# Universidad Miguel Hernández de Elche

## MASTER UNIVERSITARIO EN ROBÓTICA


# “Estudio de técnicas de Deep Learning para la diferenciación de tareas mentales”

Trabajo de Fin de Máster

2019-2020

Autor: Carlos Carrazoni Pérez
Tutor/es: José María Azorín Poveda
Eduardo Iáñez Martínez

# RESUMEN

Este trabajo aplica técnicas de Deep Learning con el objetivo de diferenciar tareas mentales relacionadas con la marcha humana, a partir de registros de señales EEG. Dentro del proceso de un sistema BCI, se enfoca en la etapa de clasificación de señales para reducirlas a comandos o tareas sencillas. Tras un estudio de dichos sistemas BCI a modo de introducción, se plantean los experimentos realizados. Se implementan dos arquitecturas neuronales convolucionales basadas en Python. A partir de ellas, se estudian los resultados de clasificación obtenidos con diferentes métodos de entrenamiento, variando parámetros y simulando la aplicación de los modelos. Los experimentos fueron realizados con dos conjuntos de datos: un dataset público y otro de nuestro propio laboratorio. Además, se analiza el efecto de la herramienta CUDA en las variables temporales. Finalmente, se ofrece una discusión y la configuración que mejor clasificó las clases de información nunca antes vista.

# ABSTRACT

This work applies Deep Learning techniques in order to differentiate mental tasks related to human gait, from EEG signal records. Within the process of a BCI system, it focuses on the signal classification stage to reduce them to simple commands or tasks. After a study of these BCI systems as an introduction, the experiments carried out are explained. Two convolutional neural architectures based on Python are implemented. From them, the classification results obtained with different training methods are studied, varying parameters and simulating the application of the models. The experiments were carried out with two data sets: a public dataset and another from our own laboratory. In addition, the effect of the CUDA tool on temporal variables is analyzed. Finally, a discussion is offered and the configuration that best classified the classes of information never seen before.

# AGRADECIMIENTOS

A los profesores que decidieron volver a ser mis tutores para un trabajo. Por ofrecerme la oportunidad no solo de trabajar sobre Inteligencia Artificial, sino por abrirme las puertas a aplicarla en un campo tan bonito. Y, quizás, tan adecuado para mí. Por responder a mis dudas y tener paciencia conmigo. Por haberme ayudado a crecer intelectualmente igual que muchos otros profesores y profesoras a los que me gustaría agradecer también. A don Jaime, don Antonio, Gaspar, Isabel, Luis y un largo etc.

A mis padres por apoyarme tanto activa como pasivamente, con su amor y sustento. Por ser las personas que más han tenido que vivir conmigo y que, por tanto, peores momentos han vivido conmigo. Por hacer que hoy sea quien soy. Porque el dinero no fuese nunca un problema, porque mi vista nunca fuera una barrera presente, por llevarme en coche a donde necesitase, por ocuparse en las sombras de lo que necesitaba sin saberlo, por permitirme soñar más de lo debido…

A mis amigos por hacer de mi realidad algo más fantástico. A Pilar, por ser mi confidente, mi espejo, mis risas en los momentos más inadecuados, mi querida persona favorita. A Carlos, por haber estado a punto de no querer conocerte y ahora querer saltar la distancia para verte. A Julián, por haber viajado conmigo a realidades distintas estos últimos años. A Jonás, por ser mi compañero de pasiones. A Miguel, a Emilio, a Manuela, a Javier, a Adrián…

A Carina, Sobre todo a ti Carina. Sin ti, muy seguramente, esto no existiría.

# ÍNDICE

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1. INTRODUCCIÓN

Los daños neurológicos en humanos como los accidentes cerebrovasculares, las lesiones de medula espinal o el debilitamiento de las articulaciones debido a la edad, limitan considerablemente la vida cotidiana de las personas que padecen estas dolencias. A finales de 2013, la Organización Mundial de la Salud (OMS) cuantificó que cada año de 250.000 a 500.000 personas sufren una lesión medular [1], con picos en edades menores de 30 años (accidentes de tráfico o violencia) y para mayores de 70 años (caídas). Este último caso surge de los avances médicos que buscan ampliar la esperanza de vida y se verá incrementado (se espera que para 2050 la población que supere los 60 años será de 10 millones en 33 países [2]). Entonces, sería lógico que estos avances centrarán una especial atención en la rehabilitación total o parcial de los afectados.

Es por este motivo que, desde hace varios años, ha aumentado el interés de desarrollar dispositivos que se puedan adaptar a la fisionomía humana para ayudar a la rehabilitación o sustituir extremidades con el fin de restituir las capacidades motoras perdidas. Estos ingenios mecánicos asisten o reemplazan a los terapeutas en las tareas de rehabilitación repetitivas, permitiendo mejorar los tratamientos de rehabilitación motora [3]. Se conocen como robots vestibles por basar su diseño en las formas y funciones del cuerpo humano, de manera que el usuario pueda "cargar" con él con facilidad. La característica diferenciadora de este tipo de robots es la interacción que se establece entre este y el humano al que acompaña. Hay diversas formas de medir esta interacción en la que se basa el control del robot. En este estudio nos hemos centrado en la captación de la intención cognitiva, la cual convierte esta interacción en una comunicación. Con ella, el conjunto humano-robot pasa a ser un sistema denominado Interfaz Cerebro-Máquina (BCI por sus siglas en inglés). Este consiste en un proceso de extracción de información para su posterior tratamiento (si es necesario) y su reconocimiento por medio de un algoritmo de clasificación. Este algoritmo transforma la información en órdenes finitas que enviar a los actuadores del mecanismo.

El correcto funcionamiento de todas las etapas que comprende un BCI es la clave de la seguridad ética del paciente y del éxito en la tarea del robot. Es por ello que en este trabajo abordamos la fase de extracción de características y clasificación de las señales captadas. Este procesamiento será realizado sobre señales EEG y mediante una Red Neuronal Convolucional (CNN) debido a su creciente aplicación en este ámbito [4].

# 2. ESTADO DEL ARTE

En esta sección se pretende poner la situación actual del estudio de los sistemas BCI, la neurofisiología, ambos como una breve introducción en primer lugar, y el Deep Learning, en segundo lugar, más en profundidad. Concretamente, los siguientes apartados responden a las siguientes preguntas:

1) ¿Cómo captaremos la intención cognitiva?
2) ¿Qué tareas mentales existen y cuáles utilizaremos?
3) ¿Qué formulación utilizaremos para introducir las señales como input?
4) ¿Qué arquitectura de Deep Learning utilizaremos?

El objetivo de esto será entender las diferentes posibilidades que disponemos en ambos campos para poder justificar las elecciones tomadas. Además, se introducirá el concepto de las Interfaces Cerebro-Máquina para entender donde se encuadran los puntos principales de este trabajo.

## 2.1. Interfaces Cerebro-Máquina

Podemos definir, entonces, una interfaz cerebro-máquina (BCI) como un sistema capaz de traducir los patrones de actividad biomédicos de un sujeto en mensajes o comandos para una aplicación concreta [5]. Esta actividad biomédica será medida, por ejemplo, usando electroencefalografía (EEG), y procesada por el propio sistema. Los sistemas BCI tienen muchas aplicaciones existentes y potenciales tanto para la vida cotidiana como para personas con algún tipo de discapacidad. Con esta tecnología es posible que un usuario mueva un cursor a la izquierda o a la derecha sin ninguna acción física, sino simplemente imaginando tareas mentales [6].

El proceso que sigue un sistema BCI comprende las siguientes etapas [7]:

- **Adquisición**: las señales son captadas gracias a un equipamiento específico produciéndose una transformación analógica-digital a través de los electrodos para su posterior procesamiento.
- **Preprocesamiento**: las señales pueden ser preprocesadas mediante el uso de filtros espaciales o espectrales. El tratamiento mejorará la calidad de las mismas para remarcar y filtrar las características más importantes de la actividad neurofisiológica de interés.
- **Extracción de características**: se extraen las características más representativas de las señales preprocesadas.

- **Clasificación**: un algoritmo llamado clasificador encuentra los patrones de la señal en el momento temporal que se realizan las tareas para crear un modelo que sea capaz de reconocer futuras señales sin la información de la tarea dada.
- **Aplicación**: la tarea reconocida por el clasificador es emitida como un comando para activar un dispositivo, que responderá realizando las acciones para las que fue diseñado.

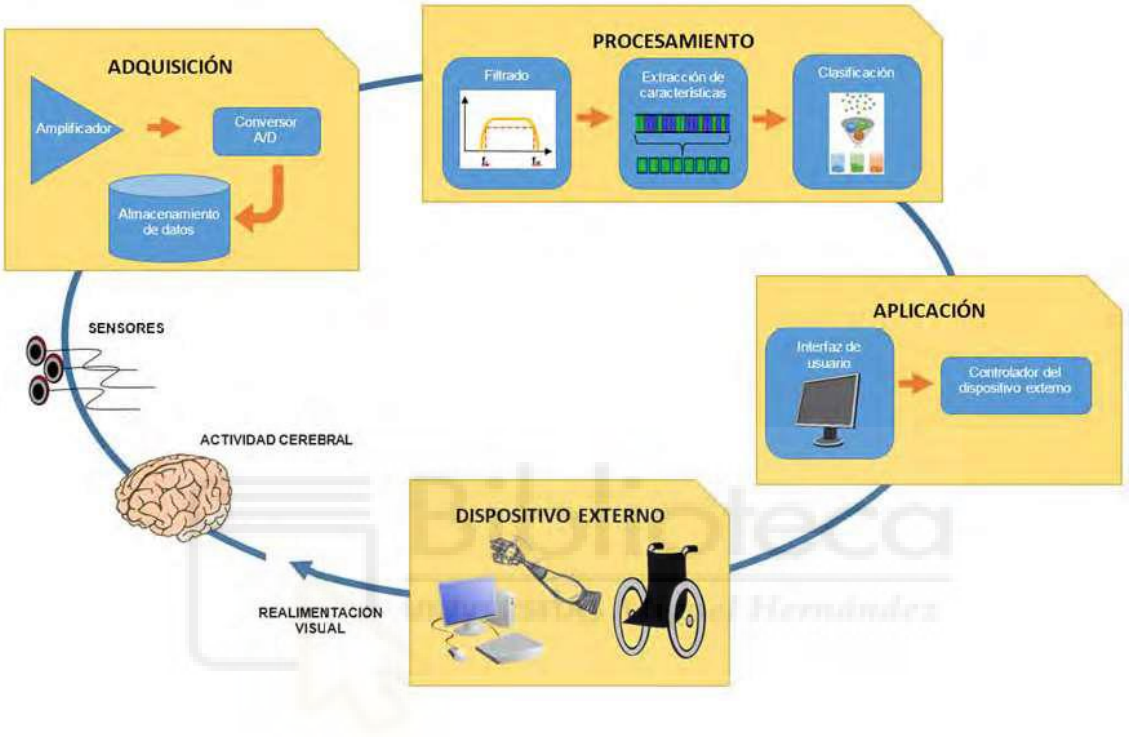


**Figura 1**. Etapas de un sistema BCI
Fuente: [7]

Para la creación de un BCI distinguimos dos fases típicas: la offline, un entrenamiento durante el cual el sistema es calibrado, y la online, donde ya es capaz de reconocer patrones de actividad cerebral y traducirlos en comandos.

En la etapa offline se calibra el algoritmo de clasificación, eligiendo desde sus parámetros más óptimos hasta escogiendo las características más útiles de múltiples canales. Con este fin, se deben recoger conjuntos de datos que contengan las señales registradas del usuario que realizó las tareas mentales de interés varias veces.

Por su parte, el proceso de un BCI online es un sistema en lazo cerrado que parte del usuario produciendo el patrón especifico que será medido. Tras su debida adquisición y preprocesamiento, las distintas muestras EEG serán clasificadas en tiempo real en las etiquetas establecidas para poder ejecutar los comandos a los que estén ligadas. Al mismo

tiempo, el usuario será realimentado mediante la visualización de los resultados obtenidos por la máquina.

## 2.2. Señales neurofisiológicas

La medida de señales procedentes del cuerpo humano permite estimar la intención motora sin pérdida o retraso alguno en comparación con otros métodos. Más formalmente, la electrofisiología cognitiva es el estudio de cómo las funciones cognitivas (percepción, lenguaje, memoria, emociones, control del comportamiento, etc.) son llevadas a cabo por la actividad eléctrica producida por grupos de neuronas [8].

Si atendemos al método utilizado para la adquisición de las señales cerebrales, los sistemas se pueden clasificar en dos grandes grupos: invasivos y no invasivos. Las primeras consisten en la implantación cerebral de colecciones de microelectrodos, los cuales proveen de una gran cantidad de información, sin embargo, presentan múltiples desventajas como: reacciones biológicas adversas, procedimientos de implantación complejos y la necesidad de algoritmos que extraigan información relevante. Es por todo esto que los métodos no invasivos, basados en los potenciales de campo, son los más comunes en la práctica. Pudiendo destacar entre ellos las señales EMG y las EEG como sus entradas (o *inputs*) más utilizadas.

Los métodos no invasivos ofrecen una excelente resolución temporal y buena cobertura espacial. Las señales electrofisiológicas abren una oportunidad para medir las oscilaciones neuronales y evaluar directamente las rápidas fluctuaciones en la coherencia neuronal que subyace en el interior de las comunicaciones del cerebro [9]. Más específicamente, electrocorticografía (ECoG), magnetoencefalografía y electroencefalografía (EEG) se encuentran en una posición única para detectar las escalas temporales más cortas de la conectividad funcional, con las medidas EEG siendo particularmente capaces de cubrir gran parte de la corteza cerebral.

Podemos entonces justificar nuestro interés en el uso de señales EEG por varias razones. La primera es que estas señales capturan las dinámicas cognitivas en el cuadro temporal en el que ocurren. Las funciones cognitivas son rápidas (muchos procesos ocurren de 10 hasta 100ms). Las técnicas con alta resolución temporal son adecuadas para capturar estos eventos rápidos, dinámicos y temporalmente secuenciales. En comparación, la precisión temporal de otras técnicas como la respuesta hemodinámica es 2-3 veces más lenta que la respuesta electroencefalográfica. La segunda razón es que las

señales EEG miden directamente la actividad neuronal. La fluctuación del voltaje medido son reflejos directos de fenómenos biofísicos al nivel de poblaciones neuronales (es más, las oscilaciones neuronales en el córtex se reflejan directamente en oscilaciones de la señal EEG). La tercera es que la señal EEG es multidimensional. Los datos EEG comprenden cuatro dimensiones: tiempo, espacio, frecuencia y potencia (la fuerza de la actividad de la banda de frecuencia concreta). Esta multidimensionalidad provee de muchas posibilidades para especificar y probar hipótesis. Esto, además, permite enlazar los descubrimientos obtenidos mediante métodos no invasivos en invasivos. En la **Figura 2** se puede observar un canal de señal EEG típica.

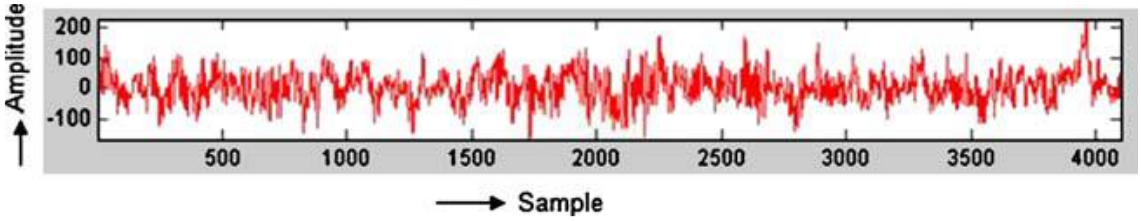


**Figura 2**. Señal EEG típica en el dominio del tiempo
Evolución de la amplitud (eje vertical, en $\mu V$) de una señal EEG a lo largo del tiempo (eje horizontal)
Fuente: [10]

Pese a todo esto, también hay que tener en cuenta que este tipo de señales contienen artefactos (valores no deseados). Dado que se valen de electrodos colocados sobre el cuero cabelludo para extraer la información de interés se considera un método simple, pero que conlleva bajos ratios señal-ruido (las fuentes que producen información no relevante para la tarea a menudo afectan más fuertemente a la señal EEG que las fuentes relevantes). Debido a la distancia a la que se encuentran del emisor, los electrodos captan señales electrofisiológicas indeseadas, como las señales electromiográficas (EMG) procedentes de los parpadeos y los músculos del cuello. También existe preocupación por los artefactos ocurridos por los movimientos de los cables y el desplazamiento de los electrodos cuando el sujeto se mueve.

