Predicción de enfermedades neurocognitivas mediante electroencefalografía y Aprendizaje Profundo

Eduardo Montoya, Jose Luis Cadavid y Sofia Muñoz
Fundamentos de Deep Learning
Entrega 2

Diciembre de 2025

1. Contexto de aplicación

Para el desarrollo del sistema de clasificación basado en *Deep Learning*, se construyó una base de datos a partir de señales de electroencefalografía (EEG) en estado de reposo. El procesamiento incluyó la segmentación mediante ventanas temporales de 5 segundos, generando un conjunto estructurado de características para cada segmento. El objetivo principal es realizar una clasificación multiclase de los sujetos en cuatro grupos diagnósticos, permitiendo distinguir tanto la presencia de la mutación E280A como distintos estadios cognitivos y un grupo control externo.

1.1. Distribución de las Clases (Diagnósticos)

El conjunto total está compuesto por 5,724 muestras (ventanas) provenientes de 124 sujetos únicos. Las clases se definen de la siguiente manera:

Cuadro 1: Distribución de sujetos y muestras por clase.

Clase	**Sujetos**	**Muestras**
GG (Portadores Mutación)	37	1,712
GU (Controles Familiares)	31	1,420
DCL (Deterioro Cognitivo Leve)	18	846
CTR (Controles Sanos)	38	1,746

1.2. Características por Muestra

Cada muestra (ventana) está representada por un vector de 128 características numéricas, diseñadas para capturar la dinámica espectral y la complejidad de la actividad cerebral. Estas características se organizan según:

- **Canales de EEG (8 electrodos):** FP1, FP2, C3, C4, P7, P8, O1, O2.
- **Bandas de Frecuencia (8 rangos):** Delta, Theta, Alpha-1, Alpha-2, Beta-1, Beta-2, Beta-3 y Gamma.

1.3. Métricas Extraídas

- **Potencia Relativa (método de Welch):** Cuantifica la energía de la señal en cada banda de frecuencia.
- **Entropía de Permutación:** Estima la complejidad y la dinámica no lineal de la serie temporal.

2. Descripción de los notebooks:

Para implementar el modelo de clasificación de las diferentes clases, se desarrolló un flujo de trabajo dividido en tres etapas. En primer lugar, se realizó un análisis exploratorio inicial de la base de datos (Notebook 01). Posteriormente, dado que el conjunto de datos presentaba un número reducido de muestras, se llevó a cabo un proceso de preparación y aumento de los datos. Finalmente, la tercera etapa consistió en la implementación de los modelos de clasificación: primero un modelo de deep learning orientado a la clasificación de los cuatro tipos de diagnóstico (Notebook 02), y posteriormente un modelo para la clasificación binaria (Notebook 03). A continuación, se describe en detalle el contenido de cada uno de los notebooks.

2.1. Descripción de la estructura del notebook 01-Carga_y_exploracion_datos.ipynb

Este notebook está orientado a la importación del conjunto de datos y a la realización de una exploración inicial para comprender su estructura, dimensiones y distribución de las variables clave. Su contenido se organiza en las siguientes etapas:

2.1.1. Cargar del DataFrame

Se inicia leyendo el archivo "EEG_WELCH_PermEntropy_por_banda.csv", que contiene datos de series de tiempo con características de entropía y potencia para distintos sujetos con diferentes diagnósticos. Este archivo se almacena en un objeto DataFrame, lo que permite disponer de la información en un formato adecuado para los análisis posteriores.

2.1.2. Exploración inicial del dataset

Se presentan métricas básicas destinadas a obtener una primera comprensión del tamaño y forma del conjunto de datos: se observa la cantidad de filas y columnas, se revisa el número de ventanas asociados a cada sujeto. Para su visualización general, se generan estadísticas descriptivas y un gráfico de barras.

2.1.3. Análisis descriptivo de los sujetos

En este paso se identifican el número total de sujetos del dataset y reportan cuantos sujetos hay por cada clase lo cual es relevante para evaluar posibles desbalances en la tarea de clasificación.

