Universidad Nacional Autónoma de México

Facultad de Ingeniería

Codificación de señales EEG usando aprendizaje *self-supervised*

TESIS

Que para obtener el título de:

Ingeniero en Computación

P R E S E N T A

Miguel Ángel López Soto

DIRECTOR DE TESIS

Dr. Iván Vladimir Meza Ruiz

Ciudad Universitaria, Cd. Mx., 2022

Agradecimientos

Eternamente agradecido con mi madre por mostrar su amor y apoyo incondicional durante todos los años que convivimos juntos. Fue la persona que estuvo acompañándome en todo momento y guiándome para convertirme en la persona que soy ahora. A pesar de que ya no pudimos coincidir en este momento, siempre estaré agradecido por todas las cosas que hizo por mi, por siempre cuidarme y ser la madre que todos desearíamos tener. Me faltan palabras para describir lo agradecido que estoy con ella pues fue la persona que más he querido y me ha querido en el mundo.

Agradecido con mi padre y mis hermanos por tanto apoyo en los buenos momentos y en los difíciles. Agradecido con mi padre que siempre ha trabajado muy duro para que nunca nos faltara nada ni a mi ni a mis hermanos. Por apoyarme en muchas de mis decisiones y hacerme saber que siempre tendré su apoyo de forma incondicional. A mis hermanos por brindarme esa seguridad, confianza y esa calidez que brinda una familia.

A mis amigos que han estado presentes durante mucho tiempo y en diferentes etapas de mi vida. A aquellos que conocí en la secundaria y que aún seguimos coincidiendo en la vida. A mis amigos de la preparatoria con los que he pasado grandes momentos y que hasta le día de hoy seguimos apoyándonos y compartiendo experiencias de vida. Y a los amigos que conocí en la facultad con quienes he compartido vivencias, intereses y metas en común. Muchas gracias a todos ellos pro ser parte importante de mi vida.

Al Dr Ivan Vladimir Meza Ruiz por apoyarme y guiarme en este proyecto de investigación. Agradecido por todo el conocimiento adquirido a lo largo de todo este tiempo que llevamos trabajando y por compartir ese entusiasmo por temas de inteligencia artificial y aprendizaje profundo. Gracias por brindar esa confianza y por compartir el conocimiento y pasión por estos temas además de servir de guía en mi vida académica.

Resumen

La interpretación de señales cerebrales (EEG) es un reto ya que requiere de expertos en el tema para extraer información de dichas señales. Éstas señales pueden indicar problemas médicos por lo que su correcta interpretación ayuda a determinar la situación del cerebro en el individuo.

El objetivo de este trabajo es explorar el uso de redes neuronales y aprendizaje *self-supervised* para la codificación de señales EEG.

Usando un conjunto de datos extraído un experimento donde se le aplicaron una serie de estímulos visuales a 22 personas neurotípicas durante 72 segundos.

Para alcanzar el objetivo se utilizó un método de aprendizaje llamado *self-supervised* que consiste en agrupar datos que tienen información similar a través de una tarea llamada *espuria* para obtener representaciones vectoriales de éstas señales. La arquitectura propuesta es una red neuronal de tipo siamesa con las siguientes variantes:

1. *Fully Conected*.

2. *Convolucional 1D*

3. *Convolucional 2D*

Además de proponer diversas tareas espurias para el tipo de aprendizaje que se está utilizando.

Como resultado se obtienen un conjunto de modelos que logran generalizar poco con un *accuracy* máximo de 0.79101 en el conjunto de datos de prueba. Además explorando el uso de una herramienta para la optimización de hiperparámetros y realizar un análisis de los hiperparámetros mas importantes en los diferentes tipos de redes neuronales.

La poca generalización de la red neuronal puede deberse a estos diferentes factores:

1. Complejidad de la tarea: Al proponer diferentes tareas, puede que la red neuronal no haya sido capaz de «aprender» la tarea propuesta por la dificultad que esto implica ya que no hay patrones que relacionen los datos o no los pudo encontrar.

2. Datos: Los modelos presentaron un sobreajuste significativo a pesar de incluir técnicas de regularización agresivas, esto puede indicar que el dominio de los datos o los datos no son suficientes para poder lograr una generalización.

3. Arquitectura: Trabajar con *contrastive self supervised learning* es complicado ya que durante el entrenamiento el modelo puede colapsar, lo que significa que los vectores generados ocupan el mismo punto o espacio vectorial [1]. Es un problema conocido y actualmente hay algunas propuestas para evitar este colapso.

Utilizar *contrastive self supervised learning* resulta interesante por las ventajas que tiene sobre los métodos *no contrastive* como las funciones *Cross Entropy* para problemas de clasificación, pero como resultado de estos experimentos puede ser viable el uso de éste método para entrenar redes neuronales que ofrezcan representaciones vectoriales útiles para diferentes tareas del área, con un conjunto reducido de datos y diferentes experimentos, se logra obtener un poco de información acerca de éstas señales.

Índice general

Índice de tablas

Índice de figuras

1 Introducción

Hacer uso del aprendizaje automático ha facilitado la exploración de los datos para obtener información útil. En este trabajo se explora el uso de *deep learning* para realizar una codificación de de las señales cerebrales EEG (Electroencefalograma) utilizando aprendizaje *self-supervised* para obtener representaciones vectoriales que permitan realizar tareas como clasificación.

1.1. Objetivo

Codificar señales EEG usando aprendizaje auto-supervisado para extraer características de la señal.

1.2. Definición del problema

La extracción de características es una parte importante cuando se quiere analizar ciertos fenómenos relacionados a diversas señales. En el caso de las señales EEG a priori no hay suficiente información que indique alguna característica importante relacionada al fenómeno que se quiere estudiar como por ejemplo la diferencia entre señales dado un estímulo visual. Usar algoritmos de *deep learning* facilita la tarea de búsqueda de patrones ya que la red neuronal se encarga de extraer características y en el caso del aprendizaje *contrastive self-supervised learning* ayuda a crear una codificación vectorial que permite utilizar el modelo en diferentes tareas, por ejemplo, clasificación de emociones.

1.3. Método

Para esta extracción de características se pueden generalizar dos tipos:

Métodos clásicos:

Se refieren a obtener características y así obtener una codificación de la señal utilizando algún tipo de transformación y diversos parámetros y cálculos que permiten disminuir las entradas de tiempo/amplitud. Un ejemplo de este tipo de métodos es obteniendo una transformada wavelet de la señal para así tener información sobre la

frecuencia y el tiempo y de esta manera realizar un análisis por ejemplo de energía para extraer características de dicha señal.

Métodos basados en aprendizaje automático:

En este tipo de métodos se busca que el sistema aprenda a reconocer patrones en los datos y de esta forma codificar la señal de entrada. Principalmente se han usado en los últimos años las redes neuronales para esta tarea, ya que hay poder computacional para entrenar este tipo de redes y una gran cantidad de datos con los que se puede extraer información relevante. Además de arquitecturas especializadas en diversos tipos de datos.

Dada la practicidad de los métodos basados en aprendizaje automático, este método es el principal objetivo. Dentro de estos métodos se utilizará aprendizaje autosupervisado ya que:

1. Se quiere extraer características de los datos para después ser usados en otras tareas

2. No se tiene una gran cantidad de datos como para recurrir al aprendizaje supervisado pero se pueden generar datos para trabajar con aprendizaje autosupervisado.

3. Se pueden analizar las diferentes estructuras internas de los datos. La arquitectura general de la red se le conoce como red siamesa la cual agrupa señales similares y aleja señales que no comparten características similares. Para entrenar la red se hace uso de una tarea de pretexto, la red aprende esta tarea y finalmente se tiene un vector de características de dicha señal.

En la estructura interna de la red se hace uso de capas de una red neuronal convolucional que permite "aprender filtros" y de esta forma procesar la señal múltiples veces encontrando de esta forma características diferentes.

1.4. Estructura de la tesis

1. Trabajos previo: Se presenta una breve introducción al aprendizaje automático y aprendizaje profundo.

2. Metodología: Se presentan las arquitecturas, datos y estructura del software utilizado.

3. Experimentos y resultados: Se muestran los resultados obtenidos por cada arquitectura y la optimización realizada.

4. Conclusiones

1.5. Resultados Esperados

Una red neuronal que sea capaz de codificar un fragmento de una señal EEG y de esta manera extraer diversas características.

2 Trabajo anterior

Para comprender la importancia de este trabajo se presenta que son las señales EEG y el aprendizaje automático que es la clave para tratar el problema de la codificación de señales.

2.1. EEG

Para el estudio de diversas partes del cuerpo se tienen señales producidas por los órganos implicados en el proceso. En el cerebro existen impulsos eléctricos viajando por ciertas partes de éste y por todo el conjunto del sistema nervioso central. Para el estudio de estas señales eléctricas se tiene como método de estudio en el que se usa un instrumento llamado EEG (Electroencefalograma) para amplificar dichas señales a través de electrodos colocados en diferentes partes del cuero cabelludo [12].

Los registros permiten realizar diferentes estudios como por ejemplo el sueño, efectos de la anestesia, diagnostico de epilepsia, trastornos mentales, entre otros. Las señales dan información acerca de todos los procesos cognitivos asociados a los estímulos que recibidos por el sujeto en cuestión. La adquisición de estas señales se hace mediante un dispositivo llamado electroencefalógrafo que para capturar las señales hace uso de distintos electrodos.

Los electrodos son pequeños dispositivos que permiten conducir la electricidad a través de una placa metálica, además en algunos casos transformar las corrientes iónicas del cuerpo a señales eléctricas [13]. Los electrodos pueden estar hechos de diferentes materiales que sean capaces de generar el suficiente potencial para el registro de la señal. De esta manera se permite detectar estas variaciones y registrarlas para su análisis posterior [13].

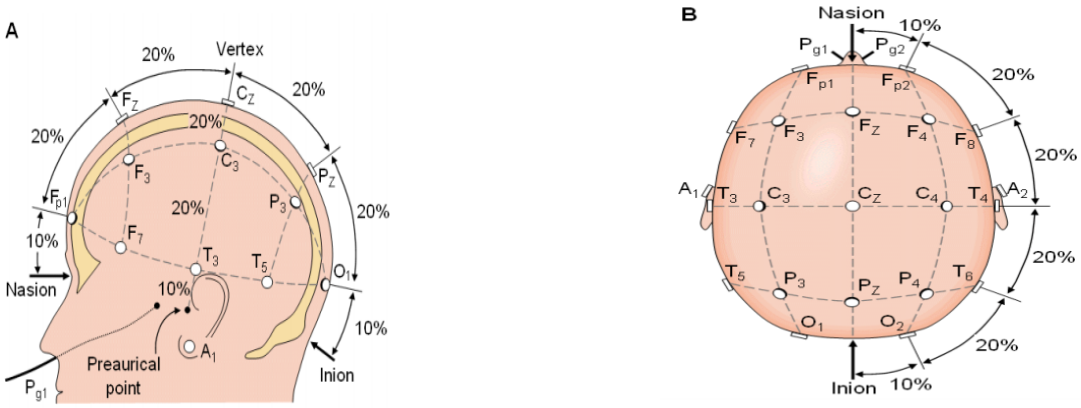
Existen diversos tipos de electrodos, los más comunes son los superficiales ya que no necesitan intervención quirúrgica. Estos dispositivos deben de estar preparados para no producir interferencia en el registro de la señal [13].

Los electrodos se pueden colocar de diferentes formas pero para estandarizar los experimentos se tiene el estándar internacional 10-20. Los números (10 y 20) se refieren a la distancia que hay entre cada uno de ellos de forma porcentual y por cada lóbulo establecido por el estándar. La nomenclatura se puede observar en la Tabla 2.1.

Electrodo	Lóbulo
F	Frontal
T	Temporal
C	Central
P	Parietal
O	Occipital

Tabla 2.1: Tabla de nomenclatura en el estandar 10-20.

Además con este estándar se definen números de electrodos los cuales 2,4,6,8 pertenecen al hemisferio derecho mientras que los electrodos con números 1,3,5,7,9 pertenecen al hemisferio izquierdo. Esto se ilustra en la Figura 2.1 [2].



(a) Diagrama A.

(b) Diagrama B.

Figura 2.1: Diagramas de colocación del estándar 10-20. [2]

Cuando se adquiere la señal por algunos de los electrodos, esta debe de ser amplificada y registrada en la computadora para su posterior análisis.

Dentro de estas señales se tiene información para detectar por ejemplo algún trastorno mental pero debido a que se trabaja con seres vivos y en consecuencia estamos en constante movimiento, se logran captar algunos artefactos que agregan ruido a las señales [13].

Los artefactos en este contexto son señales no deseadas durante el experimento. El movimiento de los músculos de la cara y otras extremidades provoca que se registre actividad en los electrodos por lo que es importante identificarlos y de ser posible eliminarlos.

Algunas de estas pueden ser controladas pero otras son parte del funcionamiento del cuerpo humano. Algunos ejemplos de artefactos son:

1. Pestañeo.

2. Movimiento de cabeza.
3. Movimiento de piernas.
4. Movimiento de dedos.
5. Movimiento de brazos.
6. Inhalación y exhalación.

Algunos artefactos se realizan de forma involuntaria por lo que se pueden eliminar para obtener una señal más limpia, a través de filtros digitales aunque esto no suele ser muy preciso. Por ello en algunos casos se necesita el trabajo de un experto para identificar artefactos y eliminarlos o etiquetarlos para su posterior análisis. Un ejemplo de artefacto se puede observar en la Figura 2.2.

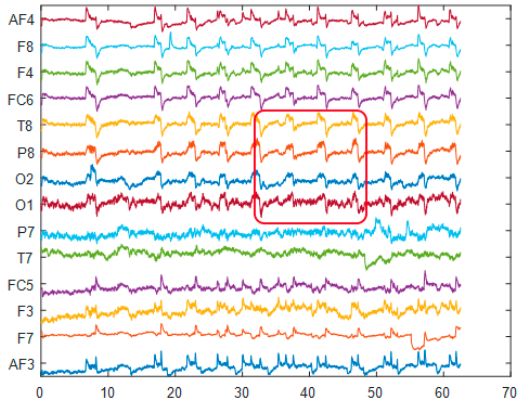


Figura 2.2: Ejemplo de registro EEG en el proceso de parpadeo.

La actividad cerebral es difícil de registrar a altas frecuencias debido a limitaciones técnicas aunque un dispositivo actual puede registrar señales desde 128 Hz hasta 512 Hz en un análisis de rutina, en investigación se llegan a usar hasta 10 KHz de frecuencia de muestreo [14].

Aunque los electroencefalógrafos pueden captar un amplio rango de voltajes,los rangos de las señales dependen de diferentes factores, la amplitud media es de 50 μV para personas jóvenes sin embargo varia dependiendo de factores ambientales como los artefactos, la actividad del sujeto y si tiene alguna patología. En general se puede definir un rango entre 10 μV y 100 μV [15].

2.2. Aprendizaje automático

Para resolver un problema en especifico como en el área de reconocimiento de patrones, se necesita de un análisis exhaustivo del problema para encontrar algoritmos y transformaciones matemáticas que permitieran procesar los datos de entrada para

así poder dar una solución al problema. Esto supone tener expertos en el área para investigar y probar diferentes métodos que permitieran crear algoritmos funcionales y de ser posible eficientes.

Este enfoque cambia completamente con el aprendizaje automático, ya que la idea detrás de esta rama de la inteligencia artificial es tener algoritmos que puedan aprender de los datos.

Dejar que la computadora aprenda suena muy complicado ya que necesitamos un sistema que sea capaz de aprender, mejorar con el tiempo y tenga una respuesta rápida a lo que esta tratando de modelar.

En 1981 Gerald Dejong trabajaba sobre una forma de aprendizaje automático con la cual consideraba 4 aspectos importantes [16].

1. Dominio: contiene el interés especifico de lo que se busca.
2. Espacio de Hipótesis: Son todas las posibles soluciones al problema o modelo.
3. Datos: Son los ejemplos con los que se pueden encontrar los patrones.
4. Criterios de Operatividad: Determina cuáles son las características reconocibles de los datos, por ejemplo, qué datos podemos obtener de uno o más sensores.

En años posteriores hubo avances importantes pero no se conocía suficiente el potencial de esta rama de la inteligencia artificial.

Las empresas de tecnología fueron importantes para darle el impulso que necesitaba el aprendizaje automático ya que por ejemplo en 2008 Microsoft lazó su servicio de Azure Machine Learning, de ahi siguieron más empresas lanzando sus productos y servicios como IBM con Watson que es un sistema capaz de responder preguntas en lenguaje natural.

Esto fue desencadenando más y más compañías que se volvían expertas en esta área y una inversión millonaria para promover la investigación [16].

Los aspectos antes mencionados eran una parte importante en los algoritmos de aprendizaje automático.

Los algoritmos tienen que enfrentar a los problemas de aprendizaje y como este término resulta algo complicado de definir, en el área se toma como "La capacidad de un sistema de mejorar en una tarea especifica con el tiempo". Y esta simplificación trae aspectos importantes que a tratar en temas de aprendizaje automático [16]. Teniendo el concepto de aprendizaje podemos destacar ciertos elementos importantes.

1. Tarea: El problema a resolver.
2. Medida de desempeño: El modelo debe de aprender con el tiempo pero ¿Cómo sabemos si esta aprendiendo? Esto lo podemos tomar con una medida que nos indique si el modelo esta realizando la tarea correctamente.

3. Experiencia: En general los datos con los que vamos a trabajar.

Esta es una aproximación bastante más clara ya que en general se sigue una metodología similar en todas las tareas relacionadas al campo.

Es importante destacar que una interpretación de los resultados esta dada por la probabilidad y estadística ya que al ser un campo donde convergen muchas disciplinas se intenta ver a un modelo de aprendizaje automático como una distribución probabilística que debe ser lo más parecida a la distribución de la vida real. Esto depende de la perspectiva en la que se mire puesto que se pueden encontrar otras interpretaciones.

Los algoritmos pueden servir para resolver múltiples tareas pero dependiendo de los datos y de otras características se pueden definir ciertas categorías dentro del aprendizaje. Dependiendo de la taxonomía que se consulte pueden haber diferentes clasificaciones entre algoritmos de aprendizaje automático, aunque los principales son [17]:

1. Aprendizaje supervisado: consiste en utilizar datos etiquetados para que el sistema pueda identificar patrones en los datos que permitan producir etiquetas para una entrada nueva.

2. Aprendizaje no supervisado: aquí se trabaja con datos no etiquetados, por lo que el algoritmo debe revisar diferencias o similitudes entre los datos para poder segmentarlos.

3. Aprendizaje reforzado: se trata con datos no etiquetados y lo que se pretende es maximizar una recompensa en la tarea con la que se pretende entrenar.

4. Aprendizaje semi-supervisado: esta es una mezcla entre el aprendizaje supervisado y no supervisado ya que los datos etiquetados muchas veces son escasos por lo que los modelos se preparan para trabajar con los dos tipos de datos.

Aprendizaje Supervisado

Este tipo de aprendizaje requiere de datos etiquetados para entrenar a un modelo que permita, dada una entrada desconocida, predecir una etiqueta establecida en los datos de entrenamiento. Esto se realiza principalmente en clasificadores pues es aquí donde se puede explotar este potencial de los datos [17].

Los algoritmos tratan de separar las clases como según esté diseñado el algoritmo y esto puede ser basándose en ajustar probabilidades o encontrar una función matemática que permita determinar a que clase pertenece la entrada dada [17].

Entre los algoritmos principales están:

1. Clasificadores Bayesianos.

2. Regresión Logística.

3. Maquinas de Soporte Vectorial.

4. Redes Neuronales.

Aprendizaje no Supervisado

En este tipo de aprendizaje no hay datos etiquetados, por lo cual se tuvo que idear otra forma de trabajar con este tipo de datos. Aquí se trabaja con similitudes. Estas similitudes se pueden calcular de diferentes formas una de ellas es utilizando una métrica con la cual se pueda ubicar en el espacio los datos con los que se quiere trabajar y la distancia a la que están dichos datos. Entonces mientras más cercanos estén los datos, más similares son. Este tipo de algoritmos se les conoce como algoritmos de *clustering* y sirven para agrupar datos que a priori no se puede tener idea sobre como están conformados los datos y podemos descubrir patrones ocultos entre los datos [17]. Otra forma de separar los datos es utilizando la correlación que hay entre algunas variables. Entre los algoritmos principales están:

1. *Clustering* Jerarquico.
2. K-Means (Clustering)
3. PCA (*Principal Component Analysis*)
4. Algoritmo Apriori (Reglas de Asociación)

Aprendizaje Reforzado

Este tipo de aprendizaje se diferencia de los anteriores en el aspecto en que no se necesitan pares entrada-salida, si no que se entrena a un modelo basándose en el concepto de «prueba y error» aplicado entre agentes y entornos. Este tipo se puede relacionar al aprendizaje que una persona podría desarrollar, si la acción a la tarea propuesta es buena entonces se le da una recompensa al sujeto mientras que si dicha acción resulta ser errónea a ojos del observador se le da un castigo [17].