Son muchos los estudios que se han enfocado en detectar estos artefactos. Trabajos que emplean métodos establecidos como el filtro de Kalman [11] o el Análisis de Componentes Independientes (ICA) [12], y enfoques más modernos como la utilización de cámaras que capten parpadeos [13] o el uso de Máquinas Vector Soporte (SVM) para complementar la aplicación de ICA [14].

## 2.3. Deep Learning

Los sistemas de Machine Learning son utilizados para identificar objetos en imágenes, transcribir el habla en texto, relacionar noticias, publicaciones o productos con los intereses de los usuarios, y seleccionar resultados relevantes de la búsqueda. Cada vez más, estas aplicaciones hacen uso de una clase de técnicas denominadas Deep Learning [15].

Estas técnicas convencionales estaban limitadas por su capacidad de trabajar con información natural sin procesar. Durante décadas, la construcción de un sistema de reconocimiento de patrones o aprendizaje automático requirió una ingeniería cuidadosa y una considerable experiencia en el dominio para diseñar un extractor de características que transformara los datos en bruto (como los valores de píxeles de una imagen) en una representación interna adecuada o un vector de características desde el cual el subsistema de aprendizaje, a menudo un clasificador, podría detectar o clasificar patrones en la entrada. Por esta razón, surgió el aprendizaje de características, un conjunto de métodos que permiten a una máquina ser alimentado con información sin procesar y hallar automáticamente las representaciones necesarias para la detección o clasificación.

Las diferentes arquitecturas de Deep Learning son uno de estos métodos con múltiples niveles de representación. Obtenidos de la composición de módulos simples, pero no lineales (neuronas) los cuales transforman la representación de la información de un nivel (desde los datos originales) a un mayor y algo más abstracto nivel. Con el número suficiente de estas transformaciones combinadas (capas de neuronas y conexiones entre estas) se pueden crear funciones muy complejas. Para tareas de clasificación, un mayor número de capas amplifican aspectos de la información que son importantes para la discriminación y eliminan variaciones innecesarias.

De esta forma, la aplicación de Deep Learning está consiguiendo grandes avances en la resolución de problemas que se han resistido a los mejores intentos de la comunidad de Inteligencia Artificial durante muchos años. Ha resultado ser muy bueno para descubrir estructuras complejas en datos de alta dimensión y, por lo tanto, es aplicable a muchos dominios de la ciencia, los negocios o el gobierno. Por estas razones y todos los éxitos conseguidos por el Deep Learning actualmente, hemos considerado interesante usar uno de estos métodos para el reconocimiento de tareas mentales.

### 2.3.1. Deep Learning y la tecnología BCI

Como hemos visto, son muchas las posibilidades dentro de las señales neurofisiológicas, por no hablar de las diferentes arquitecturas neuronales que comprende el Deep Learning. Por esta razón, actualmente, existen muchos estudios con sistemas BCI muy diversos.

Dentro de la aplicación para escritura, se ha diseñado un sistema monocanal mediante señales EEG [16]. En este estudio, se emplearon los potenciales evocados visuales en el dominio de la frecuencia para controlar un teclado virtual de 58 caracteres. Con este fin, se captaron las señales mediante un solo electrodo seco (bipolar O1-Oz) para registrar las señales desde el córtex visual. El proceso de clasificación de 5 clases diferentes fue llevado a cabo por una CNN unidimensional. Tras la fase offline para optimizar el sistema y entrenar al modelo, se realizó un experimento online donde los sujetos deletreaban la palabra "SPELLER" a través de 5 opciones en el menú visual. En la **Figura 3** se muestra la disposición experimental. Con este sistema se llegó a obtener un 97,37% de exactitud en la etapa online.

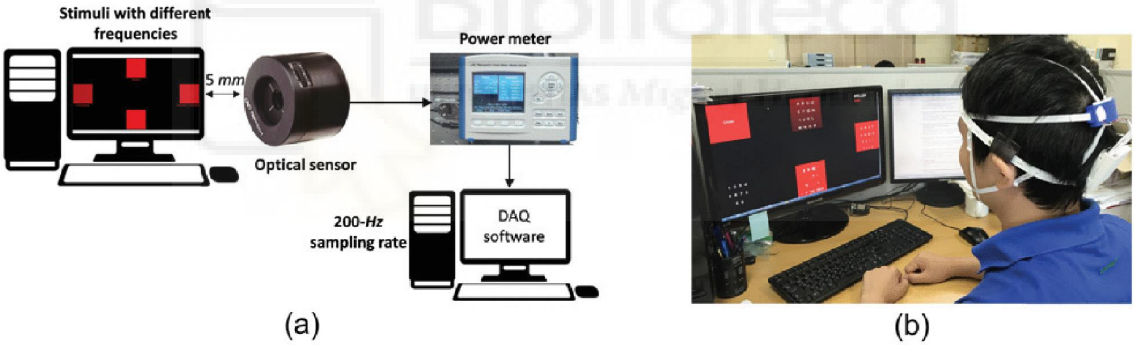


**Figura 3**. Implementación experimental de un teclado virtual controlado por señales EEG
a) medida del potencial óptico inducido por estímulos b) proceso experimental
Fuente: [16]

Para el campo de la movilidad física encontramos una silla de ruedas actuada mediante señales EEG para personas incapaces de controlar partes de su cuerpo [17]. En esta ocasión un casco provisto de 16 electrodos recoge la señal EEG y la envía a una computadora Raspberry PI. Para controlar la silla de ruedas se utilizan 4 comandos de acción (derecha, izquierda, adelante y stop), siendo reconocidas por un modelo CNN. Las pruebas para 10 sujetos consiguieron una ratio de acierto del 96%.

En un sentido más clínico, han surgido métodos para detectar y prevenir las convulsiones fruto de la epilepsia [18]. Este estudio demostró la gran utilidad en el manejo y procesamiento eficientes del Big Data para la gestión de enfermedades complejas. En

él se utilizaron métodos de Deep Learning de aprendizaje no supervisado para la extracción de características. Este sistema BCI basado en la nube supuso la oportunidad de intervenir en el proceso epileptogénico, permitiendo prevenir las convulsiones, beneficiando la vida del paciente.

### 2.3.2. Deep Learning aplicado a Intención Motora

El análisis EEG ha sido una importante herramienta con aplicaciones en neurociencia, ingeniería neuronal e, incluso, en el ámbito comercial. De forma simple, un dataset EEG consiste en una matriz de valores reales comprendidos en dos dimensiones (tiempo y canales) que representa los potenciales generados por el cerebro registrados en el cuero cabelludo asociados con condiciones específicas de las tareas de interés. Esta forma altamente estructurada hace que la información EEG sea adecuada para el Machine Learning. Por ello, muchos estudios enfocados en estas señales han usado algoritmos tradicionales de machine learning y reconocimiento de patrones para determinar información importante para la clasificación neuronal y la neuroimagen.

La disponibilidad actual de gran cantidad de estos datasets EEG y los avances en el machine learning ha permitido el desarrollo de arquitecturas de Deep Learning, especialmente en el análisis de las señales EEG y en la comprensión de la información que puede contener para la funcionalidad del cerebro. La robusta clasificación automática de estas señales es un paso importante hacia convertir las señales EEG en más prácticas para muchas funciones y menos dependientes de profesionales especializados. Esto última surge del hecho que las redes neuronales optimizan sus parámetros automáticamente en cada instante del entrenamiento, conduciendo a la idea de menor necesidad de conocimiento experto. Esta ventaja permitió la adaptación temprana en el ámbito de la imagen médica la cual, normalmente, requiere de amplios datasets difíciles de ser interpretados, incluso por expertos.

Tras seleccionar las señales EEG, para poder responder al resto de preguntas anteriormente planteadas se recurrió al estudio planteado por Craik *et al*. [4]. Este trabajo es una revisión de investigaciones realizadas sobre señales EEG en los últimos 10 años. Distinguir siete tipos de tareas mentales: reconocimiento emocional, intención motora, carga de trabajo mental, detección de convulsión epiléptica, potenciales relacionados a eventos, nivel de sueño y otros (clasificación de Alzheimer, del patrón de marcha, de género, detección de EEG anormal, etc.). De entre los 90 artículos presentados, el 22% cayó en el grupo de intención motora. Se decidió enfocar nuestro estudio en la

imaginación motora de miembro inferior con fines de extender el software de procesamiento a un sistema de exoesqueleto. Entendemos imaginación motora como tareas que envuelven que el usuario imagine ciertos movimientos musculares de las extremidades y/o la lengua.

Por otro lado, como ya se ha explicado, las señales EEG son inherentemente ruidosas, por lo que identificar y eliminar los artefactos que acompañan a las señales EEG ha sido ampliamente estudiado. Podríamos diferenciar tres formas de afrontar el problema: eliminación manual, eliminación automática y sin postprocesamiento. Ciertamente, es fácil identificar visualmente valores abruptos atípicos, por ejemplo, cuando se pierde información o cuando existen intensos artefactos EMG). Sin embargo, es difícil identificar ruidos persistentes en grabaciones temporalmente largas o ruido en sistemas con múltiples canales. Lo más importante es que el procesamiento manual es muy subjetivo, dificultando la reproducción de los procedimientos seguidos. Por su parte, podríamos destacar como ejemplos para la eliminación automática: los algoritmos de Análisis de Componentes Independientes (ICA) y la Transformada Discreta de Ondícula (DWT), o los filtros temporales y espaciales. En nuestro caso, la Red Convolucional utilizada preprocesa los datasets antes de emplearlos para crear el modelo de Deep Learning. La tipología de estos filtros y sus parámetros serán discutidos en la sección de MATERIALES Y MÉTODOS.

Ahora bien, pese a que los filtros espaciales son capaces de eliminar las perturbaciones entre electrodos, separar las señales no es una tarea trivial. Esto se debe a las características de la conducción del volumen anisotrópico en los tejidos del cerebro, cráneo, cuero cabelludo y cabello. Por ello, uno de los desafíos en el análisis de señales EEG es como formularlas para ser inputs. Se conocen tres categorías: cálculo de las características, imágenes y valores de la señal. La decisión de qué forma utilizar depende fuertemente del tipo de tarea y de la arquitectura de red a utilizar. Las señales EEG son comúnmente analizadas en el dominio de la frecuencia debido a que estas están normalmente asociadas con patrones de comportamiento. La Densidad Espectral de Potencial (PSD), la descomposición de ondícula y las medidas estadísticas de la señal (media, desviación estándar) son las tres formulaciones más comúnmente utilizadas.

Muchas redes neuronales, especialmente las CNN, usan como input espectrogramas generados de la información EEG. Los espectrogramas son tradicionalmente utilizados como una herramienta de postprocesado para visualizar el conjunto de datos. Sin

embargo, la capacidad sin precedentes de esta tipología de red para aprender de imágenes ha permitido utilizar a los espectrogramas como entrada de los clasificadores. Otras formulaciones de imagen incluyen crear mapas de características de Fourier y diseñar rejillas 2D o 3D.

Por su parte, la fuerza de la señal en el dominio del tiempo también se usa directamente como entrada. Tradicionalmente, este enfoque generalmente se asocia con características particulares del dominio del tiempo diseñadas a mano, como las características PSD. Las redes neuronales prometen aprender automáticamente características complicadas de grandes cantidades de datos, suscitando la idea de aprendizaje de extremo a extremo (esto es, aprender a partir de las señales sin ninguna selección de características a priori). Esta filosofía es la que motiva a alimentar las redes con señales sin preprocesar, lo cual podría contribuir a la práctica de análisis directo de señales EEG.

Para el caso del tipo de arquitecturas, el estudio mostró que, para todas las tareas mentales revisadas, las CNN son el marco de diseño prevalente, seguidas por las Deep Belief Netwerk (DBN). Las primeras implican alternar capas de convolución con capas de *pooling* (típicamente de *max pooling*). Sus principales características de diseño se centran en la cantidad de capas convolucionales y del tipo de clasificador final. La amplia mayoría de estas arquitecturas usan funciones ReLU (Rectified Linear Unit) como funciones de activación para sus capas convolucionales. Para las capas totalmente conectadas empleadas para el clasificador final utilizan funciones de activación SoftMax. Pese a que para la tarea de imaginación motora no se mostró ninguna preferencia en cuanto arquitectura, la decisión de utilizar CNNs se explica por dos motivos. El primero es que el uso de este tipo de red se verá beneficiado por los avances y mejoras que se consigan en otros campos como la visión por computador. El segundo es por las diferentes posibilidades de formulación de inputs que permiten.

Las CNN que utilizaron imágenes como entradas obtuvieron la misma exactitud media que las que utilizaron las características (84%). Mientras tanto, las CNN que emplearon los valores de la señal obtuvieron un 87% de exactitud. Esto va en contra de la intuición que nos dice que el mayor esfuerzo aplicado en las etapas de preprocesamiento, mejor serán los resultados de la clasificación. Estos valores hay que tomarlos con precaución, puesto que no se pueden comparar estudios que analizan diferentes tareas, sujetos y procedimientos. Sin embargo, podríamos utilizar estos

resultados y el hecho de que este menor procesamiento podría ayudar a reducir los tiempos de cálculo, para justificar trabajar con los valores de la señal.

### 2.3.3. Redes Neuronales Convolucionales

Las CNN son un tipo de red neuronal artificial que pueden aprender patrones locales en la información mediante la operación de convolución como su componente clave. Los distintos modelos varían en el número de capas convolucionales, que van desde arquitecturas superficiales con una sola capa convolucional, CNN profundas con múltiples capas consecutivas y arquitecturas muy profundas con hasta 1000 capas. Las CNN profundas pueden extraer, en primer lugar, características de bajo nivel a partir de la señal y, luego, características cada vez más globales y de mayor nivel en capas más profundas. Por ejemplo, a partir de imágenes, las CNN son capaces de aprender a detectar características visuales cada vez más complejas (bordes, formas simples, objetos completos).

Una propiedad atractiva de las CNN que ha sido aprovechada en muchas aplicaciones anteriores es que son adecuadas para el aprendizaje de extremo a extremo. Este tipo de aprendizaje puede ser interesante para la decodificación de señales cerebrales, ya que no todas las características relevantes pueden ser asumidas como conocidas a priori.

La señal EEG tiene característica que la hace diferente de las imágenes, las entradas con las que las CNN han tenido más éxito. En contraste con las imágenes estáticas de dos dimensiones, las señales EEG es una serie temporal dinámica proveniente de las medidas de electrodos obtenidas en el cuero cabelludo tridimensional. Esto, unido a su bajo ratio señal-ruido, puede hacer mucho más difícil el aprendizaje de extremo a extremo que para las imágenes. Por lo tanto, es necesario tener en cuenta la necesidad de modificar las arquitecturas CNN adoptadas de la visión por computador para que trabajen con señales EEG.

## 2.4. CUDA

Las GPU (Unidad de Procesamiento Gráfico) programables han evolucionado a unos procesadores *manycore*, altamente paralelos y multihilo [19]. La GPU sobresale en cargas de trabajo de datos paralelos que constan de miles de subprocesos que ejecutan otros hilos de programas para el sombreado de pixeles, vértices y geometrías. Este tremendo rendimiento de las GPUs modernas ha llevado a explorar el mapeo de cálculos no gráficos en ellas. Estos sistemas GPU de propósito general (GPGPU) han producido algunos

resultados impresionantes, pero las limitaciones y dificultades para conseguir esto mediante APIs gráficas son altas. Con el objetivo de usar una GPU como un dispositivo computacional paralelo más genérico, NVIDIA desarrolló una nueva arquitectura GPU unificando los gráficos y la computación, y el modelo de programación CUDA.

Mediante el escalado del número de procesadores y particiones de memoria, su matriz de procesadores multihilo masivo se convierte en una eficiente plataforma unificada tanto para gráficos como para computación paralela para aplicaciones de propósito general. En la **Figura 4** se puede observar una comparación esquemática de los componentes de una CPU (Unidad Central de Proceso) y una GPU. En ella se aprecian la gran cantidad de unidades *core* encargadas de los cálculos matemáticos. La plataforma CUDA es una capa software que da acceso directo al conjunto de instrucciones virtuales y a los componentes paralelos de la GPU para la ejecución de los kernel computacionales. Fue diseñada para funcionar con lenguajes de programación como C, Fortran o Python y permite una mejor accesibilidad que antiguas APIs como OpenACC o OpenCL, las cuales requerían de una mayor habilidad en programación gráfica.