2.2. Descripción de la estructura del notebook 02_modelo_multiclaseipynb

Este notebook abarca desde la preparación estructurada de las secuencias hasta la implementación completa del modelo y su correspondiente fase de validación. Las etapas principales incluidas son:

2.2.1. Configuración inicial y parámetros globales

Una vez se incluyen todas las librerías necesarias para este análisis, se fijan las semillas aleatorias de NumPy y TensorFlow con el fin de garantizar la reproducibilidad de los experimentos, y se establecen parámetros fundamentales para el preprocesamiento, como la longitud de las secuencias temporales (`SEQ_LENGTH`), el desplazamiento entre ventanas consecutivas (`STRIDE`), que permite un solapamiento elevado entre secuencias, y las variables `USE_AUGMENTATION` y `N_AUGMENTATIONS`, que controlan la activación y la intensidad del proceso de data augmentation.

2.2.2. Carga y verificación de los datos

Se importa el archivo principal con características EEG preprocesadas y se separan las columnas de metadatos y las variables de entrada. Además, se realiza verificaciones preliminares, tales como: conteo de features y ventanas totales, revisión y tratamiento de valores faltantes en las características (que se cambian por la mediana) y obtención de la lista de sujetos únicos y su distribución diagnóstica.

2.2.3. División estratificada por sujeto (train/test)

En esta etapa se realiza la división estratificada por sujeto en los conjuntos de entrenamiento y prueba. La estratificación se realiza tomando como referencia la clase de diagnóstico, de modo que aproximadamente el 80 % de los sujetos se asigna al conjunto de entrenamiento y el 20 % al conjunto de prueba. Esta estrategia garantiza que cada individuo pertenezca únicamente a uno de los dos conjuntos, evitando así el sobreajuste que podría presentarse si secuencias del mismo sujeto aparecieran en ambas particiones.

2.2.4. Definición de funciones de data augmentation

El data augmentation o aumento de datos es una técnica crucial cuando se trabaja con datasets pequeños, como este es un proyecto de clasificación de señales EEG. El problema fundamental que resuelve el data augmentation es el sobreajuste, ya que cuando un modelo de arquitecturas complejas como las redes LSTM se entrena con pocos datos tiende a memorizar los ejemplos específicos en lugar de aprender los patrones generales subyacentes, teniendo un modelo que funciona perfectamente con los datos de entrenamiento pero falla cuando se enfrenta a datos nuevos. Al multiplicar artificialmente el tamaño del dataset mediante transformaciones cuidadosamente diseñadas, el modelo se ve forzado a aprender características más robustas y generalizables, porque cada vez que ve un ejemplo, este es ligeramente diferente, lo que dificulta la memorización simple.
Las cuatro funciones de augmentation implementadas en el código simulan dichas variaciones que ocurren naturalmente en las señales EEG, son las siguientes:

- La función `add_noise()` añade ruido gaussiano aleatorio que emula las interferencias electromagnéticas inevitables durante la grabación de señales cerebrales, como las producidas por equipos cercanos o movimientos musculares mínimos del paciente, permitiendo que el modelo aprenda a extraer patrones relevantes incluso en presencia de estas perturbaciones.

- La función `scale_data()` modifica ligeramente la amplitud de las señales para simular variaciones individuales entre pacientes, diferencias en la impedancia de los electrodos o cambios en la conductividad del cuero cabelludo, haciendo que el modelo sea invariante a estas diferencias de escala que no deberían afectar el diagnóstico.

- La función `shift_data()` implementa desplazamientos temporales circulares que representan el hecho de que los eventos cerebrales relevantes pueden ocurrir en momentos ligeramente diferentes dentro de la ventana temporal analizada, enseñando al modelo a reconocer patrones independientemente de su posición exacta en el tiempo.

- Finalmente, `time_warp()` es la técnica más sofisticada, aplicando una deformación suave del eje temporal que simula variaciones naturales en la velocidad de los procesos cerebrales entre individuos o incluso en el mismo sujeto bajo diferentes condiciones de atención o fatiga, manteniendo la morfología general de la señal pero alterando sutilmente las duraciones relativas de sus componentes.

- La función `augment_data()`, que permite aplicar aleatoriamente y de manera repetida diferentes técnicas de aumento sobre secuencias.