Esto hará que en seres vivos, por ejemplo, se entrene para realizar de manera satisfactoria una tarea. Este concepto se aplica también en el aprendizaje automático como se mencionó anteriormente se tienen cuatro elementos principales:

1. Agente: Aquel sistema que se entrena para realizar una tarea.
2. Ambiente: El mundo en el que el agente interacciona.
3. Acción: Corresponde a la acción del agente según su entorno.
4. Recompensa: La evaluación que se le hace al agente por su acción, si fue positiva o negativa se le hará saber mediante un valor.

Esto se puede aplicar a diversos problemas que no requieran tener presentes una etiqueta en sus datos y en los que se tenga una métrica para comprar si la acción del agente es correcta o no. Un ejemplo es en los videojuegos, se tiene un agente que es el personaje, su entorno es precisamente el ambiente en el que se puede mover

e interactuar, la acción son las actividades que realiza el personaje al pulsar por ejemplo una tecla y la recompensa que puede ser mediante puntos o la pantalla de *Game Over* [17]. Entre los algoritmos más importantes en este ámbito están:

1. *Q-Learning*
2. *Policy Optimizations*
3. *Temporal difference*

Aprendizaje Semi-Supervisado

Dado que encontrar datos etiquetados es una tarea que requiere de mucho esfuerzo y tiempo de humanos para estar etiquetando por ejemplo imágenes. Existen por ejemplo micro tareas en donde se le pide a un humano categorizar, transcribir o realizar algún tipo de etiquetado en los datos pero esto solo es algo que las grandes empresas se pueden permitir para entrenar modelos más avanzados. Por el contrario, se pueden encontrar muchos datos sin etiquetar en diferentes repositorios públicos como internet por lo que ¿Por qué no mezclar las ventajas del aprendizaje supervisado y no supervisado? De esto se trata el aprendizaje Semi-supervisado, la idea general es tomar un pequeño grupo de datos etiquetados y realizar clustering para ver que tipos de datos son similares y gracias a las etiquetas que se tienen en ciertos datos etiquetar los que estén más cercanos con dicha etiqueta [17]. Con esto debemos asumir las siguientes premisas:

1. Los puntos que estan cercanos unos de otros comparten la misma etiqueta
2. Los datos se pueden dividir en diferentes clusters y que estos comparten dicha etiqueta
3. Los datos se pueden representar en una dimensionalidad menor lo que permite el uso de distancias y densidades para realizar la tarea.

Este tipo de algoritmos se usan por ejemplo para clasificar contenido de internet. Existen diversos tipos de algoritmos por ejemplo basados en grafos o modelos generativos.

Finalmente se puede observar un resumen de algoritmos de machine learning en la Figura 2.3.

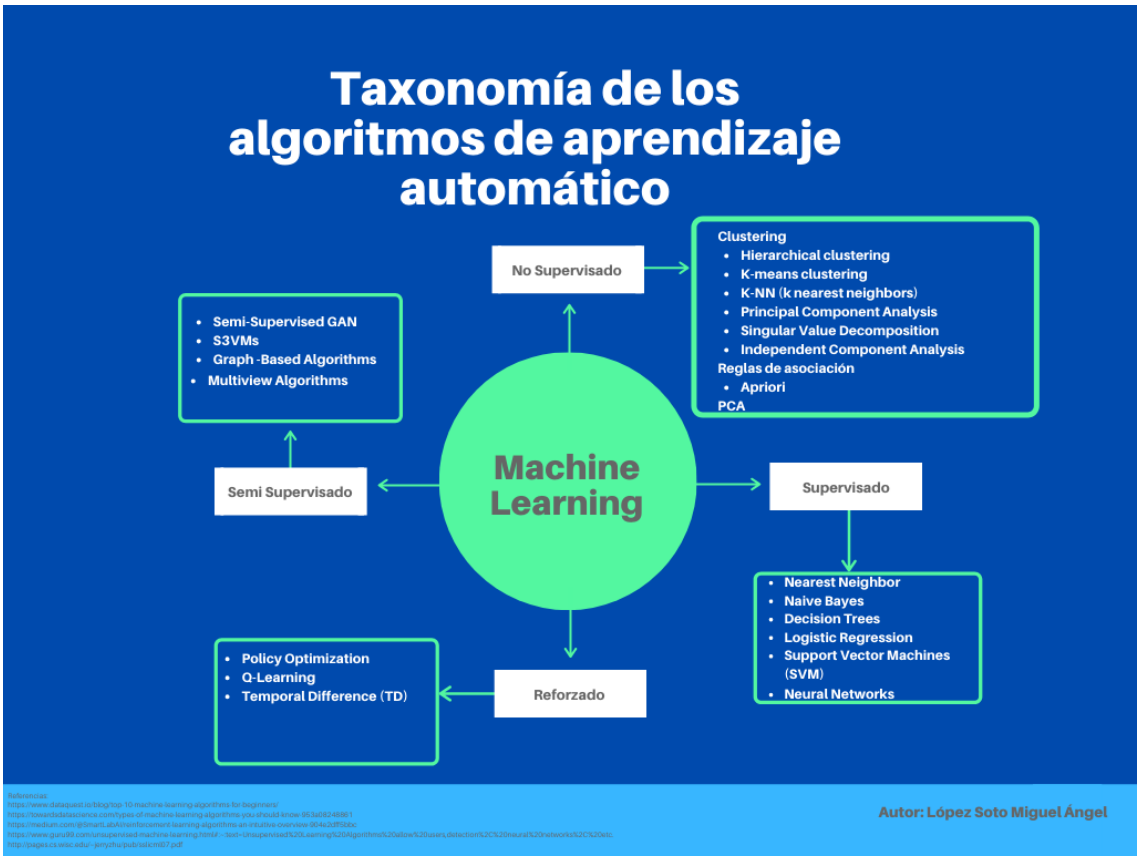


Figura 2.3: Taxonomía de algoritmos de *Machine Learning*

2.3. Aprendizaje profundo

El aprendizaje profundo esta inspirado en el cerebro humano, esto es con sus diferencias pero inicialmente se parte de la idea en cómo las neuronas del cerebro funcionan.

Las neuronas son células del sistema nervioso que se encargan de regular el estado general del sistema nervioso. Estas consisten en un cuerpo celular y dos componentes adicionales llamados:

1. Dendritas: Reciben señales químicas del axón de otras neuronas
2. Axón: Su función es transmitir información de la neurona a otras con las que esté conectada.

Este proceso de recibir información y transmitirla es lo que se le llama sinapsis. Estas neuronas están organizadas en complejas cadenas y redes que conforman nuestro sistema nervioso [3].

Esto se puede resumir en la Figura 2.4.

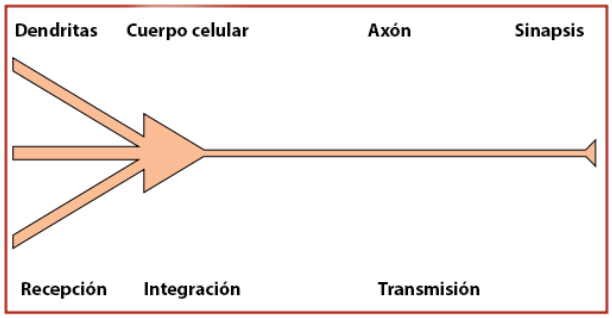


Figura 2.4: Diagrama general del funcionamiento de una neurona. [3]

Esto inspiró a Warren McCulloch y Walter Pitts en 1943 a proponer un modelo que imitaba el comportamiento de una neurona real llamada neurona artificial. Cada neurona era una unidad de umbral con dos estados de encendido y apagado. Se puede observar el modelo en la Figura 2.5. Posteriormente Frank Rosenblatt desarrollaría

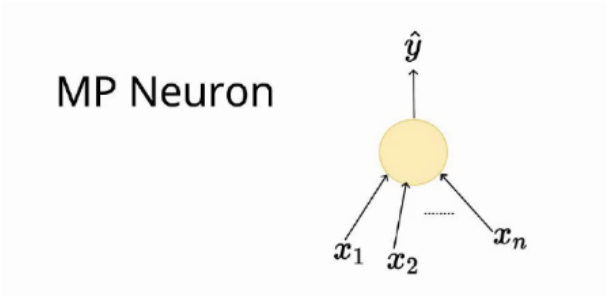


Figura 2.5: Modelo de Neuronal Artificial [4].

una versión algorítmica de dicha neurona artificial, añadiéndole un peso a cada entrada por lo que el modelo podría trabajar con valores no booleanos. Los pesos se multiplican con la entrada y si superan cierto umbral la neurona se activa de lo contrario permanece apagada [**?**].

Se pueden generalizar cuatro aspectos importantes en este tipo de neuronas artificiales:

1. Entradas(x): todos los valores de entrada a la neurona.
2. Pesos (w): Representan la fuerza de la conexión con respecto a la entrada.
3. Sesgo (b): Representa el umbral que determina si la neurona se activa o no.
4. Función de activación: Es una función que discrimina si la neurona se activa o no dependiendo de las entradas para producir una salida.

Se puede observar el modelo de Rosenblatt en la Figura 2.6.

Este modelo es bastante útil para clasificar datos linealmente separables esto quiere decir que en un espacio de $\mathbb{R}^{\times}$ hay una recta o hiper plano que logra separar los datos

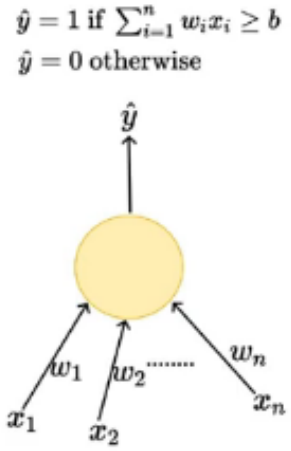


Figura 2.6: Modelo Neuronal Artificial de Rosenblatt [4].

en dos categorías. Dada esta restricción existen problemas linealmente no separables como el problema del XOR. El perceptrón multicapa rompe esta restricción añadiendo capas de neuronas artificiales conectadas entre si, esto hace una arquitectura más robusta y que permite manejar problemas de clasificación no lineales ya que permite generar conexiones más complejas [18].

Esta arquitectura se puede visualizar mejor en la Figura 2.7

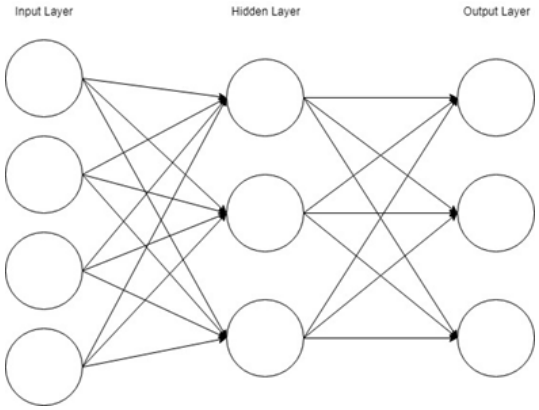


Figura 2.7: Perceptron multicapa

Durante el siglo XX el desarrollo en el campo fue perdiendo interés por las limitaciones técnicas de la época. Las operaciones con punto flotante eran costosas y dado que se usaban neuronas binarias, no se podía aplicar el algoritmo de *backpropagation*, ya que un aspecto importante es que las funciones de activación deben ser diferenciables [19].

En el capitulo anterior, las redes neuronales se pueden considerar como modelos enfocados al aprendizaje supervisado ya que necesitamos de una etiqueta para saber si la red esta aprendiendo o no y de esta manera ajustar los parámetros de la red neuronal.

Trabajando con esta arquitectura, se puede generalizar el perceptrón multicapa y en este sentido cada capa puede ser considerada un módulo y cada módulo tiene parámetros ajustables y no linealidad. A apilar este tipo de módulos precisamente se le llama aprendizaje profundo. Además una de las razones por las que se trata de buscar que cada módulo sea no lineal, es que si dos capas son lineales el resultado va a ser lineal.

Generalizando todo el proceso, se tiene un vector de entradas que se multiplica por una matriz de pesos (estos pesos son parámetros ajustables de la red), al vector resultante se le aplica una función no lineal y este proceso se repite entre todos los módulos de la red neuronal.

Para ajustar los pesos se recurre al aprendizaje supervisado en donde mediante una función de error o coste se tiene que ir minimizando para que de esta forma se puedan ajustar los parámetros de la red neuronal.

Se puede separar los componentes principales del aprendizaje profundo en 4 áreas activas de investigación:

1. Modelos: Tipos de redes que permiten manejar cierto tipo de datos.
2. Optimizadores: Son los algoritmos que permiten el cálculo de los parámetros ajustables de la red.
3. Funciones de activación: Funciones que permiten realizar la separación de clases de forma óptima.
4. Funciones de error: Permiten identificar el error que esta cometiendo la red al realizar una tarea.
5. Regularización: Permiten manejar el sobre-ajuste del modelo.

Modelos

Es un área bastante activa de investigación ya que se han propuesto diversas arquitecturas que permiten una mejor extracción de características y a su vez mejoran la relación que hay entre los datos para que de esta forma se pueda generalizar el conocimiento que tiene la red sobre los datos con los que se esta probando.

En este sentido, se puede mencionar la generalización del perceptrón multicapa que son las redes llamadas *Deep Feed Forward*.

Redes Feed Forward

La finalidad de este modelo es que la red aprenda a aproximar una función $f(x)$, que por ejemplo mapee el vector de entrada x a un valor de una categoría y establecida. Es entonces que se ajustarán los parámetros para obtener la mejor aproximación de esta función. Se le llama de tipo *Deep Feed Forward* ya que la información fluye hacia adelante en donde la salida de un módulo es la entrada de otro hasta que se da una salida y. Este tipo de redes son muy usadas por la capacidad de cambiar los datos para encontrar un espacio vectorial en el que se cumpla la tarea objetivo [20].

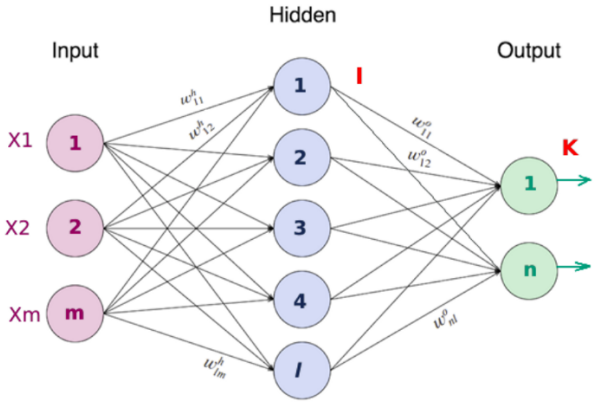


Figura 2.8: Figura ejemplo de red *feed forward* [1]

Redes Neuronales Convolucionales

Este tipo de redes son usadas para procesar datos que tienen una topología de red o señales en el tiempo. Utilizan una operación matemática llamada convolución que es una operación de dos funciones e indica la cantidad de superposición que hay entre una función g desplazada con respecto a otra función f. La operación se denota como $f * g$ y se define como:

$$f * g = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.1}$$

Siendo f y g funciones reales [21].

Este tipo de operación se utiliza en el procesamiento de señales. En *deep learning* se usa para crear tensores que sirven como filtros que al convolucionarlos con una señal de entrada permite obtener otra señal de la cual se pueden extraer características. Dado que se trabaja con valores discretos en vez de una integral se tiene la suma de

[1]Figura tomada de *Neural Networks (Part 1)* `https://medium.com/swlh/neural-networks-4b6f719f9d75` visto en junio de 2021.

convolución. Esta suma de convolución puede ser en una o más dimensiones. La expresión para la suma de convolución en una dimensión se puede ver en la ecuación 2.2 mientras que la convolución en dos dimensiones se observa en la ecuación 2.3

$$x * w(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.2}$$

$$K * I(i,j) = \sum_{m}\sum_{n} I(i-m, j-n)K(m,n) \tag{2.3}$$

En una dimensión se puede trabajar con series de tiempo a las cuales se les puede aplicar una serie de filtros para obtener señales que permitan por ejemplo a una red *fully conected* realizar una clasificación.

En dos dimensiones su aplicación más directa es en el procesamiento de imágenes, ya que de forma tradicional se definen filtros que finalmente son matrices que se convolucionan con una imagen para obtener una imagen emborronada o los bordes de una imagen. Matemáticamente se define con una convolución en dos dimensiones. Este tipo de redes han tenido éxito gracias a su gran facilidad de adaptación a las imágenes.

De forma simple y comparada con las redes de tipo *fully conected* en vez de modificar una matriz de pesos se modifica una serie de matrices que funcionan como filtros y que permiten extraer características a diferentes niveles de la imagen. Por ejemplo se puede definir un filtro laplaciano que extrae los bordes de una imagen, en este sentido la red «aprende» a generar estos filtros. Un ejemplo de estos filtros se observa en la Figura 2.9

$$\begin{bmatrix} 2 & 0 & 2 \\ 0 & -8 & 0 \\ 2 & 0 & 2 \end{bmatrix}$$

Figura 2.9: Filtro Laplaciano.

La aplicación de un filtro laplaciano a una imagen se observa en la Figura 2.10

(a) Imagen Original.

(b) Imagen aplicando un filtro laplaciano.

Figura 2.10: Aplicación de un filtro a una imagen.[2]

En arquitecturas que utilizan redes neuronales convolucionales las capas normalmente tienen tres fases: se realiza la convolución, se le aplica una función no lineal y se le hace un *pooling*. El *pooling* es una función que modifica la capa de salida con un resumen estadístico de las salidas cercanas. Esto es por ejemplo el *max pooling* toma de un conjunto de valores cercanos a un punto el valor máximo. El *pooling* hace que la representación intermedia sea invariante a pequeñas traslaciones. [20] Otro tipo de redes neuronales que permiten trabajar con datos estructurados en el tiempo como las redes de tipo recurrentes,*Long Short Term Memory*, que generan datos como los *Auto Encoders* o generan distribuciones de probabilidad como las *Variational Auto Encoders* o que trabajando con conceptos de atención como lo son *transformers*. Existe una gran diversidad de tipos de redes neuronales por lo que abarcar todos los tipos es una tarea extensa. Finalmente se puede ver un resumen de los tipos de redes neuronales más usados en la Figura 2.11.

[2]Imagen modificada de *Instagram: ChykeEGL* `https://www.instagram.com/p/BsYc4sChTfZ/` visto en junio de 2021.

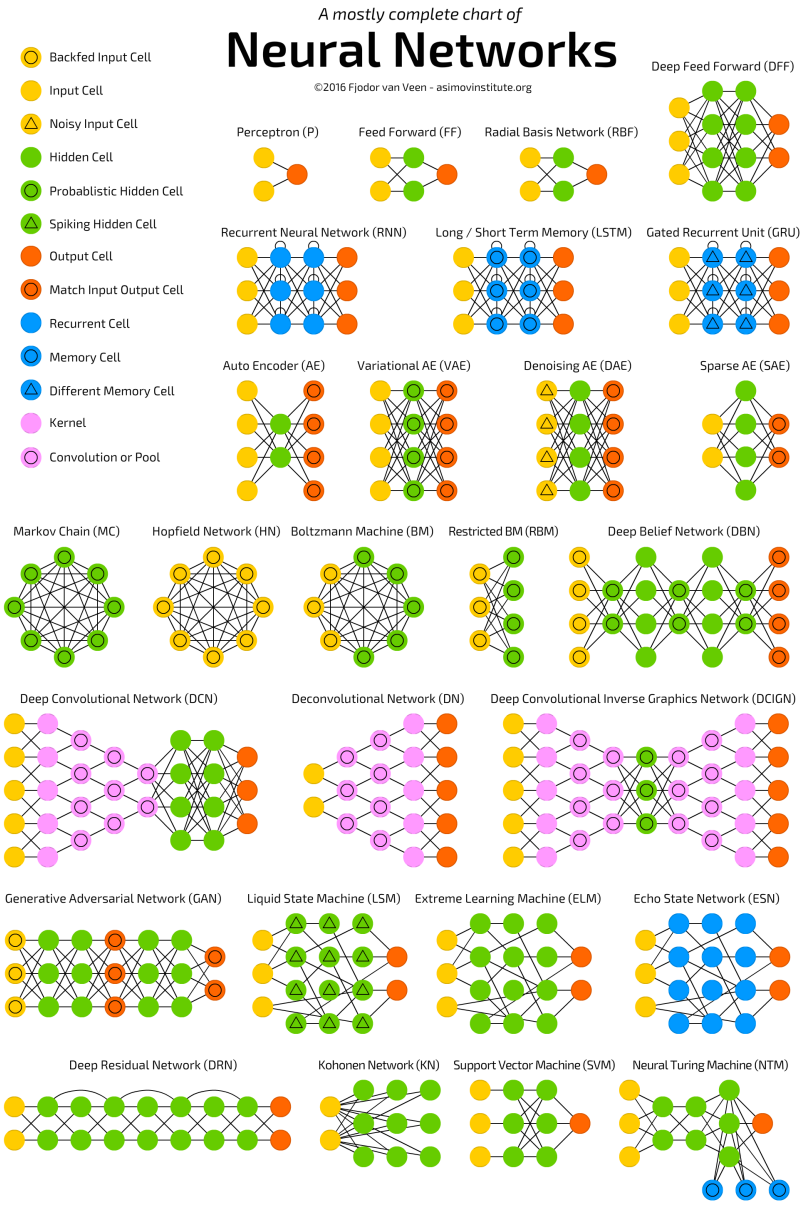


Figura 2.11: Tipos de Redes Neuronales. Tomado de *The Neuronal Network Zoo*[3]

Funciones de activación

Una característica importante en las redes neuronales es que permiten transformar los datos de entrada de forma que sean separables por una capa de la misma red neuronal. Este es un paso importante ya que como se mencionó con el perceptrón multicapa, si no se tiene una función de activación las capas de la red funcionaran solo como transformaciones lineales de los vectores de entrada y para datos complejos no se van a poder separar en las clases definidas. Una de las primeras funciones de

[3]Entrada de blog *The Asimov Institute Blog The Neuronal Network Zoo* `https://www.asimovinstitute.org/author/fjodorvanveen/` visto en agosto de 2021

activación usada para entrenar redes neuronales fue la función sigmoide 2.4.

$$s(x) = \frac{1}{1 + e^{-x}} \tag{2.4}$$

Esta función de activación fue utilizada gracias a que su derivada se puede expresar en términos de la misma función, esto facilita el proceso del gradiente descendiente.

