase de entrenamiento y test.

Una vez acabado este proceso, la base de datos se puede utilizar para entrenar los distintos modelos que se evaluarán en el estudio.

4.4.2. Implementación de modelos ‘Deep Learning’

Se aplicarán distintas arquitecturas de modelos basados en ‘Deep Learning’ para constatar su comportamiento con la base de datos anteriormente preprocesada. En el caso de estos modelos se les aplicará una capa de ‘dropout’ correspondiente al veinte por ciento y se mantendrá el uso de la función de activación ‘softmax’ en la salida de la arquitectura.

4.4.2.1. Modelos Secuenciales

Como en el estudio anterior, se testeará la base de datos con un modelo secuencial que tenga como entrada un número de neuronas equivalente al número de características a estudiar y como salida dos neuronas,



correspondiente a las dos clases de categoría. Se puede observar estas arquitecturas según su grupo de características en la Tabla 4.14.

Capa	24 características	16 características	8 características
Densa_1	64	64	64
Dropout	64	64	64
Densa_2	24	16	8
Densa (Output)	2	2	2

Tabla 4.14.- Arquitectura de modelo secuencial con su número de neuronas.

4.4.2.2. Modelos LSTM

Los modelos LSTM son un tipo redes neuronales recurrentes capaces de procesar y aprender de secuencias de datos. Esto permite aprender no solo de las muestras independientes de la base de datos sino también de su evolución. En este estudio puede ser interesante usar esta arquitectura por las características de los datos medidos.

En este caso se estudiará el uso de la tecnología LSTM con sesenta cuatro unidades de procesamiento al inicio de la arquitectura del modelo.

Capa	24 características	16 características	8 características
LSTM	64	64	64
Densa_1	64	64	64
Dropout	64	64	64
Densa_2	24	16	8
Densa (Output)	2	2	2

Tabla 4.15.- Arquitectura de modelo LSTM con su número de neuronas.



4.4.2.3. Modelos convolucionales

Las redes convolucionales son un tipo de redes de tipo ‘Deep Learning’ capaces de aprender de los patrones que presenten los datos. Es interesante hacer uso de estas redes en el caso de estudio con el uso de redes convolucionales unidimensionales.

Para este estudio se seguirá la arquitectura mostrada en la tabla 4.16. Se hará uso de dos capas de convolución y ‘max-pooling’ que permitirá muestrear las características de los datos y reducir el tamaño del muestreo para las capas siguientes respectivamente. Al final de las capas de convolución se configurará una capa de aplanamiento que permitirá convertir los resultados de las capas convolucionales en formato unidimensional para las últimas capas pertenecientes a la categorización.

Capa	24 características		16 características		8 características	
	Tamaño	Filtros	Tamaño	Filtros	Tamaño	Filtros
Convolutacional_1	24	16	16	16	8	16
Max_Pooling_1	12	16	8	16	4	16
Convolutacional_2	12	32	8	32	4	32
Max_Pooling_2	6	32	4	32	2	32
Flatten	192		128		64	
Dropout	192		128		64	
Densa	32		32		32	
Densa (Output)	2		2		2	

Tabla 4.16.- Arquitectura del modelo convolutacional con sus parámetros.



4.4.2.4. Modelos híbridos

Para comprobar su efectividad se hará uso de un modelo híbrido entre el modelo LSTM y el convolucional como se puede observar en la Tabla 4.17. Para ello se introducirá una capa LSTM después de la arquitectura convolucional antes descrita en la Tabla 4.16. La capa de ‘dropout’ evita que el modelo se ajuste demasiado a los datos, dando lugar a problemas de ‘overfeating’.

Capa	24		16		8	
	características		características		características	
	Tamaño	Filtros	Tamaño	Filtros	Tamaño	Filtros
Convolutacional_1	24	16	16	16	8	16
Max_Pooling_1	12	16	8	16	4	16
Convolutacional_2	12	32	8	32	4	32
Max_Pooling_2	6	32	4	32	2	32
LSTM	64		64		64	
Flatten	64		64		64	
Dropout	64		64		64	
Densa	32		32		32	
Densa (Output)	2		2		2	

Tabla 4.17.- Arquitectura de modelo híbrido con sus parámetros.

4.4.2.5. Modelos GRU

Los modelos GRU, al igual que los modelos LSTM, son modelos neuronales recurrentes que procesan secuencias de datos variantes en el tiempo. Es interesante implementar este tipo de modelos debido a la naturaleza de los datos de la base de datos. Su arquitectura, mostrada en la Tabla 4.18. consistirá en una capa GRU al inicio de la arquitectura de treinta y dos unidades seguidas de varias capas densas y dropout cuyas unidades dependerán del número de características.



Capa	24 características	16 características	8 características
GRU	32	32	32
Densa_1	24	16	8
Dropout	24	16	8
Densa_2	24	16	8
Densa (Output)	2	2	2

Tabla 4.18.- Arquitectura del modelo GRU con su número de neuronas.

4.4.3. Evaluación de modelos ‘Deep Learning’

Para evaluar los modelos descritos anteriormente se realizarán en total tres entrenamientos para cada una de las arquitecturas, cada uno con el uso de diferentes grupos de características. En primer lugar, se compilarán todos los modelos con el optimizador ‘Adam’, siendo este el optimizador más equilibrado para el entrenamiento. Posteriormente, después de validar qué modelos se comportan mejor con un determinado conjunto de características, se realizará otra prueba con el optimizador ‘Adamax’, ya que puede presentar una mejor optimización para secuencias de datos temporales.

Todas las evaluaciones se realizarán con validación cruzada ‘K-Folder’ con un número de conjuntos igual a diez como en el anterior estudio. Los resultados se analizarán en función de su precisión, su error, sus pérdidas, el uso de memoria y el tiempo de entrenamiento. Aunque la base de datos parezca resultar extensa, es necesario recordar que cada muestra no es independiente y obtenemos resultados de solamente cincuenta participantes. Por ello es necesario utilizar validación cruzada y no es recomendable sacar conclusiones sin esta validación.



5. Desarrollo

5.1. DESARROLLO DE TÉCNICAS DE 'DEEP LEARNING' SOBRE MÉTODOS ANALÍTICOS

Para realizar cada proceso descrito en el apartado de metodología hay que implementar las funciones o algoritmos que ejecuten cada función según el planteamiento establecido. El desarrollo de este primer apartado del estudio se centrará en tres procesos principales.

- **Preprocesado:** Referido a todos los procesos consistentes en leer y modificar la dataset para acondicionar los datos a los modelos.
- **Implementación de los modelos:** Parte del desarrollo en el que se implementarán los modelos para su posterior entrenamiento y evaluación.
- **Evaluación de los modelos:** Referido a los procesos que permiten entrenar los modelos y evaluarlos para su posterior toma de conclusiones.

5.1.1. Preprocesado de la base de datos

La base de datos adjunta en el estudio consiste en un archivo de formato '.csv' llamado 'Toddler Autism dataset July 2018.csv'. Una vez importado al proyecto se eliminarán las columnas que no interesen en el aprendizaje y aquellas que contengan algún valor nulo según el código de la Figura 5.1.



```
dataset = pd.read_csv("Toddler Autism dataset July 2018.csv")  
  
dataset = dataset.drop(columns=  
["Case_No", "Qchat-10-Score", "Who completed the test"])  
  
dataset = dataset.dropna()
```

Figura 5.1.- Importación de base de datos.

Con la base de datos ya cargada el siguiente paso consistirá en convertir todas las características categóricas en formato numérico. Esto significa que todas aquellas características que estén en formato texto deberán traducirse a un valor numérico único para el correcto funcionamiento del aprendizaje. Este proceso se ejecuta usando el código descrito en la Figura 5.2.

Las columnas pertenecientes a cada característica presentan un identificador único correspondiente al número de columna del datagrama. Alguno de estas características con su correspondiente identificador se puede observar en la Tabla 5.1. De este modo se puede utilizar dicho localizador para efectuar los cambios en cada una de las muestras identificando cada una de sus características individualmente.

Característica	Identificador de columna
Class/ASD_Traits	15
Sex	11
Family_mem_with_ASD	14
Ethnicity	12
Jaundice	13

Tabla 5.1.- Identificador de columna perteneciente a cada característica

Para comprobar que no existe ningún valor erróneo en las columnas modificadas se pueden analizar los valores únicos de cada columna. Si se han producido correctamente las modificaciones solo deberían aparecer los valores discretos



impuestos en el planteamiento, tal y como se puede comprobar utilizando el código perteneciente a la Figura 5.3.

```
for counter, element in dataset.iterrows():  
  
    if element["Class/ASD_Traits"] == "Yes": dataset.iloc[counter,15] = 1  
    elif element["Class/ASD_Traits"] == "No": dataset.iloc[counter,15] = 0  
  
    if element["Sex"] == "f": dataset.iloc[counter,11] = 1  
    elif element["Sex"] == "m": dataset.iloc[counter,11] = 0  
  
    if element["Family_mem_with_ASD"] == "yes": dataset.iloc[counter,14] = 1  
    elif element["Family_mem_with_ASD"] == "no": dataset.iloc[counter,14] = 0  
  
    if element["Jaundice"] == "yes": dataset.iloc[counter,13] = 1  
    elif element["Jaundice"] == "no": dataset.iloc[counter,13] = 0  
  
    if element["Ethnicity"] == "White European": dataset.iloc[counter,12] = 1  
  
    [. . .]  
  
    elif element["Ethnicity"] == "middle eastern": dataset.iloc[counter,12] = 11
```

Figura 5.2.- Cambio de características a formato numérico.

```
print(dataset["Class/ASD_Traits"].unique())  
print(dataset["Sex"].unique())  
print(dataset["Family_mem_with_ASD"].unique())  
print(dataset["Jaundice"].unique())  
print(dataset["Ethnicity"].unique())
```

Figura 5.3.- Comprobación de preprocesado.

Una vez realizado este proceso se deben de crear los subconjuntos de datos para entrenamiento, test y validación del modo descrito en la Figura 5.4. Además, se separarán las características de clasificación para cada uno de los subconjuntos aplicando el código presente en la Figura 5.5. Para que el modelo pueda leer los datos correctamente se precisa realizar una conversión a tipo 'float32' con el uso de la función '.astype(np.float32)' sobre la base de datos.



```
dataset_train , dataset_test_validation = train_test_split(dataset, test_size=0.2,  
                                                         random_state=42)  
  
dataset_test , dataset_validation = train_test_split(dataset_test_validation , test_size=0.2,  
                                                    random_state=42)
```

Figura 5.4.- Creación de los distintos subconjuntos de datos.

```
dataset_train_data , dataset_train_label = dataset_train.iloc[:, :-1] , dataset_train.iloc[:, -1]
```

Figura 5.5.- Separación entre características y etiquetas.

5.1.2. Implementación de modelos 'Deep Learning'

El modelo secuencial se implementará con una función para que las técnicas de evaluación puedan usarse con o sin validación. Dicha función implementará las capas del modelo sin definir el optimizador en su compilación, ya que este valor se mandará como parámetro en la función de evaluación. El código para implementar este modelo se describe en la Figura 5.6. En este caso a la capa 'dropout' se le configurará un parámetro de desactivación del diez por ciento, correspondiente a la porción de neuronas que serán desactivadas en el entrenamiento del modelo.

En cuanto a las funciones de activación, las capas de entrada y las capas ocultas se les configurará una función de tipo sigmoide, usada para la clasificación binaria. A la capa densa posterior al 'dropout' se le configurará una activación lineal 'relu' para ajustarse a la capa final de salida. En cuanto a esta última capa, se le configurará una activación de tipo 'softmax' para ajustar la suma de las probabilidades de salida a la unidad. Con ello nos aseguraremos de que los valores de predicción sean equilibrados a la hora de realizar las predicciones y en formato de proporción.



```
def sequential_model:  
  
    model = tf.keras.models.Sequential()  
    model.add(tf.keras.layers.Dense(units=15,activation='sigmoid', input_shape=(15,)))  
    model.add(tf.keras.layers.Dense(units=10, activation='sigmoid'))  
    model.add(tf.keras.layers.Dropout(0.1))  
    model.add(tf.keras.layers.Dense(units=10, activation='relu'))  
    model.add(tf.keras.layers.Dense(units=2, activation='softmax'))  
  
    return model
```

Figura 5.6.- Implementación del modelo secuencial

En el caso de los modelos 'Gradient Boosted' y 'Random Forest' se implementarán directamente en el código de evaluación, debido a que solo precisan de una instrucción a la hora de implementarlos como se puede observar en la Figura 5.7.

```
GB_model = tfdf.keras.GradientBoostedTreesModel()  
  
RF_model = tfdf.keras.RandomForestModel()
```

Figura 5.7.- Implementación de los modelos de árboles de decisión

5.1.3. Evaluación de modelos 'Deep Learning'

En primer lugar, se evaluarán los modelos sin hacer uso de la validación cruzada. Para ello se implementará una función que pida como parámetro el optimizador para la compilación. Dicha función compilará el modelo e iniciará su entrenamiento con la información de toda la base de datos dedicada al entrenamiento. Esta función devolverá el modelo ya compilado y entrenado, sus predicciones a partir del subconjunto de prueba y los diferentes parámetros de evaluación como son las pérdidas, la precisión y el tiempo de entrenamiento. Los parámetros de pérdidas y precisión se obtienen una vez entrenado el modelo y el tiempo de entrenamiento mediante la función 'time.time()'. Esta función registrará



el tiempo antes y después del entrenamiento para hacer una estimación del tiempo que ha tardado.

La Figura 5.8. indica el funcionamiento de este proceso. En él se puede comprobar cómo recibe el parámetro del optimizador por medio de la variable 'optimizer_type' y como lo usa para compilar el modelo. En segundo lugar, se realiza su entrenamiento y se toman las medidas temporales, antes y después.

Finalmente se comprueban las muestras destinadas a la fase de pruebas con el modelo ya entrenado. Estos valores se devuelven al final de la función para extraer de ellos las gráficas de matrices de confusión.

```
def sequential_model_fit_by_optimizer(optimizer_type):  
  
    model = [Modelo en Estudio]  
    model.compile(optimizer=optimizer_type,  
                  loss='sparse_categorical_crossentropy', metrics=['sparse_categorical_accuracy'])  
  
    start_time = time.time()  
  
    model_fit = model.fit(dataset_train_data, dataset_train_label, epochs=30,  
                          validation_data=(dataset_validation_data ,  
                          dataset_validation_label), verbose=0)  
  
    training_time = time.time() - start_time  
  
    Test_loss, Test_accuracy = model.evaluate(dataset_test_data, dataset_test_label)  
  
    dataset_test_predictions = np.argmax(model.predict(dataset_test_data), axis=-1)  
  
    return model_fit , dataset_test_predictions , Test_loss, Test_accuracy , training_time
```

Figura 5.8.- Evaluación del modelo sin validación cruzada

Para implementar la evaluación del modelo con validación cruzada se implementará una función que, en primer lugar, unirá los subconjuntos de entrenamiento y test. Después se realizará la subdivisión del nuevo conjunto por particiones y se iterará cada una de ellas con la porción de entrenamiento y test correspondientes. Se hará uso de la función 'kfolder' que cuenta con la función



‘.split()’ para realizar la subdivisión por grupo. Además, se crearán dos ‘arrays’, uno de pérdidas y otro de precisión, para realizar el cálculo global de todos los conjuntos.

La Figura 5.9. representa el funcionamiento de la validación cruzada. En ella se puede observar cómo se instancia el elemento ‘kfold’ con diez subconjuntos. Luego de ello, se crean los dos ‘arrays’ de métricas de precisión y pérdidas para que se almacenen cada uno de los valores correspondientes a cada iteración del entrenamiento.

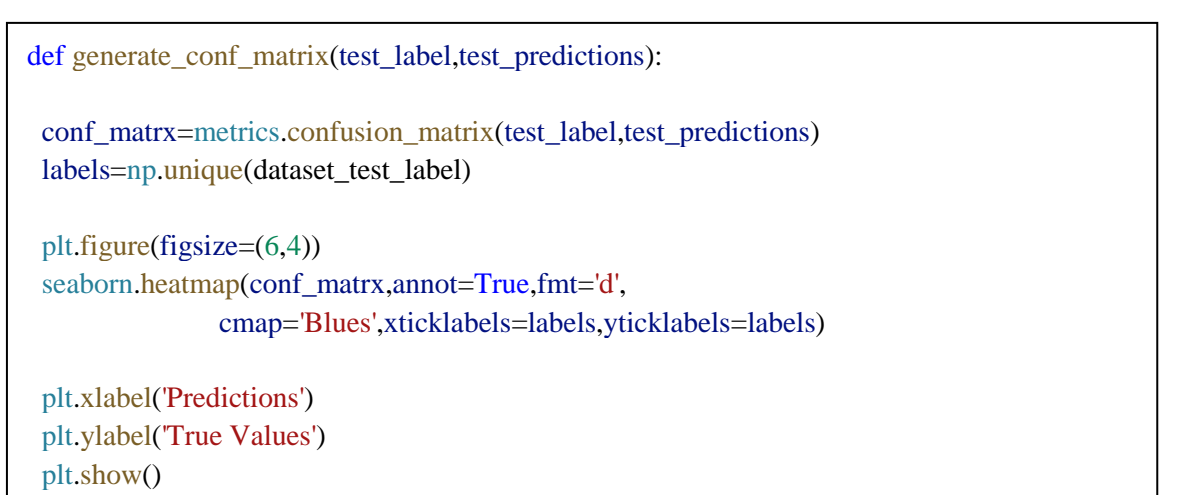
```
def sequential_model_fit_by_optimizer_kfold(optimizer_type):  
    kfolder = KFold(n_splits=10, shuffle=True)  
  
    acc_per_fold = []  
    loss_per_fold = []  
  
    #Iteración por folder  
  
    for train, test in kfolder.split(data,label):  
        model = [Modelo a Estudiar]  
        model.compile(optimizer=optimizer_type, loss='sparse_categorical_crossentropy',  
                    metrics=['sparse_categorical_accuracy'])  
  
        print(f-> Training for folder number {folder_number})  
        model_fit = model.fit(data[train], label[train], epochs=30,verbose=0)  
        values = model.evaluate(data[test], label[test])  
  
        acc_per_fold.append(values[1] * 100)  
        loss_per_fold.append(values[0])  
  
        folder_number = folder_number + 1
```

Figura 5.9.- Evaluación del modelo con validación cruzada



5.1.4. Generación de gráficas

Un modo de evaluar el comportamiento de los distintos modelos es por medio de las gráficas de los parámetros de evaluación. Para ello se implementarán funciones que generen gráficos como matrices de confusión, evolución de pérdidas y evolución de precisión.