2.2.5. Generación de secuencias mediante sliding window

Esta etapa, está dedicada a la generación de secuencias mediante el método de sliding window. A través de la `función create_sequences()` se construyen secuencias temporales fijas para cada sujeto, utilizando ventanas deslizantes con alto solapamiento. El proceso incluye el ordenamiento cronológico de las ventanas, el escalado robusto de las características mediante RobustScaler, la creación de secuencias que permanecen dentro del mismo sujeto para preservar la coherencia temporal y el almacenamiento organizado de las características, sus etiquetas correspondientes y la trazabilidad por individuo. El resultado inicial de esta etapa produce 1376 secuencias para entrenamiento y 342 para prueba.

2.2.6. Aplicación de la función data augmentation

Las secuencias del conjunto de entrenamiento se multiplican mediante las transformaciones previamente definidas, lo que incrementa significativamente el tamaño del dataset y mejora su variabilidad temporal. Tras este proceso, el conjunto aumentado alcanza un total de 5504 secuencias, con una distribución de clases ajustada representada por Counter(0: 1664, 2: 1656, 3: 1380, 1: 804).

2.2.7. Implementación del modelo Simple LSTM

Para la etapa de clasificación multiclase se diseñó un modelo denominado Simple LSTM, específicamente optimizado para trabajar con conjuntos de datos pequeños y secuencias temporales cortas, como las derivadas de los segmentos EEG utilizados en este estudio. El objetivo principal de esta arquitectura es proporcionar un equilibrio adecuado entre capacidad de representación temporal y control del sobreajuste, incorporando distintos mecanismos de regularización a lo largo de su estructura.
El modelo recibe como entrada secuencias de longitud fija `(seq_length)` compuestas por múltiples características `(n_features)` extraídas de las ventanas temporales del EEG. Sobre esta entrada, se aplica inicialmente una capa `GaussianNoise`, que introduce una cantidad reducida de ruido gaussiano durante el entrenamiento. Esta operación actúa como una forma de regularización implícita, favoreciendo que la red aprenda patrones robustos frente a variaciones menores en las señales originales y reduciendo la tendencia al sobreajuste, algo especialmente relevante en escenarios con pocas muestras.
La arquitectura incorpora una única capa LSTM con 32 unidades internas. Esta capa constituye el núcleo del modelo, ya que permite capturar dependencias temporales presentes dentro de cada secuencia mediante su memoria recurrente. Para reforzar la regularización, se emplean penalizaciones `L1 + L2` tanto en los pesos principales como en los pesos recurrentes, junto con un recurrent dropout del 30 %, lo cual ayuda a mitigar el sobreajuste al eliminar aleatoriamente conexiones internas durante el entrenamiento. La salida de esta capa resume la información temporal completa de la secuencia en un vector de características de dimensión reducida.
Posteriormente, se aplica `Batch Normalization`, técnica que estabiliza la distribución interna de activaciones y acelera la convergencia del entrenamiento. Esta normalización es seguida por una capa Dropout del 50 %, que elimina de manera aleatoria la mitad de las unidades, reforzando aún más la capacidad del modelo para generalizar. El vector resultante alimenta una pequeña capa densa intermedia de 16 neuronas con activación `ReLU`, que permite una transformación no lineal adicional antes de la clasificación final. Esta capa también incorpora regularización `L1+L2` para mantener un control estricto sobre la magnitud de los pesos. Seguidamente, se aplica una nueva capa Dropout, esta vez del 40 %, con la finalidad de incrementar la robustez del modelo frente a memorias espurias o ruido. Finalmente, el modelo concluye en una capa de salida densa con activación Softmax, cuyo número de neuronas corresponde al total de clases objetivo `(num_classes)`.

2.2.8. Creación y compilación del modelo

Tras instanciar la arquitectura, el modelo es compilado utilizando la función de pérdida *sparse categorical crossentropy*, apropiada para problemas de clasificación multiclase en los que las etiquetas se encuentran

codificadas como enteros. El optimizador empleado es Adam, al cual se le asigna una tasa de aprendizaje reducida (learning rate $= 5 \times 10^{-4}$). Como métrica principal de evaluación se incluye la exactitud (*accuracy*), que permite monitorear el rendimiento del modelo durante las diferentes etapas del entrenamiento. Finalmente, se genera un resumen estructural del modelo (`model.summary()`), el cual detalla las capas utilizadas, sus dimensiones, la cantidad de parámetros entrenables y no entrenables, y la organización general de la arquitectura (Figura 1).