Actualmente se pueden seguir usando este tipo de funciones pero empíricamente la función ReLu o rectificador lineal ha demostrado obtener buenos resultados al entrenar redes neuronales. La función relu se define de la siguiente manera:

$$ReLu(x) = max(0, x) \tag{2.5}$$

Adicionalmente gracias a la versatilidad de esta función de activación se propuso la función GELU (Unidad Lineal de Error Gausiano) que mejora en desempeño con respecto a la función ReLU. Como se puede ver en la Figura 2.12 la función GELU hace que la red neuronal pueda converger más rápido que con otras funciones de activación [5]. La función GELU se puede definir como:

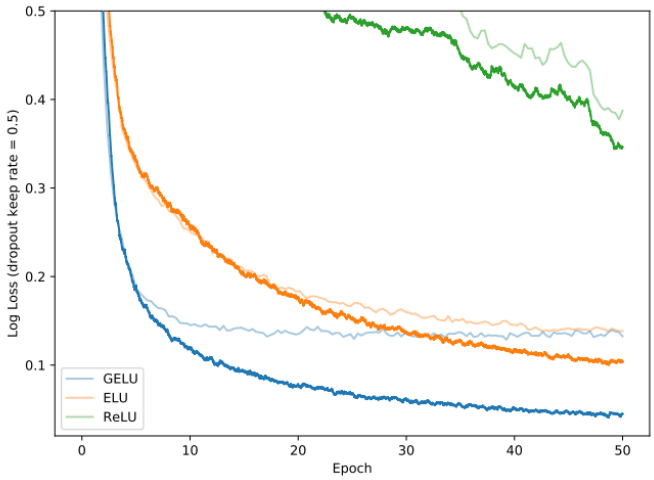


Figura 2.12: Gráfico comparativo entre la función de activación usada y la pérdida al entrenar una red neuronal [5].

$$GELU(x) = xP(X) = x\phi(x) = x\frac{1}{2}[1 + erf(x/\sqrt{2})] \tag{2.6}$$

Otra función de activación comúnmente usada es la función *Softmax* que dado un tensor mapea los valores a una distribución de probabilidad esto es que por ejemplo en un vector de N componentes la suma de todos los valores al aplicar la función softmax es igual a 1. Esto es muy útil cuando se quieren estimar probabilidades de de una clase en la que se este entrenando la red neuronal. La función *Softmax* se define como:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} paraj = 1, ..., K \tag{2.7}$$

Finalmente, elegir la función de activación adecuada es fundamental para que la red neuronal pueda converger de forma más rápida lo que conlleva menor tiempo y recursos.

Funciones de error

Las funciones de error son otra parte fundamental en el diseño y entrenamiento de las redes neuronales ya que son las funciones que se van a tener que optimizar para obtener un error mínimo en la tarea de la red neuronal.

El objetivo de un algoritmo de aprendizaje automático es reducir el error dada una función de riesgo (también llamada función de coste o *loss*). La manera sencilla de convertir un problema de aprendizaje automático en una tarea de optimización es proponiendo una distribución empírica definida para la tarea. A este método se le conoce como minimización del riesgo empírico. Para muchas tareas se conocen funciones de riesgo que permiten tener una medida para optimizar los parámetros de una red neuronal [20].

Elegir la función más adecuada es importante ya que de esto depende la optimización de los parámetros y de lo que se describe en la sección de optimizadores. Las funciones de riesgo en las redes neuronales van de la mano con la regresión lineal ya que en este caso se aplica una función llamada error cuadrático medio, esta función nos da un valor numérico indicando que tanto error hay en los parámetros de la regresión lineal, esta función se encuentra descrita en la ecuación 2.8.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.8}$$

Donde Y_i es el valor real y el valor predicho.

Para diversas tareas se tienen funciones de coste en las cuales su desempeño es mejor que otras funciones. Este es el caso de la función *Cross Entropy Loss*. Esta función mide la diferencia entre dos distribuciones de probabilidad para una variable aleatoria y un evento.

Esta función se basa en un concepto llamado información que en teoría de la información se refiere al número de bits necesarios para codificar un mensaje. Esta definición se puede transformar a un concepto más intuitivo en el que si un evento tiene mayor probabilidad de suceder entonces tiene una baja información mientras que un evento poco probable tiene mayor información. Esto se debe a que en un evento probable tenemos la certeza que va a suceder y por lo tanto no aporta mucha información a lo que estemos describiendo.

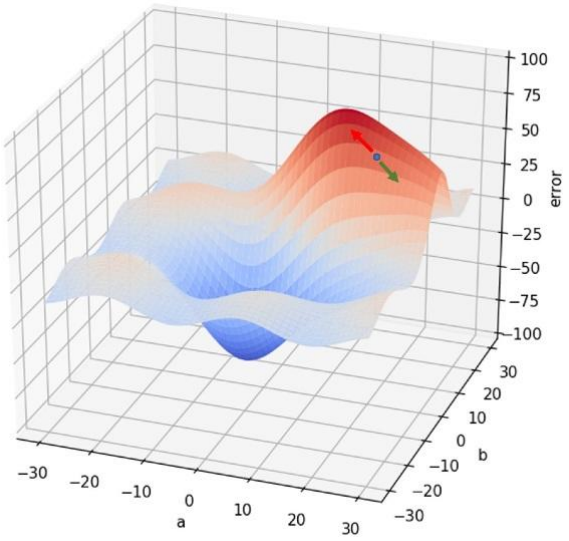
Además incluye otro concepto muy importante llamado entropía que se refiere a la cantidad de bits requeridos para transmitir un evento aleatorio en una distribución de probabilidad. En términos más simples si un evento tiene una probabilidad muy

similar a los otros eventos la entropía es alta ya que no se tiene certeza de que evento puede ocurrir mientras que si un evento tiene una mayor probabilidad comparada con los demás se dice que tiene una entropía baja ya que tenemos mayor certeza de que es lo que va a suceder. Esta función se usa tanto para clasificaciones binarias o multiclase. [22] La función *Cross Entropy Loss* es de las más populares debido a su uso en clasificadores y trabaja con distribuciones de probabilidad, la expresión de esta función esta en la ecuación 2.9.

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i),\ para\ n\ clases \tag{2.9}$$

Optimizadores

Para entrenar una red neuronal hace falta un componente importante y es la optimización de parámetros de una red neuronal. Los pesos de las capas de tipo *fully conected* o de las matrices (*kernels*) de los filtros en dos dimensiones necesitan ser optimizados para encontrar la mejor configuración de parámetros entrenables para que de esta forma se logre minimizar el error al pasar un vector por toda la red. Para encontrar estos parámetros se hace uso del gradiente descendiente. El gradiente descendiente o *gradient descent* es un algoritmo de optimización que utiliza el concepto de gradiente para realizar la optimización, el objetivo de este algoritmo es buscar el mínimo local de una función a través de las derivadas parciales de la función objetivo con respecto a sus parámetros. Es uno de los algoritmos más usados para la optimización de parámetros en redes neuronales.



: F
igura representativa del gradiente descendiente] Figura representativa del gradiente descendiente.[4]

El gradiente descendiente minimiza una función objetivo $J(\theta)$ actualizando los parámetros de la función en dirección opuesta al gradiente de la función objetivo $\nabla_\theta J(\theta)$. Tiene un parámetro importante y es la tasa aprendizaje o *learning rate* que determina el tamaño de actualización de los parámetros para encontrar el mínimo local. Existen variaciones del algoritmo que difieren en la cantidad de datos que usan para computar el gradiente de la función objetivo.

Batch Gradient Descent

También llamado *Vanilla Gradient Descent* computa el gradiente de la función de coste con respecto a todos los parámetros θ. La actualización de los parámetros se realiza con la ecuación 2.10.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{2.10}$$

Donde η es el rango de aprendizaje. La desventaja de usar este método es que solo se realiza una actualización por cada iteración además de que es lento para conjuntos de datos grandes. Este algoritmo sirvió para inspirar otros algoritmos basados en gradiente descendiente.

Gradiente Descendiente Estocástico (SGD)

En contraste con *Batch Gradient Descent*, este algoritmo actualiza los parámetros por cada ejemplo dado a la red. *Batch Gradient Descent* realiza cálculos redundantes para conjuntos de datos largos ya que recomputa los gradientes para ejemplos similares antes de actualizar cada parámetro. SGD realiza una actualización por cada paso, esto hace que sea mucho más rápido. Otro aspecto importante es la convergencia hacia los mínimos, *Batch Gradient Descent* converge solo a los mínimos locales esto causa que no siempre se tenga la mejor optimización. SGD permite que los pasos entre cada actualización sean de forma estocástica por lo que potencialmente puede encontrar mejores mínimos locales su expresión se encuentra en la ecuación 2.11.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.11}$$

Adam

Adaptive Moment Estimation (Adam) es un método que calcula un rango de aprendizaje adaptativo para cada parámetro. Adam mantiene un promedio exponencialmente decreciente de gradientes pasados esto es para adaptar el momento. Para evitar las oscilaciones en el proceso derivados de los pasos constantes, se utiliza el momento para reducir el número de pasos hasta llegar al mínimo local [23].

Backpropagation

Para actualizar los nuevos pesos, se tiene un algoritmo llamado *backpropagation* que permite actualizar todos los parámetros entrenables de la red. El algoritmo se basa

[4]Figura tomada de *Gradient Descent* `https://interactivechaos.com/es/manual/tutorial-de-machine-learning/gradient-descent` visto en agosto de 2021

en el calculo del gradiente para cada neurona de la red. Esto se hace calculando la derivada parcial de cada capa desde la función de coste hasta la capa a cambiar. Se llama de esta forma ya que va calculando y acumulando los gradientes hasta la capa inicial utilizando la regla de la cadena [6]. Por ejemplo, si se quiere actualizar los valores de la última capa, se realiza la siguiente operación:

$$\frac{\partial E}{\partial w} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial h} \cdot \frac{\partial h}{\partial w} \tag{2.12}$$

Que se puede leer como la derivada parcial del error con respecto a los pesos es igual a la derivada parcial de la función de perdida con respecto a la función de activación $\frac{\partial L}{\partial a}$ por la derivada parcial de la función de activación con respecto a la entrada de dicha función $\frac{\partial a}{\partial h}$ por la derivada parcial de la salida de la capa con respecto a los pesos de la capa a actualizar $\frac{\partial h}{\partial w}$. Estas operaciones se repiten para las capas anteriores. Dado que los cálculos son complicados y sugiere conocer las derivadas de los componentes como las funciones de activación además de que el proceso se repite para todas las capas de una red profunda. Para solucionar se construye un grafo computacional que almacena toda esta información [6]. Por ejemplo se puede tener la siguiente función:

$$f(x, y, z) = \tanh\left(\ln\left[1 + z\frac{2x}{sin(y)}\right]\right) \tag{2.13}$$

Se puede construir un grafo donde se tiene a cada elemento como una función simple y que si se sustituye cada elemento se obtiene la función f así como el de la figura 2.13. A este paso se le llama *forward pass*. Para calcular el paso *backward* se calculan las

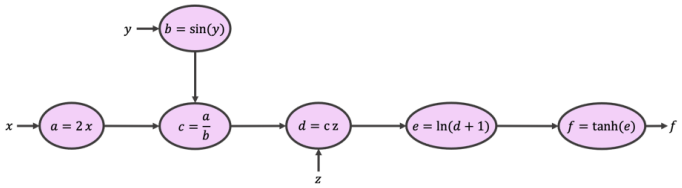


Figura 2.13: Ejemplo de Grafo Computacional [6].

derivadas de cada nodo del grafo así como se muestra en la figura 2.14.

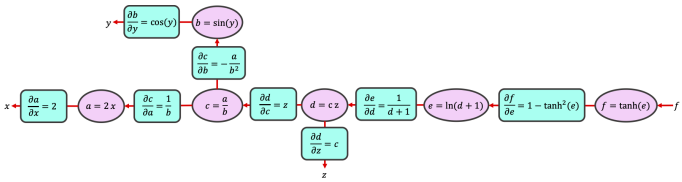


Figura 2.14: Ejemplo de Grafo Computacional con los gradientes calculados [6].

Actualmente así se calculan los gradientes y se actualizan los pesos en los frameworks especializados en redes neuronales y *deep learning*.

Regularización

Cuando se entrena un modelo, es deseable que éste pueda generalizar las predicciones fuera del conjunto de datos de entrenamiento. Con las redes neuronales es importante minimizar el *overfitting* para obtener un buen modelo. Para esto, existen algunos métodos que reducen este fenómeno como lo es el *dropout*.

El *dropout* consiste en desactivar aleatoriamente neuronas de la red para evitar que se aprendan flujos específicos en la red y distribuir las «señales de aprendizaje» por cada paso del entrenamiento. Además se puede considerar como ruido por lo que se aprenden representaciones más robustas y se obtiene un modelo más generalizable [24].

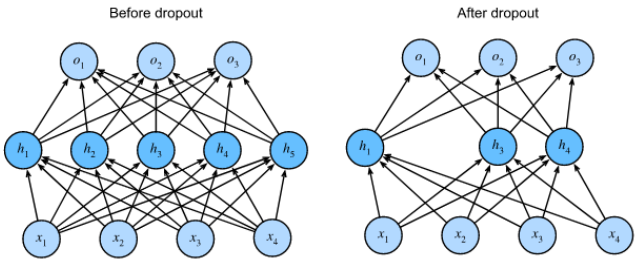


Figura 2.15: Ilustración del método *dropout*.[5]

2.4. Aprendizaje *self-supervised*

Para entrenar un modelo predictivo hacen falta tener datos etiquetados que muchas veces son difíciles de conseguir. Algunas compañías como Amazon han recurrido a estrategias basadas en micro-tareas que sirven para validar y etiquetar grandes conjuntos de datos, esto le da la oportunidad a las personas de obtener un ingreso extra y a las empresas de obtener datos válidos para entrenar sus modelos. Programas como *Amazon Mechanical Turk* son un esfuerzo para conseguir este objetivo [25].

El aprendizaje *self-supervised* es motivado por esta complejidad de acceder a datos etiquetados y de aprender mejores representaciones. Las redes neuronales necesitan datos etiquetados por lo que la idea detrás del aprendizaje *self-supervised* es utilizar una tarea de pretexto (también llamada tarea espuria) para entrenar a la red, la tarea puede ser cualquiera donde se puedan etiquetar los datos automáticamente. Esto permite a la red aprender representaciones y generalidades de los datos por lo que posteriormente es más fácil adaptar el modelo a la tarea objetivo.

La tarea de pretexto va desde predecir alguna característica de los datos asi como alguna operación dentro de los mismos. Varia dependiendo de los datos con los que

[5]Imagen tomada de *Dropout* `https://d2l.ai/chapter_multilayer-perceptrons/dropout.html` visto en septiembre de 2021

se este trabajando.

Para modelos de lenguaje comúnmente la tarea consiste en predecir la palabra siguiente, una palabra aleatoriamente oculta o dada una palabra predecir las palabras anteriores y posteriores. Gracias a esto se han generado modelos de lenguaje muy robustos que son el estado del arte.

Para visión computacional ha pasado algo similar generando modelos que son más robustos ante rotaciones o traslaciones en las imágenes comparados con redes convolucionales ordinarias. Las tareas van desde predecir la posición de un fragmento de la imagen, predecir cual es la rotación de una imagen o dadas ciertas transformaciones predecir si dos fragmentos de imagen pertenecen o no a la misma fuente.

Modelos como SimCLR han demostrado tener un buen rendimiento comparado con otras arquitecturas además de que al ser un modelo entrenado con distintas transformaciones hacen más robusto el modelo y permite realizar mejores predicciones [26].

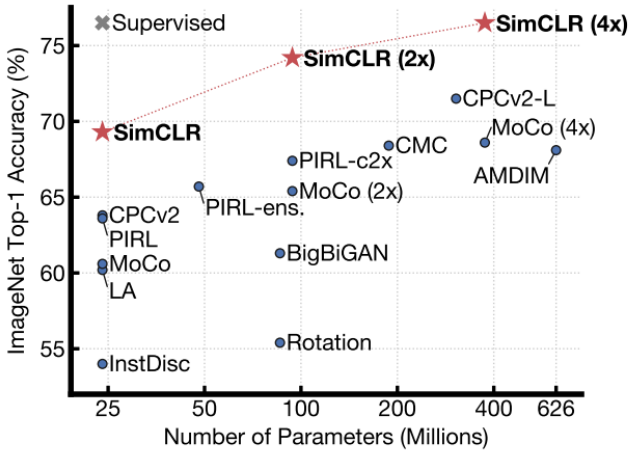


Figura 2.16: Gráfica de *accuracy* vs Número de parametros de SimCLR [7].

Asi mismo, el aprendizaje *self-supervised* permite aprender mejores representaciones de los datos en el espacio latente. En conjunto con métodos como *contrastive learning* permite codificar características similares entre los datos y agrupar los que comparten diversos patrones y de esta forma generar codificaciones que sirven para las *downstream tasks*.

2.5. *Contrastive Learning*

Contrastive Learning es una forma de aprendizaje con fundamentos en los modelos basados en energía en la que el objetivo es aprender representaciones de los datos en los cuales algunos pares son similares o disimilares a otros.

Los modelos basados en energía parten de la premisa que en lugar de clasificar un vector de entrada x a una salida y, se toma el par (x, y) y se mide si son compatibles o no. Entonces se define una función $F(x, y)$ que permite evaluar qué tan compatibles son estas dos variables.

Formalmente una función de energía se define como $F : X \times Y \rightarrow R$ donde $F(x, y)$ describe el nivel de dependencia entre las variables x y y. Una forma más sencilla de entender este concepto es comparar si el texto y es una buena traducción del texto x.

Una modificación de los modelos basados en energía es utilizar una variable extra z llamada variable latente la cual puede proveer información auxiliar al modelo para mejorar la codificación de las variables y por ende agrupar mejor las entradas similares. En términos generales, los métodos contrastivos minimizan la energía de los pares de datos similares y la aumentan con datos disimilares.

Conjuntándolo con el aprendizaje *self-supervised* se pueden definir tareas en las que por ejemplo si se toman dos fragmentos de la misma imagen, se obtiene una etiqueta positiva la cual en el proceso de entrenamiento minimizará la energía. Caso contrario si se toman dos fragmentos de imágenes totalmente distintas, la etiqueta será llamada como negativa y elevará la energía. Además para mejorar las predicciones, se hace uso de variables latentes h y h' que definen vectores de características para cada par de variables de entrada. A estos vectores se le aplica una métrica de similitud para agrupar o alejar los pares y una función de pérdida que minimiza o maximiza la energía de los pares de datos [27].

Los métodos contrastivos han demostrado ser muy útiles para codificar estas variables latentes y en consecuencia realizar una distinción entre conjuntos de datos con diversas características. Un ejemplo de es la separación entre las grabaciones de voz de 10 diferentes personas [9]. En la figura 2.18 se observa mediante una una proyección las variables latentes o *embeddings* de las grabaciones y donde cada color representa a una persona diferente.