- **Matrices de confusión:** Recibiendo como parámetros las etiquetas pertenecientes al subconjunto de test y sus predicciones por el modelo, esta función imprimirá el gráfico con la matriz de confusión en función de estos datos. Como se puede ver en la Figura 5.10. se hará uso de la librería 'seaborn' para generar de modo gráfico la matriz generada por la función 'metrics.confusion_matrix()'.


```
def generate_conf_matrix(test_label,test_predictions):  
  
    conf_mtx=metrics.confusion_matrix(test_label,test_predictions)  
    labels=np.unique(dataset_test_label)  
  
    plt.figure(figsize=(6,4))  
    seaborn.heatmap(conf_mtx,annot=True,fmt='d',  
                    cmap='Blues',xticklabels=labels,yticklabels=labels)  
  
    plt.xlabel('Predictions')  
    plt.ylabel('True Values')  
    plt.show()
```

Figura 5.10.- Generación de matriz de confusión

- **Gráficas de precisión y de pérdidas:** Tomando como parámetro el modelo entrenado y accediendo a su historial de entrenamiento como se observa en la Figura 5.11. se generarán las distintas gráficas de pérdidas y precisión. Para ello se representará en un eje el parámetro de interés y en el otro el número de épocas en las cuales se ha entrenado el modelo. Se representarán en la misma gráfica tanto los parámetros correspondientes al conjunto de test como los del conjunto de validación.



```
def generate_accuracy_graphs(model):  
    plt.figure(figsize=(12, 4))  
  
    plt.plot(model.history['sparse_categorical_accuracy'], label= 'Accuracy (Training)')  
    plt.plot(model.history['val_sparse_categorical_accuracy'], label='Accuracy(Validation)')  
  
    plt.xlabel('Epoch')  
    plt.ylabel('Precisión')  
    plt.title('Accuracy Evolution')  
    plt.legend()  
  
    plt.tight_layout()  
    plt.grid(True)  
    plt.show()
```

Figura 5.11.- Código para generar la gráfica de precisión

- **Gráficas de precisión y de pérdidas en modelos de árboles de decisión:** En el caso de los modelos de árboles de decisión se debe emplear otra función que genere el gráfico adecuado para realizar las evaluaciones (Figura 5.12.) En este caso se debe de representar la evolución de la precisión y las pérdidas según el número de árboles generados por el modelo.

```
def Decision_Tree_generate_accuracy (model):  
    logs = model.make_inspector().training_logs()  
  
    plt.figure(figsize=(12, 6))  
    plt.plot([log.num_trees for log in logs], [log.evaluation.loss for log in logs])  
  
    plt.xlabel("Number of trees")  
    plt.ylabel("Logloss (out-of-bag)")  
  
    plt.grid(True)  
    plt.show()
```

Figura 5.12.- Gráfica de pérdidas en modelos de árbol de decisión.



5.2. DESARROLLO DE TÉCNICAS DEEP LEARNING SOBRE DATOS 'EYE TRACKING'

Al igual que el desarrollo del estudio anterior, el proceso para programar todas las funciones necesarias se dividirá en tres etapas fundamentales.

- **Preprocesado:** Procesos para condicionar los datos antes del proceso de aprendizaje.
- **Implementación de los modelos:** Implementación de los distintos modelos basados en 'Deep Learning'
- **Evaluación de los modelos:** Proceso de evaluación de cada uno de los modelos.

5.2.1. Agrupación de archivos CSV

En primer lugar, es necesario agrupar todos los archivos '.csv' en un único dataframe. Para ello se utilizará la función 'glob()' para recolectar en un único 'array' el nombre de todos los archivos formato '.csv'. Después de ello se utilizará la función 'pd.concat()' para realizar la unión de todos los archivos como se observa en la Figura 5.13.

```
dataset_union = [i for i in glob.glob('DS/*.{,}.format('csv'))]  
dataset = pd.concat([pd.read_csv(f) for f in dataset_union ])
```

Figura 5.13.- Unión de todos los archivos '.csv'

Una vez unidos todos los archivos correspondientes al dataset, se procederá a cotejar cada participante con su categoría. Para ello se importará el archivo de 'metadata.csv' y se asignará cada etiqueta a la muestra correspondiente, tal y como se implementa en la Figura 5.14. Para ello se recorrerá cada muestra y se obtendrá en una variable el valor id del participante. Una vez extraída dicha 'id' se



buscará dicho identificador en la dataset de meta data y se extraerá su valor de etiqueta para agregarla a la base de datos. Los cambios se producirán en todas las muestras que tengan un número de id igual. Así evitaremos recorrer todas las muestras y hacer la comprobación individual.

```
metadata = pd.read_csv("Metadata_Participants.csv")

for row in dataset_with_labels["Participant"]:
    if row != previous_row:

        dataset_with_labels.loc[dataset_with_labels["Participant"] == row , "LABEL"] =
        metadata["Class"][metadata["ParticipantID"]
        ==int(row)].values[0]

    previous_row = row
```

Figura 5.14.- Agregado de etiqueta a las muestras.

En este caso `'dataset_with_labels.loc[dataset_with_labels["Participant"] == row , "LABEL"]'` indica la selección de todas las muestras que contengan el id específico en la nueva columna a crear llamada 'LABEL' donde se introducirá el valor de clasificación.

Este valor se agregará en función de `'metadata ["Class"][metadata["ParticipantID"] == int(row)].values[0]'` que representa el valor de clasificación del id del participante.

En la base de datos existen varias muestras que no muestran identificador del participante. Para esos casos se optará por su eliminación para que no interfieran en el entrenamiento. En este caso se etiquetará dichos valores como 'NODATA' y no se incluirán en la base de datos final.

El siguiente paso tratará de eliminar todas las características o columnas que no nos sirvan para el entrenamiento (Figura 5.15). Esto se realizará con la función `'drop()'` y una lista de variables que contendrá todas las columnas a eliminar.



```
columns_to_drop = ["Unnamed: 0", "RecordingTime [ms]",  
                  "Time of Day [h:m:s:ms]", "Trial", "Stimulus",  
                  "Export Start Trial Time [ms]", "Export End Trial Time [ms]",  
                  "Color", "Tracking Ratio [%]", "Category Group",  
                  "Category Right", "Category Left", "Index Right",  
                  "Index Left", "AOI Name Right", "AOI Name Left",  
                  "Annotation Name", "Annotation Description", "Annotation Tags",  
                  "Mouse Position X [px]", "Mouse Position Y [px]",  
                  "Scroll Direction X", "Scroll Direction Y", "Content",  
                  "Port Status", "AOI Group Right", "AOI Scope Right",  
                  "AOI Order Right", "AOI Group Left", "AOI Scope Left",  
                  "AOI Order Binocular", "groupe d'enfants"  
]  
  
dataset_with_labels = dataset_with_labels.drop(columns=columns_to_drop)
```

Figura 5.15.- Eliminación de características no usadas para el aprendizaje.

5.2.2. Preprocesado de la base de datos

Habiéndose quedado con las características válidas para el entrenamiento, es necesario asegurar que todos los valores sean válidos. Cuando un valor no se ha podido medir en la base de datos se identifica con el carácter '-' que en este caso se reemplazará con el valor nulo 'pd.NA'. Después se aproximará cada valor nulo a una aproximación según sus valores pasados y futuros o se hará una aproximación con la media de las variables. De ese modo no perderemos información de una muestra entera cuando solo falten pequeñas porciones de información en alguna de sus características. Por cada muestra y cada característica se tomarán los valores correspondientes a la muestra presente con sus correspondientes valores pasado y futuro (Figura 5.16.)

```
value = dataset_with_labels.iloc[index,column]  
last_value = dataset_with_labels.iloc[index-1,column]  
next_value = dataset_with_labels.iloc[index+1,column]
```

Figura 5.16.- Extracción del valor presente, pasado y futuro



Una vez extraídos dichos valores, se comprobará la relación existente entre ellos para realizar una acción u otra. En primer lugar, se validará si el valor presente es nulo. Si es así se harán las siguientes comprobaciones.

- **Valores presente y futuro no nulos:** Como se puede observar en la Figura 5.17. se aproximará el valor presente con sus valores pasado y futuro. Se aproximará con el mismo valor si tienen el mismo valor tanto el valor pasado como el futuro o se aproximará con el valor medio si son diferentes.

```
if (not math.isnan(last_value) and not math.isnan(next_value)):
    if (last_value == next_value):
        dataset_with_labels.iloc[index,column] = last_value
    else:
        dataset_with_labels.iloc[index,column] =
            (next_value + last_value)/2
```

Figura 5.17.- Aproximación entre valores pasado y futuro no nulos.

- **Valor futuro nulo y pasado no nulo:** Se almacenará el id del dato pasado como temporal para futuras aproximaciones (Figura 5.18.).

```
elif (not math.isnan(last_value) and math.isnan(next_value)):
    index_temp_data.append([index-1,column])
```

Figura 5.18.- Almacenar id de la muestra y su característica como valor temporal

- **Valor futuro y pasado nulos:** Se almacenará el id del dato presente en la lista de valores pendientes. Se validará también si no se trata de la última muestra perteneciente a un participante (Figura 5.20.). Esta comprobación se realiza analizando el 'id' de la muestra futura y comparándola con el id de la muestra presente (Figura 5.19.)



```
elif (last_data_of_participant == False) and  
    (math.isnan(last_value) and math.isnan(next_value)):  
  
    index_data_pending.append([index,column])
```

Figura 5.19.- Almacenar muestra como valor pendiente

```
if dataset_with_labels["Participant"].iloc[index] !=  
    dataset_with_labels["Participant"].iloc[index+1]:  
  
    last_data_of_participant = True
```

Figura 5.20.- Comprobación de última de muestra de un participante.

- **Valor futuro no nulo y pasado nulo:** Se actualizarán todos los valores pasados pendientes de aproximación. También se llevará a cabo esta acción cuando se verifique que sea la última muestra de un participante. Se extraerá el último valor de la lista de valores temporales para así aproximar al último valor pasado y se actualizarán los valores pasados. Si no se encuentra un valor temporal, éste se representará como un valor nulo. Este proceso se representa en la Figura 5.21.

```
elif (last_data_of_participant == True) or  
    (math.isnan(last_value) and not math.isnan(next_value)):  
    index_row_last_data = -1  
  
    for element_temp_data in reversed(index_temp_data):  
        if element_temp_data[1] == column:  
            index_row_last_data = element_temp_data[0]  
            break  
  
    if index_row_last_data == -1:  
        temp_data = np.nan  
    else:  
        temp_data =  
            dataset_with_labels.iloc[index_row_last_data,column]
```

Figura 5.21.- Extracción del último valor temporal de la lista.



Una vez obtenidos los valores temporales, se extraerán la posición de los datos pendientes de aproximación (Figura 5.22.) Una vez obtenidos, se aproximará cada uno de los valores de forma gradual, incrementando o decrementando el valor obtenido de la diferencia entre el valor temporal y el valor futuro dividido entre el número de datos pendientes. Este valor se denomina valor 'delta' y se obtiene del modo en el que se indica en la Figura 5.23. y la aproximación de los valores pendientes en la Figura 5.24.

```
index_row_data_pending = []
index_data_pending_updated = []

for index, data_pending_column in index_data_pending:
    if data_pending_column == column:
        index_row_data_pending.append(index)
    else:
        index_data_pending_updated.append((index, data_pending_column))

index_data_pending = index_data_pending_updated
```

Figura 5.22.- Extracción de los valores pendientes de aproximación.

```
index_row_data_pending = []

for index, data_pending_column in index_data_pending:
    if data_pending_column[1] == column:
        index_row_data_pending.append(index)

len_index_row_data_pending = len(index_row_data_pending)
delta = (next_value - temp_data)/len_index_row_data_pending
```

Figura 5.23.- Cálculo del valor 'delta'.



```
increment_counter = 1

for row in index_row_data_pending:

    dataset_with_labels.iloc[row,column] =
    temp_data + increment_counter * delta

    increment_counter += 1
```

Figura 5.24.- Aproximación de valores pendientes.

Finalmente, se deben eliminar los valores atípicos que puedan estar en la base de datos del modo en el que se indica en la Figura 5.25. Primero de todo, se modificarán todos los valores nulos que el algoritmo de aproximación no consiguió modificar, asignando a dichos valores el valor de la media correspondiente a la totalidad las muestras en su característica determinada. Para ello se recorrerá cada una de las características y se extraerán los valores de la media, por medio de la función ‘mean()’ y de los cuartiles uno y tres usando la función ‘quantile()’.

Después de ello se modificarán los valores nulos al valor de la media y se calcularán los límites superiores e inferiores según las fórmulas 4.1 y 4.2. Una vez calculados estos parámetros se modificarán todos los valores superiores e inferiores de cada uno de los límites y se modificarán por el valor establecido.



```
columns =  
dataset_with_labels.drop(columns=["Participant", "LABEL"]).columns  
  
for column in columns:  
    mean_column = dataset_with_labels[column].mean()  
    dataset_with_labels[column] = dataset_with_labels[column].fillna(mean_column)  
  
    Q1 = dataset_with_labels[column].quantile(0.25)  
    Q3 = dataset_with_labels[column].quantile(0.75)  
    IQR = Q3 - Q1  
  
    upper_limit = Q3 + 1.5 * IQR  
    lower_limit = Q1 - 1.5 * IQR  
  
    dataset_with_labels.loc[dataset_with_labels[column] > upper_limit, column] = upper_limit  
    dataset_with_labels.loc[dataset_with_labels[column] < lower_limit, column] = lower_limit
```

Figura 5.25.- Corrección de los valores atípicos.

5.2.3. Selección de características

Es importante determinar qué características serían las más indicadas para entrenar la inteligencia artificial. Para ello se calculará el coeficiente de correlación de todas las características según su valor de clasificación, tal como se indica en la Figura 5.26. El valor de correlación se extraerá a partir de la función 'pearsonr()' de la librería 'scipy' y se pasarán como parámetros cada una de las columnas de forma independiente con la columna 'LABEL' perteneciente a la categoría. Esta función devolverá el valor de la correlación y del 'p-value' que nos servirá para determinar qué características están más correlacionadas con la clasificación.

El algoritmo permitirá iterar en cada columna de la base de datos para almacenar en un array las correlaciones de cada una de las características.



```
columns = dataset_with_labels.drop(columns=["Participant","LABEL"]).columns

for column in columns:
    corr, p_value = pearsonr(dataset_with_labels[column],
                             dataset_with_labels["LABEL"].replace({'TD':0,'ASD':1}))

    if not math.isnan(corr):
        column_corr_pvalue.append([column,corr,p_value])
    else:
        column_corr_pvalue.append([column,0,0])
```

Figura 5.26.- Extracción de los coeficientes de correlación.

Seleccionando las características para el estudio, se puede dividir la base de datos de modo que se creen los conjuntos de entrenamiento y de prueba. Para balancear los datos se intercalarán los participantes según a la categoría que pertenezcan. Para llevarlo a cabo se extraerán los índices de las categorías 'TEA' y 'No TEA' y se creará un nuevo 'dataset' intercalando ambas categorías (Figura 5.27.)

```
id_ASD = dataset_with_labels_ASD["Participant"].unique()
id_TD = dataset_with_labels_TD["Participant"].unique()

id_dataset=[value for par in zip(id_ASD, id_TD) for value in par]
```

Figura 5.27.- Intercalado entre casos 'TEA' y 'No TEA'

Finalmente se crearán los dos conjuntos de datos a partir de los identificadores intercalados. Como se indica en la Figura 5.28. Se tendrá en cuenta que el tamaño de los datos de entrenamiento corresponda al 80% de toda la base de datos. El restante se agregará al 'array' correspondiente con el subconjunto de prueba del estudio.



```
dataset_with_labels_test_mix = []

len_train=int(len(id_dataset)//1.25)

for value in id_dataset[:len_train]:
    dataset_with_labels_train_mix = pd.concat([dataset_with_labels_train_mix,
        dataset_with_labels[dataset_with_labels["Participant"] ==int(value)]]))

for value in id_dataset[len_train:]:
    dataset_with_labels_test_mix.append([value,
        dataset_with_labels[dataset_with_labels["Participant"] == int(value)]]))
```

Figura 5.28.- Creación de conjunto de datos para entrenamiento y test.