Model: "Simple_LSTM"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 20, 128)	0
gaussian_noise (GaussianNoise)	(None, 20, 128)	0
lstm (LSTM)	(None, 32)	20,608
batch_normalization (BatchNormalization)	(None, 32)	128
dropout (Dropout)	(None, 32)	0
dense (Dense)	(None, 16)	528
dropout_1 (Dropout)	(None, 16)	0
dense_1 (Dense)	(None, 4)	68

Total params: 21,332 (83.33 KB)
Trainable params: 21,268 (83.08 KB)
Non-trainable params: 64 (256.00 B)

Figura 1: Resúmen de la arquitectura del modelo empleado

2.2.9. Configuración de callbacks y balanceo de clases

Para mejorar la estabilidad del entrenamiento y mitigar problemas asociados al sobreajuste y al desbalance de clases, se implementaron diversos mecanismos. En primer lugar, se incorporó el `callback EarlyStopping`, configurado para monitorear la métrica de `val_accuracy`. Este callback detiene el entrenamiento de forma anticipada cuando el modelo deja de mostrar mejoras en el conjunto de validación durante un número prolongado de épocas consecutivas (patience $= 25$). Además, se habilita la opción `restore_best_weights`, que asegura que los pesos finales del modelo correspondan a aquellos obtenidos en la época con mejor desempeño, evitando que el entrenamiento finalice en un estado subóptimo.
Como complemento, se incluyó el `callback ReduceLROnPlateau`, también supervisando `val_accuracy`. Este mecanismo reduce automáticamente la tasa de aprendizaje cuando el modelo deja de mejorar durante un cierto intervalo establecido (patience $= 10$). La reducción se realiza multiplicando el valor actual por un factor de 0.5, con un límite mínimo fijado en (1×10^{-6}). Esta estrategia permite que el optimizador Adam ajuste su ritmo de actualización de parámetros, facilitando la convergencia en fases avanzadas del entrenamiento y evitando oscilaciones en el espacio de parámetros.
Adicionalmente, se abordó el desbalance existente entre clases utilizando `class weights`, un mecanismo que asigna un peso mayor a las clases con menor representación en el conjunto de entrenamiento. Los pesos iniciales se calcularon mediante la función `compute_class_weight` bajo el esquema `balanced`, que asigna valores proporcionalmente inversos a la frecuencia de cada clase. Posteriormente, estos valores fueron amplificados mediante un factor adicional de 1.5, con el fin de incrementar la penalización por errores en clases minoritarias y favorecer un aprendizaje más equilibrado.

2.2.10. Proceso de entrenamiento del modelo

El entrenamiento se llevó a cabo mediante el método `model.fit`, empleando el 85 % de los datos de entrenamiento para el ajuste de parámetros y reservando el 15 % restante como subconjunto de validación `(validation_split = 0.15)`. El modelo fue entrenado durante un máximo de 150 épocas, utilizando un tamaño de lote de 32, seleccionado para proporcionar mayor estabilidad en la estimación de gradientes y un avance más uniforme del proceso de optimización. Se habilitó la opción `shuffle=True` con el fin de reordenar aleatoriamente las secuencias en cada época y evitar que el modelo aprenda patrones espurios relacionados con el orden del dataset. Adicionalmente, se crea un objeto `(history)` para almacenar la evolución de la pérdida y de la exactitud tanto en entrenamiento como en validación.

2.2.11. Visualización de las curvas de entrenamiento

Con el fin de analizar el comportamiento del modelo durante el proceso de optimización, se generaron dos gráficas fundamentales que permiten evaluar el aprendizaje y detectar posibles señales de sobreajuste o problemas de convergencia. La primera gráfica corresponde a la exactitud `(accuracy)` y la segunda gráfica presenta la evolución de la función de pérdida `(loss)`y se ejecutan tanto para el entrenamient, como la validación.