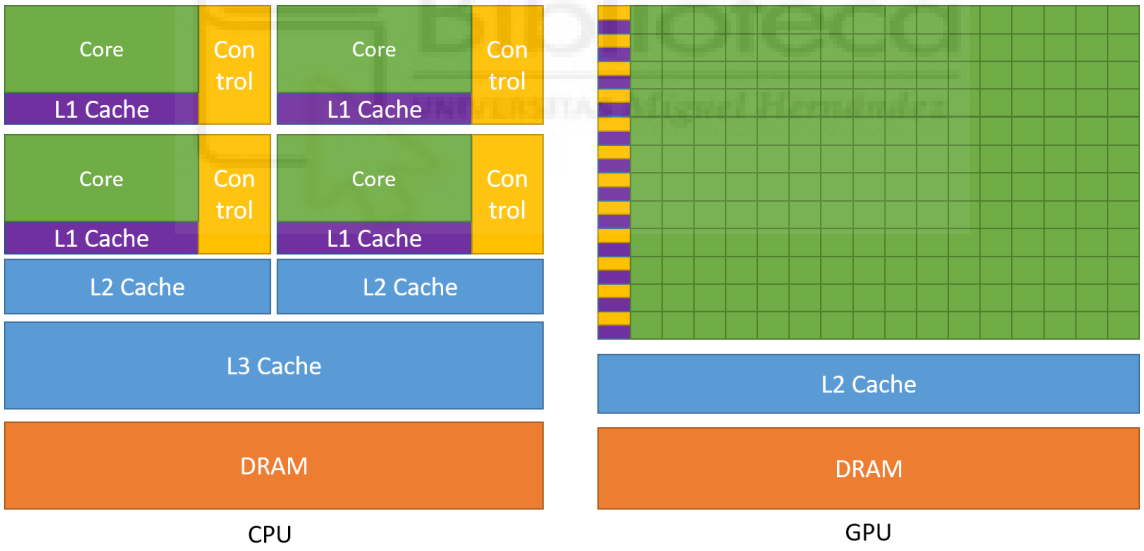


**Figura 4**. Comparación esquemática componente CPU vs GPU
Fuente: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

# 3. OBJETIVO

El presente trabajo pretende llevar un paso más allá lo conseguido por R. T. Schirrmeister.et al. en [20]. Este estudio buscaba hallar la influencia en la exactitud de los resultados de las elecciones de diseño (la configuración de la CNN en general y otros aspectos como el tipo de no-linealidad usada) y las estrategias de entrenamiento (entrenar introduciendo pruebas completas o dividiéndolas). Para ello diseñaron 3 modelos diferentes de CNN: una *Shallow CNN* (2 capas), una *Deep CNN* (5 capas) y una *Res CNN* (una red residual de 31 capas). Además de una combinación de las dos primeras. Todas ellas fueron adaptadas para los requerimientos específicos que plantea el análisis multicanal de las señales EEG y sus resultados fueron comparados con un método convencional en el campo como es el Banco de Filtros de Patrones Espaciales Comunes (FBCSP).

Las decisiones de diseño en este estudio se justificaron a través de los resultados obtenidos con el *BNCI Competition Dataset 2a*, un conjunto público de señales EEG recogidas por diversos estudios de imaginación motora. Con los parámetros de diseño establecidos, el objetivo actual es adaptar el modelo de clasificación a los datasets propios del laboratorio. Para ello, el desarrollo temporal del trabajo comenzó con la configuración tanto software como hardware para el correcto funcionamiento de los modelos CNN codificados en lenguaje Python. Tras comprobar el correcto funcionamiento del sistema, se pasó a adaptar los códigos para que pudieran trabajar con dichos datasets. A continuación, se investigó acerca de los parámetros que mejoraban la asimilación de los datasets en el entrenamiento del modelo y su reconocimiento de nuevos datos de entrada. Para esto último se codificó una simulación de experimento de recolección de señales EEG. Durante el proceso, se estudió las variables temporales que afectan al modelo y se probó con una herramienta de aceleración mediante computación paralela. Finalmente, se estudiaron los resultados para obtener una configuración óptima para la predicción.

# 4. MATERIALES Y MÉTODOS

En la siguiente sección se detallan los aspectos teóricos anteriormente expuestos. Comienza con una explicación de en qué consisten los datasets EEG utilizados y su obtención. Seguidamente se detalla cómo han sido formateados estos datos para que sean utilizados como entradas al modelo. A continuación, se expone el funcionamiento y la configuración de las dos arquitecturas CNN utilizadas. Después de esto, se exponen las bases innovadoras del trabajo: un test de los modelos existentes y un análisis de las variables temporales. Se finaliza con una explicación de la etapa de experimentación, donde se obtienen todos los resultados. Para una explicación más en detalle práctico véase el ANEXO A.

## 4.1. Obtención de los datasets

Los datasets utilizados son datos EEG recogidos de 6 usuarios. Cada usuario realizó una sesión diaria durante 5 días consecutivos (de lunes a viernes). Una sesión está compuesta por 10 pruebas (4 para crear el modelo y 6 para su testeo). El paradigma BCI, basado en señales y realimentación visual, constaba de 2 tareas de imaginación motora: la imaginación de reposo y la imaginación de la marcha. En una prueba, el usuario se posiciona frente a un monitor que le provee de la información visual sobre la tarea a realizar. Dicho usuario realiza en una sesión 20 trials (segmentos temporales que pertenecen a una de las posibles clases), 10 por clase de tarea. Durante los periodos de relajación, los usuarios debían relajar sus mentes lo máximo posible; durante los trials de Imaginación los usuarios tenían que imaginar el movimiento de la marcha. Ambas tareas aparecen aleatoriamente, pero no más de 2 veces para evitar aburrimiento o cansancio mental. Su duración individual está entre 6 y $7.4s$. Además, se utilizó una tercera tarea de "transición" de $3s$ de duración que separa las tareas de interés para permitir al usuario descansar la vista pestañeando, tragando saliva. Movimientos de cabeza o cualquier otro acto que supondría artefactos sobre las señales a estudio. La **Figura 5** ofrece una representación de la secuencia temporal de cada prueba.

Para la recolección de las señales EEG se utilizaron 30 electrodos (P7, P4, CZ, PZ, P3, P8, O1, O2, C2, C4, F4, FP2, FZ, C3, F3, FP1, C1, OZ, PO4, FC6, FC2, AF4, CP6, CP2, CP1, CP5, FC1, FC5, AF3, PO3) dispuestos según el standard 10-10 del Sistema Internacional, junto a 2 electrodos de referencia (CMS, DRL), a una frecuencia de muestreo de $500Hz$. En la ***Figura 6*** se puede observar el montaje de los electrodos utilizados.

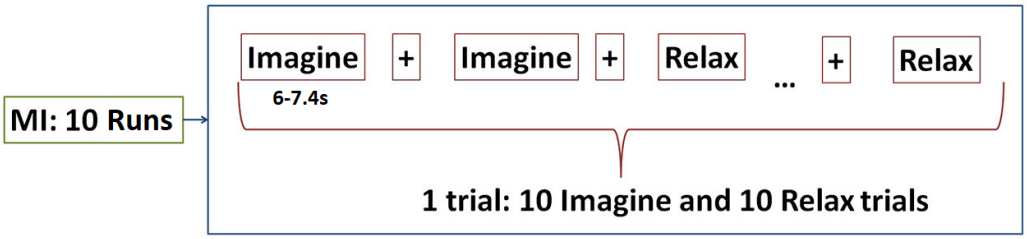


**Figura 5**. Secuencia temporal de cada sesión
Cada sujeto realiza 10 pruebas de tareas de imaginación motora. Las tareas van separadas por periodos de transición entre ellas. Cada prueba consistió en 10 tareas de imaginación de Relax y 10 de Marcha
Fuente: adaptado de [21]

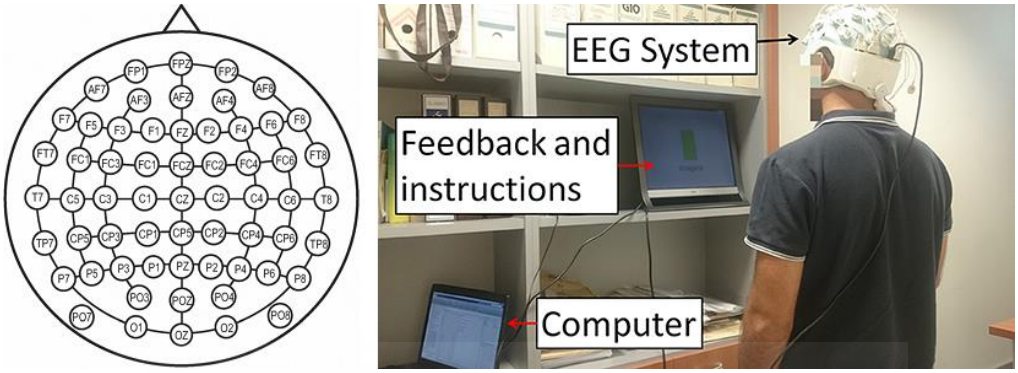


**Figura 6**. Recolección experimental señales EEG del BMILab Dataset
A la izquierda se muestra un esquema de la colocación de los electrodos empleados (P7, P4, CZ, PZ, P3, P8, O1, O2, C2, C4, F4, FP2, FZ, C3, F3, FP1, C1, OZ, PO4, FC6, FC2, AF4, CP6, CP2, CP1, CP5, FC1, FC5, AF3, PO3, CMS, DRL) para la recolección de las señales EEG del BMILab Dataset. A la derecha se observa la situación experimental
Fuente: adaptado de [21]

Para más información acerca de la adquisición experimental de estas señales, véase [21]. Con el propósito de realizar un análisis comparativo de los resultados obtenidos con dichos datasets, se realizaron los mismos procedimientos con el dataset público 2a del *BCI Competition IV* [22]. Este dataset guarda cierta similitud de procedimientos y fue el utilizado por los autores originales de la arquitectura neuronal. En la **Tabla 1** se muestra una comparación comprensible de ambos datasets.

**Tabla 1**
Comparación de Datasets

| | **BNCI Dataset IV 2a** | | **BMILab Datasets** | |
|---|---|---|---|---|
| **Sujetos** | 9 | | 6 | |
| **Sesiones** | 2 | | 5 | |
| **Pruebas** | 6 | | 4 | |
| **Trials** | 48 (12 por clase) | | 20 (10 por clase) | |
| **Muestras totales por prueba** | 54000 | | 35900-38100 | |
| **Tareas** | 4 | Mano Izq. | 2 | Relajación |
| | | Mano Der. | | |
| | | Ambos pies | | Marcha |
| | | Lengua | | |
| **Electrodos** | 22 (+3 EOG) | | 30 (+2 de referencia) | |
| **Frecuencia de muestreo** | 250 | | 500 | |

## 4.2. Formulación de los inputs y preprocesamiento

Se dispone de un conjunto de datos EEG por sujeto. Estos conjuntos que serán introducidos a la red comprenden hasta las 5 sesiones realizadas por cada sujeto, siendo 1 sesión el mínimo. Cada dataset fue formateado para utilizar solo las tareas de Relajación e Imaginación de la marcha. Los datos fueron separados en los trials realizados en cada prueba, identificados con su correspondiente etiqueta (objetivo de imaginación), y se eliminó el primer segundo de cada uno para eliminar los efectos de la respuesta del usuario frente a la indicación de la tarea.

Formalmente, tenemos un dataset $D^i = \{(X^1, y^1), \dots, (X^N, y^N)\}$, siendo $N$ el número total de trials realizados por el sujeto $i$. La matriz de entrada $X^j \in R^{E,T}$ del trial $j, 1 \leq j \leq N$ contiene la información EEG grabada por los $E$ electrodos en los T instantes temporales. La clase de la tarea $j$ viene denotada por $y^j$. Esta toma lo valores de las 2 clases posibles.

Cuando los datasets hayan sido correctamente cargados, son preprocesados. En primer lugar, los datos pasan un filtro Butterworth de $3^{er}$ orden para eliminar las frecuencias por debajo de $4Hz$ y por encima de $38Hz$. El filtro paso alto debería hacer menos probable que la red use artefactos oculares como características discriminantes. Seguidamente, se aplica una función de estandarización móvil exponencial. Esta función, en primer lugar, calcula la media móvil exponencial $m_t$ en el instante $t$ como:

$$m_t = k \cdot \text{media}(x_t) + (1 - k) \cdot m_{t-1} \quad (1)$$

con $k = 0.001$. A continuación, se computa la varianza móvil exponencial $v_t$ en el instante $t$ como:

$$v_t = k \cdot (m_t - x_t)^2 + (1 - k) \cdot v_{t-1} \quad (2)$$

Finalmente, se estandariza el valor:

$$x'_t = \frac{(x_t - m_t)}{\max\left(\sqrt{v_t}, \varepsilon\right)} \quad (3)$$

con $\varepsilon = 10^{-4}$.

Se ha decidido aplicar estos filtros para que los datasets sean preprocesados de igual forma que los originales, obteniendo, así, resultados comparables. Por motivos de la naturaleza del experimento que obtuvo las señales EEG, la duración de los trials es

variable, lo que se traduce en número de muestras variables. Esto implica que la red recibirá un número variable de inputs por trial.

**Tabla 2**
Comparación de preprocesamiento

| | **BCIC Dataset IV 2a** | **BMILab Datasets** |
|---|---|---|
| **Filtro Paso Alto** | 38 | 38 |
| **Filtro Paso Bajo** | 4 | 4 |
| **Voltios a Microvoltios** | Sí | Sí |
| **Exponential moving standardization** | Sí | Sí |
| **Offset inicial del trial** | 0.5s (125 muestras) | 1s (500 muestras) |
| **Offset final** | 0 | 0 |
| **Muestras** (tamaño del input) | 750 | 1500 |

Se asume que las Señales EEG se aproximan a una superposición lineal de patrones de voltaje espacialmente globales originados por múltiples fuentes de corriente dipolar en el cerebro. Entonces, se considera un paso básico separar estos patrones globales usando filtros espaciales, típicamente aplicado a todo el conjunto de electrodos relevantes. Todas las modulaciones EEG son globales por naturaleza, debido a la naturaleza física de la captación no-invasiva de estas señales, por tanto, no existe una composición jerarquía obvia de modulaciones globales y locales en el espacio. Sin embargo, existe una evidencia abundante sobre que las señales EEG se organizan a través de múltiples escalas temporales, como en oscilaciones compuestas que involucran modulaciones locales y globales en el tiempo. Teniendo esto en cuenta, los modelos de Red han sido diseñadas para de forma que puedan aprender filtros separadores espacialmente globales en las capas de entrada, así como jerarquías temporales de modulaciones locales y globales en los modelos más profundos. Para ello, las señales EEG se formulan como una matriz 2D con el número de instantes temporales como ancho y el número de electrodos como la altura. De esta forma, además, se consigue reducir la dimensionalidad de la entrada comparado con la señal EEG como una "imagen" (la sucesión temporal de mapas de la distribución del voltaje a lo largo del cuero cabelludo).

## 4.3. Los modelos de Red Convolucional

### 4.3.1. Deep ConvNet

El modelo de Deep CNN busca extraer un amplio rango de características, sin estar restringido a un tipo concreto. Una arquitectura tan genérica permite, en primer lugar, generar una buena precisión sin necesidad de conocimiento experto y, en segundo lugar, apoya la idea de que las redes CNN estándar se pueden entender como herramientas de

generalización para tareas de decodificación de señales cerebrales. Además, esto supone que el modelo podría beneficiarse directamente de los avances metodológicos conseguidos para el Deep Learning en otros campos.

Esta arquitectura está compuesta por 4 bloques convolucionales con *max pooling*, donde el primero fue especialmente diseñado para tratar con señales EEG, seguidos de una capa densa de clasificación *softmax*. El primer bloque convolucional está dividido en dos capas para un mejor manejo del gran número de canales de entrada (los 32 electrodos). La primera capa realiza una operación convolución a lo largo del tiempo y la segunda a lo largo del espacio (electrodos). Cada filtro en estos pasos tiene pesos para todos los electrodos (como un filtro Patrón Espacial Común (CSP)) y para los filtros de convolución temporal precedentes (como cualquier capa convolucional estándar intermedia). Dado que no existe ninguna función de activación entre ambas capas, estas podrían, en principio, estar combinadas en una sola. Sin embargo, el uso de dos capas regulariza implícitamente la convolución general al forzar una separación de la transformación lineal en una combinación de dos convoluciones (temporal y espacial). Se utiliza una función ELU como función de activación.

$$ELU(x) = \begin{cases} x, & x > 0 \\ e^x - 1, & x \leq 0 \end{cases} \quad (4)$$

Para la regularización y la normalización intermedia se utiliza la *batch normalization*, *dropout* y una nueva función de pérdida dedicada. La batch normalization estandariza las salidas internas de la red a media cero y varianza unitaria para un lote de ejemplos de entrenamiento. Esto pretende facilitar la optimización manteniendo las entradas de las capas cerca de una distribución normal durante el entrenamiento. Dropout ajusta aleatoriamente algunos inputs para la capa a $0$ en cada actualización del entrenamiento (epoch). El objetivo es prevenir la coadaptación de diferentes unidades y pude verse como análogo al entrenamiento de varias redes. Después de la primera, las entradas a todas las capas convolucionales son eliminadas con una probabilidad del $0.5$. Finalmente, se aplica una función de pérdida que fue diseñada especialmente para regularizar aún más el entrenamiento *cropped*.