Para seleccionar las características de estudio, se implementará una lista llamada 'features' en la que estarán todas las variables a estudiar (Figura 5.29.). Con esta lista se podrá realizar un filtrado sobre la base de datos original para quedarse solo con las características de interés.

```
features = [

    "Gaze Vector Right Y",
    "Gaze Vector Right Z",

    [ . . . ]

    "Pupil Size Right X [px]",
    "Eye Position Left X [mm]"
    # "Gaze Vector Right X", #Correlación 0
    # "Gaze Vector Left X" #Correlación 0

]
```

Figura 5.29.- Lista de características a seleccionar.



5.2.4. Implementación de modelos ‘Deep Learning’

En esta sección se implementará el código necesario para crear las arquitecturas de los modelos de inteligencia artificial con sus configuraciones específicas explicadas en el apartado cuatro.

En todos los modelos empleados la capa de ‘dropout’ se configurará con un parámetro del veinte por ciento, indicando el porcentaje de células que se desactivarán en el proceso de entrenamiento en la capa intermedia de ambas densas.

En cuanto a las funciones de activador, todas las capas de salida harán uso de la función ‘softmax’. Usando esta función nos cercioraremos de que las salidas tengan un formato probabilístico en la clasificación de clases, ‘No TEA’ y ‘TEA’. De este modo, ambas probabilidades sumarán una unidad. En el resto de las capas se hará uso de la función de activación ‘relu’ que adquiere a la neurona una activación lineal.

5.2.4.1. Modelo secuencial

Se implementará una función que creará una arquitectura de modelo secuencial que siga las indicaciones citadas en el apartado de metodología. El tamaño de entrada irá en función del tamaño de la lista ‘features’ como se indica en la Figura 5.30.



```
def sequential_model():  
  
    model = tf.keras.models.Sequential()  
    model.add(tf.keras.Input(shape=(len(features),1)))  
    model.add(tf.keras.layers.Dense(units=64, activation='relu'))  
    model.add(tf.keras.layers.Dropout(0.2))  
    model.add(tf.keras.layers.Dense(units=len(features), activation='relu'))  
    model.add(tf.keras.layers.Dense(units=2, activation='softmax'))  
  
    return model
```

Figura 5.30.- Función de modelo secuencial

5.2.4.2. Modelo LSTM

Modelo con arquitectura similar a la anterior con el añadido de una capa LSTM a la entrada. En la figura 5.31. se puede observar que el único cambio que existe respecto a la Figura 5.30. es el añadido de la capa LSTM.

```
def sequential_lstm_model():  
  
    model = tf.keras.models.Sequential()  
    model.add(LSTM(64, input_shape=(len(features),1)))  
    model.add(tf.keras.layers.Dense(units=64, activation='relu'))  
    model.add(tf.keras.layers.Dropout(0.2))  
    model.add(tf.keras.layers.Dense(units=len(features), activation='relu'))  
    model.add(tf.keras.layers.Dense(units=2, activation='softmax'))  
  
    return model
```

Figura 5.31.- Función de modelo LSTM.

5.2.4.3. Modelo convolucional

Arquitectura que añade dos capas convolucionales con su respectivo 'max-pooling'. Finalmente se aplica una capa de 'dropout' con una capa densa de treinta y dos unidades (Figura 5.32.)



```
def sequential_convolutional_model():  
  
    model = tf.keras.models.Sequential()  
  
    model.add(tf.keras.Input(shape=(len(features),1)))  
    model.add(tf.keras.layers.Conv1D(filters=16, kernel_size=3,  
                                     activation='relu', padding='same'))  
    model.add(tf.keras.layers.MaxPooling1D())  
  
    model.add(tf.keras.layers.Conv1D(filters=32, kernel_size=3,  
                                     activation='relu', padding='same'))  
    model.add(tf.keras.layers.MaxPooling1D())  
    model.add(tf.keras.layers.Flatten())  
  
    model.add(tf.keras.layers.Dropout(0.2))  
    model.add(tf.keras.layers.Dense(units=32, activation='relu'))  
    model.add(tf.keras.layers.Dense(units=2, activation='softmax'))  
  
    return model
```

Figura 5.32.- Función de modelo convolucional.

5.2.4.4. Modelo híbrido

Esta función añade al modelo convolucional anterior una capa LSTM al final de todas las capas convolucionales. Como se observa en la Figura 5.33. tiene una arquitectura igual a la de la Figura 5.32. con la diferencia de que, como en el caso del modelo LSTM con el secuencial, se le añade esta capa LSTM al final de las capas pertenecientes a la estructura convolucional.



```
def sequential_convolutional_lstm_model():  
  
    model = tf.keras.models.Sequential()  
    model.add(tf.keras.Input(shape=(len(features),1)))  
  
    model.add(tf.keras.layers.Conv1D(filters=16, kernel_size=3,  
                                     activation='relu', padding='same'))  
    model.add(tf.keras.layers.MaxPooling1D())  
  
    model.add(tf.keras.layers.Conv1D(filters=32, kernel_size=3,  
                                     activation='relu', padding='same'))  
  
    model.add(tf.keras.layers.MaxPooling1D())  
    model.add(LSTM(64, input_shape=(128,1)))  
    model.add(tf.keras.layers.Flatten())  
  
    model.add(tf.keras.layers.Dropout(0.2))  
  
    model.add(tf.keras.layers.Dense(units=32, activation='relu'))  
    model.add(tf.keras.layers.Dense(units=2, activation='softmax'))  
  
    return model
```

Figura 5.33.- Función de modelo híbrido LSTM + convolucional.

5.2.4.5. Modelo GRU

Esta función combina el modelo secuencial con una capa basada en 'GRU' al inicio de la arquitectura (Figura 5.34.). Es una arquitectura parecida a la Figura 5.32. solo que añadiendo una capa GRU en vez de una capa LSTM antes de la primera capa densa.



```
def gru_model():  
  
    model = tf.keras.models.Sequential()  
    model.add(tf.keras.Input(shape=(len(features),1)))  
  
    model.add(tf.keras.layers.GRU(32))  
    model.add(tf.keras.layers.Dense(len(features)))  
    model.add(tf.keras.layers.Dropout(0.2))  
  
    model.add(tf.keras.layers.Dense(units=len(features), activation='relu'))  
    model.add(tf.keras.layers.Dense(units=2, activation='softmax'))  
  
    return model
```

Figura 5.34.- Función de modelo GRU.

5.2.5. Evaluación de modelos 'Deep Learning'

Al evaluar los modelos anteriores se usará validación cruzada. La función es similar a la utilizada anteriormente, con la diferencia de la implementación de un parámetro añadido que permite iniciar el entrenamiento con un folder determinado. Se ha implementado por el tiempo que toman los modelos en entrenarse, pudiendo ser susceptibles a fallos. Implementando este parámetro se puede reiniciar el entrenamiento en un determinado folder sin la necesidad de realizar de nuevo toda la validación.

Además, en este estudio, se ha desactivado el parámetro 'shuffle' que en el estudio anterior hacía combinar los datos entre sí de forma aleatoria (Figura 5.35.). En este caso debemos priorizar que los datos preserven su orden según fueron medidos.



```
def model_fit_kfold(model, optimizer = "Adam", epochs = 50 , folder_init = 1):  
  
    kfolder = KFold(n_splits=10, shuffle=False)  
  
    [..]  
  
    for train , test in kfolder.split(data,label):  
  
        if folder_number == folder_init:  
  
            [..]  
  
            folder_init += 1  
  
        folder_number = folder_number + 1
```

Figura 5.35.- Modificaciones de la función de la validación cruzada.



6. Análisis de resultados

6.1. MÉTRICAS

Es necesario esclarecer los criterios a seguir a la hora de evaluar los modelos. En ambos casos de estudio se han tenido en cuenta parámetros como la precisión, las pérdidas, el tiempo de entrenamiento y el uso de memoria. En los modelos evaluados por validación cruzada aparece una métrica llamada error, la cual mide la variación de las métricas de precisión entre grupos.

- **Precisión:** Métrica referida a la capacidad de un modelo de realizar predicciones. Se obtiene comparando los resultados obtenidos de las predicciones con el valor categórico real de las muestras. Es un valor comprendido entre los valores cero y uno, representando de este modo una probabilidad. Cuanto más se acerque dicho valor a la unidad más preciso lo consideramos, aunque si dicho valor resulta ser la unidad puede ser indicativo de que el modelo se ha ajustado demasiado a los datos de entrenamiento.
- **Pérdidas:** Se consideran pérdidas o función de pérdidas a la métrica utilizada para evaluar la discrepancia entre las predicciones realizadas por un modelo y los valores reales. Cuando el valor tiende a cero se considera que la predicción tiene poca discrepancia respecto al valor real, lo que significa que hace predicciones más precisas. Cuando mayor sea el valor de pérdidas mayores discrepancias existirán y menos fiables serán las predicciones del modelo.
- **Tiempo de entrenamiento:** Un parámetro para tener en cuenta en el entrenamiento es el tiempo que toma en completarse. A la hora de evaluar un modelo es importante tenerlo en cuenta debido al consumo de recursos que, si este tarda más tiempo en converger, mayor uso usará de ellos. No



solo hay que tener en cuenta un modelo con buenos resultados de precisión, sino que, además, sea eficiente.

- **Uso de memoria:** Al igual que el tiempo de entrenamiento, es interesante medir el uso de recursos que un modelo necesita para cumplir el entrenamiento. Es otra métrica de eficiencia que, junto al tiempo de entrenamiento, hay que tener en cuenta en la evaluación. En este estudio se considerará la cantidad de memoria de GPU utilizada durante el entrenamiento de los modelos.

- **Parámetros en validación cruzada:** Realizando validación cruzada, debemos de tener en cuenta que obtendremos diferentes valores de precisión y pérdidas por cada grupo. Para obtener una evaluación global de modelo es necesario promediar los valores de cada folder de un modo específico.
 - **Precisión:** La precisión global del modelo será la media de los valores de precisión de cada folder.

 - **Error:** Esta métrica se refiere a la variación entre el mayor y el menor valor de precisión. Se obtendrá de la desviación estándar de los valores de precisión de cada folder.

 - **Pérdidas:** Igual que la métrica de precisión, este valor se referirá como la media de pérdidas de cada folder.

 - **Tiempo de entrenamiento:** Se considerará el tiempo total de entrenamiento de todos los conjuntos, aunque a efectos se puede considerar el tiempo de entrenamiento de un solo folder.



- **Uso de memoria:** Se tendrá en cuenta solo el uso de memoria de un folder, ya que en cómputo global se reservarán los mismos recursos de memoria durante todo el proceso de evaluación.

Para realizar las medidas de los modelos de ambos estudios se ha tomado como referencia el entorno virtual de 'Google Colaboratory' con el uso de la GPU 'NVIDIA T4'. De este modo se consigue tener un entorno libre de valores erróneos al ser un espacio controlado y ajeno a cualquier otro subproceso, además de tener una referencia común para medir todos los parámetros de todos los modelos a evaluar.

6.2. GRÁFICAS DE ANÁLISIS

Otras herramientas utilizadas para analizar el comportamiento del entrenamiento de un modelo se basan en la generación de gráficos que permitan visualizar ciertas características de interés en el estudio.

- **Matrices de confusión:** Con esta gráfica se puede evaluar la precisión de los modelos al clasificar las muestras y comprobar en qué tipo de clasificación no es tan preciso. Esta gráfica representa la cantidad de predicciones que ha realizado según su nivel de acierto, representado tanto los verdaderos positivos como los verdaderos negativos, como su nivel de error representado por sus falsos positivos y los falsos positivos.
- **Gráficas de precisión y pérdidas:** Para los modelos sin validación es posible obtener gráficamente la evolución de la precisión y las pérdidas para cada una de las épocas. De este modo se puede evaluar cómo convergen los modelos y cuánto tiempo toman en ello. En el caso de los modelos basados en árboles de decisión se grafica según árboles en vez de épocas en el eje horizontal.



6.3. RESULTADOS DE TÉCNICAS ‘DEEP LEARNING’ SOBRE MÉTODOS ANALÍTICOS

Se realizará un estudio diferenciado entre la evaluación sin validación cruzada y el otro realizado con validación cruzada ‘K-Folder’. Además, se analizarán las gráficas obtenidas de los modelos sin validación cruzada.

6.3.1. Resultados sin validación cruzada

Evaluando los modelos secuenciales de forma independiente según el uso de un determinado optimizador, obtenemos sus métricas de precisión, pérdidas, uso de memoria y tiempo de entrenamiento.

Optimizador	Precisión (%)	Pérdidas
Adam	97,61	0,052
Nadam	96,43	0,076
RMSProp	94,64	0,196
Adamax	80,95	0,441
Adagrad	67,86	0,628
Adadelta	32,14	0,801

Tabla 6.1.- Precisión y pérdidas en modelos secuenciales

Analizando los resultados del entrenamiento de la Tabla 6.1. los modelos secuenciales, en cuanto a precisión, se puede apreciar que el optimizador que consigue un mayor valor de precisión es aquel que usa optimizador ‘Adam’, con un 97,61% de precisión. Fijándose además en el valor de pérdidas, el modelo que usa optimizador ‘Adam’ presenta el valor más bajo de todos los optimizadores. Tanto los optimizadores ‘Adam’, ‘Nadam’ y ‘RMSProp’ consiguen unos valores de precisión y de pérdidas bastante óptimos, situando sus precisiones por encima del 90%. En cuanto a nivel de pérdidas, cabe destacar el valor de ambos optimizadores ‘Adam’ y ‘Nadam’ que consiguen valores cercanos a cero.



El optimizador que peores resultados ha dado ha sido el 'Adadelta' con un 32,14% de precisión. Este resultado se debe a que este tipo de optimizador se suele utilizar en datos que varían en el tiempo. Este optimizador intenta buscar patrones en las actualizaciones de los datos y, al ser cada muestra independiente de las otras, no consigue ajustar el modelo. Esto también explica la baja precisión del optimizador 'Adagrad' que se trata de otro método adaptativo. A nivel de pérdidas también es aquel que consigue el valor más alto en comparación con el resto de los optimizadores.

En términos de uso de memoria, todos los modelos secuenciales han usado un tamaño de memoria equivalente a 152 MB. En este aspecto no existe ninguna diferencia notable para su comparación.

Optimizador	Tiempo de entrenamiento (s)
Adam	8,64
Nadam	10,07
RMSProp	9,28
Adamax	9,52
Adagrad	7,82
Adadelta	7,97

Tabla 6.2.- Tiempo de entrenamiento de modelos secuenciales

Atendiendo a aspectos de tiempo de entrenamiento (Tabla 6.2.) se puede observar la rápida convergencia de los modelos con optimizador 'Adagrad' y 'Adadelta'. Sin embargo, ambos modelos han presentado valores de precisión poco significativos, indicativo de su escasa capacidad de aprendizaje. Refiriéndose a aquellos modelos con niveles aceptables de precisión, cabe destacar el modelo 'Adam' que ha sido aquel con un tiempo de entrenamiento más reducido de 8,64 segundos. Coincide además con ser el modelo más preciso de todos los demás.



Analizando los modelos basados en árboles de decisión, podemos parametrizar sus valores de precisión y de tiempo de entrenamiento. Dos aspectos importantes de estos modelos son el uso exclusivo de CPU para el entrenamiento, por lo que no se puede medir el uso de recursos por no usar la memoria RAM del acelerador gráfico, ni el valor nulo de las pérdidas por el funcionamiento de los modelos de árbol de decisión. Estos modelos buscan minimizar el valor de pérdida por ello el modelo no devuelve un valor de pérdida que pueda servir para extraer conclusiones.

Optimizador	Precisión (%)	Pérdidas
Random Forest	97,02	-
Gradient Boosted	96,43	-

Tabla 6.3.- Resultados modelos de árbol de decisión

Según los resultados de la Tabla 6.3. en términos de precisión ‘Random Forest’, presenta una ligera mejora respecto al modelo ‘Gradient Booted’ con una diferencia de apenas un 1%. Ambos modelos consiguen unos valores de precisión bastante elevados por encima de 90%.

Optimizador	Tiempo de entrenamiento (s)
Random Forest	0,97
Gradient Boosted	0,96

Tabla 6.4.- Tiempo de entrenamiento de modelos de árbol de decisión

Los tiempos de entrenamiento de ambos modelos son casi los mismos, tomando aproximadamente un tiempo de un segundo como se observa en la Tabla 6.4.

6.3.2. Gráficas de evaluación sin validación cruzada

De las métricas obtenidas de los modelos anteriores se puede analizar gráficamente los resultados de las predicciones, además del proceso de entrenamiento de los modelos.