2.2.12. Evaluación del modelo a nivel de secuencias

Una vez finalizado el entrenamiento, se procedió a evaluar el desempeño del modelo utilizando el conjunto de prueba. Para ello, el modelo generó las probabilidades de pertenencia a cada clase mediante la función model.predict, y posteriormente se obtuvo la clase predicha seleccionando el valor con mayor probabilidad `(argmax)`.
Como métrica principal se empleó el `Balanced Accuracy`, una medida especialmente útil en escenarios con desbalance de clases, ya que calcula el promedio de las tasas de acierto por clase. Esto evita que las clases mayoritarias dominen la evaluación y proporciona una visión más equitativa del rendimiento.
Además, se generó la `matriz de confusión`, que permite visualizar de forma detallada el número de aciertos y desaciertos por clase. Para complementar esta información, también se presentó una versión normalizada por filas, expresada en porcentajes, lo que facilita la interpretación de la proporción de predicciones correctas e incorrectas para cada categoría diagnóstica.
Finalmente, se generó el informe de clasificación, que incluye métricas como `precision`, `recall` y `F1-score` para cada clase. Estas métricas permiten examinar de manera general el desempeño del modelo y evaluar su efectividad tanto en la detección de verdaderos positivos como en la capacidad para evitar falsos positivos y falsos negativos.

2.2.13. Evaluación por sujeto mediante votación mayoritaria

Se incorporó un análisis adicional orientado a medir el desempeño del modelo a nivel de sujeto, lo cual resulta especialmente relevante en aplicaciones clínicas. Dado que cada individuo posee múltiples secuencias derivadas de ventanas temporales de su señal EEG, se implementó un esquema de votación mayoritaria `(majority voting)` para obtener una única predicción final por sujeto.
En primer lugar, todas las secuencias pertenecientes a un mismo individuo fueron agrupadas, registrando tanto las predicciones generadas por el modelo como la etiqueta real correspondiente a cada sujeto. Posteriormente, para cada individuo se determinó la clase final seleccionando aquella que obtuvo la mayor cantidad de votos entre las predicciones de sus secuencias asociadas.
Una vez obtenidas las predicciones por sujeto, se calcularon métricas de rendimiento específicas para este nivel de análisis. Entre ellas, se incluyó la `accuracy` y el `Balanced Accuracy`. Adicionalmente, se estimó el índice Kappa de Cohen, una métrica que evalúa el grado de acuerdo entre las predicciones del modelo y las etiquetas reales. Así mismo, se generó la correspondiente matriz de confusión a nivel de sujeto y el reporte de clasificación.

2.2.14. Análisis de confianza de las predicciones

Como etapa final del proceso de evaluación, se llevó a cabo un análisis de confianza orientado a examinar no sólo qué clase predice el modelo para cada sujeto, sino también con qué nivel de seguridad lo hace. Dado que el modelo produce probabilidades asociadas a cada clase para cada secuencia, se agruparon todas las probabilidades generadas para las secuencias pertenecientes al mismo individuo. A partir de estas probabilidades agregadas, se determinó la clase predicha final como aquella con mayor probabilidad media. Asimismo, se extrajo el valor de la probabilidad más alta como una medida de confianza del modelo en su decisión. Para complementar este análisis, se comparó dicha predicción con la etiqueta real del sujeto, identificando explícitamente si la clasificación fue correcta o incorrecta.

2.3. Descripción de la estructura del notebook 03-modelo_CTR_DCL.ipynb

Este está orientado al desarrollo de un modelo de clasificación binaria para distinguir entre los grupos CTR y DCL. Su estructura es equivalente a la empleada en el modelo multiclase, pero adaptada específicamente a la reducción del problema a dos categorías. Al igual que en los notebooks anteriores, el flujo de trabajo abarca desde la preparación del conjunto de datos hasta la evaluación final del modelo por secuencia y por sujeto.El número de secuencias generadas mediante sliding window asciende a 561 para entrenamiento y 217 para prueba, lo cual confirma que el subconjunto binario mantiene el desafío de datos limitados. Por otra parte, el proceso de augmentación multiplica el número de secuencias hasta alcanzar 2244 ejemplos de entrenamiento, con una distribución balanceada post-augmentación de 1496 secuencias para la clase CTR y 748 para la clase DCL, lo cual permite al modelo entrenar con un conjunto más robusto y representativo.