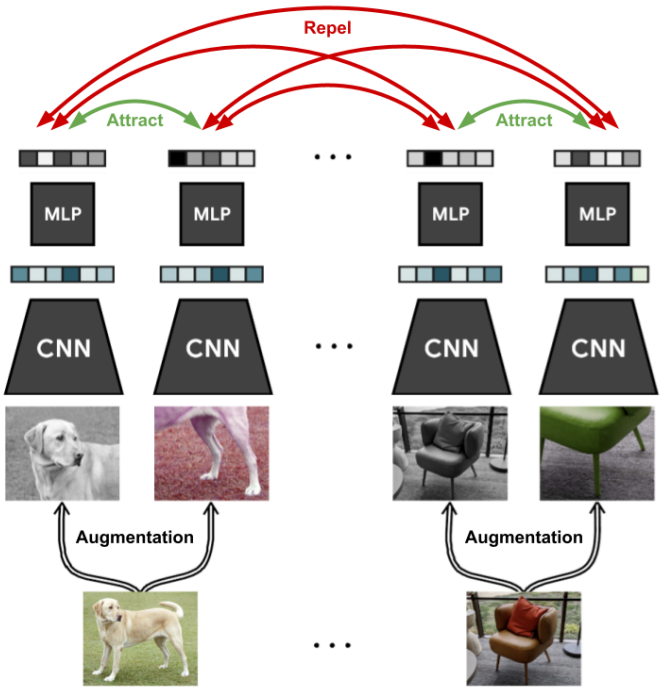


Figura 2.17: Diagrama de la arquitectura SimCLR donde se aplica *Contrastive Learning* [8].

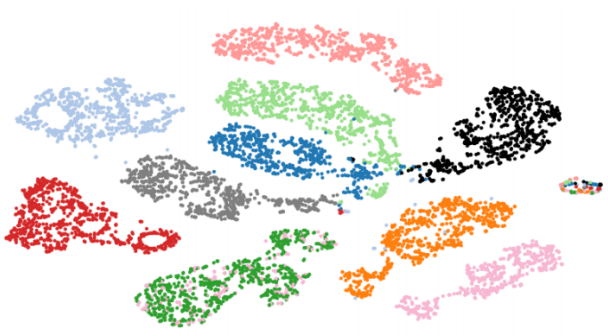


Figura 2.18: Visualización de *embeddings* entrenados con *Contrastive Learning* [9].

3 Metodología

Para comprender la metodología del presente trabajo se describirán las herramientas utilizadas para la generación del modelo de *deep learning*.

3.1. Redes Neuronales de tipo Siamesas

Las redes neuronales siamesas son una arquitectura que consiste en dos redes neuronales idénticas en las cuales se aprenden representaciones internas por cada vector de entrada. La red es de tipo *feed forward* y emplea *backpropagation* para entrenar los parámetros. Los vectores generados por este tipo de redes son comparados entre si mediante una métrica como la distancia euclidiana o la distancia coseno para definir la similitud entre los vectores [28].

La idea de usar redes siamesas proviene un artículo de 1994 que describe un algoritmo basado en redes neuronales para la verificación de firmas. En esta investigación se comparaban dos firmas escritas a mano en una tableta electrónica y la red neuronal debía predecir si las dos firmas pertenecían a la misma persona o eran diferentes [29].

Esta idea permitió manejar datos complicados como las firmas ya que la firma de alguien al ser realizada por un humano nunca es idéntica y causaban conflicto con los métodos que se tenían en la época.

Al mencionar que se tienen dos redes neuronales idénticas, los pesos de las redes son los mismos por lo que al hacer el paso *forward* se calcula la salida de los pares de entradas con respecto a la misma red y al hacer el paso *backward* se actualizan los pesos de la red solo una ocasión. En la figura 3.2 se muestra un diagrama ilustrativo de una red siamesa.

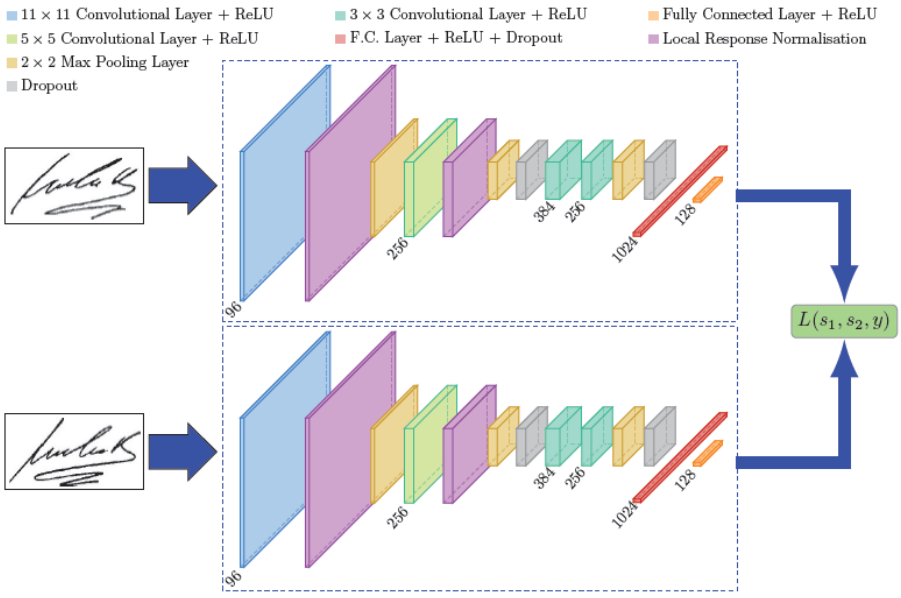


Figura 3.1: Diagrama de una red neuronal de tipo siamesa.[1]

Para ilustrar que los pesos de la red siamesa son los mismos, en el paso *forward* ambas entradas de la red pasan por el mismo método, la salida del método se genera los *embeddings*, que posteriormente se utilizan en la función de perdida y en la métrica.

```
def forward(self, input1, input2):
    # forward pass of input 1
    output1 = self.forward_once(input1)
    # forward pass of input 2
    output2 = self.forward_once(input2)
    return output1, output2
```

Figura 3.2: Código de ejemplo en el paso *forward*.

3.2. Funciones de perdida

Para entrenar la red neuronal utilizando *contrastive learning* es necesario decidir el tipo de función *loss* a utilizar.

[1]Imagen Tomada de *A friendly introduction to Siamese Networks* `https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942` visto en agosto de 2021

Contrastive Loss

La función general se expresa de la siguiente forma

$$L(W,(Y,\vec{X_1},\vec{X_2})^i) = (1-Y)L_s(D_w^i) + YL_d(D_w^i) \tag{3.1}$$

Y se refiere al resultado esperado dados los dos vectores $\vec{X_1}, \vec{X_2}$ codificado de la siguiente forma: para datos similares $Y = 0$ y para entradas disimilares $Y = 1$.

El termino L_s representa la función a aplicar cuando la tupla $(\vec{X_1}, \vec{X_2})$ es similar, lo mismo aplica para L_d cuando los datos son disimilares.

D_w^i representan la función de similitud. Dada la codificación anterior, se toma una función que mapee ambos vectores resultantes de la red siamesa a un valor entre 0 y 1. Esta función normalmente es la distancia euclidiana aunque puede ser cualquier parámetro α de la distancia de Minkowski. La formulación matemática se muestra en la ecuación 3.2:

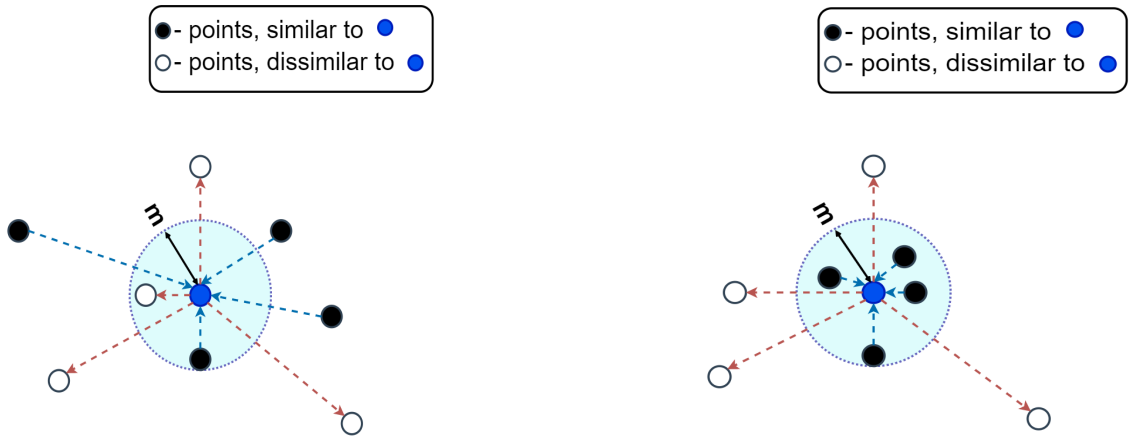
$$D_w(\vec{X_1},\vec{X_2}) = \|G_w(\vec{X_1}) - G_w(\vec{X_2})\|_2 \tag{3.2}$$

Donde G_w es una función que permite conservar la similitud semantica de las entradas con respecto a las salidas.

La ecuación final para una tupla de valores $(Y,\vec{X_1},\vec{X_2})^i$ utilizada es la mostrada en la ecuación 3.3.

$$L(W,Y,\vec{X_1},\vec{X_2}) = (1-Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}\{max(0,m-D_w)\}^2 \tag{3.3}$$

L_s es la distancia euclidiana por lo que si dos vectores son similares se va a minimizar la distancia. Para L_d se maximiza la distancia entre los puntos. Para entender la expresión L_d hay que recordar que la clasificación tiene dos valores principales: similares y disimilares, por lo que se necesita de un margen para agrupar todos los puntos similares que caen dentro de este margen y empujar los puntos disimilares fuera del margen [30].



(a) Acción de *Push*. (b) Acción de *Pull*.

Figura 3.3: Diagramas representativos de la acción *push* y *pull* en *Contrastive Loss*. [10]

3.3. Adquisición de los datos

Los datos fueron recabados siguiendo dos etapas mostradas en la figura 3.4 y recolectados por un grupo de investigadores (Arias García MA, Calderón Ortíz VM, Hernández Martínez R) del Laboratorio de Neurociencias perteneciente a la Facultad de Psicología de la Universidad Nacional Autónoma de México.

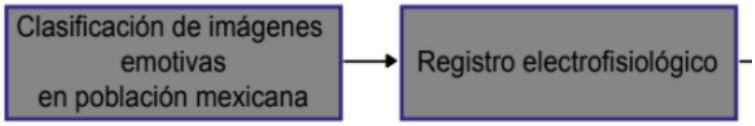


Figura 3.4: Flujo de trabajo en la adquisición de los datos.

Clasificación de imágenes emotivas en población mexicana.

Debido a la subjetividad en la clasificación de imágenes se le presentaron 60 imágenes a 200 participantes donde el 44 % fueron hombres y el 66 % mujeres. Los participantes estaban en un rango de edad de 18 a 67 años quienes clasificaron las imágenes en tres categorías: placentero, displacentero y neutro [31].

Figura 3.5: Ejemplo de imagen placentera.

Figura 3.6: Ejemplo de imagen displacentera.

Figura 3.7: Ejemplo de imagen neutra.

Registro electrofisiológico.

Se realizaron registros EEG a 22 participantes sanos 10 hombres y 12 mujeres de 20 a 62 años de edad. Estos registros fueron tomados mientras los sujetos eran sometidos a un estimulo visual con imágenes etiquetadas como placenteras, displicenteras y neutras. Estos estímulos son videos de 72 segundos en los que por cada imagen mostrada hay una imagen intermedia que sirve como punto de fijación. Estos puntos son de 2 segundos cada uno y las imágenes mostradas tienen una duración de 5 segundos [31].

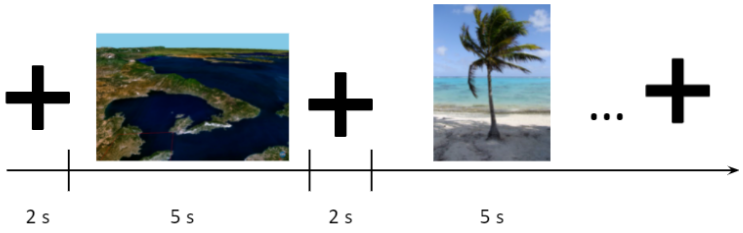


Figura 3.8: Diagrama del video estimulo.

Los experimentos se dividieron en tres bloques correspondientes a cada una de las categorías, con diez imágenes por cada categoría y previamente nombradas así como se muestra en la tabla 3.1.

Todos los registros fueron capturados con un electroencefalógrafo llamado *Emotiv EPOC*(vease figura 3.9) de 14 canales utilizando el estándar internacional 10-20. Este dispositivo cuenta con una frecuencia de muestreo de 128 muestras por segundo. Una resolución de 14 bits que captura hasta 0.51 μV. Captura señales desde 0.2 hasta 45 Hz. Los datos obtenidos por este dispositivo son guardados en formato CSV como

	Bloque 1: Placenteras	Bloque 2: No placenteras	Bloque 3: Neutras
1	Tacos	Basura cañon del sumidero	Cesto
2	Chamorro	Bebé decerebrado	Casuelas
3	Cabos	Baño	Quisoco
4	Monte bello	Perro con bala	Museo Soumaya
5	Pies Pareja	Descuartizado	Pizarrón
6	Dinero	Metro DF	Sillas Muac
7	Graduacion	Carro chocado	Boliche
8	Chica Guapa	Cabeza de cordero	Universum
9	Playa	Rata perfusion	Jueguetes niños
10	Chico Guapo	Pie diabético	Atlixco

Tabla 3.1: Tabla con el contenido de cada estimulo.

series de tiempo.

Figura 3.9: Imagen demostrativa del *Emotiv EPOC*.

En la tabla 3.2 Se muestra un resumen de la población analizada.

Número de Sujetos Estímulos Presentados	Género	Rango de Edad	Tiempo de Estímulo
10 Placenteros, No Placenteros y Neutros	Hombres	20 a 62 años	72 segundos
12 Placenteros, No Placenteros y Neutros	Mujeres	20 a 62 años	72 segundos

Tabla 3.2: Tabla con el resumen de la población.

3.4. Tareas Espurias

Debido a la estructura de los datos se propusieron dos formas de evaluarlos. Tomando un fragmento de un canal dando así un vector de $\mathbb{R}^n$ y evaluándolo en redes lineales y convoluciones de una dimensión. Y tomar uno o más canales y de esa manera tomar una Matriz $\mathbb{M}_{NxM}$ donde N representa los canales y M representa la longitud establecida de la señal.

Para todas las tareas se definieron grupos de sujetos y dentro de estos grupos se tiene la opción de combinar los datos de dos sujetos o no para las tarea «Un Canal».

3.4.1. Un Canal

Tomando información de un electrodo o canal de los 16 disponibles y las propiedades de los datos se encontraron las siguientes características: sujetos, canales y temporalidad.

Mismo Sujeto

En tareas de audio es común encontrar la tarea de clasificación por hablante por lo que la tarea propuesta es la clasificación por sujeto. Sean dos vectores de entrada obtenidos por alguno de los 14 canales de la señal clasificar si esas dos señales pertenecen al mismo sujeto o no. Los datos pueden pertenecer o no al mismo canal por lo que la tarea es robusta al no tener una característica estática.

Mismo Canal

Al tener 14 canales en los datos es factible clasificar las señales mediante su canal de origen. Los datos son tomados del mismo o de diferente sujeto según la configuración deseada por lo que la tarea es robusta.

Consecutividad

Un aspecto importante en las señales es el aspecto de temporalidad por lo que se pro-

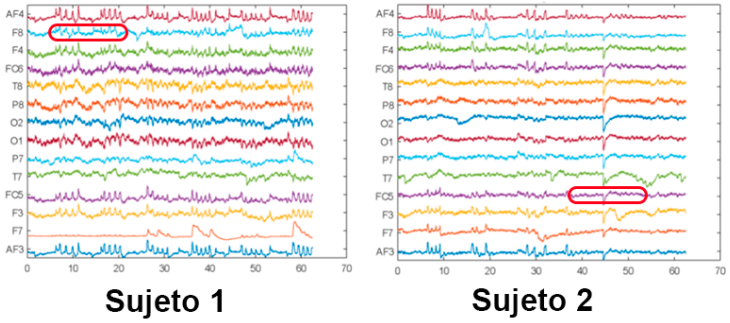


Figura 3.10: Diagrama de la tarea «Mismo Sujeto».

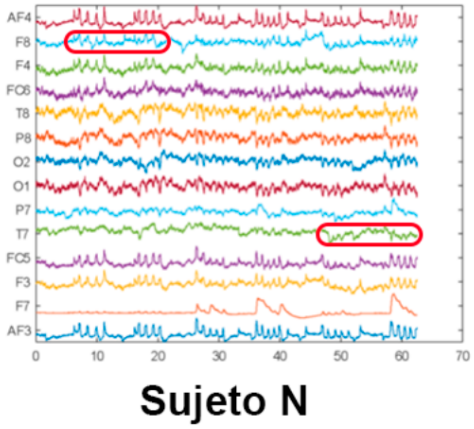


Figura 3.11: Diagrama de la tarea «Mismo Canal».

pone una tarea en donde la red neuronal aprenda a diferenciar entre dos segmentos de señales si son consecutivos o no. Debido a la tarea los datos pertenecen al mismo sujeto y al mismo canal.

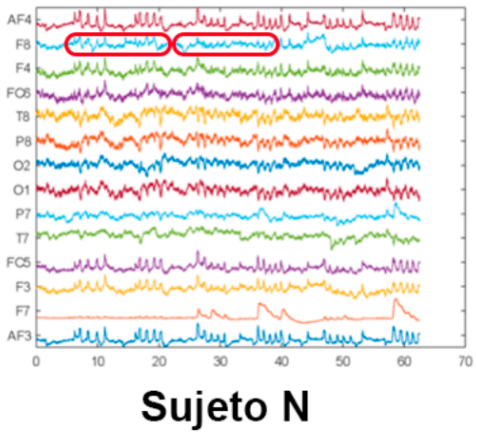


Figura 3.12: Diagrama de la tarea «Consecutivo».

3.4.2. Múltiples Canales

Relative Positioning

La idea de esta tarea es tomar dos segmentos de datos representados en la figura 3.13 como x_t y $x_{t'}^+$ o $x_{t'}^-$ y definir si estos dos segmentos pertenecen o no al mismo contexto. El contexto hace referencia a si están próximos un fragmento del otro determinado por un factor τ_{pos} y por τ_{neg}. Entonces definidos estos dos margenes se determina si el par de datos es menor al factor τ_{pos} la categoría es positiva mientras que si son mayores a l factor τ_{neg} la etiqueta sera negativa. Esta tarea toma datos de múltiples canales y es sensible al tiempo por lo que los datos deben pertenecer al mismo sujeto.

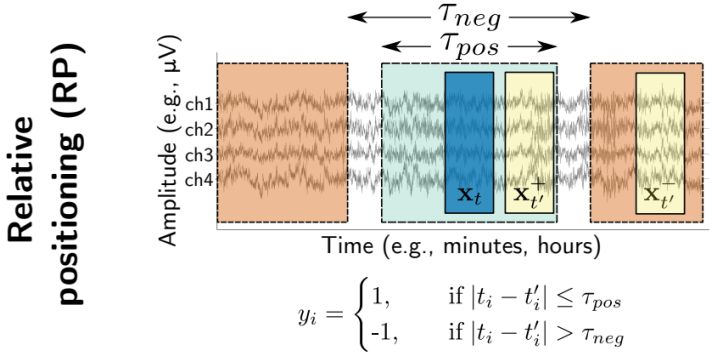


Figura 3.13: Diagrama de la tarea «*Relative Positioning*» [11].

3.5. Arquitecturas

Durante la experimentación se utilizaron diferentes tipos de redes neuronales comenzando desde lo básico con capas lineales hasta algo más complejo como lo son

las redes convolucionales en una dimensión y en dos dimensiones utilizando una red neuronal de tipo siamesa.

3.5.1. Lineal

Como primera arquitectura se propone una red siamesa utilizando capas *fully connected*. Una entrada de 256 dimensiones y 5 capas *fully connected* de 256,128,64,128 y 256 dimensiones. Todas ellas con función de activación *ReLu*, *Batch Normalization* y un *dropout* de 0.2. Los datos se tomaron de un canal de manera secuencial con y sin superposición. En la figura 3.14 se puede ver una representación de esta arquitectura.