#### 4.3.2. Shallow ConvNet

La Shallow ConvNet es una arquitectura mucho más sencilla que fue diseñada específicamente para extraer características de las bandas de frecuencia. Las primeras dos capas realizan una convolución temporal y espacial igual que la Deep ConvNet. La

convolución temporal tiene un tamaño de kermel superior (25 frente a 10 de la Deep ConvNet). Permitiendo un mayor rango de transformaciones en esta capa. A continuación, el input pasa por una no linealidad cuadrática, una capa de *mean pooling* y una función de activación logarítmica. Debido a las varias regiones de pooling dentro de un trial, esta arquitectura puede aprender una estructura temporal de los cambios en las bandas de potencia de dicho trial.

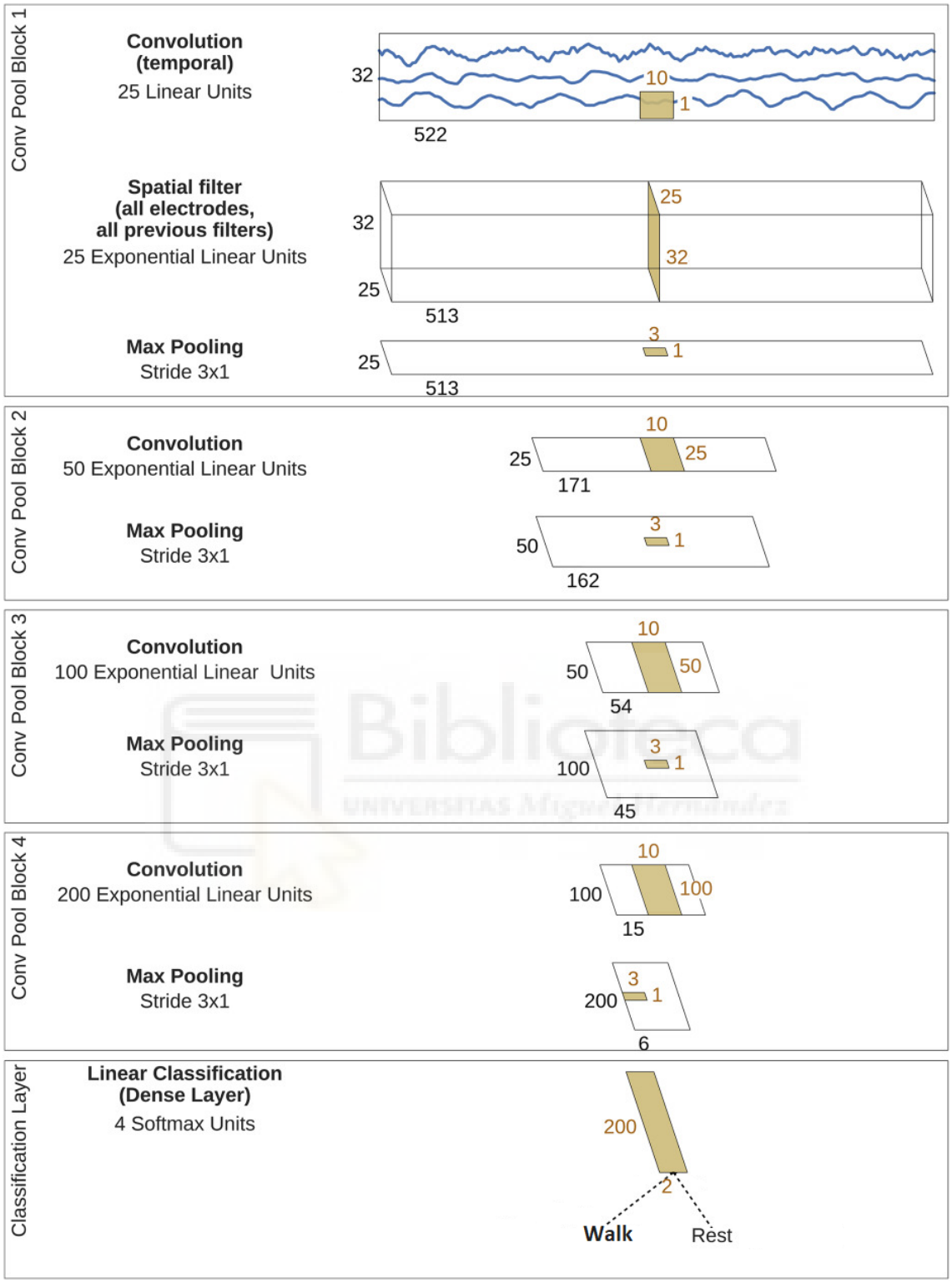


**Figura 7.** Arquitectura Deep ConvNet
La señal EEG es progresivamente transformada como muestra la secuencia en la imagen (de arriba a abajo). Cubos negros: entradas/mapas de características; cubos marrones: kermels de convolución/pooling; Los tamaños correspondientes están indicados en negro y marrón, respectivamente. Nótese que los tamaños y proporciones de estos esquemas son aproximados
Fuente: adaptado de [20]

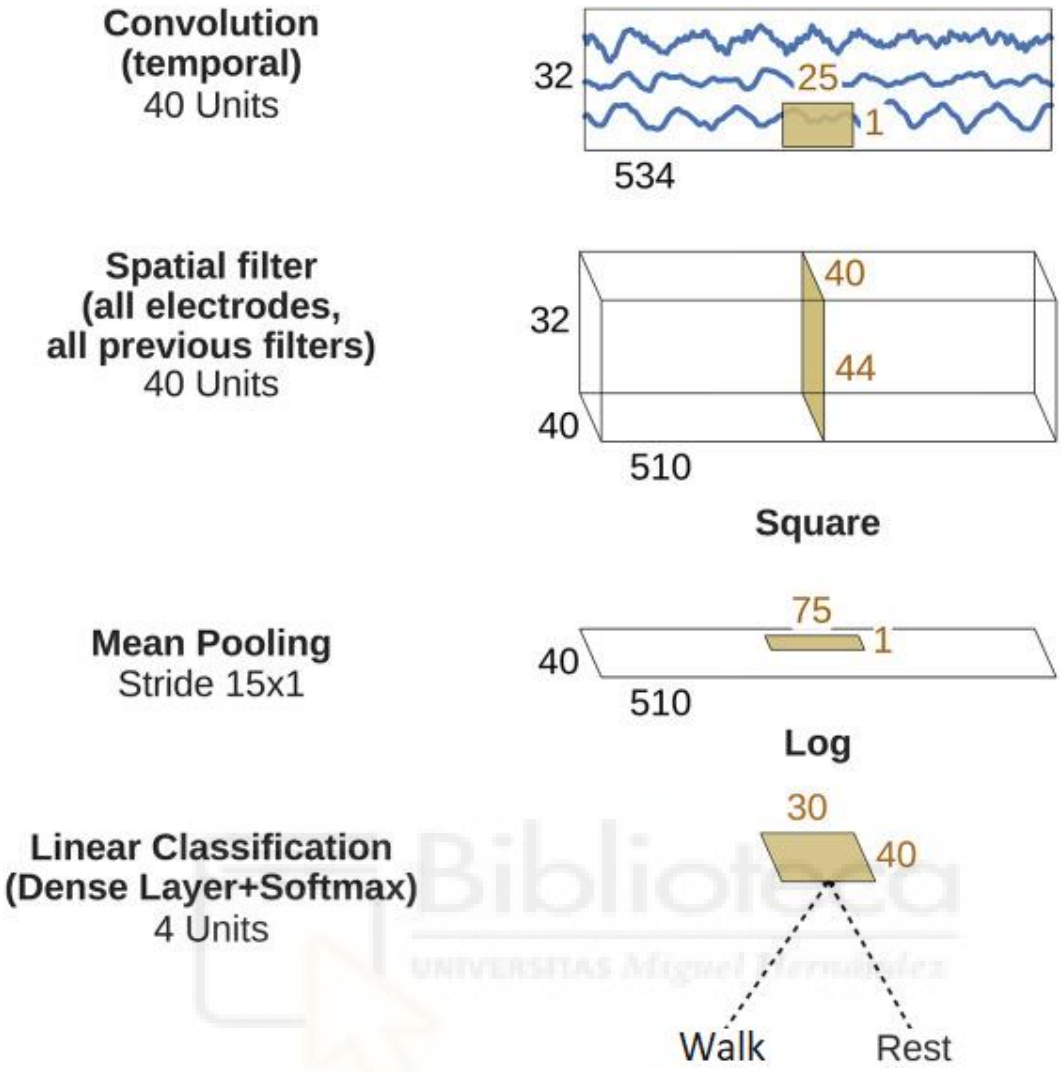


**Figura 8.** Arquitectura Shallow ConvNet
La señal EEG es progresivamente transformada como muestra la secuencia en la imagen (de arriba a abajo). Cubos negros: entradas/mapas de características; cubos marrones: kermels de convolución/pooling; Los tamaños correspondientes están indicados en negro y marrón, respectivamente. Nótese que los tamaños y proporciones de estos esquemas son aproximados
Fuente: adaptado de [20]

## 4.4. Métodos de entrenamiento

### 4.4.1. Predicción única por ventana

Esta estrategia de entrenamiento utiliza la duración completa de una ventana de entrada. Para cada ventana, la señal EEG es utilizada como entrada y su correspondiente clase como la salida para entrenar a la red.

En el entrenamiento, se ajustan todos los parámetros (todos los pesos y los biases) de la ConvNet. La red computa una función $f(X^j;\theta): R^{E-T} \rightarrow R^K$ donde $\theta$ son los parámetros de la función, $E$ el número de electrodos, $T$ el número de instantes temporales y $K$ el número de posibles objetivos de salida (clases). Para utilizar la ConvNet como

clasificador, la salida es normalmente transformada en probabilidades condicionales de una etiqueta $l_k$ dada una entrada $X^j$ usando la función softmax

$$\mathrm{p}\left(l_k \mid f_k\left(X^j_{t\ldots t+T'};\theta\right)\right) = \frac{\exp\left(f_k\left(X^j_{t\ldots t+T'};\theta\right)\right)}{\sum_{k=1}^{K}\exp\left(f_k\left(X^j_{t\ldots t+T'};\theta\right)\right)} \quad (5)$$

Dado que trabajamos por sujeto, la salida de la función nos provee de distribución condicional especifica del sujeto sobre las $K$ clases. Se puede entrenar la ConvNet para asignar altas probabilidades a las etiquetas correctas minimizando el sumatorio de la pérdida (error) de cada ejemplo:

$$\theta^* = arg\ \min_{\theta}\sum_{j=1}^{N} loss\left(y^j, p\left(l_k \mid f_k\left(X^j_{t\ldots t+T'};\theta\right)\right)\right) \quad (6)$$

donde:

$$loss\left(y^j, p\left(l_k \mid f_k\left(X^j_{t\ldots t+T'};\theta\right)\right)\right) = \sum_{k=1}^{K} -\log\left(p\left(l_k \mid f_k\left(X^j_{t\ldots t+T'};\theta\right)\right)\right)\cdot\delta\left(y^j = l_k\right) \quad (7)$$

es la probabilidad logarítmica negativa de las etiquetas. Como es común para el entrenamiento de las Redes convolucionales, los parámetros son optimizados mediante descenso del gradiente estocástico mini-batch usando gradientes analíticos computados vía *backpropagation*.

Esta función que computa la ConvNet se puede ver como la combinación de función de extracción de características y una función de clasificación. La función de extracción de características $\varphi\left(X^J;\theta_\varphi\right)$ con parámetros $\theta_\varphi$ es aplicada por todas las capas hasta la penúltima. La función de clasificación $g\left(\varphi\left(X^J;\theta_\varphi\right);\theta_g\right)$ con parámetros $\theta_g$ usa la salida de la función de extracción de características como entrada y se ejecuta por la última capa, la de clasificación. Con la optimización conjunta de ambas funciones, una CNN aprende tanto una representación de las características para la tarea como un clasificador discriminativo. Esto es especialmente útil para conjuntos de datos grandes, donde el modelo es capaz de aprender características más útiles y no sufre de overfitting al ruido.

#### 4.4.2. Predicciones múltiples por ventana

Para el caso del entrenamiento *cropped* se utilizan ventanas de datos deslizantes dentro de cada trial, lo que conduce a un mayor número de ejemplos para el entrenamiento de la red. Formalmente, dada una ventana de datos original $X^j \in R^{E-T}$, se crea un

conjunto con múltiples muestras de tamaño $T'$ tal que $C^j = \{X^j_{1\dots E,t\dots t+T} | t \in 1\dots T - T'\}$. Todas estas $T - T'$ muestras son nuevos ejemplos de entrenamiento.

Para reducir la carga computacional del mayor número de ejemplos se decodifica un grupo de muestras vecinas juntas y se reutilizan las convoluciones intermedias. El método funciona proveyendo a la ConvNet de una entrada que contiene varias muestras para que produzca las predicciones para cada una de ellas en una única pasada. De aquí surge un nuevo hiperparámetro: el número de muestras que son procesadas al mismo tiempo. Cuanto mayor sea este número, mayor será la velocidad que se puede obtener. Un mayor número de muestras procesadas al mismo tiempo durante el entrenamiento también implica actualizaciones de parámetros a partir de gradientes calculados con un mayor número de muestras de la misma prueba, con el riesgo de un entrenamiento menos estable. Este método de procesamiento resulta en las mismas predicciones para una única muestra como input cuando se utilizan convoluciones sin *padding*. Para el análisis de este trabajo. no se ha utilizado padding, pero se podría aplicar si solo se está interesado en la media de las predicciones para una misma prueba.

Para regularizar aún más la ConvNet entrenadas con este método, se diseñó una nueva función objetivo, la cual penaliza discrepancias entre muestras vecinas. Esta función calcula la pérdida en la predicción de cada muestra. En esta función se añade la entropía cruzada de dos predicciones vecinas a la función de pérdida habitual de probabilidad logarítmica negativa de las etiquetas. Denotando la predicción $\mathrm{p}\left(l_k | f_k\left(X^j_{t\dots t+T'}; \theta\right)\right)$ para la muestra $X^j_{t\dots t+T'}$ del instante t al $t + T'$ como $\mathrm{p_{f,k}}\left(X^j_{t\dots t+T'}\right)$, el error ahora tiene en cuenta la siguiente muestra $\mathrm{p_{f,k}}\left(X^j_{t\dots t+T'+1}\right)$ y convierte la función objetivo en:

$$\mathrm{loss}\left(y^j, \mathrm{p_{f,k}}\left(X^j_{t\dots t+T'}\right)\right) = \sum_{k=1}^{K} -\log\left(\mathrm{p_{f,k}}\left(X^j_{t\dots t+T'}\right)\right) \cdot \delta(y^j = l_k) + \sum_{k=1}^{K} -\log\left(\mathrm{p_{f,k}}\left(X^j_{t\dots t+T'}\right)\right) \cdot \mathrm{p_{f,k}}\left(X^j_{t\dots t+T'+1}\right) \qquad (8)$$

Este método de entrenamiento tiene como objetivo forzar al modelo a aprender características presentes en todas las muestras de la ventana, dado que ya no puede usar las diferencias entre las muestras y la estructura temporal global de las características presentes en la ventana.