3. Resultados y análisis

3.1. Clasificación Multiclase

Las gráficas de *accuracy* obtenidas durante el entrenamiento y la validación (Figura 2) muestran que el modelo inicia con un valor cercano al 43 % en entrenamiento y aumenta progresivamente hasta estabilizarse alrededor del 96 % después de la época 20. En contraste, la precisión de validación comienza en un valor más alto (aproximadamente 71 %) y alcanza de forma rápida niveles cercanos al 100 %, estabilizándose desde la época 10.
Por su parte, la curva de pérdida de entrenamiento desciende continuamente desde un valor inicial cercano a 2.0 hasta valores inferiores a 0.2. La pérdida de validación presenta un comportamiento similar, convergiendo a valores cercanos a 0.05 y manteniéndose incluso ligeramente por debajo de la pérdida de entrenamiento durante gran parte del proceso (Figura 3).

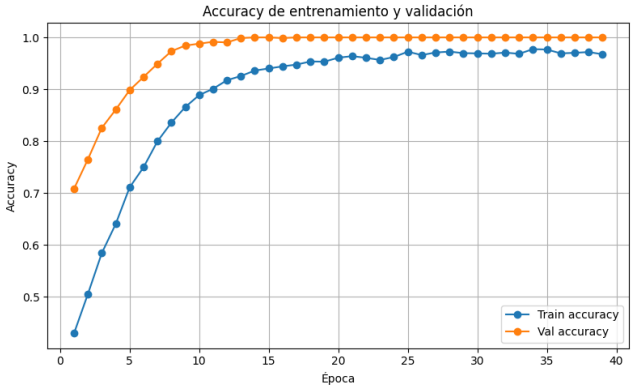

Figura 2: Accuracy de entrenamiento y validación

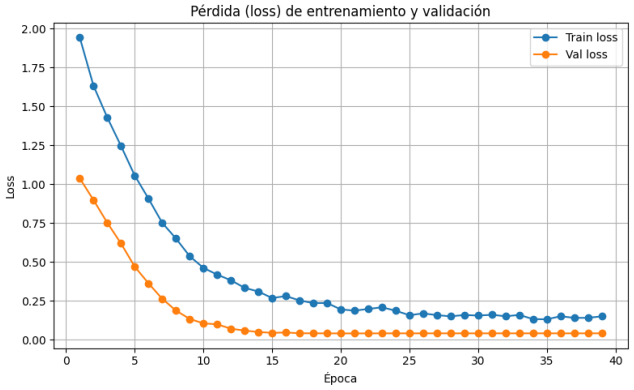

Figura 3: Pérdida de entrenamiento y validación

3.1.1. Evaluación por Secuencias

A nivel general, el modelo alcanzó un Accuracy de 0.56. Si bien este valor podría parecer moderado a primera vista, es importante contextualizar frente al azar estadístico, que en un problema de cuatro clases sería de 0.25. Esto confirma que la red neuronal efectivamente está logrando extraer patrones distintivos en las señales EEG, aunque todavía enfrenta dificultades para realizar una separación limpia entre todos los grupos, especialmente al tratar con los controles externos a la familia.

Se observó que el grupo GG actúa como una especie de «atractor» para el modelo. Aunque la capacidad de detección es alta (Recall del 0.71), la precisión cae notablemente a 0.44 (Cuadro 3). Esto indica que el modelo tiende a sobrepredicir esta clase. Al revisar la matriz de confusión (Cuadro 2), se notó que una gran parte de los sujetos CTR (40.4 %) y DCL (47.4 %) fueron clasificados erróneamente como portadores (Cuadro 2), lo que sugiere que las características de potencia o entropia en el EEG de los portadores podrían estar solapándose con las de los otros grupos.

Por otra parte, el grupo de no portadores pertenecientes a la familia (GU) tuvo un desempeño bastante equilibrado (63.7 %) (Cuadro 2). Lo interesante aquí es el patrón de error: su mayor confusión es con sus propios familiares portadores (GG, 27.5 %). Este resultado es biológicamente consistente y esperable, dado que ambos grupos comparten un fondo genético base y factores ambientales, lo que probablemente hace que sus firmas electrofisiológicas sean más similares entre sí que con los controles externos.