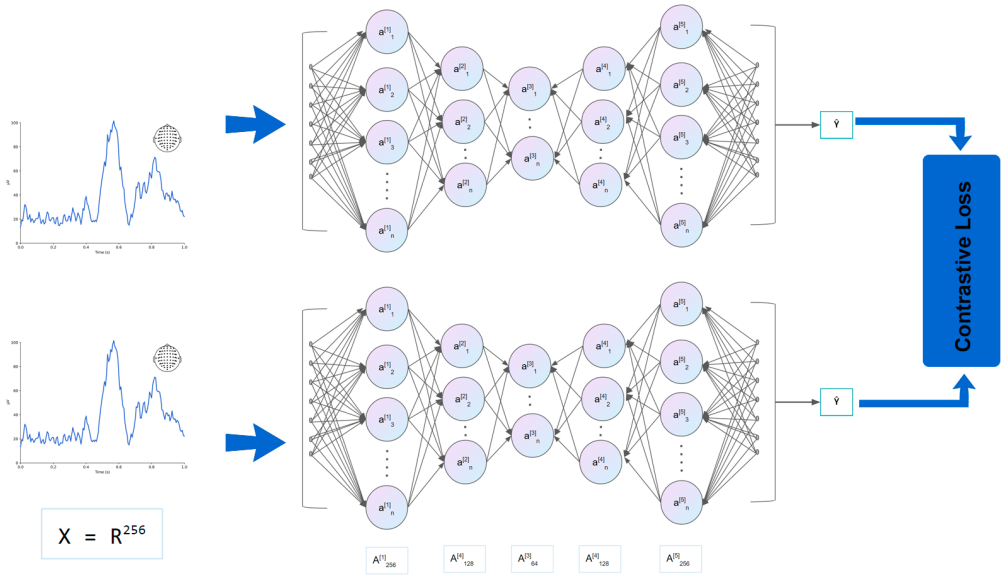


Figura 3.14: Red Siamesa con capas *fully connected*.

3.5.2. Convolucional 1D

Como una alternativa a las capas *fully connected*, se propone utilizar capas convolucionales 1D ya que al ser series de tiempo, funcionan como filtros en una dimensión. La red consta de 5 módulos de redes convolucionales 1D, todas con un *dropout* de 0.2, *batch normalization* y función de activación *GeLu*. Las caracteristicas de las capas son las siguientes:

- Conv1D(1,16,8,1) + MaxPool1d(5,2)
- Conv1D(16,64,16,1) + MaxPool1d(5,1)

- Conv1D(64,128,8,1) + MaxPool1d(8,2)
- Conv1D(128,64,8,1) + MaxPool1d(8,1)
- Conv1D(64,16,8,1) + MaxPool1d(8,2)

Siguiendo la nomenclatura Conv1D(Canales de entrada, Canales de Salida, *kernel*, *stride*) y MaxPool1d(*kernel*, *stride*). Estos módulos son acompañados de una capa *Fully Connected* con una salida de 256 dimensiones. En la figura 3.15 se observa una representación de la arquitectura.

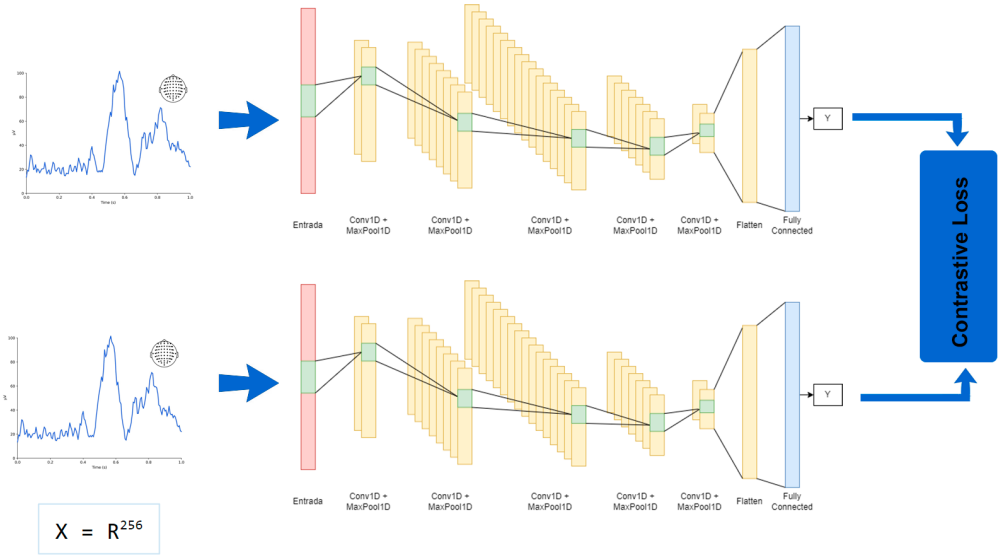


Figura 3.15: Red Siamesa con capas convolucionales 1D.

3.5.3. Convolucional 2D

Finalmente para explorar el uso de más canales y por ende más datos en los registros se propone el uso de una red basada en StageNet [11]. Esta compuesta de una red convolucional espacial que realiza una transformación de los datos que permite obtener una invariancia en la entrada de la red. Además cuenta con 4 módulos de capas convolucionales 2d, *Max Pooling 2D* con diferentes configuraciones. Todas llevan una función de activación *GeLu*, *Batch Normalization* y un *dropout* de 0.2. Las caracteristicas de las capas son las siguientes:

- Conv2D(1,256,(1,64),1) + MaxPool2d((1,2),(1,2))
- Conv2D(256,128,(1,32),1) + MaxPool2d((1,2),(1,1))

- Conv2D(128,64,(1,16),1) + MaxPool2d((1,2),(1,2))
- Conv2D(64,32,(1,8),1) + MaxPool2d((1,2),(1,1))

Al finalizar las capas convolucionales, se realiza la operación *flatten* y se pasa por una capa *Fully Connected* de 512 dimensiones de salida. En la figura 3.16 se tiene un diagrama general de la arquitectura.

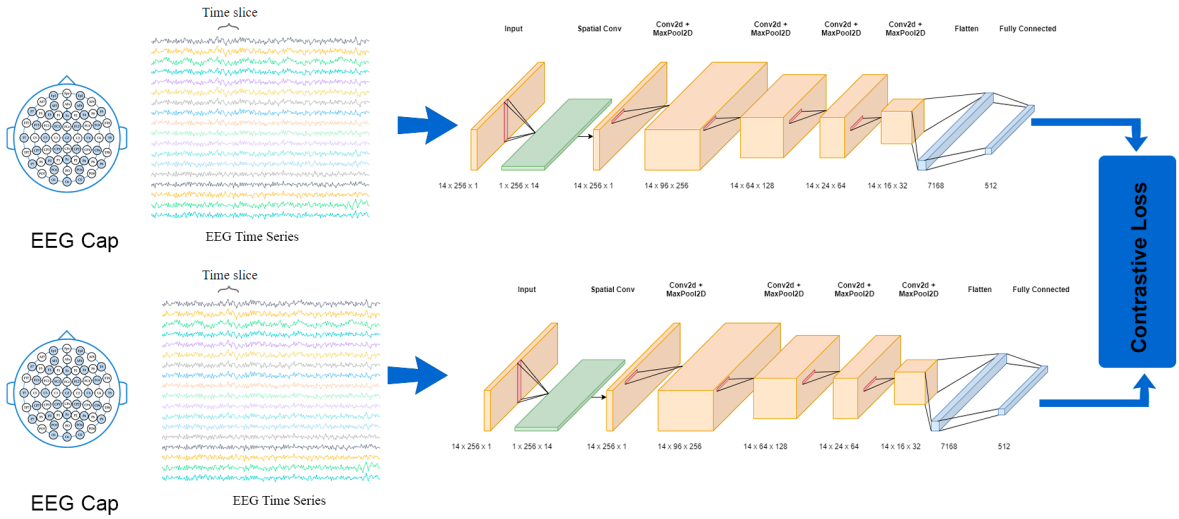


Figura 3.16: Red Siamesa con capas convolucionales 2D.

3.6. Arquitectura del Software

Para realizar los experimentos se desarrolló un paquete de utilidades para preprocesar los datos, crear las tareas, crear el dataset, cargar los modelos, entrenar las redes y evaluar los modelos conforme a diferentes métodos y aplicando reducción de dimensionalidad para la visualización de los *embeddings* En la figura 3.20 se muestra un diagrama parcial de la estructura general del proyecto. El código fuente se puede encontrar en `https://github.com/TheJorseman/EmotionNeuralInterface`.

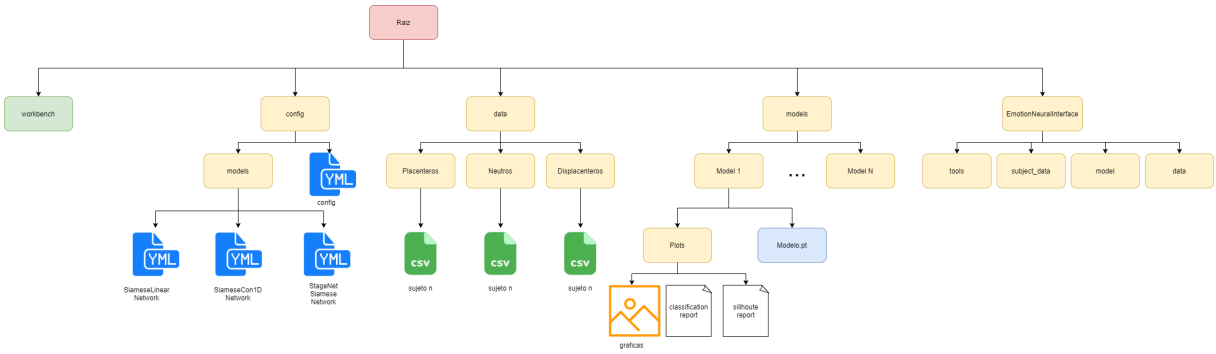


Figura 3.17: Distribución parcial de carpetas del proyecto.

3.6.1. *Config*

Esta carpeta tiene la definición en formato *.yml* sobre como se va a realizar la configuración del modelo y del experimento o entrenamiento que se quiere correr. Por una parte se tiene la configuración general, aqui se definen los hiperparametros del modelo, la ubicación de los datos, tareas, tamaño de los *batches* entre otros parámetros que permiten una mayor flexibilidad a la hora de probar una arquitectura o probar diferentes hiperparametros. En la figura 3.18 se observa un fragmento del archivo de configuración. El ejemplo completo se encuentra en `https://github.com/TheJorseman/EmotionNeuralInterface/blob/main/config/config.yaml`

El código cuenta con la posibilidad de generar un modelo dinámico basado en un archivo de configuración yml. En este archivo se definen las capas que va a tener el modelo, funcion de activación, si va a tener *batch normalization* y el valor de *dropout* que va a tener cada capa. Así mismo permite definir el tipo de capa a utilizar y los parámetros más importantes a definir en dicha capa. En la figura 3.19 se puede ver un fragmento del archivo de configuración del modelo. Un ejemplo completo se encuentra en `https://github.com/TheJorseman/EmotionNeuralInterface/blob/main/config/models/conv1d.yaml`, dentro de la misma carpeta se pueden encontrar diferentes configuraciones.

3.6.2. *Data*

Esta es una carpeta opcional ya que se pueden utilizar datos propios en otra carpeta y referenciandolos en el archivo de configuración o utilizar un link de *Google Drive* para descargar, descomprimir y utilizar los datos desde un archivo en la nube. La estructura general para este dataset y para otros es tener subcarpetas con los experimentos y dentro de estas carpetas tener los datos en formato csv y un archivo

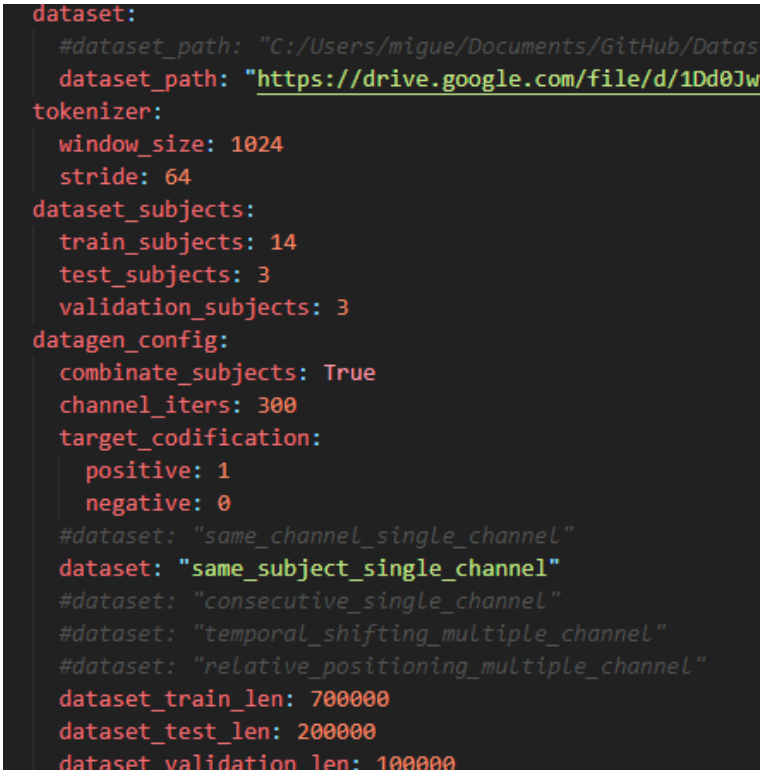


Figura 3.18: Fragmento del archivo de configuración.

por cada sujeto. Esto para facilitar la búsqueda del programa y de esta forma generar de forma correcta las clases necesarias para etiquetar los datos.

3.6.3. *Models*

Esta carpeta puede cambiar de nombre según la configuración general del experimento. En esta carpeta se guardan los experimentos creando una carpeta con un identificador único y dentro de esta carpeta se guardan los checkpoints del entrenamiento, la configuración utilizada en el experimento, la configuración del modelo, realiza gráficas de los datos de entrenamiento para su análisis guardando los vectores resultantes y reduciendo su dimensionalidad utilizando *UMAP* y *T-SNE*. Guarda los datos de entrenamiento como *accuracy* y *loss* utilizando *Tensorboard* para su análisis con dicha herramienta. Además genera una análisis mostrando el la exactitud, precisión, *F1-Score*, matriz de confusión, los parámetros de las capas y un análisis mediante *sillhouete score* para medir la similitud entre datos cercanos.

3.6.4. *EmotionNeuralInterface*

Este paquete contiene todas los módulos necesarios para preprocesar los datos, etiquetarlos de acuerdo a su origen, crear los diferentes modelos propuestos y crear el dataset dada la tarea espuria definida. En la figura 3.20 se puede observar un diagrama general del paquete.

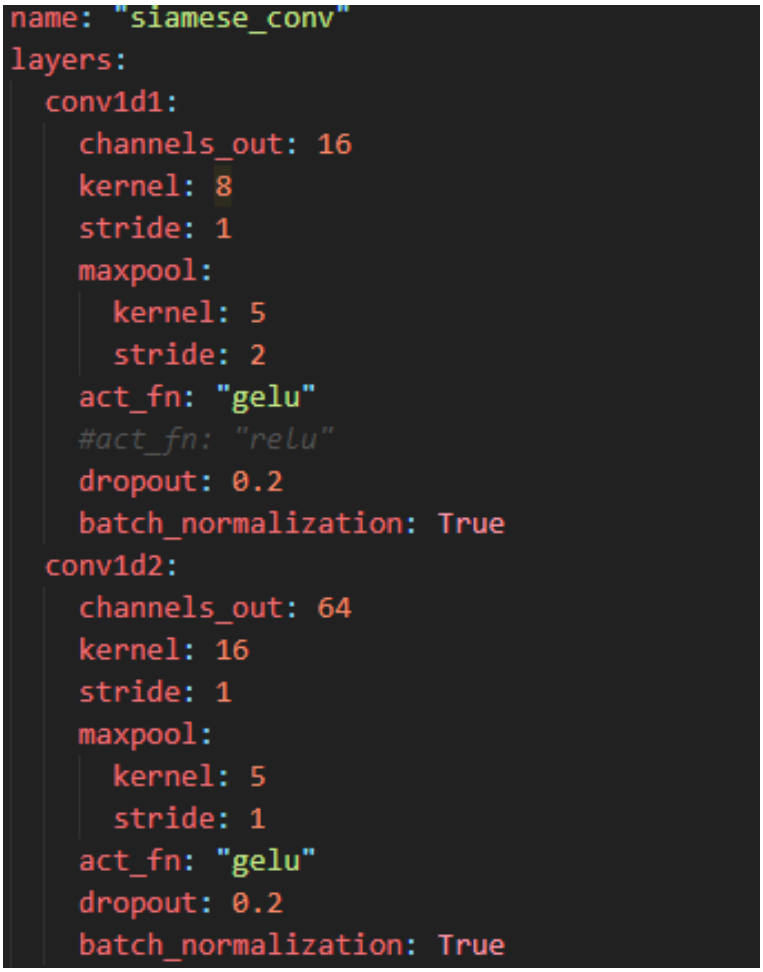


Figura 3.19: Fragmento del archivo de configuración del modelo.

Tools

Esta sección contiene todas las utilidades para cargar los datos desde una ruta definida o desde internet usando un link de *Google Drive* para su descarga, descompresión y uso. Sirve para ejecutar el código en cualquier computadora en la que se quiera correr el código y entrenar la red neuronal.

Subject Data

En esta sección se leen los datos desde una carpeta especificada, se crean las clases experimento, sujeto y data para mantener un mejor control de cada csv y generar los datos de entrenamiento de la red neuronal. Se genera una relación en la que cada experimento tiene referenciado a varios sujetos y cada sujeto tiene un dato (csv).

Model

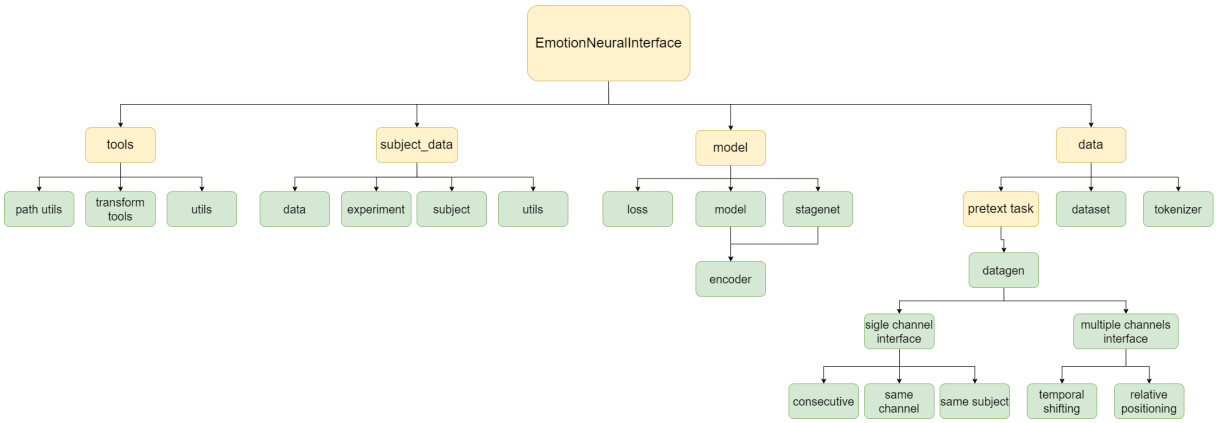


Figura 3.20: Diagrama General del Paquete *EmotionNeuralInterface*.

En esta parte se definen los modelos y las funciones de *loss* con los que se van a trabajar. Así mismo se tienen archivos separados donde se encuentran los modulos «*encoder*» que son bloques de capas convolucionales 1D y 2D, *max pooling*, *batch normalization* y *dropout*. Los modelos permiten generar redes neuronales siamesas de tipo *fully connected*, convolucionales 1D y 2D, estas redes son generadas de manera dinámica mediante un archivo de configuración.

data

Esta es una de las secciones más extensas ya que es la parte encargada de generar los datos mediante las diferentes tareas espurias tanto en un canal como en múltiples canales. Se desarrollo la herramienta *tokenizer* que divide todas las señales según los parámetros *stride* que representa el «salto» entre cada división del canal y el tamaño de la ventana. Además para gastar memoria duplicando los datos al momento de generar las divisiones se les asigna un numero con el que serán identificados al momento de generar el *dataset* y al momento de acceder a ellos en el entrenamiento, evaluación y prueba.

Se cuenta con una clase heredada de la herramienta *Pytorch* que permite generar datasets personalizados. La carpeta *Pretext Tasks* contiene una clase que es la generadora de los datos (datagen) y que es la clase padre de la cual resultan dos interfaces que contienen definiciones necesarias para generar datos a partir de un solo canal (*single channel interface*) o múltiples canales (*multiple channel interface*). Por cada tarea propuesta se tiene una clase que hereda de la interfaz correspondiente y que permite agregar la lógica necesaria para etiquetar los datos de forma automática. Esta diseñado de esta manera para que en dado caso que se proponga una nueva tarea sea más fácil incluirla.

3.7. Métricas

Para evaluar que tan buenos son los modelos, se utilizan diferentes métricas dependiendo de la tarea que va a realizar el modelo, para ésta arquitectura las métricas utilizadas son:

Exactitud

En inglés *accuracy* nos indica la fracción de predicciones correctas del modelo [32]. Para la arquitectura utilizada, si la distancia euclidiana es mayor al margen utilizado en la función de pérdida, se toma como un ejemplo negativo, de otro modo se toma como positivo. En la ecuación 3.4 se describe ésta métrica.

$$Acc = \frac{Número \quad de \quad predicciones \quad correctas}{Numero \quad total \quad de \quad predicciones} \tag{3.4}$$

F1 Score

Es usada para evaluar modelos de clasificación binarios. Combina la precisión y *recall* resultando en una media armónica de ambos valores [33] para entender el *F1 score* hay que revisar la precisión y el *recall*.