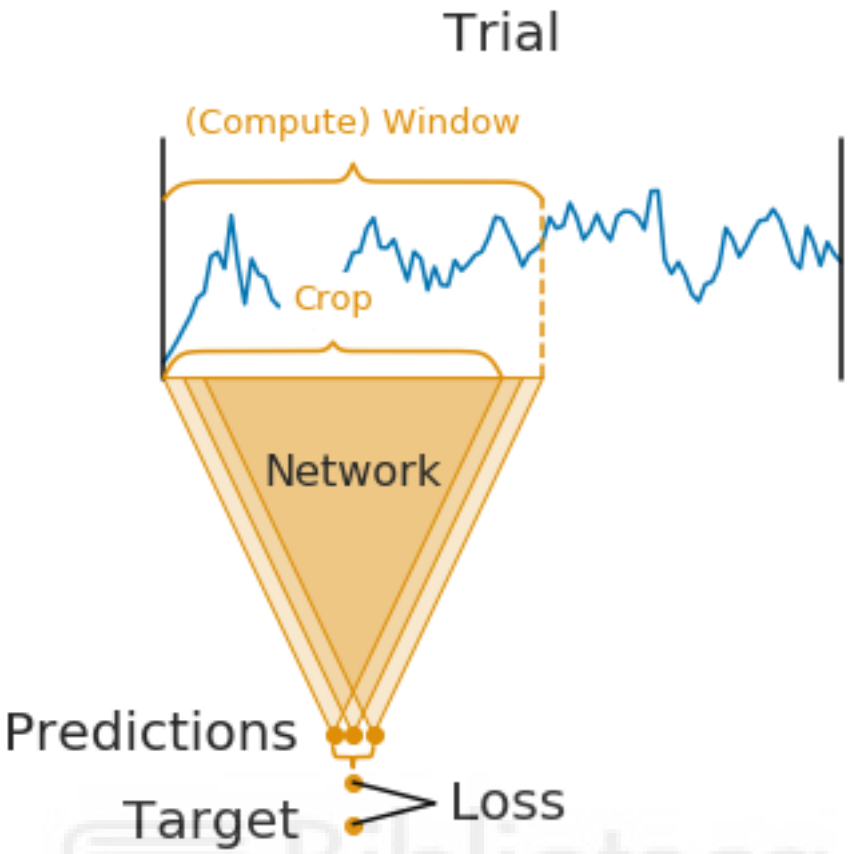


**Figura 9.** Estrategia de entrenamiento Cropped
Las pruebas presentes en cada señal EEG es separada en ventanas que a su vez son separadas en muestras para generar un mayor número de predicciones que servirán para promediar una predicción definitiva para la ventana.
Fuente: https://braindecode.org/auto_examples/plot_bcic_iv_2a_moabb_cropped.html

## 4.5. Test del modelo

Pese a que, en el artículo original, los autores presentan porcentajes de acierto en la clasificación de tareas, en la librería que han publicado vía web no existen instrucciones para poner a prueba un modelo ya entrenado y validado. Por este motivo se decidió añadir líneas de código que testean un modelo con los parámetros ya establecidos. Además, se diseñó el test de forma *pseudo-online*: se provee al modelo entrenado de una ventana de información en cada iteración para obtener una predicción de la clase a la que pertenece. Tras definir la parte del dataset original dedicado al test, se realizan las predicciones, comparándolas con las reales para obtener la exactitud de clasificación que es capaz de obtener el modelo. Durante la ejecución se muestra por pantalla la tarea predicha por si, en el futuro, se desea utilizar esta función para experimentos con feedback visual. Al final de la ejecución, el tiempo de test, el promedio de tiempo por predicción y la exactitud de clasificación del modelo se muestran por pantalla también. Todo esto se entiende dentro de la motivación de aplicar estos modelos CNN en futuros experimentos online de recolección y aplicación de señales EEG

## 4.6. Análisis temporal

Se ha considerado importante el estudio de los valores temporales por dos motivos. El primero es que los modelos de deep learning necesitan de una gran información para extraer características relevantes que permitan obtener buenos resultados de clasificación. Esta gran cantidad de información afectaría a la creación del modelo, aumentando el tiempo de lectura de datos y ajuste de parámetros durante el entrenamiento. Por ello, se ha considerado el análisis del tiempo medio de cada *epoch* (instante de entrenamiento donde se lee el input, se obtiene un output y se ajustan parámetros). El segundo tiene que ver con el tamaño de las ventanas de datos entrantes. Este supone un valor de suma importancia si se desea utilizar cualquiera de los modelos aquí propuestos en un experimento online. Conocida la frecuencia de muestreo, la cantidad de datos se puede extraer de la duración temporal de los mismos. En primer lugar, a mayor tiempo de pruebas, mayor cantidad de datos para introducir a la red, sin embargo, la clave estaría en utilizar la información dentro de la escala temporal donde se puedan extraer características relevantes y repetibles a lo largo de una misma clase. De esta forma, la gran cantidad de datos podría repartirse en más entradas. En segundo lugar, el tamaño de la entrada repercutiría directamente en el tiempo necesario para que el modelo tome su primera decisión. A partir de esa primera predicción, el tiempo necesario para la siguiente sería el *stride* (desplazamiento de la ventana de datos a lo largo del eje X). Si hay solapamiento entre ventanas, este tiempo Sería menor que el tiempo para la primera elección, pero habría que tener en cuenta las ventanas con dos clases diferentes. Este caso sería interesante si existen características relevantes cuando se va a producir la transición entre clases diferentes. Si no hay solapamiento, el tiempo de decisión siempre será el mismo. Por todo esto resulta importante estudiar si el tiempo necesario para que un modelo entrenado tome una decisión se adapta a la latencia del equipo, a la frecuencia de muestreo y a una seguridad ética y funcionalidad lógica para el usuario. Con este objetivo, se mide el tiempo medio necesario para tomar una decisión sobre las ventanas de datos de un conjunto de test en un experimento pseudo-online.

## 4.7. Procedimiento de experimentación

En esta etapa, con los dos modelos y los diferentes métodos de entrenamiento expuestos, se plantea útil un estudio sobre la variación de ciertos parámetros que podrían mejorar el rendimiento de la red.

En primer lugar, atendiendo a su objetivo de aplicación, entendemos rendimiento del modelo como su exactitud de clasificación, esto es: su capacidad para predecir la clase a la que pertenece una ventana de información EEG. Por ello, este será nuestro resultado a maximizar. A partir de ahora, para referirnos a ello lo denominaremos "test accuracy" y será el número de clases correctamente reconocidas por un modelo entrenado entre todos los posibles eventos.

$$test\ accuracy = \frac{clases\ correctas}{clases\ totales} \tag{9}$$

Téngase en cuenta, que para el BMILab Dataset solo existen 2 clases posibles, mientras que para el BNCI Dataset 2a son 4 las clases posibles (ver ***Tabla 1*** para más información).

Dado que la train accuracy depende enteramente del entrenamiento del modelo, antes debemos analizar esa etapa. Para poder entender como de bien o mal ha resultado el entrenamiento de un modelo debemos fijarnos en dos valores: la exactitud y la pérdida o error. Tendremos, por tanto, "train accuracy" y "valid accuracy", y "train loss" y "valid loss", respectivamente para el dataset de entrenamiento y el de validación. Los dos primeros vuelven a ser exactamente iguales que en la ecuación (9). En esta ocasión, estos valores servirán para que el modelo modifique sus pesos sinápticos (parámetros internos del modelo). Para el caso del train loss y el valid loss, estos valores se calcularán según la ecuación (7) o (8), si el entrenamiento es por ventana única o cropped. Durante el entrenamiento, el modelo automáticamente buscará maximizar el train accuracy y el train loss. Los respectivos valores para la validación servirán para revelar si el modelo sufre de overfitting (el modelo se especializa y solo es capaz de reconocer el dataset de entrenamiento) o underfitting (el modelo no es capaz de encontrar información discriminatoria con la información de entrenamiento suministrada). Este proceso es clave para que el modelo sea capaz de reconocer información nunca antes analizada.

Estos valores se ven influenciados por dos pilares fundamentales: el "tiempo de entrenamiento" y la cantidad de información suministrada para entrenar. El "tiempo de entrenamiento" se refiere al número de pasadas de un dataset de entrenamiento por un modelo para ajustar sus parámetros internos. Este número de pasadas se conoce como "epoch" o instante de entrenamiento. A mayor número de epochs el modelo podrá ajustarse mejor a la información de entrada, sin embargo, esto podría conllevar overfitting. Por ello, estudiaremos el efecto de variar este valor. Por otro lado, mucha

información para el entrenamiento podría servir para encontrar características más discriminativas, pero también existe el riesgo de proveer de información muy diferente, donde no exista un patrón común. Por ello, también variaremos el número de muestras con los que se entrenará un modelo.

El entrenamiento cropped ya está diseñado para multiplicar el número de entradas, sin embargo, aún existen formas de aumentar este valor. En primer lugar y el más claro es el tamaño de una ventana: el "Input simples". En cada epoch el dataset completo es introducido, pero las ventanas son tratadas de forma independiente. Dentro de cada ventana será donde se busquen las relaciones y características temporales. Un menor número de muestras crearía más ventanas independientes temporal y espacialmente. Otra forma de aumentar el número de entradas es el solapamiento entre ventanas o "stride". Con el solapamiento se pueden crear nuevas ventanas donde la parte final de la anterior ventana se concatena con nueva información. Este es el concepto que utiliza el entrenamiento cropped, pero se utiliza para obtener varias predicciones que luego se promedian. Para el entrenamiento de ventana única variaremos el valor de muestras solapadas entre ventanas o "Input stride". De esta forma, obtendremos más ventanas de entrada independientes por clase tanto para el entrenamiento como para el test.

También dentro de la intención de aumentar los datos de entrada y debido a que los autores originales entrenan y validan los modelos con dos sesiones realizadas en diferentes días, se planteó entrenar, validar y testear a partir de sesiones. Sin embargo, las señales EEG de un sujeto varían de un día para otro, lo cual podría resultar en peores precisiones al comprobar un modelo entrenado con sesiones realizadas en diferentes días. Por ello, se realizaron ambas pruebas: 2 sesiones para entrenamiento, 2 sesiones para validación y 1 sesión para test; y, dentro de una misma sesión, 2 pruebas para entrenamiento, 1 prueba para validación y 1 prueba para test. Para el caso del BNCI Dataset 2a, solo se disponen de 2 sesiones por usuario, por lo que se utilizaron, de ambas sesiones, 2 pruebas para entrenamiento, 2 pruebas para validación y 2 pruebas para test.

# 5. RESULTADOS

En la siguiente sección se exponen y discuten los diferentes resultados obtenidos. Comenzamos con una revisión de los resultados del entrenamiento (train accuracy, train loss, valid accuracy y valid loss). Seguidamente se pasa a estudiar la exactitud obtenida con los diferentes modelos (test accuracy). Para discernir si los resultados están condicionados por los datos aportados, se comparan los resultados con los obtenidos a partir del dataset público anteriormente expuesto. A continuación, se comprueba la aplicación o no de CUDA tanto en el entrenamiento como en la etapa de predicción.

Cada uno de estos apartados se dividen para los dos modelos CNN probados (Shallow CNN y Deep CNN) y dos métodos de entrenamiento (ventana única y cropped), con las variantes de datasets por prueba y por sesión para el BMILab Dataset. Dentro de estas posibilidades, se han variado los input samples por ventana, el stride (para el método de ventana única) y las epochs de entrenamiento.

Los resultados han sido promediados entre los diferentes usuarios disponibles y las sesiones realizadas (para el caso de entrenamiento por prueba). Al final de la sección se ofrece la configuración que ha resultado óptima para maximizar el train accuracy.

## 5.1. Resultados del entrenamiento

### Shallow CNN – Ventana única

**Tabla 3**
Resultados entrenamiento 1.1: BMILabD Shallow CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|---|
| 250 | 125 | 50 | 90,733 | 0,399 | 55,603 | 0,834 |
| 250 | 250 | 50 | 98,311 | 0,205 | 57,687 | 0,922 |
| 500 | 125 | 50 | 99,263 | 0,208 | 56,610 | 0,887 |
| 500 | 250 | 50 | 99,993 | 0,057 | 59,244 | 1,121 |
| 500 | 500 | 50 | 98,987 | 0,166 | 60,036 | 0,886 |
| 1000 | 125 | 50 | 99,979 | 0,120 | 57,335 | 0,888 |
| 1000 | 250 | 50 | 99,959 | 0,031 | 60,809 | 1,173 |
| 1000 | 500 | 50 | 98,965 | 0,103 | 60,309 | 0,955 |
| 1000 | 1000 | 50 | 99,561 | 0,068 | 58,679 | 1,010 |
| 250 | 125 | 100 | 100,000 | 0,052 | 55,654 | 0,990 |
| 250 | 250 | 100 | 98,126 | 0,237 | 55,477 | 0,939 |
| 500 | 125 | 100 | 99,728 | 0,069 | 57,782 | 1,146 |
| 500 | 250 | 100 | 100,000 | 0,071 | 57,254 | 1,062 |
| 500 | 500 | 100 | 100,000 | 0,013 | 59,701 | 1,366 |
| 1000 | 125 | 100 | 99,767 | 0,064 | 59,993 | 1,039 |
| 1000 | 250 | 100 | 100,000 | 0,037 | 59,368 | 1,014 |
| 1000 | 500 | 100 | 100,000 | 0,008 | 60,641 | 1,376 |
| 1000 | 1000 | 100 | 99,845 | 0,044 | 61,616 | 1,077 |

**Tabla 4**
Resultados entrenamiento 1.2: BNCID2a Shallow CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|---|
| 250 | 125 | 50 | 61,066 | 1,093 | 39,969 | 1,295 |
| 250 | 250 | 50 | 80,367 | 0,738 | 47,794 | 1,225 |
| 500 | 125 | 50 | 80,367 | 0,738 | 47,794 | 1,225 |
| 500 | 250 | 50 | 96,923 | 0,299 | 57,610 | 1,079 |
| 500 | 500 | 50 | 96,923 | 0,299 | 57,610 | 1,079 |
| 1000 | 125 | 50 | 96,923 | 0,299 | 57,610 | 1,079 |
| 1000 | 250 | 50 | 99,711 | 0,162 | 55,932 | 1,082 |
| 1000 | 500 | 50 | 99,711 | 0,162 | 55,932 | 1,082 |
| 1000 | 1000 | 50 | 99,711 | 0,162 | 55,932 | 1,082 |
| 250 | 125 | 100 | 99,711 | 0,162 | 55,932 | 1,082 |
| 250 | 250 | 100 | 70,628 | 0,949 | 41,570 | 1,305 |
| 500 | 125 | 100 | 92,376 | 0,468 | 49,067 | 1,303 |
| 500 | 250 | 100 | 92,376 | 0,468 | 49,067 | 1,303 |
| 500 | 500 | 100 | 99,769 | 0,109 | 59,857 | 1,184 |
| 1000 | 125 | 100 | 99,769 | 0,109 | 59,857 | 1,184 |
| 1000 | 250 | 100 | 99,769 | 0,109 | 59,857 | 1,184 |
| 1000 | 500 | 100 | 100,000 | 0,040 | 60,012 | 1,103 |
| 1000 | 1000 | 100 | 100,000 | 0,040 | 60,012 | 1,103 |

**Tabla 5**
Resultados entrenamiento 1.3: BMILabD Shallow CNN, ventana única, por sesión

| Input Samples | Input stride | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|---|
| 250 | 125 | 50 | 90,447 | 0,379 | 76,255 | 0,514 |
| 250 | 250 | 50 | 98,903 | 0,147 | 83,634 | 0,448 |
| 500 | 125 | 50 | 99,242 | 0,155 | 83,959 | 0,433 |
| 500 | 250 | 50 | 99,991 | 0,033 | 85,396 | 0,456 |
| 500 | 500 | 50 | 99,984 | 0,053 | 85,315 | 0,412 |
| 1000 | 125 | 50 | 99,950 | 0,081 | 84,481 | 0,387 |
| 1000 | 250 | 50 | 100,000 | 0,008 | 86,793 | 0,471 |
| 1000 | 500 | 50 | 100,000 | 0,017 | 86,318 | 0,417 |
| 1000 | 1000 | 50 | 100,000 | 0,027 | 85,795 | 0,386 |
| 250 | 125 | 100 | 100,000 | 0,034 | 84,708 | 0,382 |
| 250 | 250 | 100 | 96,797 | 0,245 | 80,324 | 0,485 |
| 500 | 125 | 100 | 99,978 | 0,055 | 84,517 | 0,506 |
| 500 | 250 | 100 | 100,000 | 0,051 | 84,452 | 0,468 |
| 500 | 500 | 100 | 100,000 | 0,008 | 85,341 | 0,560 |
| 1000 | 125 | 100 | 100,000 | 0,014 | 85,332 | 0,496 |
| 1000 | 250 | 100 | 100,000 | 0,020 | 85,151 | 0,435 |
| 1000 | 500 | 100 | 100,000 | 0,003 | 86,670 | 0,567 |
| 1000 | 1000 | 100 | 100,000 | 0,004 | 86,477 | 0,489 |

Para el caso del entrenamiento por pruebas (**Tabla 3** y **Tabla 4**) vemos valores altos de exactitud de entrenamiento, mientras que la pérdida es baja, para ambos datasets. Todo lo contrario, ocurre con la validación: exactitud baja y errores superiores a la unidad, esto podría indicar overfitting. El tiempo de entrenamiento no parece ser el causante de esto, sino más bien la cantidad de información suministrada. Esta hipótesis se ve reforzada al observar los altos porcentajes de exactitud en la validación por sesiones y la baja pérdida (**Tabla 5**).