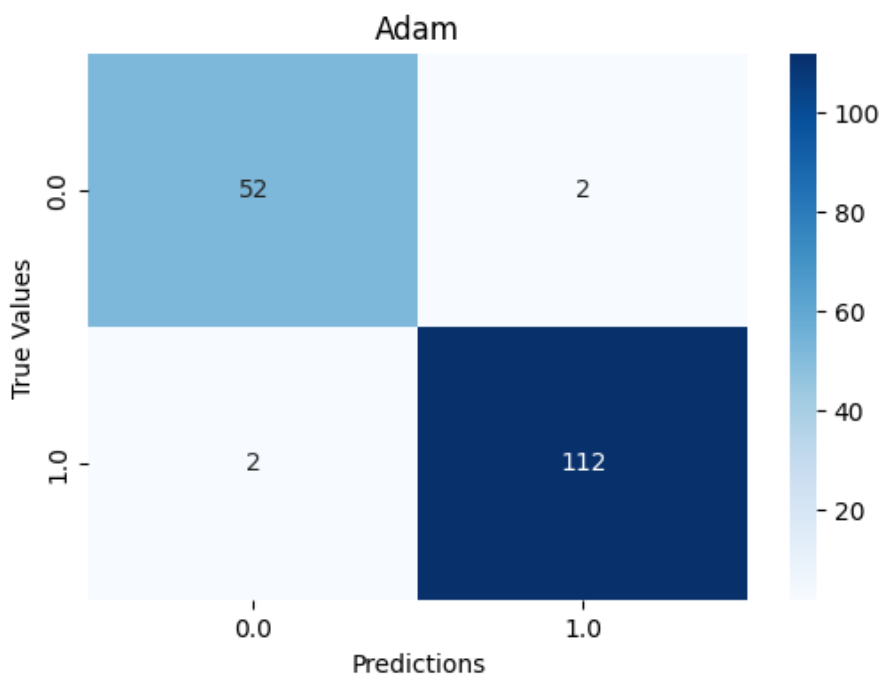


Figura 6.1.- Matriz de confusión de optimizador ‘Adam’

El modelo con optimizador ‘Adam’ es aquel que mejor ha maximizado los verdaderos positivos y verdaderos negativos de la base de datos. Solo ha clasificado erróneamente cuatro casos (dos falsos positivos y dos falsos negativos) (Figura 6.1.).

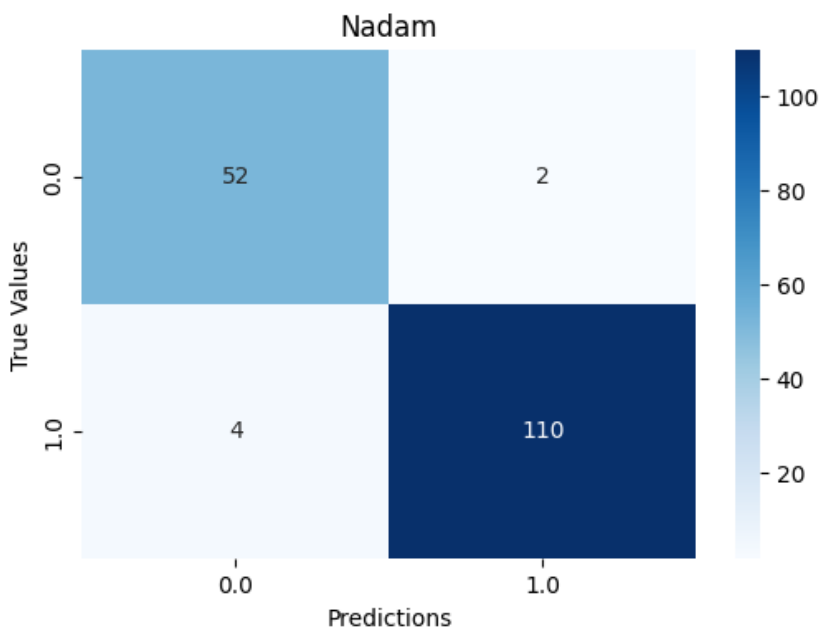


Figura 6.2.- Matriz de confusión de optimizador ‘Nadam’

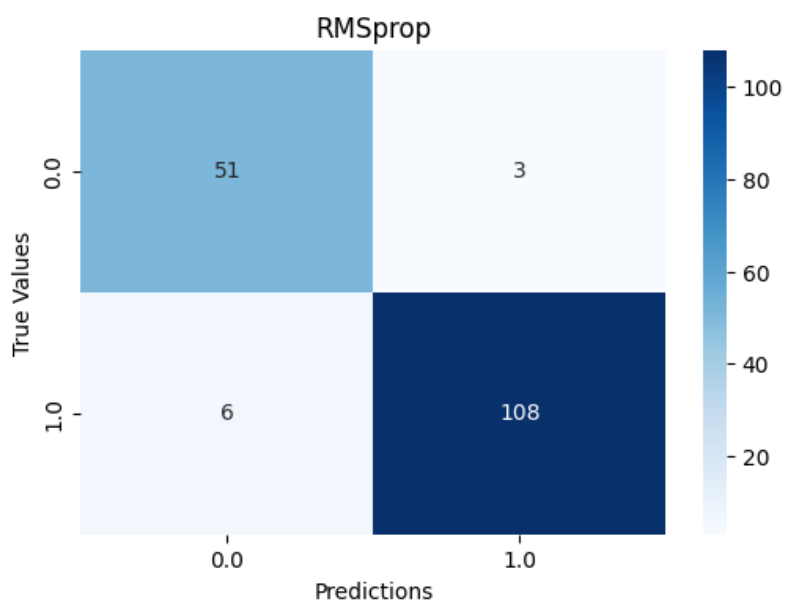


Figura 6.3.- Matriz de confusión de optimizador 'RMSProp'

En el caso de los optimizadores 'Nadam' y 'RMSProp' presentan una leve disminución en la precisión de la clasificación de los valores reales, con una clasificación errónea de seis casos (Figura 6.2.) y nueve casos (Figura 6.3.) respectivamente.

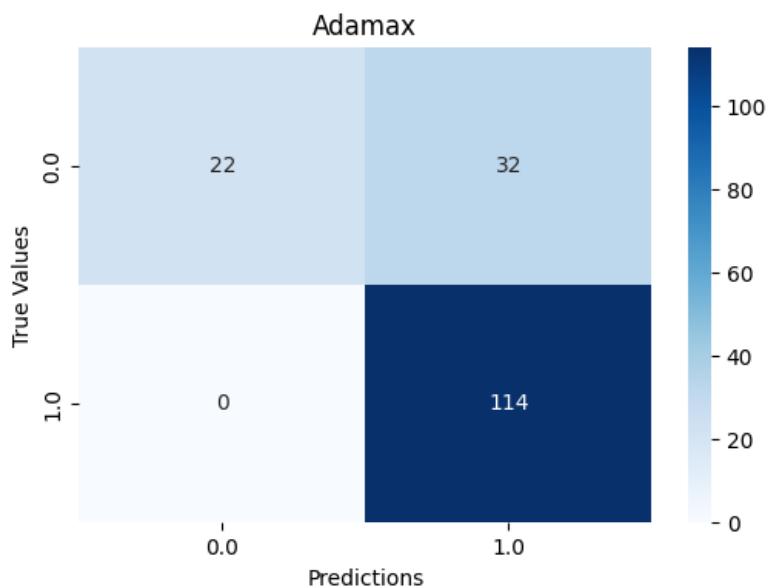


Figura 6.4.- Matriz de confusión de optimizador 'Adamax'



El modelo con optimizador 'Adamax' tiene la peculiaridad de haber fallado en los casos de clasificación solo de falsos positivos, mientras que la clasificación de falsos negativos es nula (Figura 6.4.)

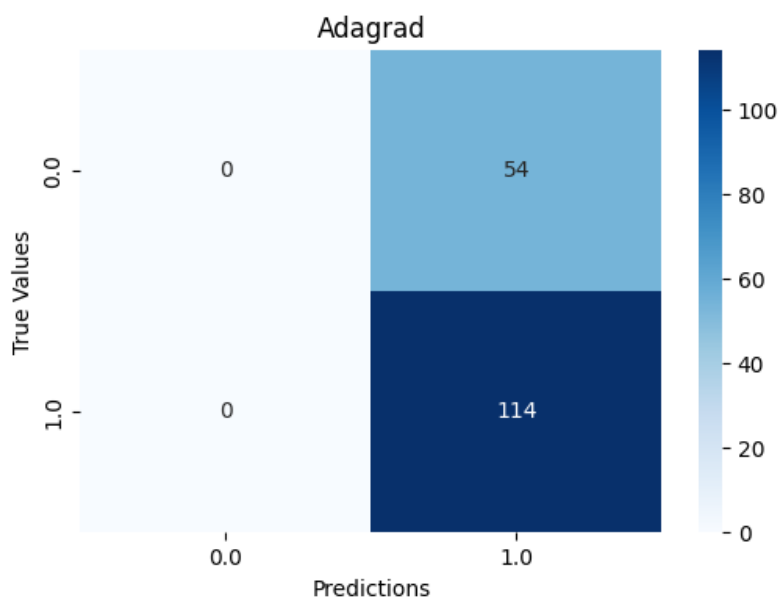


Figura 6.5.- Matriz de confusión de optimizador 'Adagrad'

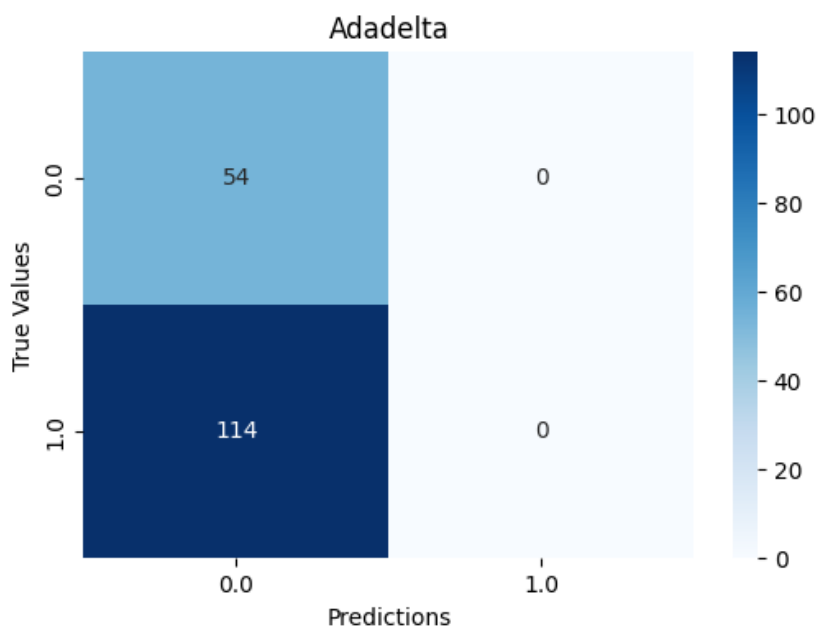


Figura 6.6.- Matriz de confusión de optimizador 'Adadelta'



Las gráficas de confusión de los modelos que han obtenido peor precisión de todos presentan precisión nula en los verdaderos positivos en el caso del optimizador ‘ (Figura 6.5.) y precisión nula en los verdaderos negativos del modelo ‘Adadelta’ (Figura 6.6.).

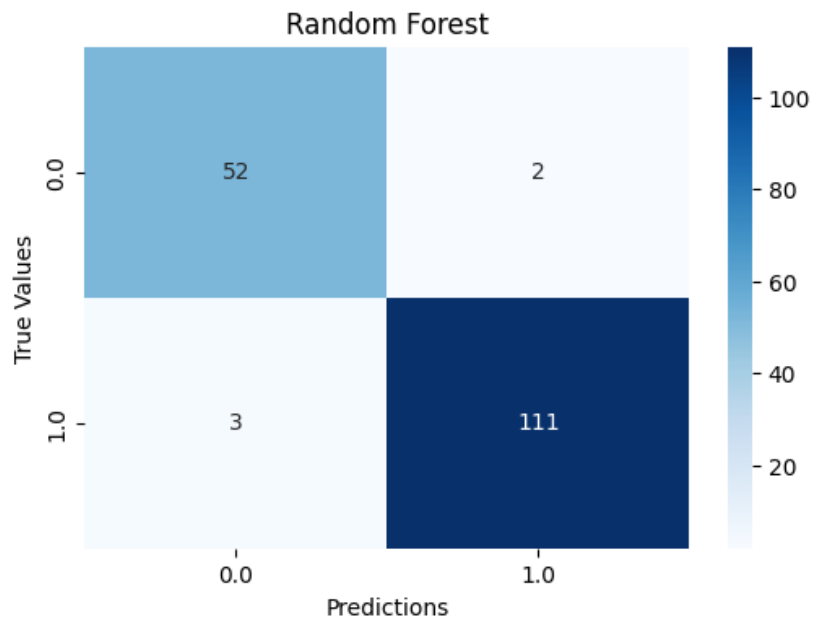


Figura 6.7.- Matriz de confusión de modelo ‘ Random Forest’.

El modelo ‘Random Forest’ consigue una imprecisión de cinco casos (dos falsos positivos y tres falsos negativos) (Figura 6.7.) adquiriendo una precisión parecida al modelo secuencial con optimizador ‘Adam’.

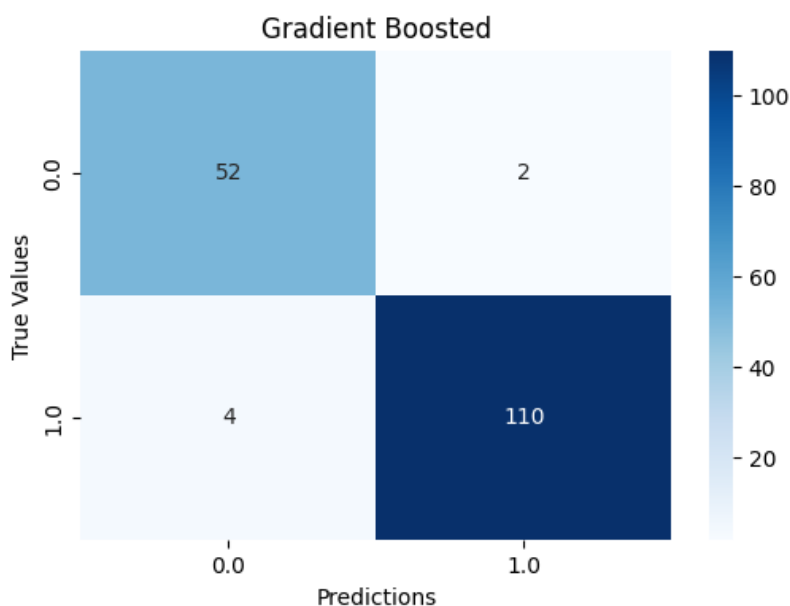


Figura 6.8.- Matriz de confusión de modelo 'Gradient Boosted'.

El modelo 'Gradient Boosted' empeora ligeramente respecto al otro modelo basado en árboles de decisión en tan solo un caso erróneo (Figura 6.8.). Ambos modelos consiguen prácticamente el mismo nivel de precisión en el entrenamiento específico de esta base de datos.

Atendiendo a la evolución de la precisión y pérdidas durante el entrenamiento a lo largo de las épocas se puede observar cómo en los modelos 'Adam' y 'Nadam' la precisión y pérdidas del conjunto de datos, tanto de entrenamiento como de validación, han seguido una evolución que denota la aceptable adaptación del entrenamiento a los datos. Las curvas de precisión de entrenamiento en ambos modelos, Figura 6.9. del modelo 'Adam' y Figura 6.11. del modelo 'Nadam', han crecido progresivamente desde la época cinco y alcanzaron su precisión máxima en la última época del entrenamiento.

En el caso de las pérdidas, tanto el optimizador 'Adam' (Figura 6.10.) como el optimizador 'Nadam' (Figura 6.13.) obtienen su valor mínimo aproximadamente a partir de la época veinte

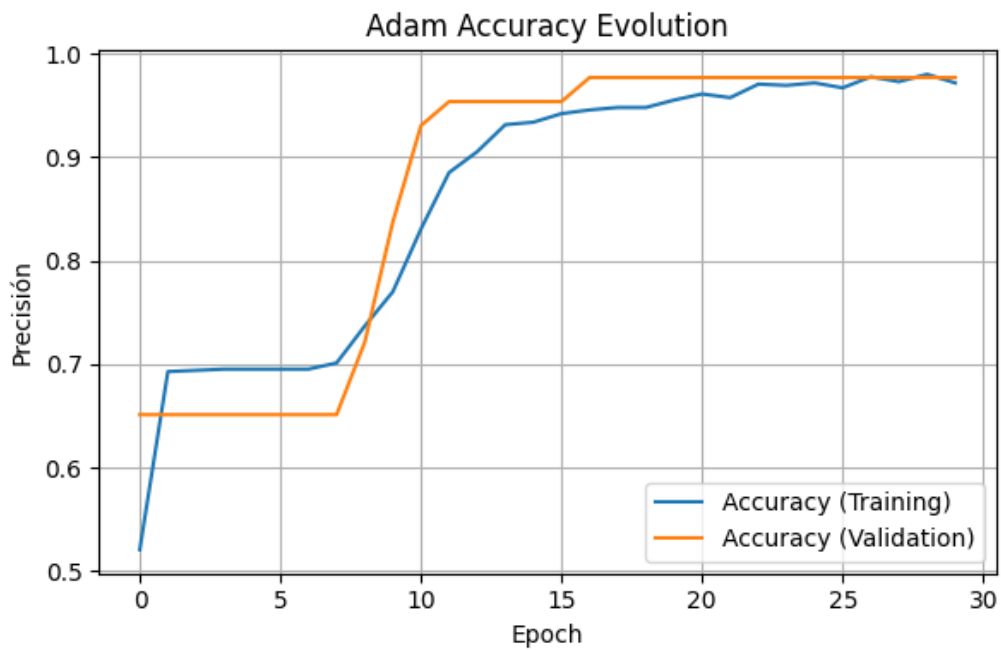


Figura 6.9.- Evolución de precisión del modelo 'Adam'.