La precisión se puede definir como la fracción entre los datos relevantes entre todas los datos recuperados, en términos de una clasificación binaria es la relación entre los verdaderos positivos entre todos los positivos [34]. En la ecuación 3.5 se describe la precisión.

$$precision = \frac{tp}{tp + fp} \tag{3.5}$$

El *Recall* se refiere a la sensitividad de los resultados, es la fracción de los datos recuperados entre todos los datos relevantes. En términos de un clasificador binario es la relación entre los verdaderos positivos entre la suma de los verdaderos positivos y los falsos negativos [34]. En la ecuación 3.6 se describe el *recall*.

$$recall = \frac{tp}{tp + fn} \tag{3.6}$$

Entendiendo la precisión y el *recall*, la fórmula usada para la métrica *f1-score* es la mostrada en la ecuación 3.7.

$$F_1 = \frac{2}{\frac{1}{recall} \times \frac{1}{precision}} = 2 \times \frac{precision \times recall}{precision + recall} = \frac{tp}{tp + \frac{1}{2}(fp + fn)} \tag{3.7}$$

Sillhoute Score

Es una métrica usada para determinar la calidad de los *clusters* generados en la predicción de los datos [35]. Al trabajar con *embeddings* o vectores, los datos resultantes pueden ser catalogados como positivos o negativos creando así diferentes *clusters* en los que la función de perdida debería mapear los datos de entrada.

Sillhoute Score es la media del coeficiente *sillhoute* de todos los datos. El coeficiente *sillhoute* es calculado teniendo en cuenta la distancia media *intra-class* a y la distancia media hacia el *cluster* más cercano b [35]. La distancia puede ser calculada como una distancia euclidiana o con algún otro método. El coeficiente *sillhoute* se calcula como se muestra en la ecuación 3.8.

$$S = \frac{(b - a)}{max(a, b)} \tag{3.8}$$

Si el valor de *sillhoute score* es cercano 1 significa que los datos están agrupados en el *cluster* correspondiente.

Si el valor es cercano 0 significa que los datos podrían pertenecer a otro *cluster*.

Si el valor es cercano a -1 los datos están en el *cluster* equivocado.

4 Experimentos y resultados

En este capitulo se presentaran diferentes resultados alcanzados utilizando diferentes configuraciones de hiperparámetros y divididos por arquitectura. Todos los experimentos fueron hechos con una GPU *RTX 2060* de 6 GB de VRAM. Se usaron 14 sujetos para el dataset de entrenamiento, 3 sujetos para validación y 3 sujetos para las pruebas. Se utilizo el algoritmo *T-SNE* implementado en la biblioteca *sklearn* y el algoritmo *UMAP* para la reducción de dimensionalidad.

Como métricas se usaron *sillhouete score* que mide el grado de cohesión o de similitud entre puntos vecinos dada una etiqueta, esta metrica varia entre -1 y 1 donde -1 significa un nulo grado de similitud y 1 significa un alto grado de similitud. Valores cercanos a 0 representan grupos de datos que se sobreponen entre si. Como métricas de clasificación se tienen *accuracy* y *F1-Score*.

Para todos los experimentos los hiperparametros modificados son:

1. dataset train len: Tamaño del Dataset de Entrenamiento.
2. dataset validation len: Tamaño del Dataset de Validación.
3. dataset test len: Tamaño del Dataset de Test.
4. window size: Tamaño del fragmento de señal.
5. stride: Distancia entre dos fragmentos de señal contiguos.
6. train batch size: Tamaño del *batch* en el entrenamiento.
7. margin: Valor del margen en la función de *loss*.
8. learning rate: Valor de actualización para el optimizador.
9. optimizer: Optimizador utilizado.
10. epochs: Épocas de entrenamiento.

4.1. Red Neuronal Siamesa de tipo *Fully Connected*

A continuación se presentan las características de los experimentos divididos por tarea, resultados de las métricas utilizadas así como una reducción de dimensionalidad de los *embeddings* para poder visualizar su estructura utilizando 2 y 3 componentes. Al final de esta sección se presenta un resumen de los resultados más relevantes.

4.1.1. Tarea: Mismo Sujeto

Para este experimento se utilizó configuración de la tabla 4.1.

dataset train len	700,000
dataset validation len	100,000
dataset test len	200,000
window size	1024
stride	64
train batch size	186
margin	9.1867
learning rate	0.001216
optimizer	adam
epochs	20

Tabla 4.1: Configuración usada para el experimento.Tarea: Mismo Sujeto, Arquitectura: *Fully Connected*.

Como resultado del reporte de clasificación se obtuvo la tabla 4.2. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.3.

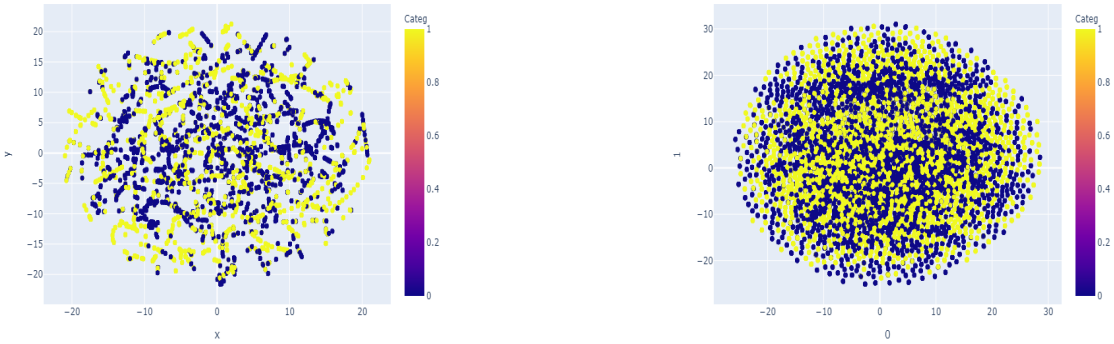
Accuracy	F1-Score-0	F1-Score-1
0.66	0.51	0.74

Tabla 4.2: Resultados del reporte de clasificación.

Categoría	Sujeto	Canal	Estimulo
0.01567	0.031192	-0.3294	-0.1376

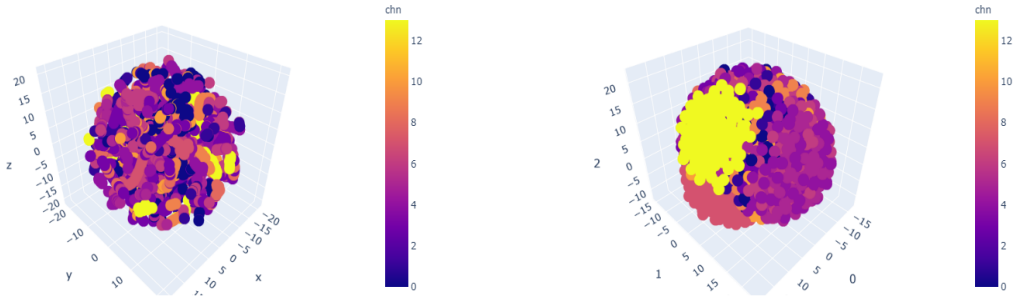
Tabla 4.3: Resultados de *Sillhouete score*

En las figuras 4.1, 4.2, 4.3, 4.4 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.



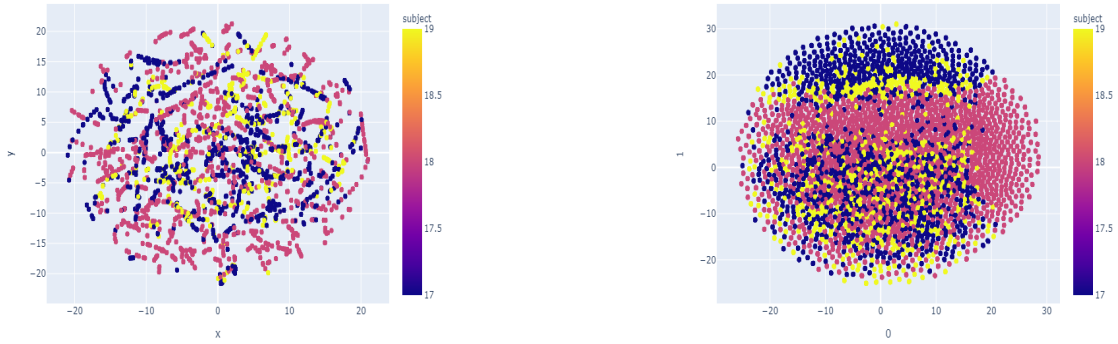
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.1: Visualización de los datos utilizando como etiqueta la tarea definida.



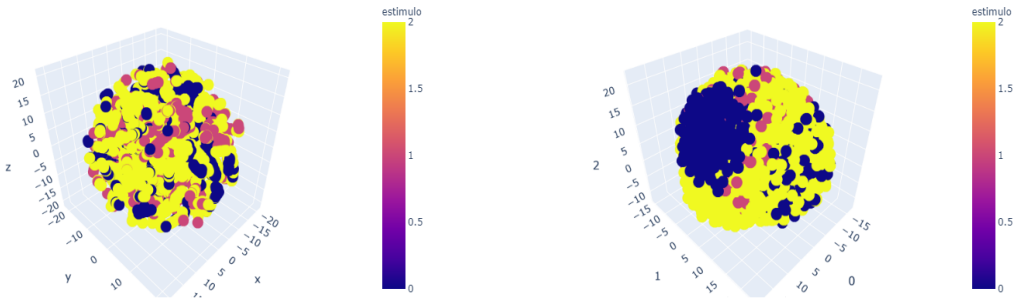
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.2: Visualización de los datos utilizando como etiqueta los canales.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.3: Visualización de los datos utilizando como etiqueta los sujetos.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.4: Visualización de los datos utilizando como etiqueta el estímulo.

4.1.2. Tarea: Mismo Canal

Para este experimento se utilizó configuración de la tabla 4.4. Como resultado

dataset train len	700,000
dataset validation len	100,000
dataset test len	200,000
window size	1024
stride	64
train batch size	77
margin	3.876866
learning rate	0.0001192
optimizer	adam
epochs	30

Tabla 4.4: Configuración usada para el experimento.Tarea: Mismo Canal, Arquitectura: *Fully Connected*.

del reporte de clasificación se obtuvo la tabla 4.5. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.6.

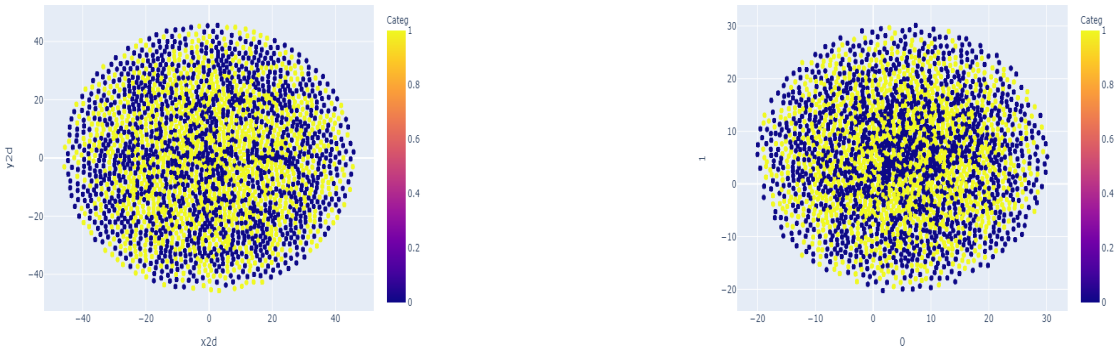
Accuracy	F1-Score-0	F1-Score-1
0.63	0.42	0.73

Tabla 4.5: Resultados del reporte de clasificación.

Categoría	Sujeto	Canal	Estimulo
0.0391	-0.2667	-0.4147	-0.15926

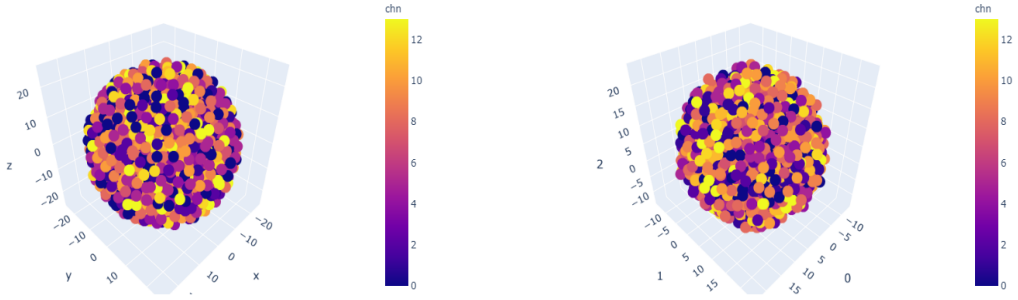
Tabla 4.6: Resultados de *Sillhouete score*

En las figuras 4.5, 4.6, 4.7, 4.8 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.



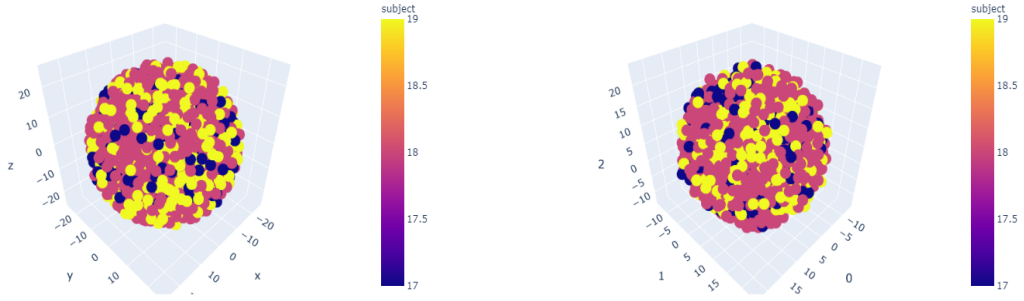
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.5: Visualización de los datos utilizando como etiqueta la tarea definida.



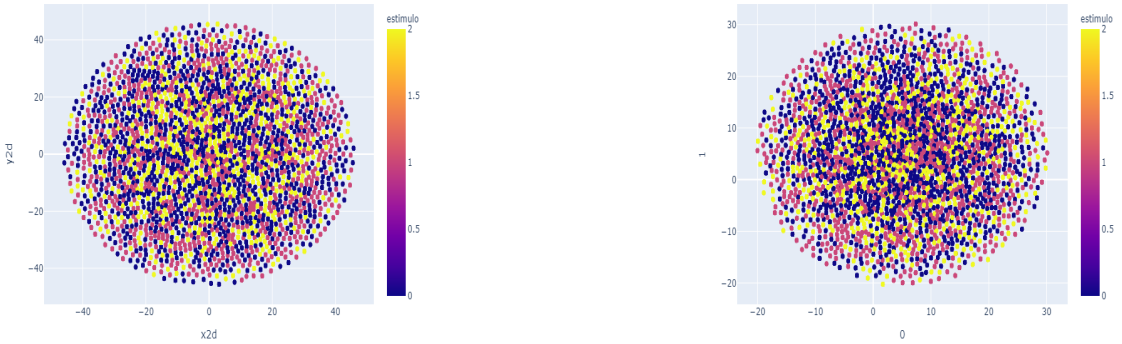
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.6: Visualización de los datos utilizando como etiqueta los canales.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.7: Visualización de los datos utilizando como etiqueta los sujetos.



(a) Gráfica Usando T-SNE.

(b) Gráfica Usando UMAP.

Figura 4.8: Visualización de los datos utilizando como etiqueta el estímulo.

4.1.3. Tarea: Segmento Consecutivo

Para este experimento se utilizó configuración de la tabla 4.7. Como resultado

dataset train len	93,800
dataset validation len	21,280
dataset test len	19,068
window size	256
stride	128
train batch size	256
margin	0.6
learning rate	0.0001
optimizer	adam
epochs	100

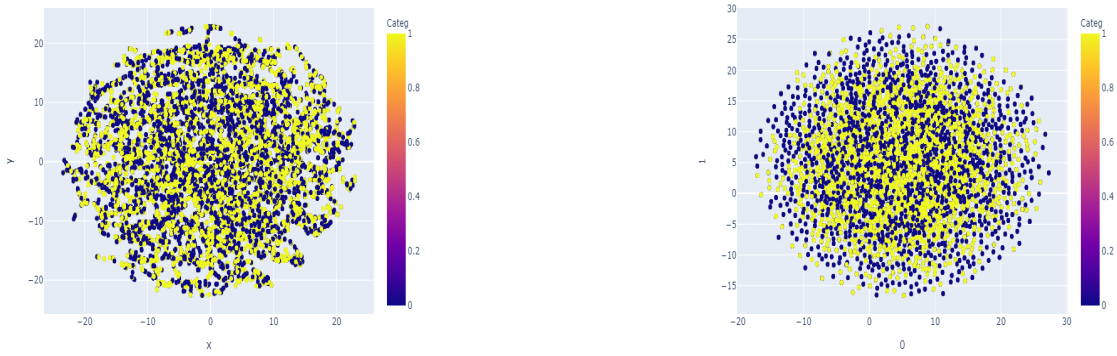
Tabla 4.7: Configuración usada para el experimento.Tarea: Segmento Consecutivo, Arquitectura: *Fully Connected*.

del reporte de clasificación se obtuvo la tabla 4.8. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.9.

Accuracy	F1-Score-0	F1-Score-1
0.50	0.00	0.67

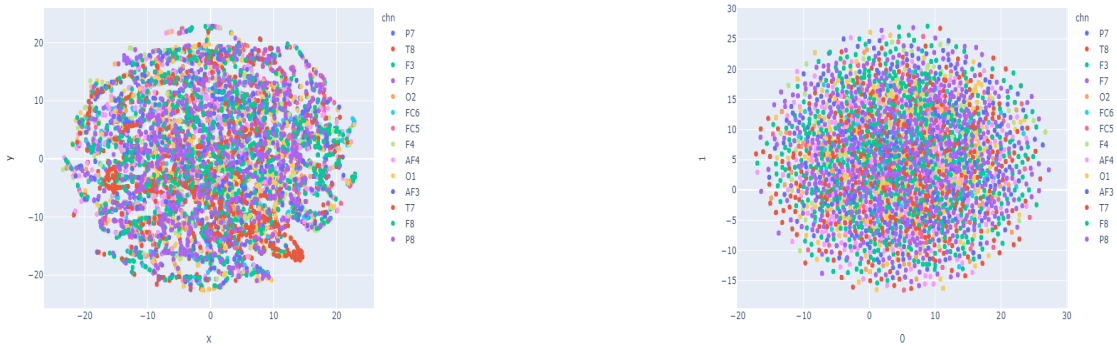
Tabla 4.8: Resultados del reporte de clasificación.

En las figuras 4.9, 4.10, 4.11, 4.12 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.



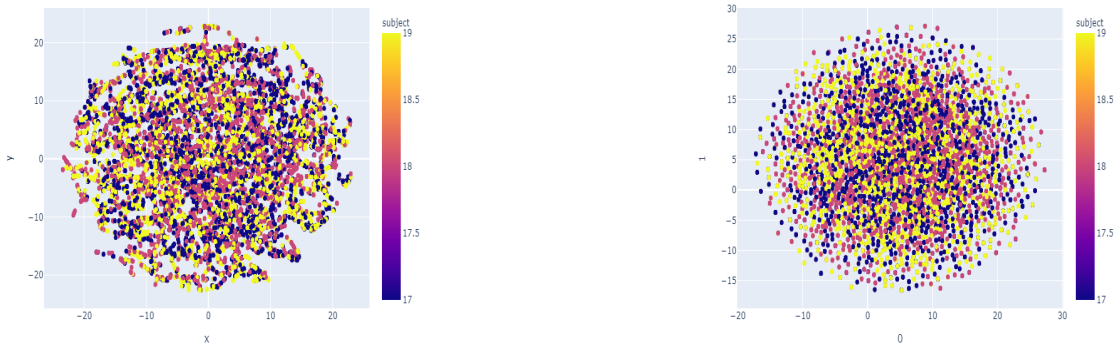
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.9: Visualización de los datos utilizando como etiqueta la tarea definida.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.10: Visualización de los datos utilizando como etiqueta los canales.