### Shallow CNN - Cropped

**Tabla 6**
Resultados entrenamiento 2.1: BMILabD, Shallow CNN, ventana cropped, por prueba

| Input Samples | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|
| 500 | 50 | 100,000 | 0,000 | 63,333 | 2,195 |
| 1000 | 50 | 99,583 | 0,222 | 63,167 | 0,976 |
| 500 | 100 | 100,000 | 0,000 | 62,833 | 2,646 |
| 1000 | 100 | 100,000 | 0,080 | 62,333 | 1,198 |

**Tabla 7**
Resultados entrenamiento 2.2: BNCID2a, Shallow CNN, ventana cropped, por prueba

| Input Samples | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|
| 500 | 50 | 100,000 | 0,005 | 71,759 | 1,807 |
| 500 | 100 | 100,000 | 0,015 | 71,528 | 2,139 |
| 1000 | 50 | 99,363 | 0,227 | 67,998 | 1,025 |
| 1000 | 100 | 100,000 | 0,083 | 68,750 | 1,140 |

**Tabla 8**
Resultados entrenamiento 2.3: BMILabD, Shallow CNN, ventana cropped, por sesión

| Input Samples | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|
| 500 | 50 | 100,000 | 0,000 | 85,938 | 1,099 |
| 1000 | 50 | 100,000 | 0,049 | 86,771 | 0,454 |
| 500 | 100 | 100,000 | 0,000 | 85,521 | 1,311 |
| 1000 | 100 | 100,000 | 0,011 | 86,771 | 0,527 |

En la, **Tabla 6**, **Tabla 7** y **Tabla 8** volvemos a observar la misma tendencia que para el método de la ventana única: alta exactitud de validación y baja pérdida para entrenamiento por sesiones (**Tabla 8**). Se expone, además, cierta mejora para el BNCI Dataset 2ª (***Tabla 7***).

### Deep CNN – Ventana única

**Tabla 9**
Resultados entrenamiento 3.1: BMILabD Deep CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|---|
| 250 | 125 | 50 | 86,824% | 0,382 | 54,602% | 0,911 |
| 250 | 250 | 50 | 90,490% | 0,344 | 53,543% | 0,957 |
| 500 | 125 | 50 | 96,440% | 0,146 | 55,431% | 1,426 |
| 500 | 250 | 50 | 92,725% | 0,262 | 51,989% | 1,378 |
| 500 | 500 | 50 | 98,392% | 0,117 | 51,540% | 1,891 |
| 1000 | 125 | 50 | 94,906% | 0,117 | 55,583% | 3,251 |
| 1000 | 250 | 50 | 91,909% | 0,535 | 54,314% | 3,953 |
| 1000 | 500 | 50 | 97,935% | 0,192 | 53,866% | 5,114 |
| 1000 | 1000 | 50 | 99,035% | 0,021 | 52,156% | 8,715 |
| 250 | 125 | 100 | 98,434% | 0,125 | 55,059% | 1,448 |
| 250 | 250 | 100 | 99,935% | 0,085 | 53,781% | 1,523 |
| 500 | 125 | 100 | 99,711% | 0,024 | 56,648% | 2,197 |
| 500 | 250 | 100 | 98,215% | 0,098 | 52,684% | 1,900 |
| 500 | 500 | 100 | 100,000% | 0,023 | 52,572% | 2,592 |
| 1000 | 125 | 100 | 98,915% | 0,028 | 56,126% | 4,392 |
| 1000 | 250 | 100 | 96,815% | 0,162 | 54,099% | 4,290 |
| 1000 | 500 | 100 | 98,913% | 0,066 | 55,172% | 5,806 |
| 1000 | 1000 | 100 | 99,888% | 0,007 | 52,650% | 10,159 |

**Tabla 10**

Resultados entrenamiento 3.2: BNCID2a Deep CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|---|
| 250 | 125 | 50 | 97,446% | 0,447 | 33,659% | 1,887 |
| 250 | 250 | 50 | 97,280% | 0,445 | 33,688% | 1,884 |
| 500 | 125 | 50 | 94,145% | 0,379 | 40,664% | 1,699 |
| 500 | 250 | 50 | 94,203% | 0,379 | 41,088% | 1,692 |
| 500 | 500 | 50 | 94,165% | 0,376 | 40,953% | 1,691 |
| 1000 | 125 | 50 | 83,247% | 0,529 | 37,876% | 2,026 |
| 1000 | 250 | 50 | 83,218% | 0,529 | 38,050% | 2,025 |
| 1000 | 500 | 50 | 83,304% | 0,529 | 37,500% | 2,026 |
| 1000 | 1000 | 50 | 83,420% | 0,528 | 37,616% | 2,024 |
| 250 | 125 | 100 | 100,000% | 0,087 | 34,238% | 2,807 |
| 250 | 250 | 100 | 100,000% | 0,088 | 34,339% | 2,812 |
| 500 | 125 | 100 | 100,000% | 0,043 | 42,197% | 2,600 |
| 500 | 250 | 100 | 100,000% | 0,044 | 42,612% | 2,604 |
| 500 | 500 | 100 | 100,000% | 0,042 | 42,419% | 2,582 |
| 1000 | 125 | 100 | 99,942% | 0,072 | 38,426% | 3,067 |
| 1000 | 250 | 100 | 99,971% | 0,072 | 39,034% | 3,076 |
| 1000 | 500 | 100 | 99,942% | 0,073 | 38,773% | 3,078 |
| 1000 | 1000 | 100 | 99,942% | 0,073 | 38,715% | 3,071 |

**Tabla 11**

Resultados entrenamiento 3.3: BMILabD Deep CNN, ventana única, por sesión

| Input Samples | Input stride | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|---|
| 250 | 125 | 50 | 97,223% | 0,209 | 82,159% | 0,496 |
| 250 | 250 | 50 | 98,320% | 0,180 | 81,675% | 0,487 |
| 500 | 125 | 50 | 91,421% | 0,097 | 76,980% | 0,824 |
| 500 | 250 | 50 | 92,037% | 0,167 | 77,339% | 0,619 |
| 500 | 500 | 50 | 95,626% | 0,181 | 77,432% | 0,666 |
| 1000 | 125 | 50 | 74,531% | 0,057 | 63,477% | 1,734 |
| 1000 | 250 | 50 | 99,890% | 0,037 | 80,357% | 0,965 |
| 1000 | 500 | 50 | 98,410% | 0,044 | 78,614% | 1,101 |
| 1000 | 1000 | 50 | 99,584% | 0,066 | 81,392% | 1,093 |
| 250 | 125 | 100 | 99,985% | 0,061 | 84,914% | 0,625 |
| 250 | 250 | 100 | 100,000% | 0,037 | 82,834% | 0,646 |
| 500 | 125 | 100 | 99,759% | 0,018 | 85,098% | 0,966 |
| 500 | 250 | 100 | 100,000% | 0,027 | 84,294% | 0,676 |
| 500 | 500 | 100 | 100,000% | 0,028 | 80,598% | 0,916 |
| 1000 | 125 | 100 | 97,057% | 0,009 | 82,253% | 1,271 |
| 1000 | 250 | 100 | 100,000% | 0,004 | 81,790% | 1,311 |
| 1000 | 500 | 100 | 100,000% | 0,008 | 81,827% | 1,494 |
| 1000 | 1000 | 100 | 100,000% | 0,012 | 82,528% | 1,572 |

Los altos errores de validación por prueba (**Tabla 9** y **Tabla 10**) muestran un obvio mal funcionamiento de la arquitectura Deep CNN. Esto se puede deber tanto a overfitting (a menor número de epochs, menor error) o a underfitting (necesidad de mayor cantidad de información de entrada). Para el caso del método por sesión (**Tabla 11**) los valores de exactitud vuelven a ser altos, junto a valores de error bajos, observándose cierta mejora a mayor tiempo de entrenamiento. En cualquier caso, la exactitud del entrenamiento obtenida es menor que la obtenida con el modelo Shallow CNN, lo cual podría marcar un límite para la generalización ante nuevos datos nunca antes vistos.

### Deep CNN - Cropped

**Tabla 12**
Resultados entrenamiento 4.1: BMILabD, Deep CNN, ventana cropped, por prueba

| Input Samples | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|
| 250 | 50 | 98,333% | 0,123 | 60,000% | 2,008 |
| 500 | 50 | 78,167% | 0,565 | 57,333% | 0,747 |
| 1000 | 50 | 68,500% | 0,636 | 56,833% | 0,703 |
| 250 | 100 | 99,667% | 0,030 | 57,667% | 2,977 |
| 500 | 100 | 92,333% | 0,342 | 61,333% | 1,090 |
| 1000 | 100 | 82,583% | 0,521 | 58,833% | 0,788 |

**Tabla 13**
Resultados entrenamiento 4.2: BNCID2a, Deep CNN, ventana cropped, por prueba

| Input Samples | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|
| 500 | 50 | 69,502% | 1,012 | 56,424% | 1,215 |
| 500 | 100 | 92,998% | 0,726 | 62,789% | 1,301 |
| 1000 | 50 | 56,481% | 1,152 | 50,058% | 1,222 |
| 1000 | 100 | 79,398% | 0,856 | 60,880% | 1,171 |

**Tabla 14**
Resultados entrenamiento 4.3: BMILabD, Deep CNN, ventana cropped, por sesión

| Input Samples | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|
| 250 | 50 | 99,688% | 0,085 | 81,250% | 1,053 |
| 500 | 50 | 66,250% | 0,462 | 56,146% | 0,745 |
| 1000 | 50 | 68,958% | 0,568 | 58,333% | 0,667 |
| 250 | 100 | 98,125% | 0,038 | 79,792% | 1,508 |
| 500 | 100 | 83,304% | 0,268 | 67,946% | 0,884 |
| 1000 | 100 | 65,208% | 0,398 | 56,771% | 0,871 |

En esta ocasión, las **Tabla 12**, **Tabla 13** y **Tabla 14** muestras errores de validación disminuyen considerablemente, mejorando la exactitud para el entrenamiento por prueba. Sin embargo, la exactitud sigue siendo baja tanto para el entrenamiento por prueba como por sesión.

## 5.2. Resultados de la clasificación

### Shallow CNN – Ventana única

**Tabla 15**
Resultados clasificación 1.1: BMILabD Shallow CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|---|
| 250 | 125 | 50 | 299,667 | 55,849 |
| 250 | 250 | 50 | 279,667 | 56,205 |
| 500 | 125 | 50 | 153,300 | 57,331 |
| 500 | 250 | 50 | 239,667 | 58,441 |
| 500 | 500 | 50 | 133,300 | 59,470 |
| 1000 | 125 | 50 | 81,233 | 57,174 |
| 1000 | 250 | 50 | 159,667 | 58,398 |
| 1000 | 500 | 50 | 93,300 | 59,027 |
| 1000 | 1000 | 50 | 61,233 | 56,825 |
| 250 | 125 | 100 | 44,067 | 56,785 |
| 250 | 250 | 100 | 299,667 | 55,170 |
| 500 | 125 | 100 | 279,667 | 56,916 |
| 500 | 250 | 100 | 153,300 | 58,032 |
| 500 | 500 | 100 | 239,667 | 58,410 |
| 1000 | 125 | 100 | 133,300 | 59,797 |
| 1000 | 250 | 100 | 81,233 | 57,405 |
| 1000 | 500 | 100 | 159,667 | 59,350 |
| 1000 | 1000 | 100 | 93,300 | 58,982 |

**Tabla 16**
Resultados clasificación 1.2: BNCID2a Shallow CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|---|
| 250 | 125 | 50 | 864,000 | 38,863 |
| 250 | 250 | 50 | 768,000 | 45,992 |
| 500 | 125 | 50 | 768,000 | 45,992 |
| 500 | 250 | 50 | 576,000 | 56,539 |
| 500 | 500 | 50 | 576,000 | 56,539 |
| 1000 | 125 | 50 | 576,000 | 56,539 |
| 1000 | 250 | 50 | 192,000 | 55,671 |
| 1000 | 500 | 50 | 192,000 | 55,671 |
| 1000 | 1000 | 50 | 192,000 | 55,671 |
| 250 | 125 | 100 | 192,000 | 55,671 |
| 250 | 250 | 100 | 864,000 | 40,509 |
| 500 | 125 | 100 | 768,000 | 47,541 |
| 500 | 250 | 100 | 768,000 | 47,541 |
| 500 | 500 | 100 | 576,000 | 58,179 |
| 1000 | 125 | 100 | 576,000 | 58,179 |
| 1000 | 250 | 100 | 576,000 | 58,179 |
| 1000 | 500 | 100 | 192,000 | 59,144 |
| 1000 | 1000 | 100 | 192,000 | 59,144 |

**Tabla 17**
Resultados clasificación 1.3: BMILabD Shallow CNN, ventana única, por sesión

| Input Samples | Input stride | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|---|
| 250 | 125 | 50 | 1198,167 | 61,843 |
| 250 | 250 | 50 | 1118,167 | 67,679 |
| 500 | 125 | 50 | 613,000 | 67,834 |
| 500 | 250 | 50 | 958,167 | 70,379 |
| 500 | 500 | 50 | 533,000 | 70,283 |
| 1000 | 125 | 50 | 324,667 | 68,762 |
| 1000 | 250 | 50 | 638,167 | 72,248 |
| 1000 | 500 | 50 | 373,000 | 71,961 |
| 1000 | 1000 | 50 | 244,667 | 70,684 |
| 250 | 125 | 100 | 177,000 | 69,697 |
| 250 | 250 | 100 | 1198,167 | 64,061 |
| 500 | 125 | 100 | 1118,167 | 68,531 |
| 500 | 250 | 100 | 613,000 | 68,601 |
| 500 | 500 | 100 | 958,167 | 70,611 |
| 1000 | 125 | 100 | 533,000 | 70,314 |
| 1000 | 250 | 100 | 324,667 | 69,992 |
| 1000 | 500 | 100 | 638,167 | 72,020 |
| 1000 | 1000 | 100 | 373,000 | 72,186 |

Para la etapa de predicción observamos (**Tabla 15**, **Tabla 16** y **Tabla 17**) lo que podían dejar prever los resultados del entrenamiento: una mayor exactitud de validación permite una mejor exactitud de entrenamiento. Por tanto, la reflexión sería la misma: el método de entrenamiento por sesión obtuvo la mejor exactitud clasificación. Es decir, más datos de entrada permite al modelo obtener características más relevantes y generalizar mejor. Se observa cierta mejoría con mayores tamaños de ventana, aunque la mayoría de combinaciones de tamaños y solapamientos son capaces de mantenerse a la par.

### Shallow CNN – Cropped

**Tabla 18**
Resultados clasificación 2.1: BMILabD, Shallow CNN, ventana cropped, por prueba

| Input Samples | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|
| 500 | 50 | 2249,467 | 57,656 |
| 1000 | 50 | 67,000 | 58,326 |
| 500 | 100 | 2249,467 | 58,241 |
| 1000 | 100 | 67,000 | 58,935 |

**Tabla 19**
Resultados clasificación 2.2: BNCID2a, Shallow CNN, ventana cropped, por prueba

| Input Samples | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|
| 500 | 50 | 5184,000 | 59,979 |
| 1000 | 50 | 192,000 | 65,567 |
| 500 | 100 | 5184,000 | 60,228 |
| 1000 | 100 | 192,000 | 65,820 |

**Tabla 20**
Resultados clasificación 2.3: BMILabD, Shallow CNN, ventana cropped, por sesión

| Input Samples | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|
| 500 | 50 | 8992,000 | 69,472 |
| 1000 | 50 | 267,667 | 71,158 |
| 500 | 100 | 8992,000 | 69,424 |
| 1000 | 100 | 267,667 | 71,330 |

Para el caso de nuestro propio dataset, los resultados de clasificación vuelven a ser parecidos, tanto para el entrenamiento por prueba (**Tabla 18**) como por sesión (**Tabla 20**). Volvemos a encontrar una mejoría en los resultados del BNCI Dataset 2a (**Tabla 19**) gracias al método de entrenamiento cropped.