Así mismo, se observó que el grupo de controles sanos (CTR) presentó una precisión alta (0.90) (Cuadro 3), sin embargo, tiene un Recall muy bajo (34.6 %), dejando escapar a la mayoría de los controles reales y clasificándolos como GG. Por otro lado, el grupo DCL fue el más difícil de caracterizar, con un rendimiento cercano al azar (Recall 52 %) y una confusión masiva con el grupo GG, lo que indica que el modelo no logra distinguir con claridad las alteraciones cognitivas leves çomunes"de las provocadas por la mutación.

Cuadro 2: Matriz de confusión y matriz normalizada por fila

Matriz de confusión

	CTR	DCL	GG	GU
CTR	36	15	42	11
DCL	0	30	27	0
GG	1	10	72	18
GU		4	22	51

Matriz normalizada (% por fila)

	CTR	DCL	GG	GU
CTR	34.6 %	14.4 %	40.4 %	10.6 %
DCL	0.0 %	52.6 %	47.4 %	0.0 %
GG	1.0 %	9.9 %	71.3 %	17.8 %
GU	3.8 %	5.0 %	27.5 %	63.7 %

Cuadro 3: Reporte de clasificación

Clase	Precisión	Recall	F1
CTR	0.90	0.35	0.50
DCL	0.51	0.53	0.52
GG	0.44	0.71	0.55
GU	0.64	0.64	0.64
Accuracy		0.55	

3.1.2. Evaluación por Sujeto

Al agrupar las predicciones por sujeto mediante votación mayoritaria, se observó una ligera mejora en el rendimiento (Accuracy 0.56 vs 0.55 en secuencias). El índice Cohen's Kappa, alcanzó un valor 0.4086, que indica un nivel de acuerdo moderado, lo cual refuerza la idea de que el modelo identifica señales discriminativas útiles, aunque aún existe solapamiento entre categorías.

Por otra parte, el reporte de clasificación muestra que la clase CTR presenta buena precisión (1.00), pero su recall es bajo (0.38), lo que implica que el modelo es conservador: cuando predice CTR, suele acertar,

pero no detecta todos los sujetos CTR verdaderos. La clase DCL se comporta de manera equilibrada, aunque con valores moderados (precision y recall de 0.50). En cuanto a la clase GG, se observa un recall alto (0.71), lo que indica que la mayoría de los sujetos GG son identificados correctamente, aunque la precisión es menor (Cuadro 5). Respecto a la clase GU es la más balanceada, con valores de precisión, recall y F1 de 0.67 (Cuadro 5).Estos resultados sugieren que el modelo capta patrones relevantes a nivel de sujeto, pero persisten confusiones entre las clases GG–GU y CTR–GG (Cuadro 4).

Cuadro 4: Matriz de confusión

	CTR	**DCL**	**GG**	**GU**
CTR	3	1	3	1
DCL	0	2	2	0
GG	0	1	5	1
GU	0	0	2	4

Cuadro 5: Reporte de clasificación

Clase	Precisión	Recall	F1
CTR	1.00	0.38	0.55
DCL	0.50	0.50	0.50
GG	0.42	0.71	0.53
GU	0.67	0.67	0.67
Accuracy		0.56	

3.2. Clasificación binaria

Esta etapa, cosistió en clasificar entre Controles Sanos (CTR) y pacientes con Deterioro Cognitivo Leve (DCL), excluyendo por un momento a los sujetos de la familia E280A. Después de entrenar el modelo con las dos clases, se analizó que, al igual que en las gráficas obtenidas para la clasificación multiclase, el *accuracy* de entrenamiento y validación alcanza valores superiores a 0.90 después de aproximadamente 10 épocas (Figura 4). De manera consistente, la curva de pérdida disminuye progresivamente a medida que avanzan las épocas. Los valores de pérdida en la validación son ligeramente menores que los del entrenamiento y se estabilizan a partir de la época 8 (Figura 5).