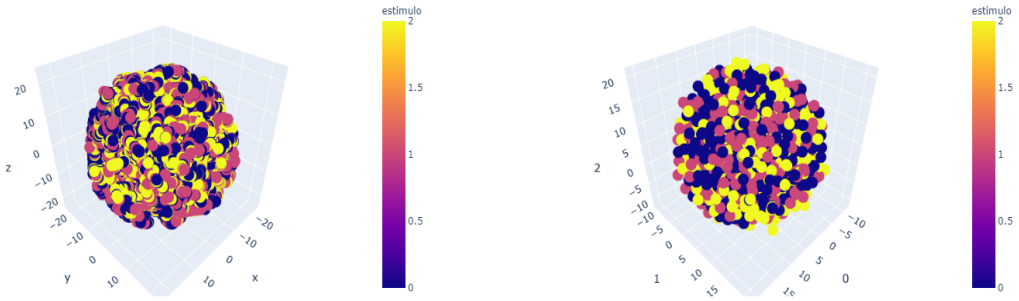


(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.11: Visualización de los datos utilizando como etiqueta los sujetos.

Categoría	Sujeto	Canal	Estimulo
-3.6317e-05	-0.0266	-0.1618	-0.0229

Tabla 4.9: Resultados de *Sillhouete score*



(a) Gráfica Usando T-SNE.

(b) Gráfica Usando UMAP.

Figura 4.12: Visualización de los datos utilizando como etiqueta el estímulo.

De todos los experimentos realizados se puede destacar que la tarea es importante para la arquitectura ya que obtiene un mejor *accuracy* con la tarea «Mismo Sujeto» y como se explico al inicio del capitulo, mientras el valor de *sillhouete* sea más alto, se puede decir que es mejor. Esto sucede con la tarea antes mencionada y en la figura 4.3b se observa hasta cierto punto una agrupación de puntos del mismo color lo que indica que esos «embeddings» comparten características. Otro resultado interesante es el valor de *sillhouete* en la tabla 4.6 evaluado en categoría o tarea espuria . Es el valor más alto de todos pero no se logra apreciar una agrupación clara en la figura 4.5. Esto puede ser debido a los algoritmos usados para reducir la dimensionalidad ya que tienen hiperparametros que pueden modificar el resultado de la gráfica. Otra razón es que mantienen una agrupación en la dimensión alta pero no es realmente significativa. Por ultimo la tarea «Segmento Consecutivo» es la que peor desempeño tiene, en la tabla 4.8 logra un *accuracy* de 0.5 por lo que el resultado es el mismo que el azar. Los resultados se observan en las figuras 4.9, 4.10, 4.11, 4.12 donde se muestra una distribución aleatoria, contrario a lo esperado y a los resultados de las otras dos tareas. Esto puede explicarse debido a la baja cantidad de datos que se pueden obtener por la tarea y a que no captura propiedades generales de las señales.

4.2. Red Neuronal Siamesa de tipo Convolucional 1D

A continuación se presentan las características de los experimentos divididos por tarea, resultados de las métricas utilizadas así como una reducción de dimensionalidad de los *embeddings* para poder visualizar su estructura utilizando 2 y 3 componentes. Al final de esta sección se presenta un resumen de los resultados más relevantes.

4.2.1. Tarea: Mismo Sujeto

Para este experimento se utilizó configuración de la tabla 4.10.

dataset train len	700,000
dataset validation len	100,000
dataset test len	200,000
window size	1072
stride	192
train batch size	205
margin	19.3014
learning rate	0.0227
optimizer	sgd
epochs	25

Tabla 4.10: Configuración usada para el experimento.Tarea: Mismo Sujeto, Arquitectura: *Convolucional 1D*.

Como resultado del reporte de clasificación se obtuvo la tabla 4.11. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.12.

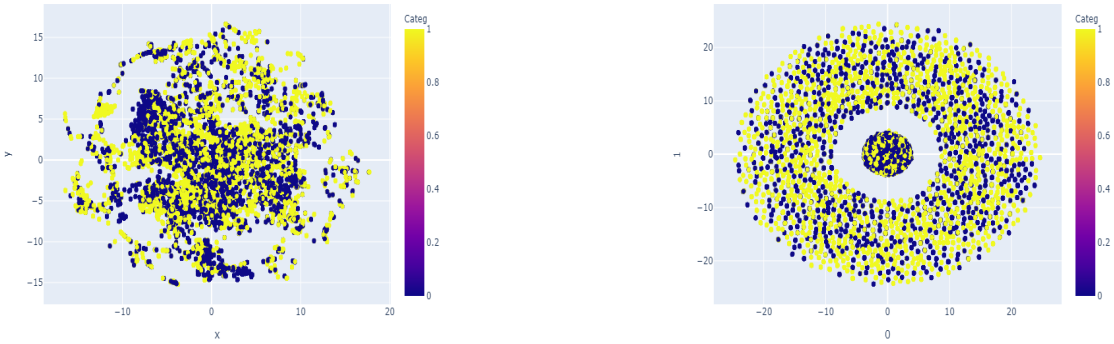
Accuracy	F1-Score-0	F1-Score-1
0.52	0.33	0.62

Tabla 4.11: Resultados del reporte de clasificación.

Categoría	Sujeto	Canal	Estimulo
0.0065	-0.0760	-0.3837	-0.0718

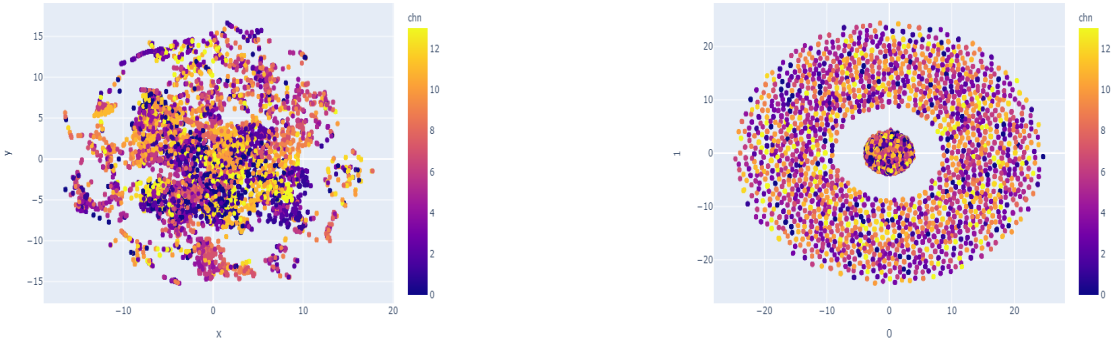
Tabla 4.12: Resultados de *Sillhouete score*

En las figuras 4.13, 4.14, 4.15, 4.16 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.



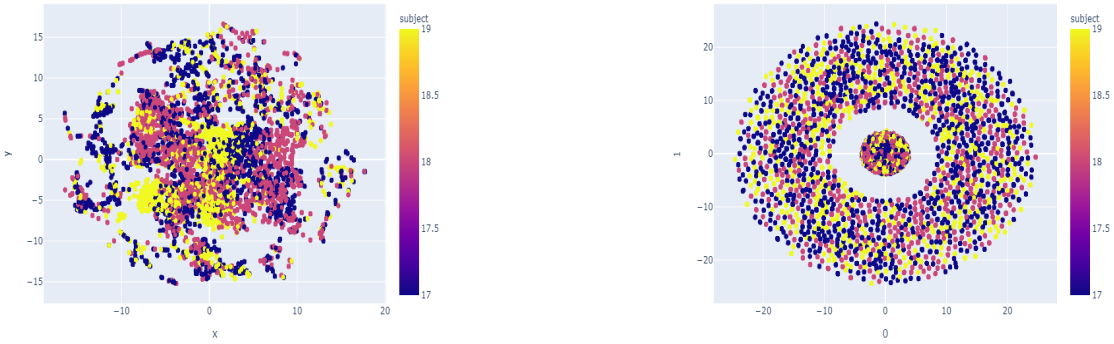
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.13: Visualización de los datos utilizando como etiqueta la tarea definida.



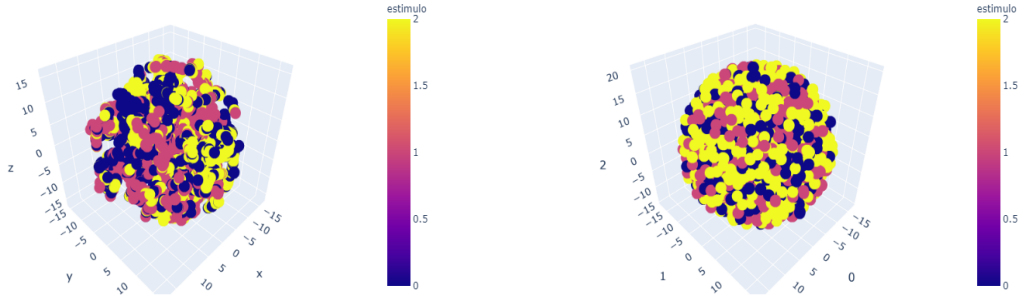
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.14: Visualización de los datos utilizando como etiqueta los canales.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.15: Visualización de los datos utilizando como etiqueta los sujetos.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.16: Visualización de los datos utilizando como etiqueta el estímulo.

4.2.2. Tarea: Mismo Canal

Para este experimento se utilizó configuración de la tabla 4.13. Como resultado

dataset train len	700,000
dataset validation len	100,000
dataset test len	200,000
window size	256
stride	128
train batch size	256
margin	0.6
learning rate	0.0001
optimizer	adam
epochs	50

Tabla 4.13: Configuración usada para el experimento.Tarea: Mismo Canal, Arquitectura: *Convolucional 1D*.

del reporte de clasificación se obtuvo la tabla 4.14. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.15.

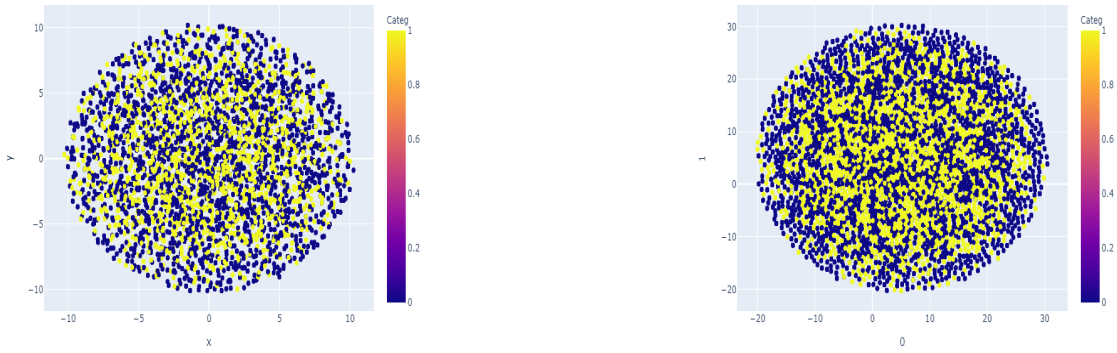
Accuracy	F1-Score-0	F1-Score-1
0.52	0.09	0.67

Tabla 4.14: Resultados del reporte de clasificación.

Categoría	Sujeto	Canal	Estimulo
0.0040	-0.1200	-0.4446	-0.0520

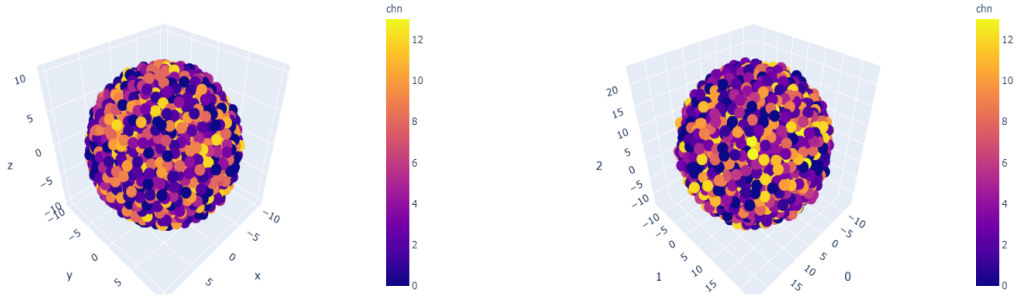
Tabla 4.15: Resultados de *Sillhouete score*

En las figuras 4.17, 4.18, 4.19, 4.20 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.



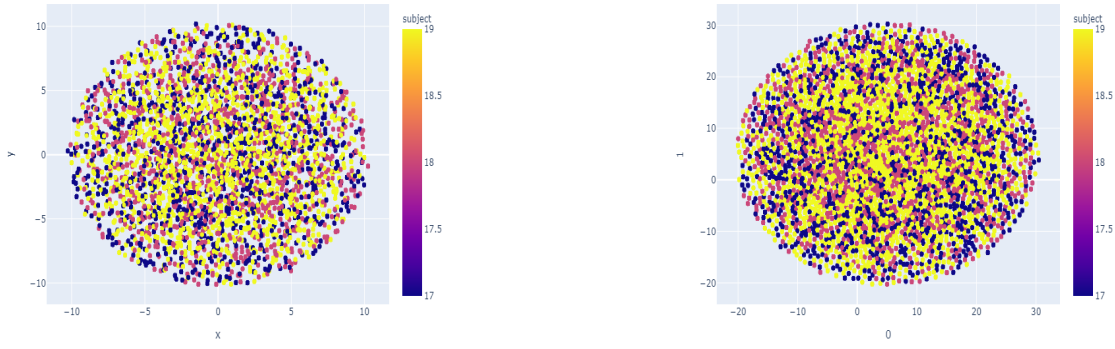
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.17: Visualización de los datos utilizando como etiqueta la tarea definida.



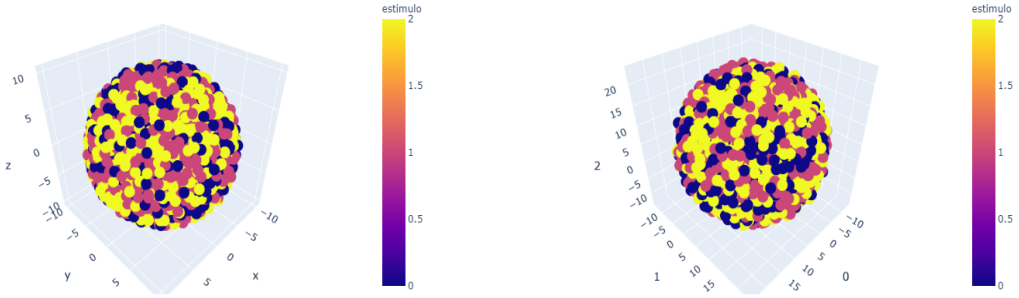
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.18: Visualización de los datos utilizando como etiqueta los canales.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.19: Visualización de los datos utilizando como etiqueta los sujetos.



(a) Gráfica Usando T-SNE.

(b) Gráfica Usando UMAP.

Figura 4.20: Visualización de los datos utilizando como etiqueta el estímulo.

4.2.3. Tarea: Segmento Consecutivo

Para este experimento se utilizó configuración de la tabla 4.16. Como resultado

dataset train len	93,800
dataset validation len	21,280
dataset test len	19,068
window size	256
stride	128
train batch size	256
margin	0.6
learning rate	0.0005
optimizer	adam
epochs	100

Tabla 4.16: Configuración usada para el experimento.Tarea: Segmento Consecutivo, Arquitectura: *Convolucional 1D*.

del reporte de clasificación se obtuvo la tabla 4.17. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.18.

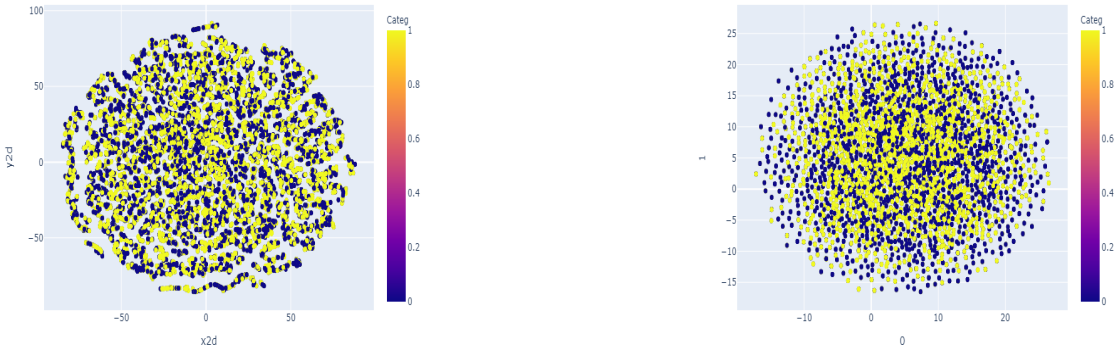
Accuracy	F1-Score-0	F1-Score-1
0.51	0.05	0.67

Tabla 4.17: Resultados del reporte de clasificación.

En las figuras 4.21, 4.22, 4.23, 4.24 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.

Categoría	Sujeto	Canal	Estimulo
-4.0092e-05	-0.0124	-0.1120	-0.0141

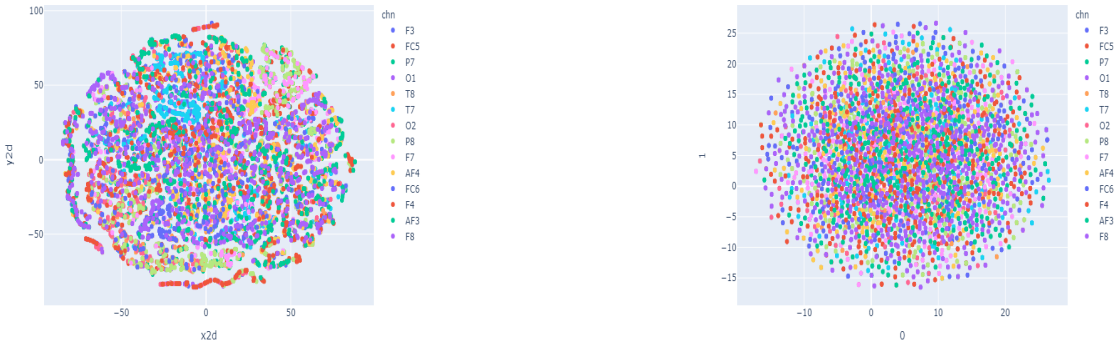
Tabla 4.18: Resultados de *Sillhouete score*



(a) Gráfica Usando T-SNE.

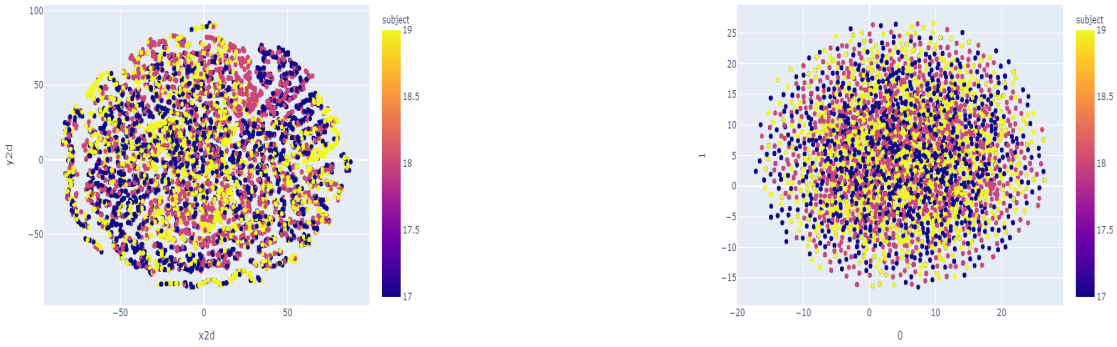
(b) Gráfica Usando UMAP.

Figura 4.21: Visualización de los datos utilizando como etiqueta la tarea definida.



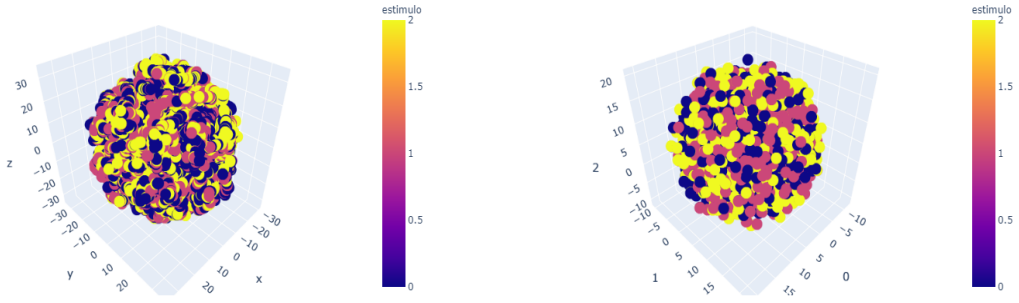
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.22: Visualización de los datos utilizando como etiqueta los canales.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.23: Visualización de los datos utilizando como etiqueta los sujetos.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.24: Visualización de los datos utilizando como etiqueta el estímulo.