### Deep CNN – Ventana única

**Tabla 21**
Resultados clasificación 3.1: BMILabD Deep CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|---|
| 250 | 125 | 50 | 279,667 | 53,985 |
| 250 | 250 | 50 | 153,300 | 53,357 |
| 500 | 125 | 50 | 239,667 | 56,326 |
| 500 | 250 | 50 | 133,300 | 54,867 |
| 500 | 500 | 50 | 81,233 | 52,240 |
| 1000 | 125 | 50 | 159,667 | 55,391 |
| 1000 | 250 | 50 | 93,300 | 54,400 |
| 1000 | 500 | 50 | 61,233 | 52,442 |
| 1000 | 1000 | 50 | 44,067 | 53,065 |
| 250 | 125 | 100 | 279,667 | 54,944 |
| 250 | 250 | 100 | 153,300 | 53,551 |
| 500 | 125 | 100 | 239,667 | 56,504 |
| 500 | 250 | 100 | 133,300 | 54,925 |
| 500 | 500 | 100 | 81,233 | 51,716 |
| 1000 | 125 | 100 | 159,667 | 55,873 |
| 1000 | 250 | 100 | 93,300 | 54,800 |
| 1000 | 500 | 100 | 61,233 | 53,039 |
| 1000 | 1000 | 100 | 44,067 | 53,480 |

**Tabla 22**

Resultados clasificación 3.2: BNCID2a Deep CNN, ventana única, por prueba

| Input Samples | Input stride | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|---|
| 250 | 125 | 50 | 768,000 | 34,317 |
| 250 | 250 | 50 | 768,000 | 34,664 |
| 500 | 125 | 50 | 576,000 | 40,644 |
| 500 | 250 | 50 | 576,000 | 40,683 |
| 500 | 500 | 50 | 576,000 | 40,664 |
| 1000 | 125 | 50 | 192,000 | 36,863 |
| 1000 | 250 | 50 | 192,000 | 36,748 |
| 1000 | 500 | 50 | 192,000 | 36,863 |
| 1000 | 1000 | 50 | 192,000 | 36,921 |
| 250 | 125 | 100 | 768,000 | 33,984 |
| 250 | 250 | 100 | 768,000 | 34,071 |
| 500 | 125 | 100 | 576,000 | 41,416 |
| 500 | 250 | 100 | 576,000 | 41,647 |
| 500 | 500 | 100 | 576,000 | 41,744 |
| 1000 | 125 | 100 | 192,000 | 38,889 |
| 1000 | 250 | 100 | 192,000 | 39,410 |
| 1000 | 500 | 100 | 192,000 | 39,178 |
| 1000 | 1000 | 100 | 192,000 | 39,005 |

**Tabla 23**

Resultados clasificación 3.3: BMILabD Deep CNN, ventana única, por sesión

| Input Samples | Input stride | Windows to test | Test Accuracy |
|---|---|---|---|
| 250 | 125 | 1136,000 | 68,222 |
| 250 | 250 | 622,000 | 65,863 |
| 500 | 125 | 976,000 | 65,540 |
| 500 | 250 | 542,000 | 65,283 |
| 500 | 500 | 329,000 | 61,753 |
| 1000 | 125 | 656,000 | 56,911 |
| 1000 | 250 | 382,000 | 62,042 |
| 1000 | 500 | 249,000 | 58,166 |
| 1000 | 1000 | 176,000 | 60,038 |
| 250 | 125 | 1136,000 | 70,158 |
| 250 | 250 | 622,000 | 67,471 |
| 500 | 125 | 976,000 | 71,448 |
| 500 | 250 | 542,000 | 71,402 |
| 500 | 500 | 329,000 | 63,982 |
| 1000 | 125 | 656,000 | 70,122 |
| 1000 | 250 | 382,000 | 63,656 |
| 1000 | 500 | 249,000 | 62,651 |
| 1000 | 1000 | 176,000 | 60,795 |

En las ***Tabla 21***, ***Tabla 22*** y ***Tabla 23*** encontramos precisiones bajas, sobre todo para el BNCI Dataset 2a. Estos resultados podrían reforzar la idea de que la arquitectura Deep CNN es demasiado compleja para este tipo de datos.

### Deep CNN - Cropped

**Tabla 24**
Resultados clasificación 4.1: BMILabD, Deep CNN, ventana cropped, por prueba

| Input Samples | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|
| 250 | 50 | 910,233 | 55,876 |
| 500 | 50 | 121,533 | 55,933 |
| 1000 | 50 | 52,067 | 57,540 |
| 250 | 100 | 910,233 | 55,819 |
| 500 | 100 | 121,533 | 55,948 |
| 1000 | 100 | 52,067 | 56,450 |

**Tabla 25**
Resultados clasificación 4.2: BNCID2a, Deep CNN, ventana cropped, por prueba

| Input Samples | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|
| 500 | 50 | 384,000 | 47,454 |
| 500 | 100 | 384,000 | 52,286 |
| 1000 | 50 | 192,000 | 49,363 |
| 1000 | 100 | 192,000 | 59,201 |

**Tabla 26**
Resultados clasificación 4.3: BMILabD, Deep CNN, ventana cropped, por sesión

| Input Samples | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|
| 250 | 50 | 3692,000 | 62,468 |
| 500 | 50 | 491,000 | 53,700 |
| 1000 | 50 | 213,000 | 58,294 |
| 250 | 100 | 3692,000 | 61,480 |
| 500 | 100 | 451,286 | 55,731 |
| 1000 | 100 | 213,000 | 51,800 |

Las **Tabla 24**, **Tabla 25** y **Tabla 26** demuestran que esta última configuración es la peor en cuanto a exactitud de clasificación. Pese a disponer de más muestras para el entrenamiento, la arquitectura Deep CNN parece no ser capaz de extraer características marginales útiles.

## 5.3. Resultados temporales

Si bien es importante comparar los resultados obtenidos por los diferentes métodos de entrenamiento, los valores temporales siguen tendencias muy similares en todos los casos. Estos solo varían a causa de la cantidad de información de entrada y según la arquitectura neuronal.

El primer valor sería el tiempo de entrenamiento, el cual se puede ver drásticamente incrementado según la cantidad de ventanas de información disponibles. Debido a esto vemos mayores tiempos totales de entrenamiento para el método cropped, al igual que para mayor número de muestras por ventana. Para el caso de la arquitectura Shallow CNN entrenada por sesión cropped y ventanas $500$ muestras, se ha requerido $216{,}544s$ de media para cada instante de entrenamiento, mientras que para $1000$ muestras por ventana se necesitó $17{,}435s$. Esos valores se ven reducidos ($17{,}971$ y $1{,}018s$, respectivamente) gracias a la aplicación de aceleración del procesamiento mediante el uso de la GPU, reduciendo, así el tiempo total necesario para entrenar un modelo. Para el caso de la arquitectura Deep CNN se observa menores incrementos de tiempo de entrenamiento para la misma cantidad de información de entrada.

Para la etapa de predicción pasamos a hablar de tiempos medios del rango de los milisegundos. Gracias a la aplicación de CUDA, la arquitectura Shallow CNN ha obtenido en muy pocas ocasiones valores superiores a los $3ms$ para cada predicción, viéndose aumentando en ocasiones por el preprocesamiento de los datos a $4ms$. La arquitectura Deep CNN ha llegado a obtener valores no superiores a 5ms. Por su parte, el método de entrenamiento de ventana única se vuelve a mostrar como la opción con menor carga de trabajo por sus menores tiempos por predicción. Para esta ocasión el desvió de procesamiento a la GPU parece no ser tan crucial como para el entrenamiento, obteniéndose sin ella mayoritariamente aumentos de entre 1-$3ms$ (en el peor de los casos, encontramos $12ms$ por predicción).

## 5.4. Desglose de la configuración óptima

Antes de nada, sería necesario entender que una configuración óptima solo se entiende por su aplicación. Podríamos escoger el modelo Shallow CNN, entrenado con ventanas únicas y por sesión, y con ventanas de $1000$ muestras sin solapamiento. Sin embargo, para una frecuencia de muestreo de $500Hz$, esto se traduciría en tomas de decisión cada $2s$.

Por tanto, si el uso de estos modelos comprendiera una clasificación automatizada de señales EEG donde el tiempo no es una variable a tener en cuenta, entonces podríamos fijarnos únicamente en la exactitud de clasificación. Por el contrario, si el tiempo es una variable a tener en cuenta, deberíamos fijarnos en el tamaño de las ventanas de entrada y su solapamiento al mismo nivel que la exactitud de clasificación.

En primer lugar, la combinación de modelo y entrenamiento que hemos podido comprobar mejor sería el modelo Shallow CNN entrenado con ventanas únicas y por sesión. En un ámbito como son los BCIs, donde obtener una gran cantidad de datasets es difícil, parece que el modelo Shallow CNN se adapta mejor a la información disponible. Por su parte, se ha demostrado que es capaz de generalizar mejor entre sesiones, permitiendo, así, una mayor información de entrada. Dentro de este caso, podemos elegir 100 epochs y ventanas de 1000 muestras sin solapamiento. A falta de un estudio más exhaustivo del entrenamiento para discernir si existe overfitting, 100 epochs se ha destacado como el tiempo de entrenamiento adecuado. Por otro lado, 1000 muestras por ventana y sin solapamiento parece ser la cantidad de información de entrada necesaria.

En las **Tabla 27** y **Tabla 28** se puede observar los resultados de esta configuración para cada usuario (sus promedios se pueden encontrar en las **Tabla 5** y **Tabla 17**).

**Tabla 27**
Resultados entrenamiento configuración óptima
BMILabD Shallow CNN, ventana única, por sesión, 1000 inputs simples, 1000 input stride, 100 epochs

| User | Input Samples | Input stride | Epochs | Train Accuracy | Train loss | Validation Accuracy | Validation loss |
|---|---|---|---|---|---|---|---|
| A09 | 1000 | 1000 | 100 | 100,00 | 0,008 | 92,90 | 0,281 |
| A13 | 1000 | 1000 | 100 | 100,00 | 0,014 | 78,86 | 0,669 |
| B29 | 1000 | 1000 | 100 | 100,00 | 0,012 | 79,94 | 0,596 |
| B79 | 1000 | 1000 | 100 | 100,00 | 0,006 | 81,71 | 0,392 |
| B80 | 1000 | 1000 | 100 | 100,00 | 0,007 | 89,83 | 0,359 |
| B81 | 1000 | 1000 | 100 | 100,00 | 0,007 | 87,85 | 0,299 |

**Tabla 28**
Resultados clasificación configuración óptima
BMILabD Shallow CNN, ventana única, por sesión, 1000 inputs simples, 1000 input stride, 100 epochs

| User | Input Samples | Input stride | Epochs | Windows to test | Test Accuracy |
|---|---|---|---|---|---|
| A09 | 1000 | 1000 | 100 | 176 | 82,386 |
| A13 | 1000 | 1000 | 100 | 183 | 60,109 |
| B29 | 1000 | 1000 | 100 | 175 | 61,714 |
| B79 | 1000 | 1000 | 100 | 174 | 62,069 |
| B80 | 1000 | 1000 | 100 | 181 | 77,901 |
| B81 | 1000 | 1000 | 100 | 173 | 78,035 |

# 6. CONCLUSIÓN

Tras analizar los resultados podríamos concluir que las Redes Neuronales Convolucionales, en concreto la arquitectura Shallow CNN, ofrecen una prometedora aplicación para la clasificación automática de las señales cerebrales. Llegando a obtener un promedio del 72,186% de eventos correctamente clasificados con el modelo Shallow CNN entrenado con ventanas únicas (1000 muestras sin solapamiento) y por sesión durante 100 epochs. Si bien los mejores resultados obtenidos en un principio no son seguros para un uso práctico con humanos, podría llegar a tener precisiones a la altura de los métodos convencionales actuales.

En primer lugar, hemos podido descartar la posibilidad de datasets defectuosos gracias a la comparación con otro dataset público, obteniendo similares resultados, incluso mejores en algunos casos. Seguidamente hemos llegado a la conclusión que la arquitectura Shallow CNN sobrepasa a la Deep CNN en todos los aspectos. Si se deseara profundizar y obtener resultados similares con la Deep CNN sería necesario otro estudio en profundidad, donde se exploren diferentes valores no analizados en este trabajo.

Por otro lado, el tamaño de la información de entrada se plantea como el pilar central de un buen funcionamiento. Gracias al estudio de utilizar información de diferentes sesiones para entrenar y comprobar el modelo, podemos concluir que los modelos son capaces de extraer características útiles para reconocer nueva información nunca antes vista. Los modelos se benefician de cuanta mayor información se les provea, pudiendo emular un patrón común dentro de las señales EEG. La librería ofrece una función que permitiría reentrenar a un modelo con parámetros ya establecidos, configurándolo de forma que no se perdiera la capacidad de reconocer la información antigua. Por tanto, sería muy interesante comprobar la evolución de un modelo entrenado y validado a lo largo del tiempo, cada vez con más información. Incluso, se podría analizar el rendimiento de un modelo entrenado con diferentes usuarios. Por otro lado, las combinaciones del método cropped y tamaños de ventana más bajos fueron imposibles de realizar debido a la incapacidad de computar la carga de trabajo tanto por parte de la GPU como de la función de creación de ventanas. Sería necesario una optimización de estas funciones si se desea estudiar este caso.

El problema fundamental de utilizar una mayor cantidad de datos de entrada repercute lógicamente en el tiempo de procesamiento. Después de los resultados, el tiempo de

entrenamiento sería la siguiente barrera a superar por las CNN. Lo cierto es que el entrenamiento de un modelo no entraña ningún peligro humano, sin embargo, habría que tener en cuenta la optimización de este parámetro para una mayor utilidad y comodidad de aplicación. El valor temporal verdaderamente importante para el posible uso de las CNN en un experimento online sería el tiempo de predicción. Para tomas de decisión superiores a, por ejemplo, $50ms$, los modelos entrenados parecen ser capaces de tomar decisiones. De cualquier forma, sería necesario un estudio que añadiera el retraso del hardware físico. Por su parte, CUDA se ha demostrado como una herramienta útil para reducir los tiempos de procesamiento. Su uso más lógico sería para el entrenamiento del modelo. Si bien, podría servir para asegurar menores tiempos en una aplicación real, la mejora no valdría la pena debido al coste del hardware dedicado.

# BIBLIOGRAFÍA


[1] Orgnización Mundial de Salud (OMS), 19 11 2013. [En línea]. Available: https://www.who.int/es/news-room/fact-sheets/detail/spinal-cord-injury. [Último acceso: 26 12 209].

[2] United Nations Population Fund (UNFPA), 2012. [En línea]. Available: https://www.unfpa.org/sites/default/files/pub-pdf/Ageing%20report.pdf. [Último acceso: 26 12 19].

[3] H. Igo Krebs, N. Hogan, M. L. Aisen, and B. T. Volpe, "Robot-aided neurorehabilitation," IEEE Trans. Rehabil. Eng., 1998.

[4] A. Craik, Y. He, and J. L. Contreras-Vidal, "Deep learning for electroencephalogram (EEG) classification tasks: a review," J. Neural Eng., vol. 16, no. 3, p. 031001, Jun. 2019.

[5] F. Lotte et al., "A review of classification algorithms for EEG-based brain-computer interfaces: A 10 year update," Journal of Neural Engineering. 2018.

[6] Y. Li et al., "An EEG-based BCI system for 2-D cursor control by combining Mu/Beta rhythm and P300 potential," IEEE Trans. Biomed. Eng., 2010.

[7] Quesada. H., E., « Doctorado en Tecnologías Industriales y de Telecomunicación. Interfaces cerebro-máquina para asistencia y rehabilitación de personas con movilidad.,» 2015.

[8] M. X. Cohen, Analyzing Neural Time Series Data: Theory and Practice., Cambridge, MA, USA: MIT Press, 2014.

[9] G. C. O'Neill, P. Tewarie, D. Vidaurre, L. Liuzzi, M. W. Woolrich, and M. J. Brookes, "Dynamics of large-scale electrophysiological networks: A technical review," NeuroImage. 2018.

[10] D. P. Subha, P. K. Joseph, R. Acharya U, and C. M. Lim, "EEG signal analysis: a survey.," J. Med. Syst., 2010.

[11] P. Sykacek and G. Dorffner, “Detection of the EEG Artifacts by the Means of the (Extended) Kalman Filter,” Science (80-. )., 2001.

[12] T. P. Jung et al., “Removing electroencephalographic artifacts: Comparison between ICA and PCA,” in Neural Networks for Signal Processing - Proceedings of the IEEE Workshop, 1998.

[13] B. Noureddin, P. D. Lawrence, and G. E. Birch, “Online removal of eye movement and blink EEG artifacts using a high-speed eye tracker,” IEEE Trans. Biomed. Eng., 2012.

[14] C. Y. Sai, N. Mokhtar, H. Arof, P. Cumming, and M. Iwahashi, “Automated classification and removal of EEG artifacts with SVM and wavelet-ICA,” IEEE J. Biomed. Heal. Informatics, 2018.

[15] G. H. Yann LeCun, Yoshua Bengio, “Deep learning,” Nature, 2015.

[16] T. H. Nguyen and W. Y. Chung, “A single-channel SSVEP-based BCI speller using deep learning,” IEEE Access, 2019.

[17] W. Zgallai et al., “Deep Learning AI Application to an EEG driven BCI Smart Wheelchair,” in 2019 Advances in Science and Engineering Technology International Conferences, ASET 2019, 2019.

[18] M.-P. Hosseini, D. Pompili, K. Elisevich, and H. Soltanian-Zadeh, “Optimized Deep Learning for EEG Big Data and Seizure Prediction BCI via Internet of Things,” IEEE Trans. Big Data, 2017.

[19] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA,” Queue, 2008.

[20] R. T. Schirrmeister et al., “Deep learning with convolutional neural networks for EEG decoding and visualization,” Hum. Brain Mapp., 2017.

[21] M. Rodríguez-Ugarte, E. Iáñez, M. Ortiz, and J. M. Azorín, “Improving Real-Time Lower Limb Motor Imagery Detection Using tDCS and an Exoskeleton,” Front. Neurosci., 2018.