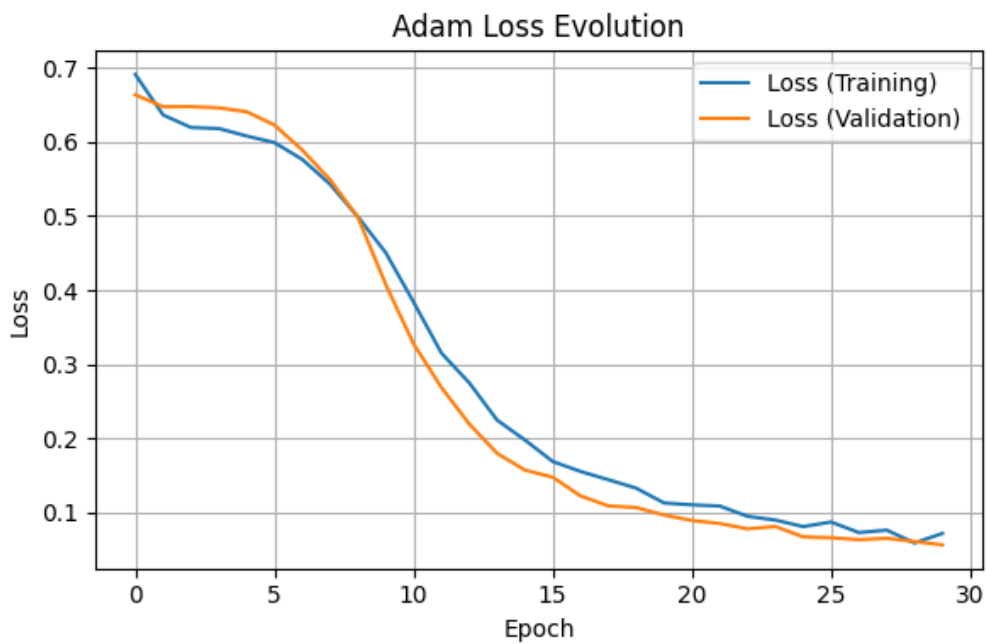


Figura 6.10.- Evolución de pérdidas del modelo 'Adam'.

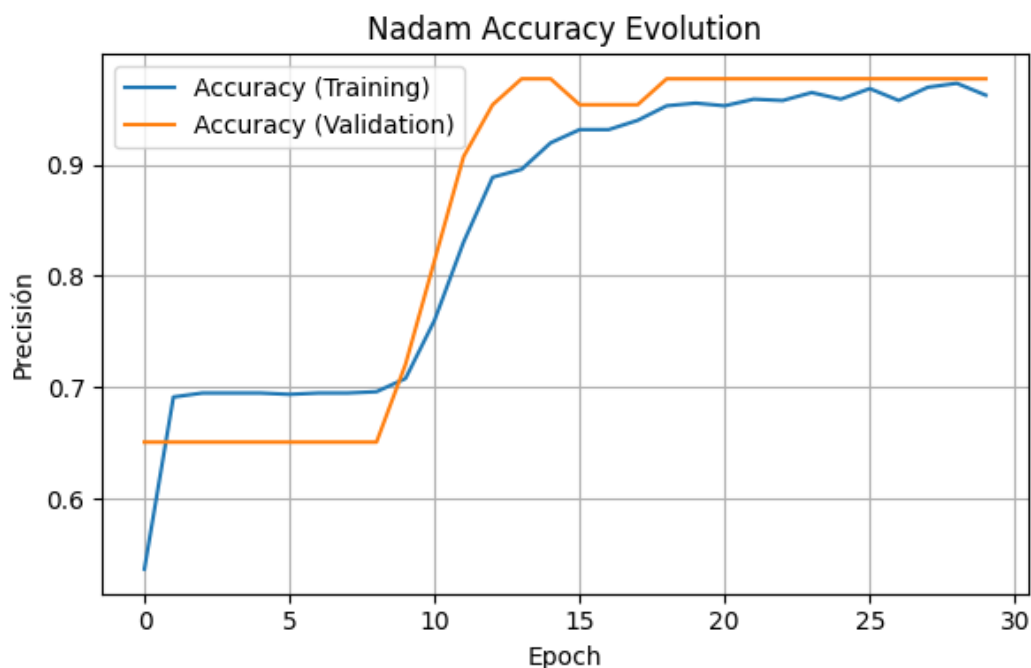


Figura 6.11.- Evolución de precisión del modelo 'Nadam'.

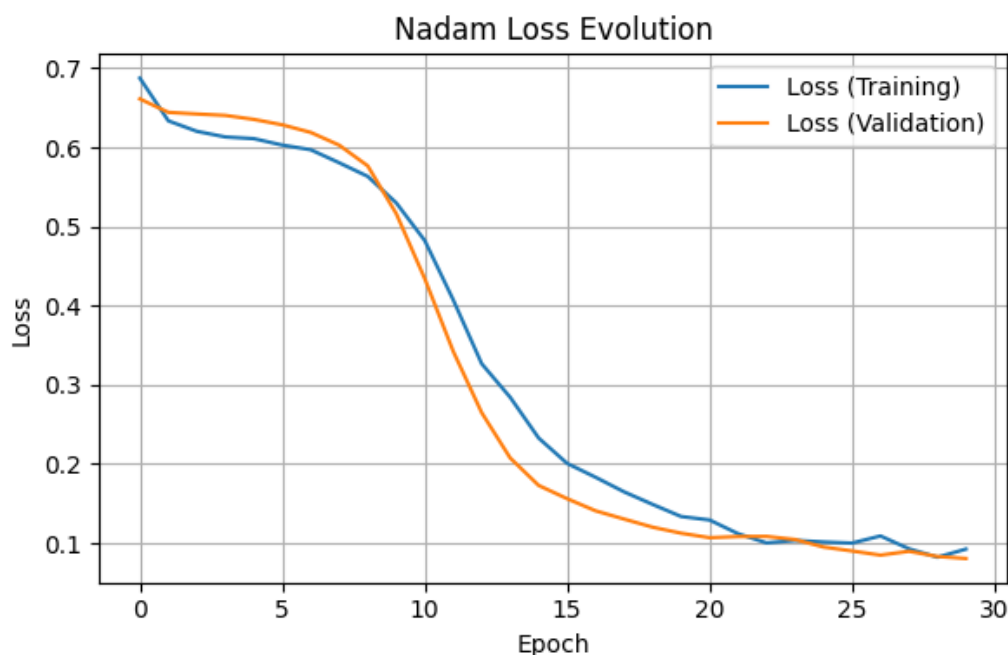


Figura 6.12.- Evolución de pérdidas del modelo 'Nadam'.

Si se analizan los dos modelos que peor resultado de precisión han obtenido, se puede observar cómo ambos valores de precisión se estancan prácticamente al inicio del entrenamiento, adquiriendo un valor no superior al 70% en el optimizador 'Adagrad' (Figura 6.13.) y 35% con el optimizador 'Adadelta'



(Figura 6.15.). Esto se debe a que ambos optimizadores pretenden ajustar el modelo según las variaciones de los datos. Al tener muestras independientes en toda la base de datos, este modelo no es capaz de aprender. La evolución de las gráficas de precisión de entrenamiento y validación siguen una evolución dispar, siendo la precisión de validación la misma durante todas las épocas, y las evoluciones de las pérdidas no disminuyen durante el entrenamiento como se puede observar en la Figura 6.14. para el optimizador ‘Adagrad’ y en la Figura 6.16. para el optimizador ‘Adadelata’, lo que denota la falta de aprendizaje de los modelos.

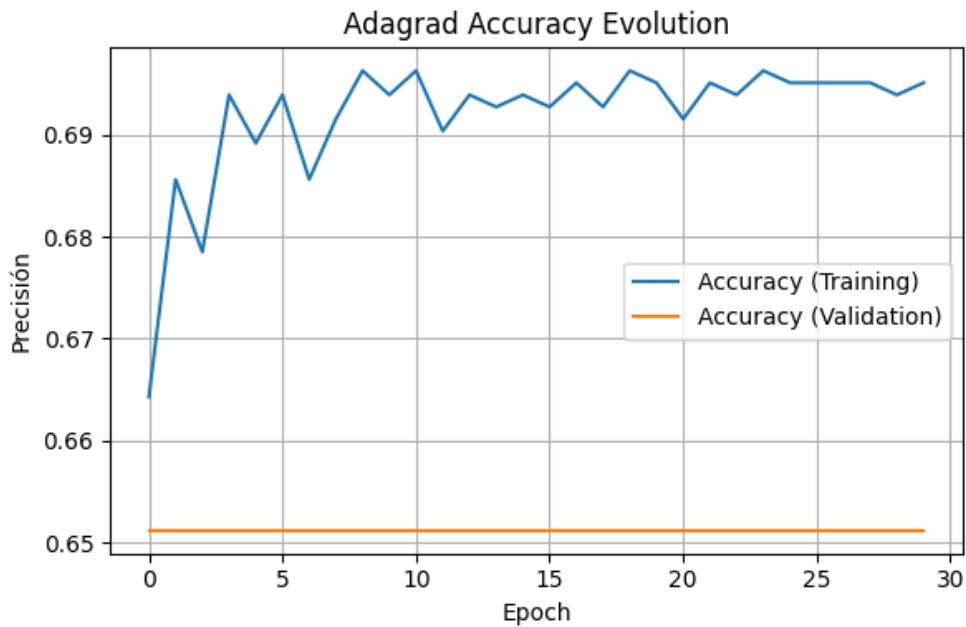


Figura 6.13.- Evolución de precisión del modelo ‘Adagrad’.

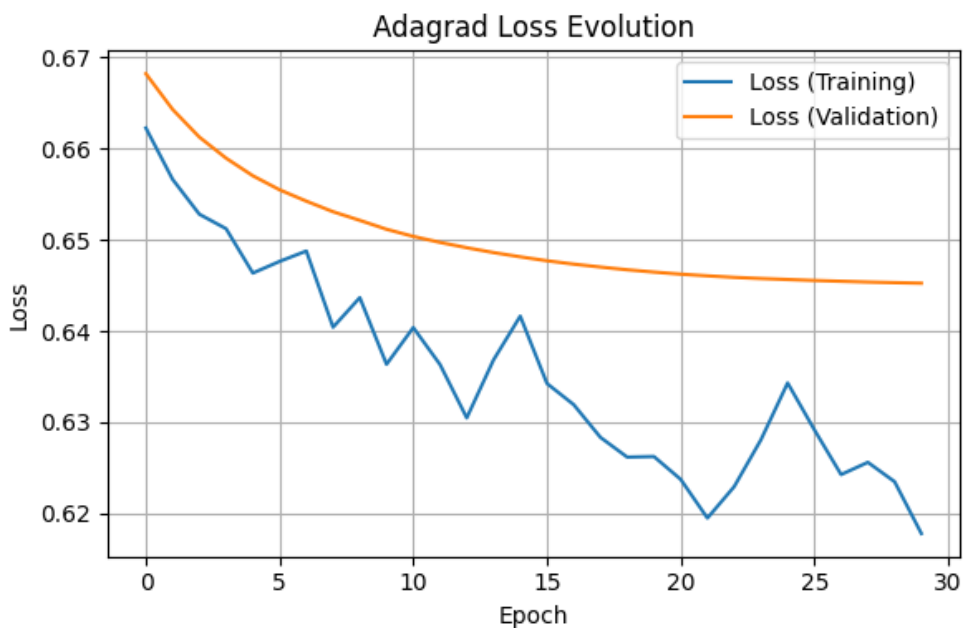


Figura 6.14.- Evolución de pérdidas del modelo 'Adagrad'.

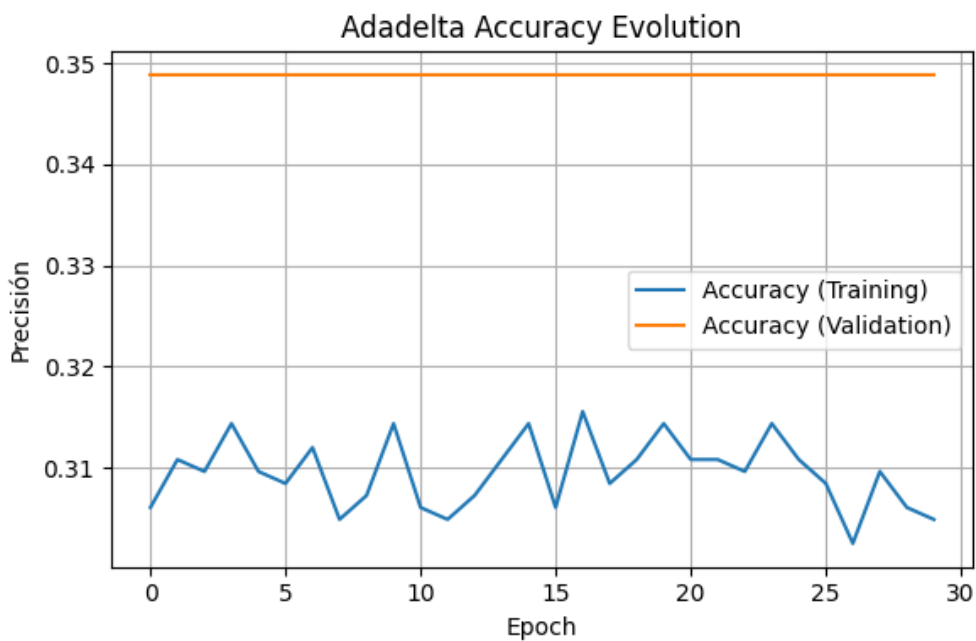


Figura 6.15.- Evolución de precisión del modelo 'Adadelta'.

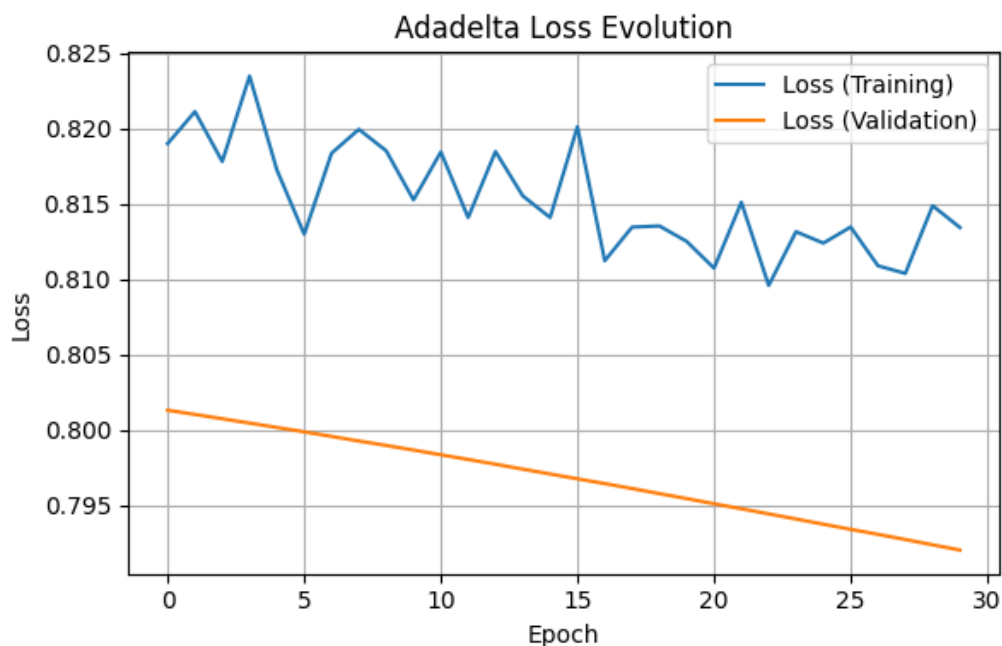


Figura 6.16.- Evolución de pérdidas del modelo 'Adadelta'.

Si estudiamos la evolución de la precisión de los métodos basados en árboles de decisión, se observa que el modelo que converge antes, en términos de creación de árboles, es el modelo 'Gradient Boosted' (Figura 6.17.). Este modelo converge alrededor de los 25 árboles frente al estancamiento del modelo 'Random Forest' entorno al 95%. (Figura 6.18.). La curva de evolución de las pérdidas presenta una evolución decreciente más pronunciada en el modelo 'Gradient Boosted' situándose en un valor de 0,1 aproximadamente a los cincuenta árboles y con tendencia a cero en su evolución (Figura 6.19.). El valor de pérdidas del modelo 'Random Forest' alcanza su valor mínimo aproximado de 0,25 y se estanca en ese valor a lo largo del resto de árboles (Figura 6.20.).

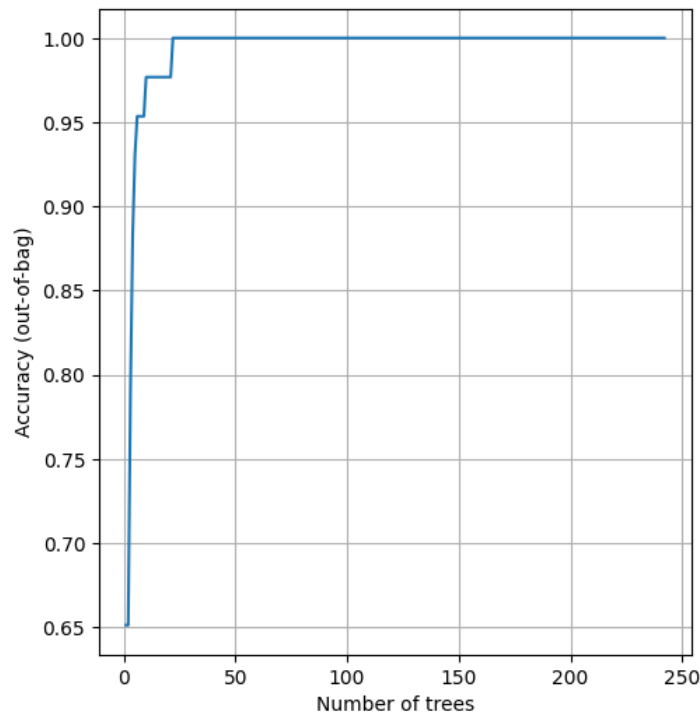


Figura 6.17.- Evolución de precisión del modelo 'Gradient Boosted'.