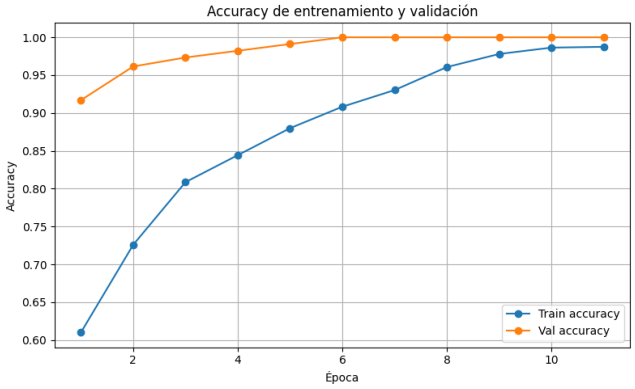

Figura 4: Accuracy de entrenamiento y validación

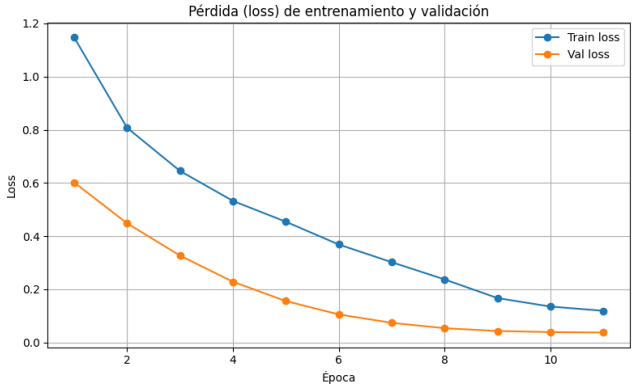

Figura 5: Pérdida de entrenamiento y validación

3.2.1. Evaluación por Secuencias

El modelo logró una Exactitud (Accuracy) por sujeto del 68.75 %, que contrastandolo con la literatura actual sobre EEG, el desempeño es bastante estándar para este tipo de tareas. Sin embargo, aunque el resultado es prometedor desde el punto de vista académico, todavía está lejos del umbral del 85 % que generalmente se exige para que una herramienta sea viable en un entorno clínico real. Por otro lado, al revisar los resultados detallados, se notó que el modelo mostró un sesgo claro hacia la clase mayoritaria (los controles sanos) (Cuadro 6). En la evaluación por secuencias, el grupo CTR obtuvo mejores métricas de Recall (68.5 %) y Precision (0.79) (Cuadro 7) que el grupo con deterioro.

Cuadro 6: Matriz de confusión (secuencias)

	CTR	**DCL**
CTR	100	46
DCL	27	44

Cuadro 7: Reporte de clasificación (secuencias)

Clase	**Precision**	**Recall**	**F1-score**
CTR	0.79	0.68	0.73
DCL	0.49	0.62	0.55
Accuracy		0.66	
Macro Avg	0.64	0.65	0.64
Weighted Avg	0.69	0.66	0.67

3.2.2. Evaluación por Sujeto (Votación Mayoritaria)

La estrategia de agrupar las predicciones por sujeto mediante votación mayoritaria volvió a ser útil, elevando la sensibilidad para detectar controles del 68.5 % al 72.7 %. No obstante, es importante considerar la fiabilidad de esta mejora. El índice Kappa de 0.31 indica un nivel de acuerdo moderado, pero también sugiere que esta métrica puede estar influenciada por el desbalance del conjunto de prueba (11 sujetos sanos frente a solo 5 con DCL) (Cuadro 8). Esto quiere decir que, aunque el accuracy aumentó (Cuadro 9), parte de ese incremento podría estar relacionado con la predominancia de la clase mayoritaria y no necesariamente con una separación completamente robusta entre sujetos sanos y con deterioro cognitivo.

Cuadro 8: Matriz de confusión (sujetos)

	CTR	**DCL**
CTR	8	3
DCL	2	3

Cuadro 9: Reporte de clasificación (sujetos)

Clase	**Precision**	**Recall**	**F1-score**
CTR	0.80	0.73	0.76
DCL	0.50	0.60	0.55
Accuracy		0.69	
Macro Avg	0.65	0.66	0.65
Weighted Avg	0.71	0.69	0.69