[22] C. Brunner, R. Leeb, G. Mu¨ller-Putz, A. Schlo´gl, and G. Pfurtscheller, “BCI Competition 2008—Graz Data Set a,” Laboratory of Brain-Computer Interfaces, Inst. for Knowledge Discovery, Graz Univ. of Technology, http://www.bbci.de/competition/iv/ #dataset2a, 2008.

# ANEXOS

## ANEXO A. Adaptación para trabajar con la librería Braindecode

La librería utilizada a lo largo de la realización de este trabajo está codificada en lenguaje Python. Esto implicó, en primer lugar, realizar un **curso de iniciación a Python e Inteligencia Artificial** debido a que el autor no había recibido aún dichos conocimientos a lo largo de su vida académica. En segundo lugar, fue necesario un proceso de adaptación del computador para que funcionase con Python y con todas las librerías necesarias.

Para poder utilizar Python existen muchas posibilidades. En este caso se instaló la edición individual de *Anaconda* desde su página web oficial (**www.anaconda.com**). Anaconda es una distribución de Python gratuita y open-source que tiene por objetivo simplificar la administración y despliegue de los diferentes paquetes básicos para la computación científica. A parte de esto último, se escogió Anaconda por ser compatible con el Sistema Operativo de Windows y por incluir el entorno de desarrollo (IDE) Spyder. Este software, también open-source, provee de una interfaz gráfica cómoda para leer, revisar, editar y ejecutar código, comprobar las variables y sus valores, y demás resultados devueltos por pantalla.

Con esta base ya se podían instalar el resto de paquetes necesarios. La versión utilizada de la librería Braindecode se puede encontrar y descargar en su propio **repositorio de Github**. La librería contiene, en principio, todo lo necesario para ser ejecutado, así como ejecutables muy útiles, pero ya se profundizará en eso más adelante. Lo cierto es que fue necesario reinstalar, actualizar e incluso editar unas pocas líneas de código de ciertos paquetes de los que depende la librería dado que al lanzar los modelos básicos la ejecución se detenía devolviendo errores. Cuando la librería ya funcionaba sin problemas se pasó a instalar **CUDA**. Para ello fue necesario instalar la **librería PyTorch**. Con ella y comprobado que la **GPU era compatible**, ya era posible acceder a esta mediante las funciones adecuadas. En este punto ya era posible ejecutar los modelos por defecto.

La librería Braindecode permite usar diversos modelos de redes convolucionales (Shallow CNN, Deep CNN, Residual CNN, Hybrid CNN y EEG CNN), varios clasificadores, datasets y paquetes de funciones dedicadas al análisis de señales

neurofisiológicas como MNE. Se pueden encontrar tutoriales sobre su instalación o ejecutables en **braindecode.org**: A partir de aquí, el objetivo era adaptar los códigos para que funcionaran con los datasets propios del laboratorio (BMILab Datasets). Para aprovechar todas las dependencias y funciones ya hechas, y con el objetivo de evitar el mayor número de modificaciones posibles para una mejor comparación, se decidió formatear los datasets. Para ello fue necesario utilizar **MATLAB**. Los datasets originalmente están en formato .mat dentro de una estructura llamada *session* como puede verse en la **Figura 5**:

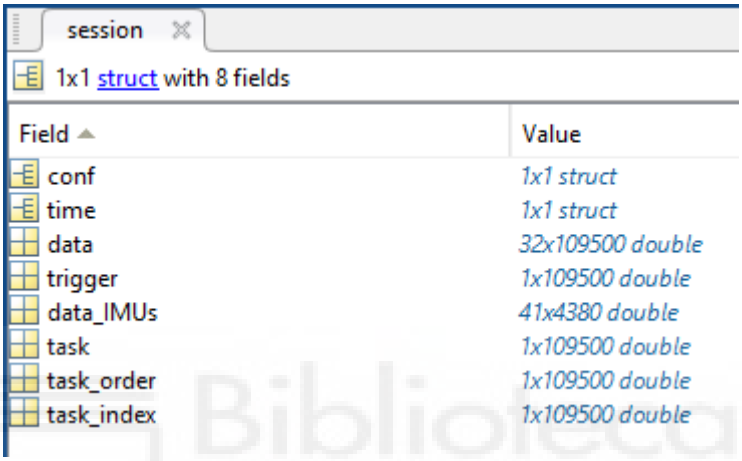


**Figura 10.** Campos de datos disponibles en cada dataset del BMILab

Para que pudieran funcionar con Braindecode se decidió comprobar los archivos .mat de los datasets utilizados por la librería, disponibles en el **sitio web del proyecto BNCI**. Con ellos se pudo copiar su formato. Para ello se creó el script *datasetsformat.m* que crea un directorio por sesión realizada. Dentro de este directorio, se encuentran las 4 pruebas realizadas en cada sesión. Estas pruebas contienen los datos que se pueden observar en la ***Figura 11***.

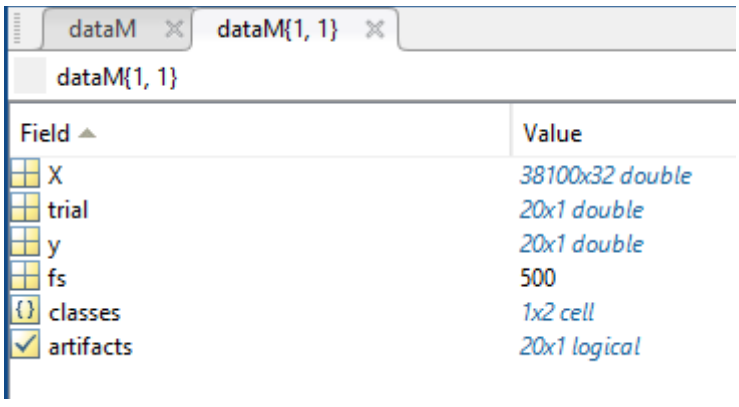


**Figura 11**. Campos de datos disponibles en una prueba de un dataset formateado del BMILab

La variable *X* contiene los datos en cada instante recogidos por los 32 electrodos tras ser eliminados los no referidos a las clases de interés (i.e. la etapa de transición y 1*s* al inicio de cada tarea). La variable *y* contiene el tipo de clase y *trial* el instante temporal en el que comienza cada una. *fs* es la frecuencia de muestreo en hercios, la variable *classes* contiene el nombre de las clases presentes en el dataset en formato string y la variable *artifacts* indica si los artefactos fueron eliminados o no mediante preprocesamiento. Con estas variables ya era posible utilizar las funciones dentro de Braindecode como el preprocesamiento mediante MNE.

Tras unos cuantos cambios en las dependencias de la librería era posible cargar los datasets como cualquier otro, sin embargo, cuando se trataba de crear las ventanas de datos se devolvía un error. Esto se debía a que la función encargada de esto no era capaz de crear las ventanas a partir de los eventos, ya que estos tienen longitudes temporales variables. Se intentó solucionar creando ventanas de tamaño fijo, pero seguía dando problemas, con lo que se decidió acudir a **la sección de problemas** en el repositorio en Github de la librería. Allí, Robin Tibor (uno de los autores), ayudó en la modificación del código para cargar los datasets.

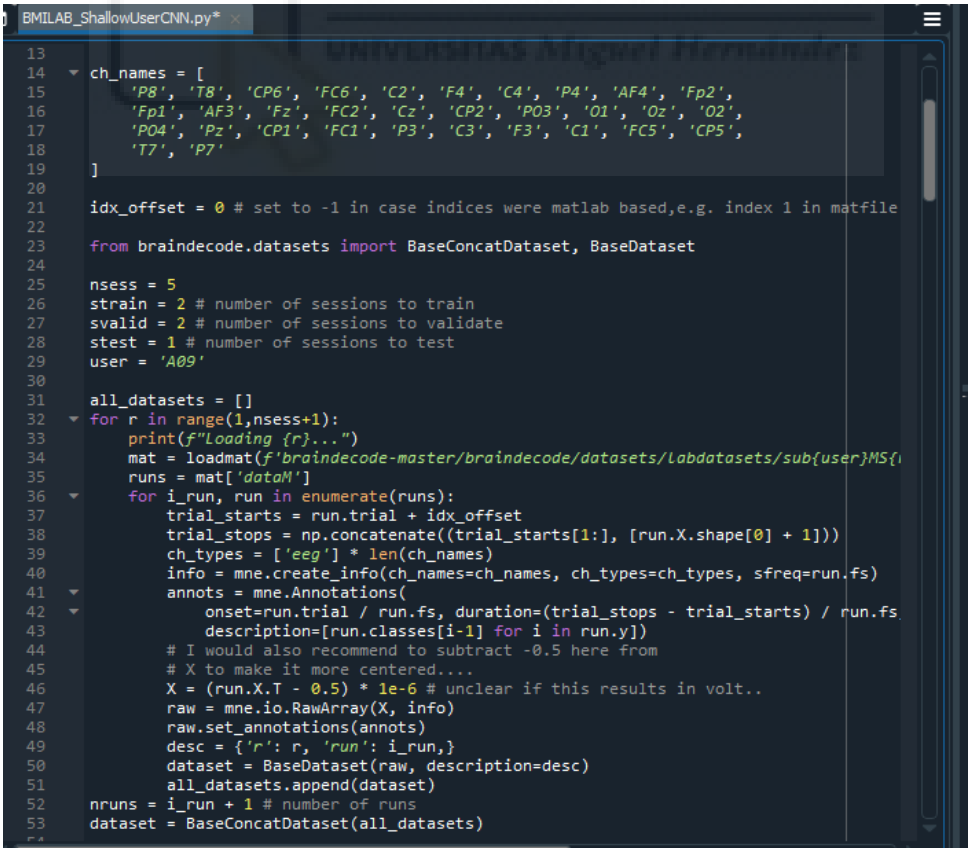


**Figura 12**. Código en Python para cargar un dataset del BMILab

Tras cargar los datasets, estos son preprocesados, se crean las ventanas de entrada y se dividen para el test, la validación y el test. Al mismo tiempo se crea el modelo y, posteriormente, se entrena. Finalmente, la librería ofrece la posibilidad de observar la evolución temporal de la exactitud y la pérdida durante el entrenamiento, pero dado que este trabajo iba a ejecutar cientos de veces el mismo código se decidió prescindir de estas gráficas. Como se explicó en la sección 4.5, el código original no tenía previsto un test para el modelo entrenado ni una forma de almacenar la información temporal, así que ese fue el siguiente objetivo.

Primero se adoptó el módulo *time* disponible en Python para obtener las variables temporales como el tiempo utilizado en preprocesar el dataset del test o el tiempo necesario para cada predicción. Por ejemplo, en la **Figura 13** se puede observar la creación de la variable *prep_time_0* justo antes de crear los parámetros necesarios para el preprocesado y la ejecución de la propia función de preprocesado. Posteriormente, cuando se han creado las ventanas y se conoce el tamaño del dataset, se obtiene el promedio del tiempo necesario para preprocesar una muestra de información mediante dicha variable (**Figura 14**).

```
import time

prep_time_0 = time.time()

low_cut_hz = 4.  # low cut frequency for filtering
high_cut_hz = 38.  # high cut frequency for filtering
# Parameters for exponential moving standardization
factor_new = 1e-3
init_block_size = 1000

preprocessors = [
    # keep only EEG sensors
    MNEPreproc(fn='pick_types', eeg=True, meg=False, stim=False),
    # convert from volt to microvolt, directly modifying the numpy array
    NumpyPreproc(fn=lambda x: x * 1e6),
    # bandpass filter
    MNEPreproc(fn='filter', l_freq=low_cut_hz, h_freq=high_cut_hz),
    # exponential moving standardization
    NumpyPreproc(fn=exponential_moving_standardize, factor_new=factor_new,
        init_block_size=init_block_size)
]

# Transform the data
preprocess(dataset, preprocessors)

prep_time = time.time() - prep_time_0
```

**Figura 13**. Código para el preprocesamiento de los datasets

```
from braindecode.models.util import to_dense_prediction_model, get_output_shape
to_dense_prediction_model(model)

n_preds_per_input = get_output_shape(model, n_chans, input_window_samples)[2]

from torch.utils.data import ConcatDataset

from braindecode.datautil.windowers import create_windows_from_events

# Create windows using braindecode function for this. It needs parameters to define
# trials should be used.
windows_dataset = create_windows_from_events(
    dataset,
    trial_start_offset_samples=0,
    trial_stop_offset_samples=0,
    window_size_samples=input_window_samples,
    window_stride_samples=n_preds_per_input,
    drop_last_window=False,
    preload=True,
)

# Save the average time needed to preprocess a smmple
pwt = prep_time/windows_dataset.cumulative_sizes[len(windows_dataset.cumulative_size
```

**Figura 14**. Código para la creación de ventanas de entrada

```
#########################################################################
# Now we test the trained and validated model in a "pseudo-online" way,
# i.e. the model gets windows of recorded data as if it was used in an
# EEG recording online experiment
#
# With sessiones
test_set = ConcatDataset([r_to_set[i] for i in range(5,6)])
test_data = test_set.datasets[0].datasets[0].windows._data
y_test = test_set.datasets[0].datasets[0].y
if nruns > 1:
    for i in range(1,nruns):
        test_data = np.append(test_data, test_set.datasets[0].datasets[i].windows._d
        y_test = np.append(y_test, test_set.datasets[0].datasets[i].y)

predictions = []
evaltime = []
correct = 0

for i in range(test_data.shape[0]):
    start_time = time.time()

    prediction = clf.predict_proba(test_data[i:i+1,:,:])
    prediction = np.mean(prediction, axis = 2)
    predicted = torch.max(torch.from_numpy(prediction),1)[1]
    predictions.append(predicted)

    correct += (predicted == y_test[i])

    if predicted == 0:
        print('Task detected:  RELAX')
    else:
        print('Task detected:  GAIT')

    evaltime.append(time.time() - start_time)
```

**Figura 15**. Código para el test de un modelo entrenado

```
test_set = ConcatDataset([run_to_set[i].datasets[s-1] for i in range(rtrain+rvalid,
test_data = test_set.datasets[0].windows._data
y_test = test_set.datasets[0].y
```

**Figura 16**. Código para crear el dataset por prueba para test

Por otra parte, en la **Figura 15** podemos observar las líneas de código necesarias para el test de un modelo entrenado. Las primeras líneas separan el dataset destinado al test en la señal EEG recogida por los electrodos y preprocesada (*test_data*) y en las clases correspondientes de cada ventana (y_test). Se crea, además, un bucle for para concatenar la información temporal del resto de pruebas de una sesión. Nótese que estas líneas son para datasets por sesión, los datasets por prueba eliminan la variable de la sesión, acortando las líneas como se aprecia en la **Figura 16**. A continuación, con intención de simular la recolección online de la señal EEG, se crea otro bucle para introducir en cada iteración una ventana de información al modelo, el cual predecirá su clase. Seguidamente, esta predicción es comparada con la clase real, si estas coinciden la variable *correct* se incrementará en 1. Se añadieron, además, líneas de código para mostrar por pantalla el tipo de clase predicha. Finalmente, se almacena el tiempo de la iteración en una matriz que será utilizada para calcular el tiempo total del test y el promedio de cada iteración.

Con esto ya se pudo comenzar a hacer pruebas con diferentes modelos, tamaños de ventana, usuarios, etc. Para automatizar el proceso de experimentación, los códigos se almacenaron como funciones que serían llamadas continuamente como se muestra en la ***Figura 17***. Al final de la ejecución se generaría una hoja de cálculo donde se almacenarían todos los resultados para su posterior análisis.

```
260  win_test = []
261  prepw_time = []
262  samples_win = []
263
264  users = ['A09','A13','B29','B79','B80','B81']
265  sessions = [1,2,3,4,5]
266  epochs = [50,100]
267  windows = [125,250,500,1000]
268  cuda = True
269
270  for e in epochs:
271      for window in windows:
272          for u in users:
273              train = 2
274              valid = 2
275              test = 1
276              ta,tl,va,vl,tt,tat,pa,pt,pat,wt,pwat = shallowusercnn(u,e,window,train,v
277
278              train_accuracy.append(ta)
279              train_loss.append(tl)
280              valid_accuracy.append(va)
281              valid_loss.append(vl)
282              train_time.append(tt)
283              train_avetime.append(tat)
284              test_accuracy.append(pa)
285              test_time.append(pt)
286              test_avetime.append(pat)
287              user.append(u)
288              epoch.append(e)
289              win_test.append(wt)
290              prepw_time.append(pwat)
291              samples_win.append(window)
292
293
294  df = DataFrame({'User': u,'Imput Samples': samples_win, 'Epochs': e, 'Train Accuracy
295  df.to_excel('shallowusercnn.xlsx', sheet_name='Hoja1', index=False)
```

**Figura 17**. Código para automatizar la experimentación de las diferentes configuraciones