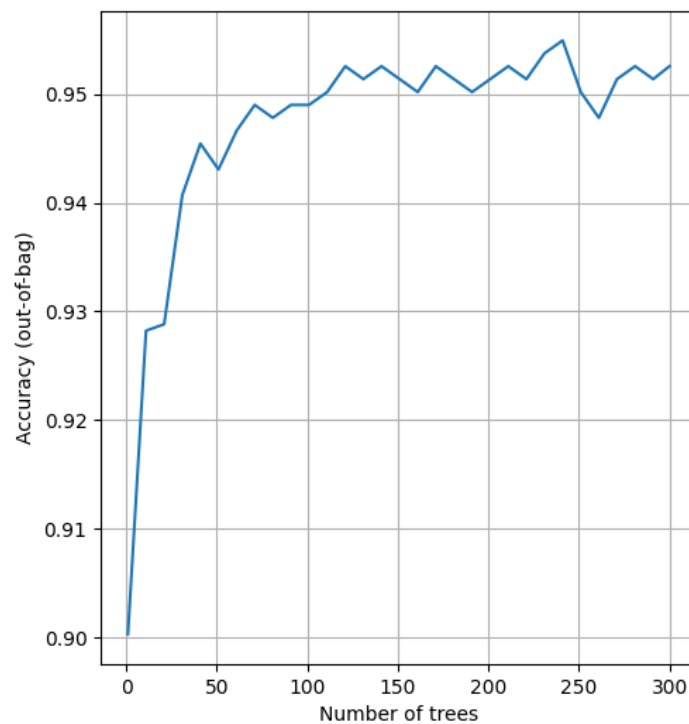


Figura 6.18.- Evolución de precisión del modelo 'Random Forest'.

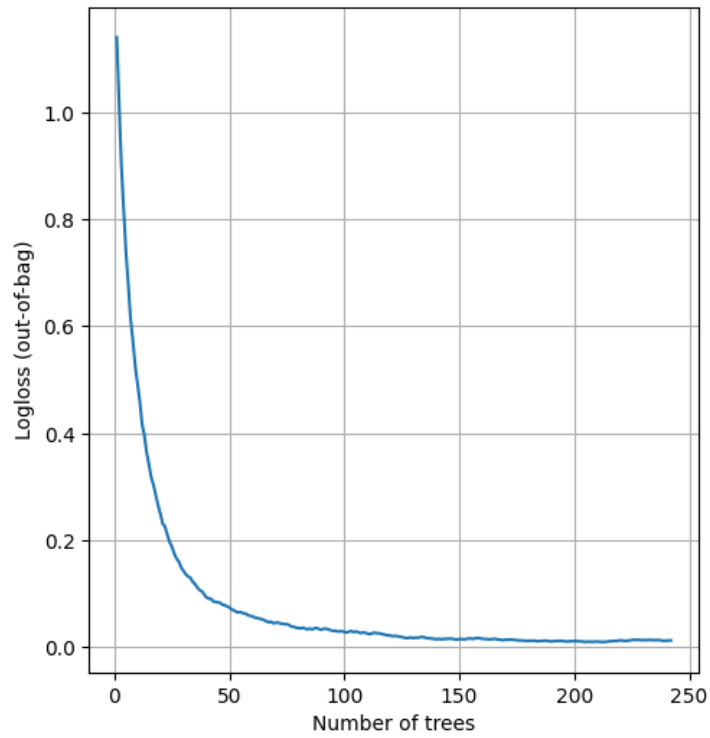


Figura 6.19.- Evolución de pérdidas del modelo 'Gradient Boosted'.

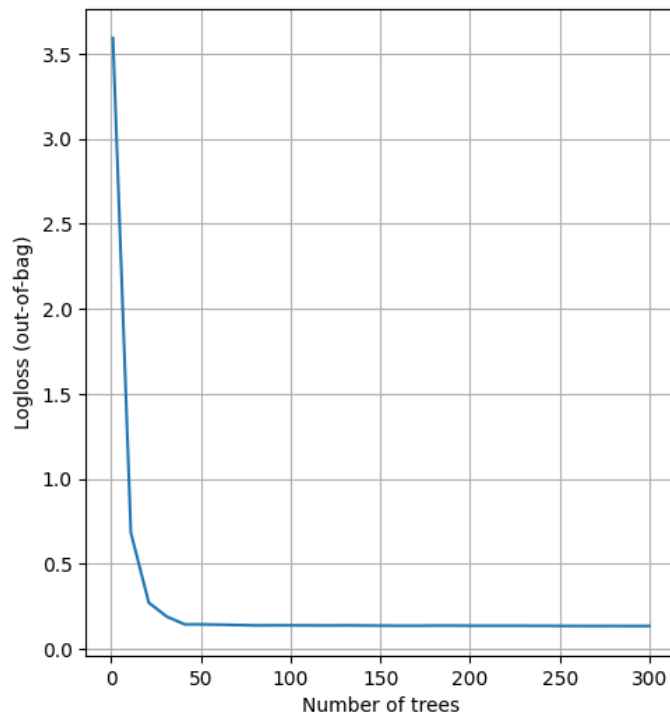


Figura 6.20.- Evolución de pérdidas del modelo 'Random Forest'.



6.3.3. Resultados con validación cruzada

Validando los modelos del apartado anterior con validación cruzada 'K-Folder' usando un número de diez conjuntos podremos obtener una estimación de la precisión y un error correspondiente a los diferentes valores de precisión de cada uno de los conjuntos.

Optimizador	Precisión y Error (%)	Pérdidas
Adam	97,53 ± 1,55	0,074
Nadam	97,63 ± 1,54	0,065
RMSProp	96,54 ± 1,35	0,116
Adamax	80,70 ± 8,77	0,395
Adagrad	69,25 ± 4,58	0,622
Adadelata	52,05 ± 19,64	0,754

Tabla 6.5.- Resultados modelos secuenciales.

Siguiendo los datos de la Tabla 6.6. el optimizador que ha obtenido el mayor valor de precisión se trata del modelo que usa 'Nadam' con un 97,63% de precisión. Como en la evaluación anterior, los modelos que han conseguido mejores resultados de precisión por encima del 90% son los modelos que hacen uso de optimizador 'Adam', 'Nadam' y 'RMSProp'. De dichos modelos el que ha conseguido el nivel de error más reducido se trata del 'RMSProp' con un error del 1,35%. El modelo con peor precisión ha sido aquel que hace uso de optimizador 'Adadelata', obteniendo una precisión del 52% y un error de casi un 20%. En cuanto a pérdidas, el modelo que mejor valor ha obtenido ha sido el optimizador 'Nadam' con unas pérdidas de 0,065.

Optimizador	Precisión y Error (%)	Pérdidas
Random Forest	95,15 ± 1,68	-
Gradient Boosted	96,04 ± 2,08	-

Tabla 6.6.- Resultados modelos de árbol de decisión.



Evaluando los modelos de árbol de decisión (Tabla 6.7.) se puede concluir que el modelo con mayor precisión es el ‘Gradient Boosted’ con un 96%. En cuanto a error el modelo ‘Random Forest’ presenta una ligera mejora frente al ‘Gradient Boosted’ con una diferencia de aproximadamente un 0,5%.

6.4. RESULTADOS DE TÉCNICAS ‘DEEP LEARNING’ SOBRE DATOS ‘EYE TRACKING’

Todas las evaluaciones en este estudio se hicieron con validación cruzada. En primer lugar, se mostrarán los resultados de los modelos compilados con optimizador ‘Adam’ y entrenados con los tres diferentes grupos de características. De los modelos que presenten mejores datos se analizarán sus resultados con el uso del optimizador ‘Adamax’ y con el grupo de características que mejor se haya comportado.

6.4.1. Resultados con optimizador ‘Adam’

Modelo	24 características	16 características	8 características
Secuencial	68,97 ± 12,61	64,50 ± 11,34	66,39 ± 10,03
LSTM	67,63 ± 17,49	68,62 ± 14,43	65,33 ± 7,58
Convolutacional	65,72 ± 13,84	64,01 ± 14,31	67,82 ± 10,97
Conv. + LSTM	66,00 ± 14,56	65,32 ± 12,50	66,24 ± 10,70
GRU	65,29 ± 18,87	66,91 ± 14,66	59,21 ± 7,90

Tabla 6.7.- Precisión y error de los modelos según grupo de características.

Analizando los resultados del modelo secuencial de la Tabla 6.7. podemos observar que obtuvo un mejor nivel de precisión al entrenarse con las veinticuatro primeras características, con una precisión de aproximadamente 69 %. En cuanto al error, se muestra con el nivel más pequeño en el entrenamiento realizado en el



grupo de ocho características con un 10% frente al error de aproximadamente un 13% con el entrenamiento de las veinticuatro características. No obstante, el valor de precisión es menor con una diferencia de alrededor de un 4%. El peor valor de precisión se ha observado en el modelo entrenado con dieciséis características, obteniendo una precisión del 64,5%.

Los resultados del modelo LSTM reflejan una mejor precisión en el entrenamiento con dieciséis características, obteniendo un valor aproximado del 69%. Pese a ello, obtiene un valor de error superior al modelo entrenado con ocho características con una diferencia de un 7%. Aunque dicho modelo presente una precisión inferior al resto, es aquel que presenta una tasa de error notablemente inferior respecto a los otros modelos LSTM. El modelo que peores datos presenta es aquel entrenado con veinticuatro características, ya que contiene una tasa de error de aproximadamente un 17%.

Para los modelos, tanto convolucional como híbrido los mejores resultados se han obtenido con el entrenamiento de ocho características, obteniendo un valor de 67,82% y 66,24% respectivamente. Ambos modelos han conseguido unos valores de precisión bastante parecidos, indicando que el añadido de una capa LSTM no ha mejorado significativamente en el modelo convolucional. El valor mínimo de error coincide en ambos modelos con el entrenamiento de ocho características, obteniendo ambos un valor aproximado del 11%.

El modelo GRU ha obtenido un valor de precisión máximo en el entrenamiento con dieciséis características con un valor aproximado del 67%. El peor resultado, también en comparación de otros modelos, lo ha obtenido en el entrenamiento con ocho características obteniendo una precisión del 59%.

En términos globales, se observa que los modelos que mejores valores de precisión han obtenido son el modelo secuencial con veinticuatro características, el modelo LSTM con dieciséis características y el modelo convolucional con ocho características.



Modelo	24 características	16 características	8 características
Secuencial	68,97 ± 12,61	-	-
LSTM	-	68,62 ± 14,43	-
Convolutacional	-	-	67,82 ± 10,97

Tabla 6.8.- Mejores resultados de precisión y error obtenidos.

Como se observa en la Tabla 6.8. en el que se ven los modelos con mejores resultados en cuanto precisión y error, destaca la precisión del modelo secuencial y el error del modelo convolutacional.

Modelo	24 características	16 características	8 características
Secuencial	0,75	0,76	0,68
LSTM	1,33	0,98	0,83
Convolutacional	0,90	0,83	0,89
Convolutacional + LSTM	0,73	0,66	0,74
GRU	1,01	0,83	0,74

Tabla 6.9.- Pérdidas de los modelos según grupo de características.

Los modelos con mejores resultados de pérdidas (Tabla 6.9.) son los que usan ocho características como entrenamiento, excepto el modelo híbrido, que obtiene el peor resultado respecto a las otras características. Las predicciones que presentan menor fiabilidad son aquellos que usan todas las características de estudio.



Modelo	24 características	16 características	8 características
Secuencial	118 s / epoch	132 s / epoch	130 s / epoch
LSTM	212 s / epoch	194 s / epoch	189 s / epoch
Convolutacional	156 s / epoch	151 s / epoch	149 s / epoch
Conv. + LSTM	227 s / epoch	202 s / epoch	210 s / epoch
GRU	207 s / epoch	190 s / epoch	184 s / epoch

Tabla 6.10.- Tiempo de entrenamiento.

Los modelos que más tardan en entrenarse y por ende más recursos consumen son los que entrenan con veinticuatro características (Tabla 6.10.). Por otro lado, los más livianos en términos temporales son aquellos que menor cantidad de características usan para su entrenamiento. A pesar de ello, el modelo que menor tiempo ha tardado en entrenarse ha sido el secuencial con veinticuatro características. Se puede comprobar que el modelo híbrido atrasa el tiempo de entrenamiento significativamente.

Modelo	24 características	16 características	8 características
Secuencial	383 MB	383 MB	255 MB
LSTM	391 MB	391 MB	263 MB
Convolutacional	383 MB	383 MB	255 MB
Conv. + LSTM	391 MB	391 MB	263 MB
GRU	391 MB	391 MB	263 MB

Tabla 6.11.- Uso de memoria en el entrenamiento.

En términos de memoria (Tabla 6.11.) los modelos de veinticuatro características y de dieciséis usan la misma cantidad de memoria. Se puede afirmar que los modelos que usan menos cantidad de memoria son aquellos que usan ocho características para el entrenamiento. Se observa además que los modelos que menos memoria necesitan para su entrenamiento son los secuenciales y los convolucionales, sin incluir el modelo híbrido. Aquellos modelos que menos uso



de memoria hacen el entrenamiento son el modelo secuencial y el modelo convolucional que usa ocho características de entrenamiento (Tabla 6.12.).

Modelo	N.º Caract	Precisión	Pérdidas	Entrenamiento	Memoria
Sec.	24	68,97 ± 12,61	0,75	118 s/epoch	383 MB
LSTM	16	68,62 ± 14,43	0,98	194 s/epoch	391 MB
Conv.	8	67,82 ± 10,97	0,89	149 s/epoch	255 MB

Tabla 6.12.- Comparativa de métricas de los mejores modelos.

6.4.2. Resultados con optimizador ‘Adamax’

Modelo	N.º Caract.	Precisión	Pérdidas	Entrenamiento	Memoria
Secuencial	24	73,05 ± 12,71	0,77	135 s/epoch	383 MB
LSTM	16	76,64 ± 17,14	1,13	178 s/epoch	391 MB
Conv.	8	65,17 ± 8,70	0,70	162 s/epoch	255 MB

Tabla 6.13- Comparativa de métricas de los modelos ‘Adamax’.

Según los resultados obtenidos, se ha observado mejoría en todos los modelos con el uso del optimizador ‘Adamax’ excepto el modelo convolucional, obteniendo este último un menor resultado de precisión comparando con el optimizador ‘Adam’ obteniendo aproximadamente un 3% menos (Tabla 6.13.).

El modelo que mejor resultado de precisión ha obtenido ha sido el LSTM con un 77% de precisión frente al 73% del modelo secuencial. Pese a ello, el modelo LSTM presenta un nivel más elevado de error en comparación que el resto de las modelos del 17%, empeorando en un 3% respecto al modelo de optimizador ‘Adam’. Los valores de pérdidas se incrementan además respecto al optimizador ‘Adam’, indicando una variación más contrastada en los resultados de los conjuntos. Para analizar mejor el comportamiento de los modelos se analizarán sus resultados en cada uno de los conjuntos, ya que mucho de los resultados



pueden deberse al empeoramiento de uno o varios conjuntos que puedan empeorar el resultado global.

	Secuencial	LSTM	Convolucional
Folder 1	66,49%	72,00%	52,80%
Folder 2	82,84%	78,83%	66,96%
Folder 3	81,15%	81,35%	68,72%
Folder 4	77,39%	85,02%	63,27%
Folder 5	60,73%	75,17%	53,74%
Folder 6	67,93%	73,04%	70,40%
Folder 7	86,05%	86,84%	69,05%
Folder 8	80,05%	91,46%	78,28%
Folder 9	84,28%	90,63%	75,14%
Folder 10	43,56%	32,04%	53,33%

Tabla 6.14.- Comparativa de precisión para cada folder.

Analizando los resultados de cada folder (Tabla 6.14.) podemos observar que el modelo que mejor resultado obtiene es el LSTM, solo con el número diez como peor resultado respecto a los otros modelos. Este valor provoca que en promedio el modelo presente un error tan elevado. El que peor resultado de precisión presenta es el convolucional, con prácticamente todos los valores de precisión por debajo de los otros modelos.



	Secuencial	LSTM	Convolucional
Folder 1	1,00	1,22	0,73
Folder 2	0,37	0,49	0,73
Folder 3	0,41	0,39	0,64
Folder 4	0,48	0,49	0,65
Folder 5	0,91	0,62	0,84
Folder 6	0,59	0,51	0,59
Folder 7	0,36	0,27	0,52
Folder 8	0,42	0,12	0,42
Folder 9	0,32	0,26	0,48
Folder 10	2,82	6,95	1,41

Tabla 6.15.- Comparativa de pérdidas para cada conjunto.

En cuanto a los resultados de pérdidas (Tabla 6.15.) el modelo LSTM presenta el mayor valor en el último folder. Esto provoca que el resultado global de pérdidas sea elevado. Analizando el resto de los conjuntos, el valor de pérdidas suele ser inferior al resto de modelos.

Optimizador	Precisión	Pérdidas	Entrenamiento	Memoria
Adam	68,97 ± 12,61	0,75	118 s/epoch	383 MB
Adamax	73,05 ± 12,71	0,77	135 s/epoch	383 MB

Tabla 6.16.- Comparación de modelo secuencial con distintos optimizadores.

Optimizador	Precisión	Pérdidas	Entrenamiento	Memoria
Adam	68,62 ± 14,43	0,98	194 s/epoch	391 MB
Adamax	76,64 ± 17,14	1,13	178 s/epoch	391 MB

Tabla 6.17.- Comparación de modelo LSTM con distintos optimizadores.



Optimizador	Precisión	Pérdidas	Entrenamien to	Memoria
Adam	67,82 ± 10,97	0,89	149 s/epoch	255 MB
Adamax	65,17 ± 8,70	0,70	162 s/epoch	255 MB

Tabla 6.18.- Comparación de modelo convolucional con distintos optimizadores.

El modelo que más mejora con el uso del optimizador ‘Adamax’ es el modelo LSTM (Tabla 6.17.). Este adquiere mayor nivel de precisión y menor tiempo de entrenamiento. También se observa una mejora en cuanto a precisión el modelo secuencial (Tabla 6.16.) aunque empeore en el resto de las métricas, sin incluir el uso de memoria que se mantiene con el mismo valor. El que peores resultados obtiene es el modelo convolucional (Tabla 6.18.) que empeora su precisión y tarda más tiempo en completar su entrenamiento.

En términos de uso de memoria, no existe ninguna mejoría en ninguno de los modelos con la implementación del optimizador ‘Adamax’.



7. Conclusiones y trabajos futuros

7.1. MÉTODOS ANALÍTICOS

De todos los modelos evaluados sobre los estudios analíticos, aquel modelo que mejores resultados ha obtenido, teniendo en cuenta su precisión y sus pérdidas, ha sido el modelo secuencial con optimizador 'Nadam' (Tabla 7.1.). Pese a su tiempo de entrenamiento ligeramente superior respecto a otros optimizadores, su uso para la detección del TEA respecto a los otros optimizadores y a los modelos basados en árboles de decisión.

Cabe destacar que todos los modelos, a excepción de los modelos secuenciales que han utilizado optimizadores 'Adamax', 'Adagrad' y 'Adadelta', han conseguido resultados más que aceptables y con un muy buen rendimiento fijándonos en su uso de memoria y tiempo de entrenamiento.

Pese a todo ello, la base de datos cuenta con reducidas muestras lo que limita el uso de estos modelos para detectar otros casos. Una posible mejora es la recolección de una base de datos más extensa que cuente con más participantes.

La mejora en los modelos se puede analizar haciendo un estudio de cada característica con su coeficiente de correlación. De este modo se puede evaluar su uso para la categorización. Al igual que se hizo en el segundo estudio, se puede hacer un ranking con las características que mayor valor de coeficiente de correlación presenten para estudiar el modelo en distintos grupos de características. Los resultados cosechados en este estudio son mejor que suficientes para los datos de esta dataset.



	Precisión y Error (%)	Pérdidas	Entrenamiento	Memoria
Secuencial	97,63 ± 1,54	0,065	10,07 s.	152 MB

Tabla 7.1.- Modelo que mejores resultados obtiene en los métodos analíticos.

7.2. MÉTODOS SOBRE 'EYE TRACKING'

En este estudio existen diversos aspectos a considerar. En primer lugar, la base de datos se puede considerar no tan extensa como para detectar casos reales de TEA. Este dataset puede contar con ciertas anomalías que hayan podido afectar en el entrenamiento de los modelos. Estas anomalías podrían ser muestras con valores vacíos en ciertas características o muestras desechadas debido a la falta de identificación del participante.

Pese a estas características, el modelo que ha presentado mejores resultados, en términos de precisión, pérdidas y rendimiento ha sido el modelo LSTM con el uso de dieciséis características para su entrenamiento y optimizador 'Adamax' (Tabla 7.2.). Aunque es el modelo que mayor uso de memoria presenta, su tiempo de entrenamiento es notablemente menor que el resto de los modelos con una precisión y pérdidas parecidas.

	Precisión (%)	Pérdidas	Entrenamiento	Memoria
LSTM	76,64 ± 17,14	1,13	178 s/epoch	391 MB

Tabla 7.2 - Modelo que mejores resultados obtiene sobre 'Eye Tracking'.

Los valores de precisión y pérdidas están aún lejos de considerarse aceptables, ya que el método aún presenta cierta imprecisión. Al tratarse de una base de datos limitada y con características temporales complejas para su detección, hacen falta pruebas con modelos LSTM u otros modelos basados en redes neuronales recurrentes con más capacidad de procesamiento y abstracción. Se ha demostrado que un acercamiento al entrenamiento de este tipo de datos



basados en la captura de visión es con modelos de tipo LSTM con optimizadores especializados en datos variantes en el tiempo.

En cuanto al nivel de error, aunque dicho modelo cuenta con el mayor valor de error respecto a otros modelos, se ha verificado en su evaluación por validación cruzado que presenta mejores resultados en prácticamente todos los conjuntos menos en el último de ellos. Esto produce un resultado global mucho peor al promediar las distintas precisiones y pérdidas.

Los modelos secuenciales han sido los que mejor rendimiento ha tenido en este estudio. Para trabajos futuros sería interesante evaluar otras arquitecturas más complejas capaces de detectar patrones más complejos para realizar predicciones de un modo más preciso.

Otros modelos que pueden mostrar cierta mejora para este tipo de bases de datos son aquellos basados en modelos 'Transformers', como los modelos que utilizan arquitecturas 'Multi Head'. Los modelos se usan para su uso en traducción simultánea y pueden adaptarse para estos datos.

Otro factor que considerar en este estudio es el tiempo que tomaron los modelos en entrenarse. Aun usando tarjetas gráficas de grandes prestaciones desde el entorno virtual de 'Google Colaboratory', se han presentado casos en los que los modelos han tardado mucho tiempo en entrenarse, limitando en algunos casos la cantidad de evaluaciones a realizar. En el ejemplo del modelo LSTM entrenado, con un tiempo de entrenamiento igual a 178 segundos por época, ha tomado en total unas dos horas y media entrenarlo. Aun así, al realizar validación cruzada con diez conjuntos, el tiempo de validación se ha multiplicado por diez pasando a tardar un día entero. Como es uno de los modelos que, en promedio, menos tardó en entrenarse, las evaluaciones para cinco modelos con tres grupos de características han tomado mucho tiempo y recursos. Con el uso de equipos con mayor capacidad de cómputo, como por ejemplo una granja de aceleradores gráficos, se podrían hacer más pruebas para comprobar diferentes arquitecturas, características y optimizadores.



7.3. CONCLUSIONES GENERALES

Como conclusión general se puede decir que se han alcanzado los objetivos iniciales del proyecto. Estos estudios han aportado en muchos aspectos al campo de investigación y han permitido alcanzar nuevas conclusiones que dan un valor añadido a los estudios realizados anteriormente.

En el primer estudio se han ampliado los métodos de clasificación clínicos con el uso de nuevas técnicas de 'Deep Learning'. Los resultados han superado en ciertos aspectos a los métodos basados en 'Machine Learning' del estudio original, además que ha permitido indagar en los optimizadores que mejor se pueden comportar con este tipo de bases de datos. El hecho de comparar estas arquitecturas con los modelos 'Machine Learning' basados en árboles de decisión ha permitido realizar una comparación más exacta con los modelos más actualizados. Pese a la evidente mejora de algunos modelos de aprendizaje profundo, los modelos 'Machine Learning' presentan una evidente ventaja en términos de eficiencia y precisión.

En cuanto al estudio de detección por 'Eye Tracking', se ha podido aportar un modelo de detección funcional a un proyecto basado en la mera recolección de datos 'Eye Tracking'. Esto podría suponer un punto de partida para la mejora de futuros modelos de detección que utilicen metodologías o arquitecturas más precisas. En ese aspecto, el estudio realizado ha alcanzado su objetivo principal. Además, se ha podido comprobar la relación de ciertas características con la determinación de patrones de supuestas características del espectro autista. Por último, se ha comprobado como los optimizadores especializados en detección de variaciones temporales han significado en una notable mejora en los resultados.

En cuanto a las técnicas empleadas y el conocimiento requerido para este estudio caben destacar varias conclusiones. Pese a que el campo de investigación de la inteligencia artificial sea novedoso y que la documentación en algunos casos de estudio sea escasa, los conocimientos adquiridos en ciertos ámbitos universitarios



han demostrado la posibilidad de hacer frente a un proyecto de estas características con su resolución. La base informática y de programación ha sido decisiva para la realización de estos estudios. Asignaturas como 'Programación', 'Servicios de comunicaciones básicos' o 'Aplicaciones Telemáticas' consiguen establecer una base de conocimientos que hacen posible ser más resolutivo con ciertas tareas a resolver.

En el campo de la inteligencia artificial son esenciales los conceptos matemáticos, por ello es importante tener una buena base en este ámbito. Asignaturas como 'Estadística', 'Álgebra lineal', 'Cálculo' o 'Métodos numéricos' ha aportado en el entendimiento de ciertos algoritmos usados por las inteligencias artificiales para su funcionamiento. Otro aspecto importante ha sido el tratamiento de los datos pertenecientes a las datasets. Para este procedimiento han sido útiles ciertos conocimientos adquiridos en asignaturas como 'Gestión de redes', donde se enseña a trabajar con tablas, datos y sus interpretaciones. Además, otra asignatura que ha aportado en el uso de datos y tablas de información ha sido 'Sistemas de información de la ingeniería'.



8. Presupuesto

Para realizar este estudio ha sido necesario el trabajo de dos personas y el uso de dos ordenadores dotados de aceleración gráfica. Para acelerar el proceso y dotar al estudio de recursos más eficientes se hará uso del servicio 'Google Colaboratory' para el entrenamiento de los modelos. En el caso del personal, se considerará el suelo acordado en el convenio de trabajadores específico de los ingenieros técnicos informáticos, aproximado a 27.000 € al año. Hay que considerar que la duración total del estudio es de diez meses, por lo que se tendrá que considerar el trabajo de ambos trabajadores en ese periodo. El precio correspondiente a los recursos dependerá de varios factores, debido al uso y a su amortización. El gasto bruto del proyecto se puede observar desglosado en la Tabla 8.1.

- **Ordenadores:** Se tendrá en cuenta el precio de cada ordenador con sus respectivos componentes. Además, se tendrá en cuenta un porcentaje correspondiente al veintiséis por ciento para su amortización en equipos informáticos.
- **Servicios en línea:** Se tendrá en cuenta la suscripción al servicio 'Google Colaboratory' en toda la fase de desarrollo. Se tendrá en cuenta además el precio de los créditos de procesamiento, que pagando la mensualidad de 'Google Colab Pro' habrá suficientes créditos para entrenar los modelos. Se equilibrará de todas formas el entrenamiento entre los dos otros ordenadores para no necesitar contratar más créditos de procesamiento. Cada mes de procesamiento cuesta doce euros y, teniendo en cuenta que la fase de desarrollo dura cinco meses en total, habrá que destinar unos sesenta euros en total.



- **Licencias de Software:** Se incluirán también los gastos pertenecientes a la adquisición de las licencias de software. En este proyecto se considerarán los gastos del sistema operativo 'Windows', de unos ciento treinta y dos euros cada licencia, y la licencia de uso anual de 'Office', destinada a la creación de la documentación, de unos noventa y nueve euros anuales cada una de ellas.

	Descripción	Cantidad	Precio
Personal	Sueldo correspondiente a un director de proyecto y el de un investigador.	2 trabajadores durante 10 meses	45.000,00 €
Ordenadores + Complementos	Equipo destinado a realizar todo el proceso de creación, entrenamiento y evaluación de los modelos.	2 unidades	2.200,00 €
Licencias de software	Correspondientes a dos licencias del sistema operativo Windows (264€) para el manejo de los programas y Office (198€) para la documentación.	2 unidades	462,00€
Servicios en línea	Servicio utilizado para entrenar los modelos basados en 'Deep Learning'	5 mensualidades	60,00 €
TOTAL			47.722,00 €

Tabla 8.1. Gastos en el proyecto en bruto.

Además de los gastos anteriores, se tendrá en cuenta también el veintiséis por ciento de amortización para el apartado de ordenadores. Otros conceptos para tener en cuenta serán aquellos correspondientes al trece por ciento destinado a gastos generales, calculados sobre el coste bruto, el seis por ciento destinado



como beneficio industrial y el IVA correspondiente al veintiún por ciento de todo el proyecto. Estos gastos se ven reflejados en la Tabla 8.2.

	Descripción	Precio
Amortización de equipo	Porcentaje del 26% anuales para la amortización de los ordenadores y sus complementos. Teniendo en cuenta los 10 meses de duración del proyecto.	476,67 €
Gastos Generales	Porcentaje del 13% correspondiente a los costes fijos que se realizarían a lo largo de la realización del proyecto.	6.265,83 €
Beneficio Industrial	Se tendrá en cuenta un porcentaje del 6% sobre el coste total del proyecto para sus costes fijos, teniendo en cuenta además los gastos generales.	3.239,27 €
IVA	Importe sobre el valor añadido del proyecto de un 21% sobre el coste bruto del proyecto.	10.021,62 €
TOTAL		20.003,39 €

Tabla 8.2. Conceptos a añadir sobre el precio bruto del proyecto.



	Descripción	Precio
Coste bruto	Coste bruto del proyecto.	47.722,00 €
Devengos	Costes extra calculados en el apartado anterior.	20.003,39 €
TOTAL		67.725,39 €

Tabla 8.3 Coste total del proyecto.

Teniendo en cuenta su precio bruto y los diferentes costes añadidos, se puede concluir que el precio total del proyecto será de sesenta y siete mil setecientos veinticinco mil euros con treinta nueve céntimos, tal como se puede observar en la Tabla 8.3.



9. Manual de instalación

En este apartado se seguirán los pasos para la instalación de los distintos elementos que conforman el proyecto para el preprocesamiento, implementación y evaluación de los modelos. Se hará distinción entre los dos estudios puesto que cada uno de ellos cuenta con diferentes librerías y diferentes versiones que permitan el correcto funcionamiento de cada una de sus funciones.

9.1. INSTALACIÓN DE CUDA

A la hora de entrenar los modelos, debemos cerciorarnos de que el entorno en el que trabajamos pueda acceder al hardware de aceleración gráfica. Para ello se debe de considerar los pasos a seguir para instalar el software pertinente.

9.1.1. Actualización de los drivers de 'NVIDIA'

Para instalar los distintos elementos es primordial asegurarse de que tenemos la última versión de los drivers de 'NVIDIA'. De este modo nos aseguraremos del buen funcionamiento de todos los elementos.

Para llevar a cabo este paso, se debe de ir a la página oficial de 'NVIDIA', ir a descarga de drivers y seleccionar el tipo de gráfica con la que estemos trabajando.



9.1.2.Instalación de kit de herramientas ‘CUDA’

Para trabajar con el dispositivo de aceleración gráfica y con la plataforma de computación en paralelo debemos de instalar las herramientas ‘CUDA’. Este elemento permitirá compilar y entrenar los modelos con una mayor efectividad ya que optimiza el trabajo de la ‘GPU’. Esta herramienta se debe de descargar en el sitio oficial de ‘NVIDIA’ en el panel de drivers.

En la página de instalación se puede observar que existen diversas versiones disponibles. Es importante asegurar qué tipo de software ‘CUDA’ es compatible con nuestra ‘GPU’. Para ello existe documentación en la misma página de descarga con los modelos de tarjeta gráfica compatibles.

Es importante instalar la versión local en formato ‘.exe’ con la arquitectura ‘Windows’ del software que estemos usando.

9.1.3.Instalación de las herramientas ‘cuDNN’

Es necesario instalar el software que nos permita optimizar el trabajo con las redes neuronales de aprendizaje profundo. Para ello existen una serie de herramientas de ‘NVIDIA’ que son capaces de trabajar eficientemente con este tipo de tecnologías. Esta biblioteca de herramientas es ‘cuDNN’. Se puede descargar directamente de la página oficial de ‘NVIDIA’. Es importante instalar la herramienta ‘cuDNN’ que sea compatible con el software ‘CUDA’ que se ha instalado anteriormente.

Para ver la compatibilidad, en el apartado de documentación de ‘NVIDIA’, se puede verificar el soporte de las diferentes librerías ‘cuDNN’ con la versión ‘CUDA’ correspondiente.



9.1.4. Verificación en 'Python'

Una vez instaladas las herramientas que nos permitirán trabajar con la tarjeta gráfica, se debe comprobar que el entorno de trabajo reconozca el hardware de aceleración gráfica. En el caso particular de este proyecto, en el que se trabaja con 'Tensorflow', se puede utilizar el código perteneciente a la Figura 9.1. De este modo nos podemos asegurar de que el entorno reconoce de forma correcta el hardware. 'Tensorflow' cuenta con una función que evalúa si está disponible la tarjeta gráfica llamada 'tf.test.is_gpu_available()'. En el caso que la detecte, se podrá acceder a la información sobre dicho dispositivo.

```
if tf.test.is_gpu_available():
    print("GPU disponible. TensorFlow utilizará la GPU.")
    tf.config.experimental.set_memory_growth(
        tf.config.list_physical_devices('GPU')[0], True)
else:
    print("No se encontró GPU. TensorFlow utilizará la CPU.")
```

Figura 9.1.- Verificación de detección de 'GPU'.

9.2. ENTORNO PYTHON

Para ejecutar cada uno de los procesos relativos a cada uno de los estudios, es recomendable hacerlo desde el archivo 'Jupyter' del proyecto. De este modo se podrá ejecutar por separado cada elemento para poder llevar cierto control sobre el proceso. Si se ejecuta el código sin usar dicho archivo, se ejecutarán todo el proceso secuencial y podría ser confuso al evaluarlo. Además, si se ejecuta todo el código, se empezará el proceso de preprocesamiento desde cero conllevando tiempos de procesamiento y usos de memoria más largos.



El entorno más recomendado para ejecutar el código es mediante el visor de ‘Google Colaboratory’ (Figura 9.2.)

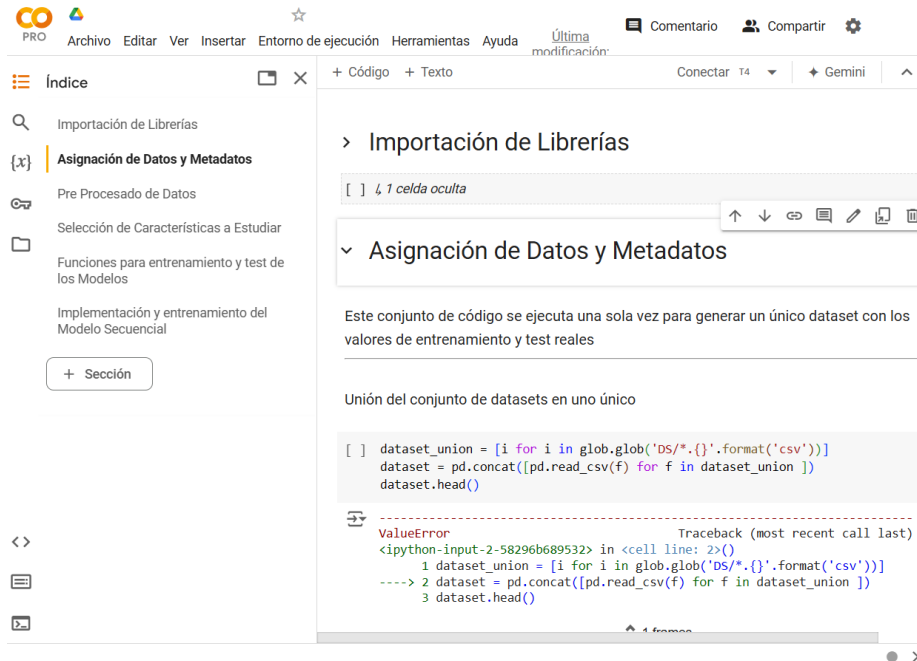


Figura 9.2.- Entorno de trabajo de ‘Google Colaboratory Pro’.

También es posible realizar esta ejecución desde las aplicaciones de ‘Visual Studio Code’ o desde la aplicación ‘Jupyter Notebooks’. Ambos programas pueden acceder a un entorno gestionado por ‘Anaconda Navigator’. En esta aplicación se configurará un entorno con la versión más actualizada de Python y las librerías citadas.

- **Anaconda Navigator:** Se puede instalar dicho programa desde su página oficial de forma totalmente gratuita. Una vez dentro se creará un entorno con la última versión disponible de Python. Una vez creado se pueden instalar las librerías a través de la terminal.
- **Visual Studio Code:** Podremos acceder a esta aplicación desde el mismo ‘Anaconda’. Una vez iniciado, se puede acceder al archivo ‘Jupyter’ que contiene todas las secciones de código. Si se ejecutan cada una de ellas se puede seleccionar el entorno creado en el paso anterior de ‘Anaconda’.

También se puede ejecutar desde la base 'Visual Studio' (Figura 9.3.). En este caso, habría que instalar de nuevo todas las dependencias.

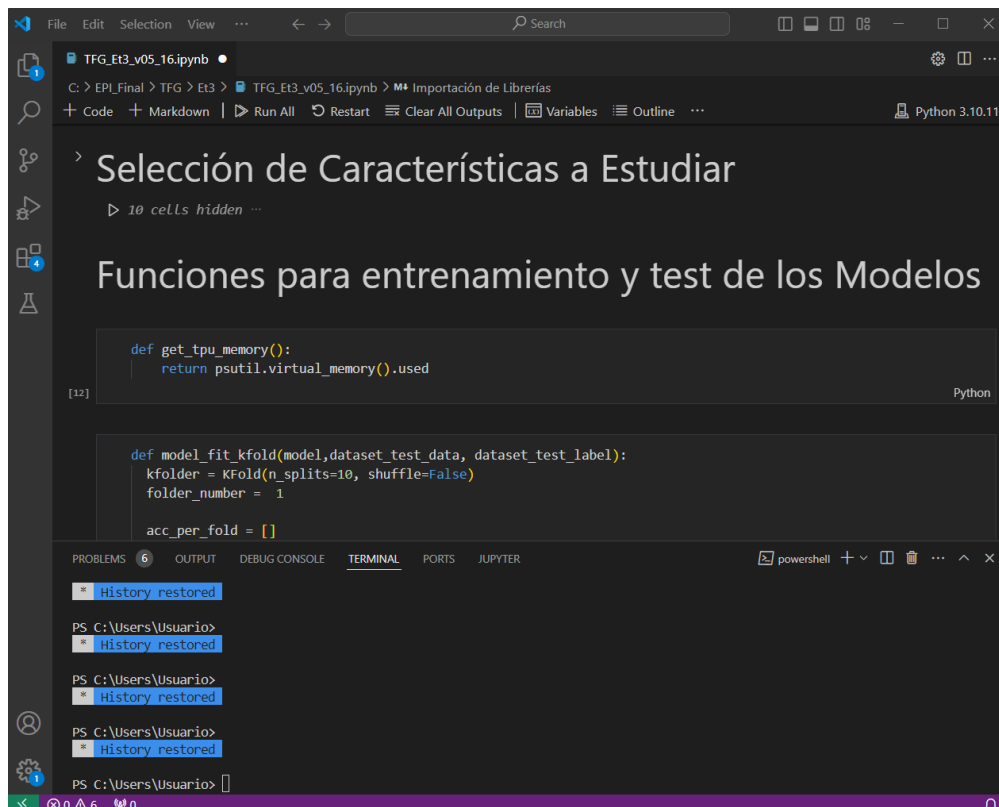


Figura 9.3.- Entorno de trabajo de 'Visual Code'.

- **Jupyter Notebooks:** Desde la aplicación 'Anaconda Navigator' Se puede acceder a este entorno gráfico que permite ejecutar los archivos 'Jupyter' del mismo modo que podríamos hacer desde 'Google Colaboratory' o 'Visual Studio Code' (Figura 9.4.).

```
Modelos de Árboles de Decisión

In [ ]: def Decision_Tree_model_fit(model_type):

#Creación del Modelo

if model_type=="GB":
    model = tfdf.keras.GradientBoostedTreesModel()
else:
    model = tfdf.keras.RandomForestModel()

#Compilación y Entrenamiento
model.compile(metrics=['accuracy'])
import gc
start_time = time.time()

model.fit(TF_train, validation_data=(TF_validation))

training_time = time.time() - start_time

tf.keras.backend.clear_session()
gc.collect()
from numba import cuda
device = cuda.get_current_device()
device.reset()

#Evaluación
Test_loss, Test_accuracy = model.evaluate(TF_test)
dataset_test_predictions_RF = model.predict(TF_test)
dataset_test_predictions_RF = np.round(dataset_test_predictions_RF)

return model, dataset_test_predictions_RF, Test_accuracy, training_time
```

Figura 9.4.- Entorno de trabajo de 'Jupyter Notebook'.

9.3. IMPORTACIÓN DE LAS DATASETS

Para usar las bases de datos, hay que considerar el dónde disponerlas y cómo acceder a ellas. En el caso de ejecutar el código desde 'Google Colaboratory' se montará la unidad de 'Google Drive' en el proyecto. De este modo podremos alojar las bases de datos en la nube y se podrá acceder a ellas sin tener que subir los archivos cada vez que iniciemos el servicio, consumiendo tiempo y recursos que podríamos ahorrar.

Es importante escribir el código Python de la Figura 9.5. en una nueva celda antes de la carga de los archivos para poder acceder a ellos.

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

Figura 9.5.- Montaje de unidad de 'Google Drive' en 'Google Colaboratory'



En el caso de usar 'Visual Studio Code' podemos alojar los archivos directamente en la carpeta raíz. Así, el código reconocerá el nombre de cada archivo para usarlos en cada proceso sin necesitar ninguna acción extra.

Si se utiliza 'Jupyter Notebooks' será necesario importar los archivos al entorno, al menos la primera vez que lo iniciemos. Existe un apartado en la aplicación donde podemos importar las bases de datos y estas quedarán disponibles para cualquier uso que se realice en el proceso.

En el caso del segundo estudio, como el proceso de preprocesamiento es tan extenso, se realizará un guardado de la base de datos, independiente a la base inicial, con los cambios realizados. De este modo tendremos diferentes versiones de la misma base de datos con modificaciones diferente. El propio programa guarda estos archivos y tiene en cuenta la importación de dichos datos.

9.4. Estudio sobre datos analíticos

Al instalar las librerías, hay que considerar que algunas de ellas deben cumplir ciertos requisitos en cuanto a su versión. Es el caso, por ejemplo, de las librerías 'tensorflow' y 'tensorflow_decision_forest'.

- **Tensorflow:** Se considerará usar la versión '2.15.0' o '2.16.1'. Depende qué versión se utilice hay que tener en cuenta que debe de ser compatible con la librería 'tensorflow_decision_forest', como se indicará en su sección. Otro aspecto para tener en cuenta es que estas versiones más actualizadas tienen implementado el uso de GPU, tras eliminar la librería 'tensorflow-gpu'. Es importante probar que tensorflow reconozca el acelerador gráfico para que haga uso de el en los entrenamientos de modelos.
- **Tensorflow decisión forest:** Se considerará usar las versiones '1.8' o '1.9'. La versión '1.8' solo es compatible con la versión tensorflow '2.15.0' y



la versión '1.9' está disponible para la versión de tensorflow '2.16.1'. Es importante instalar las versiones adecuadas puesto que, en caso contrario, nos advertirá de un error por incompatibilidad. La instalación de esta librería es menos problemática en el entorno virtual de 'Google Colaboratory', ya que en otros entornos pide además la actualización o desactualización de la librería 'keras'.

- **Sklearn:** Esta librería está considerada como deprecada. Aun así, para cumplir con la compatibilidad del resto de librerías, se instalará la librería 'scikit-learn' y se activará la opción de uso de la librería deprecada 'sklearn' según el código de la Figura 9.6. en la terminal.

```
> conda env config vars set  
SKLEARN_ALLOW_DEPRECATED_SKLEARN_PACKAGE_INSTALL=True
```

Figura 9.6.- Comando para usar método deprecado 'sklearn' en el entorno.

- **Otras librerías:** Otras librerías a instalar, que no presentan problemas de compatibilidad, son 'pandas', 'numpy', 'csv', 'matplotlib', 'seaborn' y 'time'. Para poder trabajar con estas librerías, lo primero de todo, será instalarlas desde la terminal del entorno Python que se esté utilizando (Figura 9.7.).

```
> pip install tensorflow==2.16.1  
> pip install scikit.learn  
> pip install pandas numpy csv matplotlib seaborn time
```

Figura 9.7.- Instalación de librerías desde la terminal para el primer estudio.

9.5. Estudio sobre datos 'Eye Tracking'

La instalación de las librerías referentes a este estudio es prácticamente igual al estudio anterior sin la necesidad de instalar la librería de decisión de árbol. Además, se instalarán las librerías correspondientes a 'glob' y 'math'.



Como ocurre en el estudio anterior, es necesario hacer uso de la librería deprecada 'sklearn' por lo que es necesario introducir el código de la Figura 9.8. en la terminal para hacer uso de las herramientas de dicha librería.

```
> pip install tensorflow==2.16.1  
> pip install tensorflow_decision_forest==2.19  
> pip install scikit.learn  
> pip install pandas numpy csv matplotlib seaborn time  
> pip install glob math
```

Figura 9.8- Instalación de librerías desde la terminal para el segundo estudio.

El código generará un archivo del preprocesamiento de la base de datos en cada proceso. De este modo se ahorrarán recursos cuando se hagan ciertas modificaciones correspondientes a etapas futuras.

Para ejecutar las celdas se tendrá en cuenta aquellas no necesarias y se importará el archivo de base de datos necesario. Si solo se van a probar los modelos o se quiere realizar algún cambio de arquitectura se importará la base de datos correspondiente al último generado por el proceso de preprocesamiento.



Bibliografía

- [1] European Commission. Joint Research Centre., *AI Watch, historical evolution of artificial intelligence: analysis of the three main paradigm shifts in AI*. LU: Publications Office, 2020. Accedido: 5 de marzo de 2024. [En línea]. Disponible en: <https://data.europa.eu/doi/10.2760/801580>
- [2] «Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).»
- [3] L. Zwaigenbaum, J. A. Brian, y A. Ip, «Early detection for autism spectrum disorder in young children», *Paediatrics & Child Health*, vol. 24, n.º 7, pp. 424-432, oct. 2019, doi: 10.1093/pch/pxz119.
- [4] S. Raj y S. Masood, «Analysis and Detection of Autism Spectrum Disorder Using Machine Learning Techniques», *Procedia Computer Science*, vol. 167, pp. 994-1004, 2020, doi: 10.1016/j.procs.2020.03.399.
- [5] «Markelle Kelly, Rachel Longjohn, Kolby Nottingham, The UCI Machine Learning Repository, <https://archive.ics.uci.edu>».
- [6] K. Vakadkar, D. Purkayastha, y D. Krishnan, «Detection of Autism Spectrum Disorder in Children Using Machine Learning Techniques», *SN COMPUT. SCI.*, vol. 2, n.º 5, p. 386, sep. 2021, doi: 10.1007/s42979-021-00776-5.
- [7] A. S. Heinsfeld, A. R. Franco, R. C. Craddock, A. Buchweitz, y F. Meneguzzi, «Identification of autism spectrum disorder using deep learning and the ABIDE dataset», *NeuroImage: Clinical*, vol. 17, pp. 16-23, 2018, doi: 10.1016/j.nicl.2017.08.017.
- [8] R. Nur Syahindah Husna, A. R. Syafeeza, N. Abdul Hamid, Y. C. Wong, y R. Atikah Raihan, «FUNCTIONAL MAGNETIC RESONANCE IMAGING FOR AUTISM SPECTRUM DISORDER DETECTION USING DEEP LEARNING», *Jurnal Teknologi*, vol. 83, n.º 3, pp. 45-52, abr. 2021, doi: 10.11113/jurnalteknologi.v83.16389.
- [9] M. Beary, A. Hadsell, R. Messersmith, y M.-P. Hosseini, «Diagnosis of Autism in Children using Facial Analysis and Deep Learning».



- [10] «Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. CoRR, abs/1704.04861. Retrieved from <http://arxiv.org/abs/1704.04861>».
- [11] A. Lu y M. Perkowski, «Deep Learning Approach for Screening Autism Spectrum Disorder in Children with Facial Images and Analysis of Ethnoracial Factors in Model Development and Application», *Brain Sciences*, vol. 11, n.º 11, p. 1446, oct. 2021, doi: 10.3390/brainsci11111446.
- [12] I. A. Ahmed *et al.*, «Eye Tracking-Based Diagnosis and Early Detection of Autism Spectrum Disorder Using Machine Learning and Deep Learning Techniques», *Electronics*, vol. 11, n.º 4, p. 530, feb. 2022, doi: 10.3390/electronics11040530.
- [13] B. S. V, «Grey Level Co-Occurrence Matrices: Generalisation and Some New Features», *IJCSEIT*, vol. 2, n.º 2, pp. 151-157, abr. 2012, doi: 10.5121/ijcseit.2012.2213.
- [14] C. Szegedy *et al.*, «Going Deeper with Convolutions». arXiv, 16 de septiembre de 2014. Accedido: 26 de febrero de 2024. [En línea]. Disponible en: <http://arxiv.org/abs/1409.4842>
- [15] K. He, X. Zhang, S. Ren, y J. Sun, «Deep Residual Learning for Image Recognition». arXiv, 10 de diciembre de 2015. Accedido: 26 de febrero de 2024. [En línea]. Disponible en: <http://arxiv.org/abs/1512.03385>
- [16] F. Cilia, R. Carette, M. Elbattah, J.-L. Guérin, y G. Dequen, «Eye-Tracking Dataset to Support the Research on Autism Spectrum Disorder».
- [17] M. Radhakrishnan, K. Ramamurthy, K. K. Choudhury, D. Won, y T. A. Manoharan, «Performance Analysis of Deep Learning Models for Detection of Autism Spectrum Disorder from EEG Signals», *TS*, vol. 38, n.º 3, pp. 853-863, jun. 2021, doi: 10.18280/ts.380332.
- [18] O. M. Olaniyan, A. I. Oyedeji, y O. I. Ifeka, «Development of a microcontroller EEG-based system for diagnosis of autism spectrum disorder in developing countries», *J. Phys.: Conf. Ser.*, vol. 1734, p. 012033, ene. 2021, doi: 10.1088/1742-6596/1734/1/012033.



- [19] S. Katoch, S. S. Chauhan, y V. Kumar, «A review on genetic algorithm: past, present, and future», *Multimed Tools Appl*, vol. 80, n.º 5, pp. 8091-8126, feb. 2021, doi: 10.1007/s11042-020-10139-6.
- [20] M. Abadi *et al.*, «TensorFlow: A system for large-scale machine learning».