De todos los experimentos realizados se puede destacar que la mejor tarea es «Mismo Sujeto» ya que en *accuracy* y *F1-score* obtiene valores más altos. Para la métrica *sillhouete* de igual forma se obtiene un valor más alto en la tarea antes mencionada pero no difiere de forma significativa con la tarea «Mismo Canal». En la figura 4.13b se observa una agrupación al centro del plano separado por un margen. Este fenómeno no se replica en la figura 4.13a debido a que los algoritmos reducen la dimensionalidad de forma distinta. En el caso de la figura 4.13b se puede decir que representa el *contrastive learning* y a pesar que no se logra ver una agrupación de puntos de la misma categoría, la red neuronal codificó características diferentes a la tarea establecida. Algo interesante a destacar es que los *embeddings* permanecen dentro de la mitad del valor del margen. En el caso de las figuras que utilizan el algoritmo *UMAP* la separación se encuentra justo en este valor. Por otro lado, en la tarea «Mismo Canal» en la figura 4.17b se aprecia una agrupación en distancias cortas lo cual corresponde con los valores obtenidos en las tablas 4.14 y 4.15 que muestran un nivel de agrupación en el que se traslapan los conjuntos o *clusters*.

Al igual que la red *Fully Connected* la tarea «Segmento Consecutivo» no parece tener un buen desempeño comparada con las otros dos tareas sin embargo algo destacado es que en la figura 4.22a se observan secuencias de datos del mismo color lo que representa una continuidad o similitud entre las señales.

4.3. Red Neuronal Siamesa de tipo Convolucional 2D

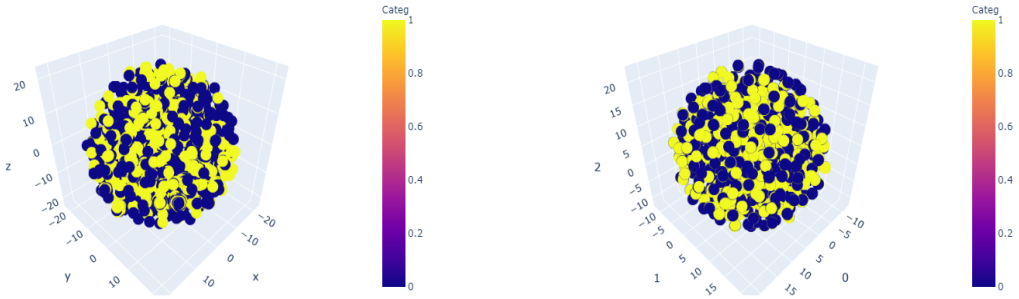
A continuación se presentan las características de los experimentos, resultados de las métricas utilizadas así como una reducción de dimensionalidad de los *embeddings* para poder visualizar su estructura utilizando 2 y 3 componentes. Al final de esta sección se presenta un resumen de los resultados más relevantes.

4.3.1. *Relative Positioning*: Experimento 1

Para este experimento se utilizó configuración de la tabla 4.19.

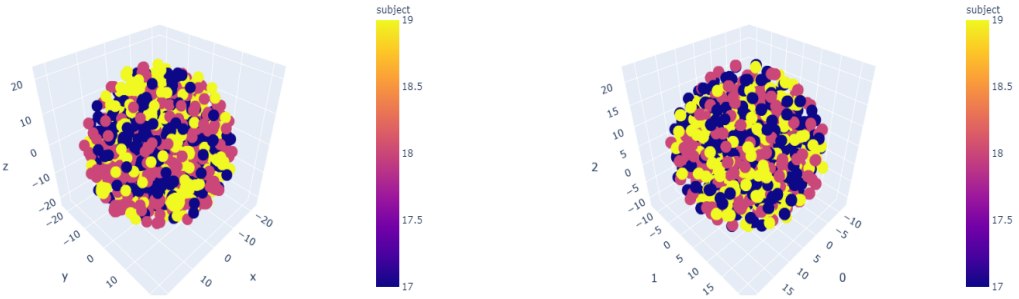
Como resultado del reporte de clasificación se obtuvo la tabla 4.20. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.21.

En las figuras 4.25, 4.26, 4.27 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.



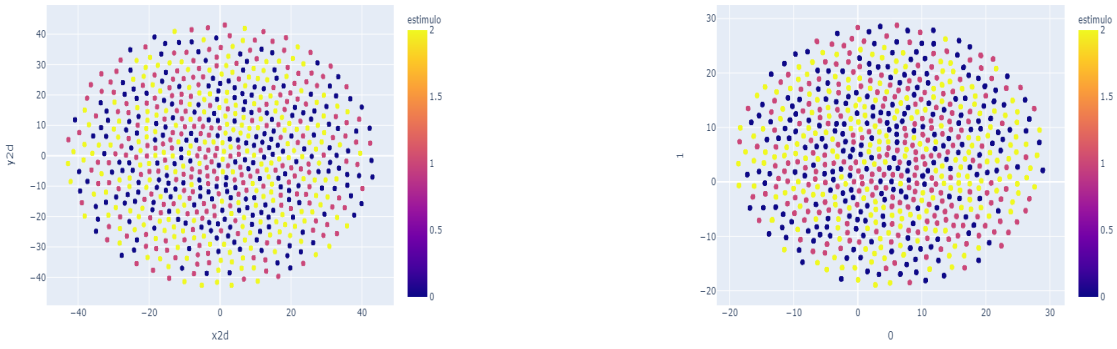
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.25: Visualización de los datos utilizando como etiqueta la tarea definida.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.26: Visualización de los datos utilizando como etiqueta los sujetos.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.27: Visualización de los datos utilizando como etiqueta el estímulo.

dataset train len	1,200,000
dataset validation len	100,000
dataset test len	200,000
window size	256
stride	128
train batch size	64
margin	0.6
learning rate	0.0002
optimizer	adam
epochs	40

Tabla 4.19: Configuración usada para el experimento.Tarea: *Relative Positioning*, Arquitectura: *Convolucional 2D*.

Accuracy	F1-Score-0	F1-Score-1
0.50	0.00	0.67

Tabla 4.20: Resultados del reporte de clasificación.

4.3.2. *Relative Positioning*: Experimento 2

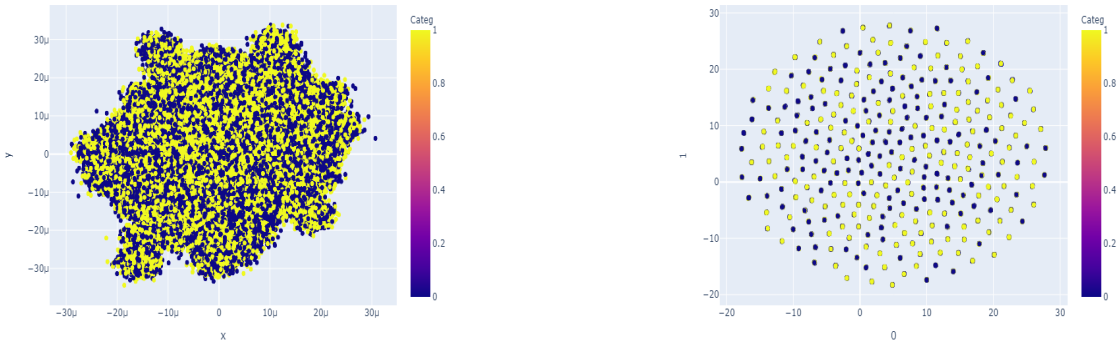
Para este experimento se utilizó configuración de la tabla 4.22.

Como resultado del reporte de clasificación se obtuvo la tabla 4.23. Y como resultado de la métrica *Sillhouete* se tiene la tabla 4.24.

En las figuras 4.28, 4.29, 4.30 se observan las diferentes visualizaciones en 2D y en 3D de los vectores resultantes de la red neuronal y etiquetados con diversas categorías.

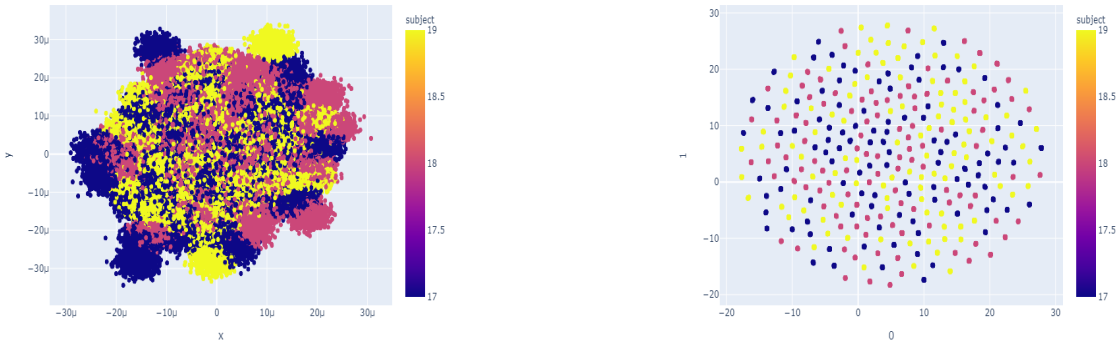
Categoría	Sujeto	Canal	Estimulo
5.0273e-05	0.0613	NA	0.0471

Tabla 4.21: Resultados de *Sillhouete score*



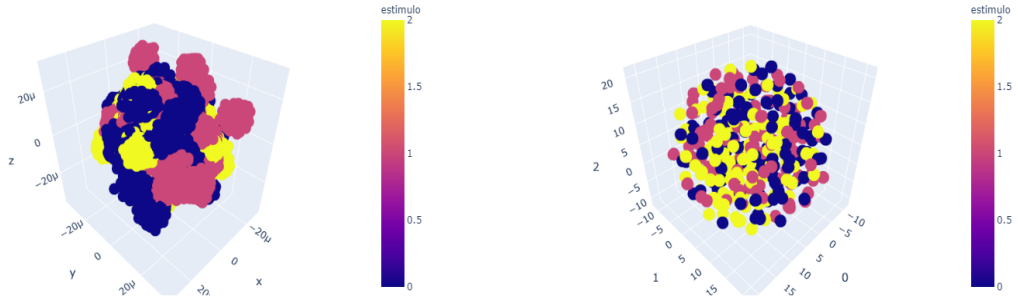
(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.28: Visualización de los datos utilizando como etiqueta la tarea definida.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.29: Visualización de los datos utilizando como etiqueta los sujetos.



(a) Gráfica Usando T-SNE. (b) Gráfica Usando UMAP.

Figura 4.30: Visualización de los datos utilizando como etiqueta el estímulo.

dataset train len	700,000
dataset validation len	100,000
dataset test len	200,000
window size	1024
stride	256
train batch size	16
margin	0.6
learning rate	0.0001
optimizer	adam
epochs	20

Tabla 4.22: Configuración usada para el experimento.Tarea: *Relative Positioning*, Arquitectura: *Convolucional 2D*.

Accuracy	F1-Score-0	F1-Score-1
0.50	0.02	0.67

Tabla 4.23: Resultados del reporte de clasificación.

Ambos experimentos fueron realizados con la misma tarea por lo que los resultados en *accuracy*, *F1-Score* y *Sillhouete* son muy parecidos. Al ser una arquitectura compleja, requiere de más poder de computo y más memoria VRAM lo que limita la experimentación. El uso de esta red aumenta el tiempo de ejecución y hace que el valor de *batch size* sea reducido y como se muestra más adelante, tener valores altos mejora el desempeño de la red neuronal. A pesar de que el valor de *accuracy* resulta muy bajo, los valores de *sillhouete* por sujeto son los más altos de todos los experimentos presentados siendo el experimento 1 mejor en esta métrica. Sin embargo la figura 4.25 muestra una distribución visualmente no separable. Caso contrario el experimento 2 que por ejemplo en las figuras 4.29a y 4.30a se muestran colores similares agrupados lo que indica posiblemente una mejor agrupación en la dimensión original. A pesar de la diferencia mínima entre los valores de *sillhouete* entre todos los experimentos y arquitecturas muestran características diferentes obtenidas por las redes neuronales.

4.4. Optimización de Hiperparámetros con *Optuna*

La búsqueda de hiperparámetros es una tarea importante que permite obtener la mejor versión de un modelo de *machine learning*. Los cambios pueden disparar el rendimiento del modelo dada una métrica establecida y a su vez permite encontrar los rangos de valores en los que la red neuronal obtiene los mejores resultados. *Optuna* es

Categoría	Sujeto	Canal	Estimulo
4.4545e-05	0.0394	NA	0.0010

Tabla 4.24: Resultados de *Sillhouete score*

un optimizador de hiperparámetros diseñado especificamente para *macchine learning*. Este *framework* es ligero y permite una fácil integración con cualquier *framework* de *Deep Learning* además tiene implementado un *dashboard* que permite visualizar de manera sencilla los *trials* que son directamente los experimentos realizados [36]. Estos experimentos son comparados basándose en una métrica establecida por el desarrollador.

Se realizaron tres experimentos correspondientes a las arquitecturas *Fully Connected*, Convolucional 1D y Convolucional 2D. Para todos los casos los hiperparámetros modificados fueron:

1. Optimizador: *Adam* o *SGD*.

2. *Learning Rate*: en el intervalo $[1x10^{-5}, 0.7]$.

3. *Window size*: varia dependiendo la arquitectura.

   Fully Connected $[64, 2048]$

   Convolucional 1D $[240, 2048]$

   Convolucional 2D $[192, 2048]$.

   Estos cambios son debido a las propiedades de las redes convolucionales.

4. *Stride* : en el intervalo $[64, 1024]$.

5. *Batch Size*: varia dependiendo la arquitectura.

   Fully Connected y Convolucional 1D en el intervalo $[32, 256]$

   Convolucional 2D en el intervalo $[2, 32768/window\ size]$

   Los cambios son debido a la poca VRAM disponible para el modelo convolucional 2D.

6. Margen : en el intervalo $[0.1, 10]$. Aunque en otros casos se probo un margen más alto.

7. Tareas : Selección dependiendo del modelo.

   Fully Connected y Convolucional 1D: «Mismo Canal», «Mismo Sujeto» y Segmento «Consecutivo».

   Convolucional 2D: *Relative Positioning*.

Como métrica de evaluación se toma el *accuracy* con el *dataset* de test.

4.4.1. Fully Connected

Se realizaron 121 experimentos, los primeros 100 fueron utilizando un *dataset* reducido de 100,000 elementos para entrenamiento y los siguientes 21 con el *dataset* completo. El resumen de los resultados se pueden ver en las figuras 4.31, 4.32, 4.33 y 4.34.

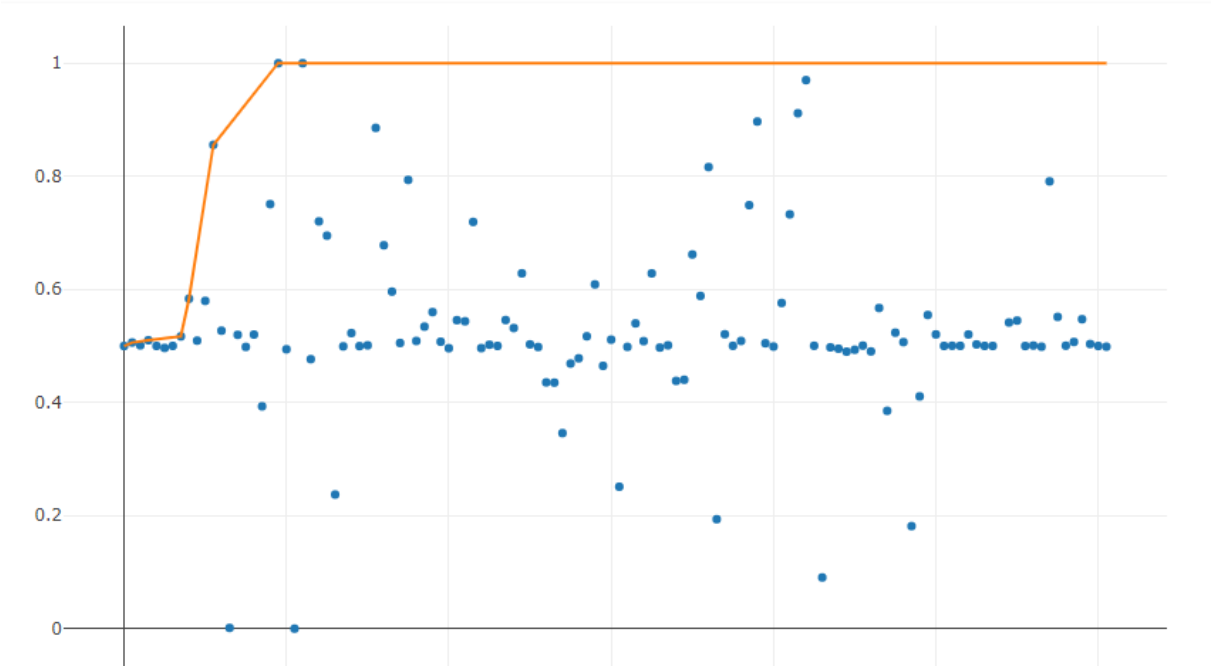


Figura 4.31: Gráfica del *accuracy* por cada experimento.

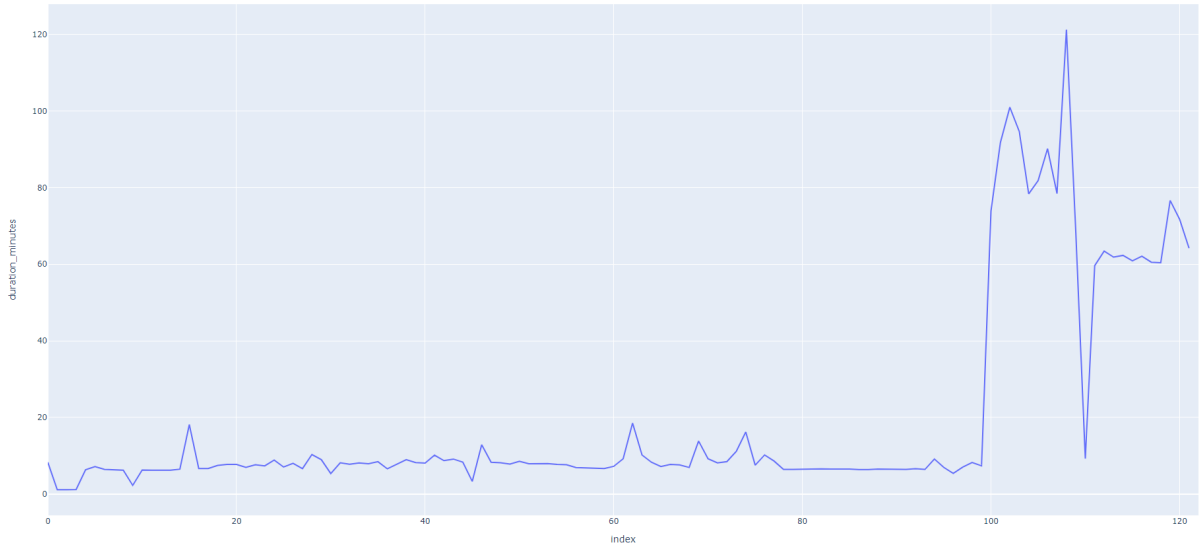


Figura 4.32: Gráfica de la duración de los experimentos.

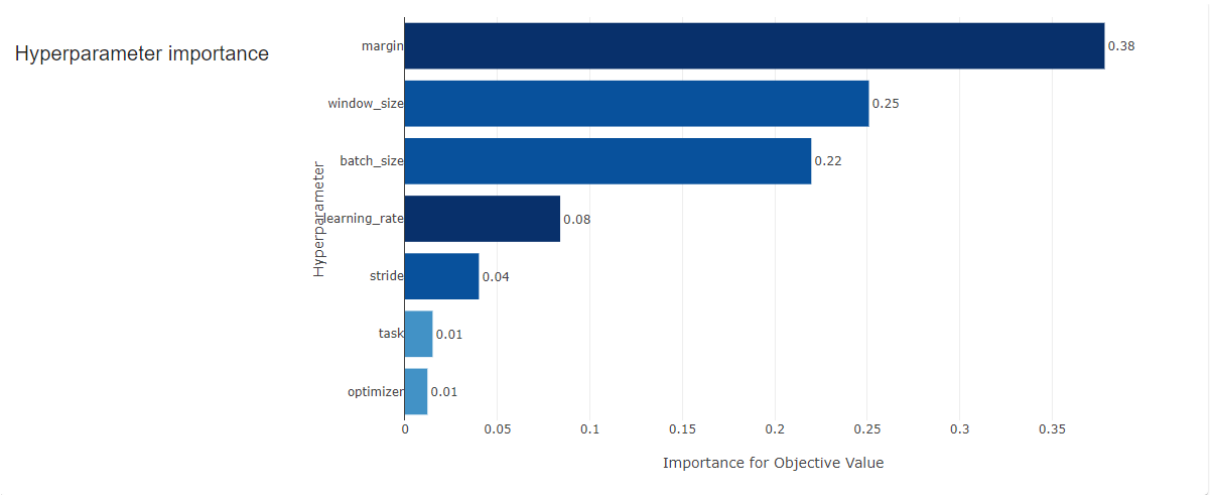


Figura 4.33: Gráfica de importancia de los hiperparámetros.

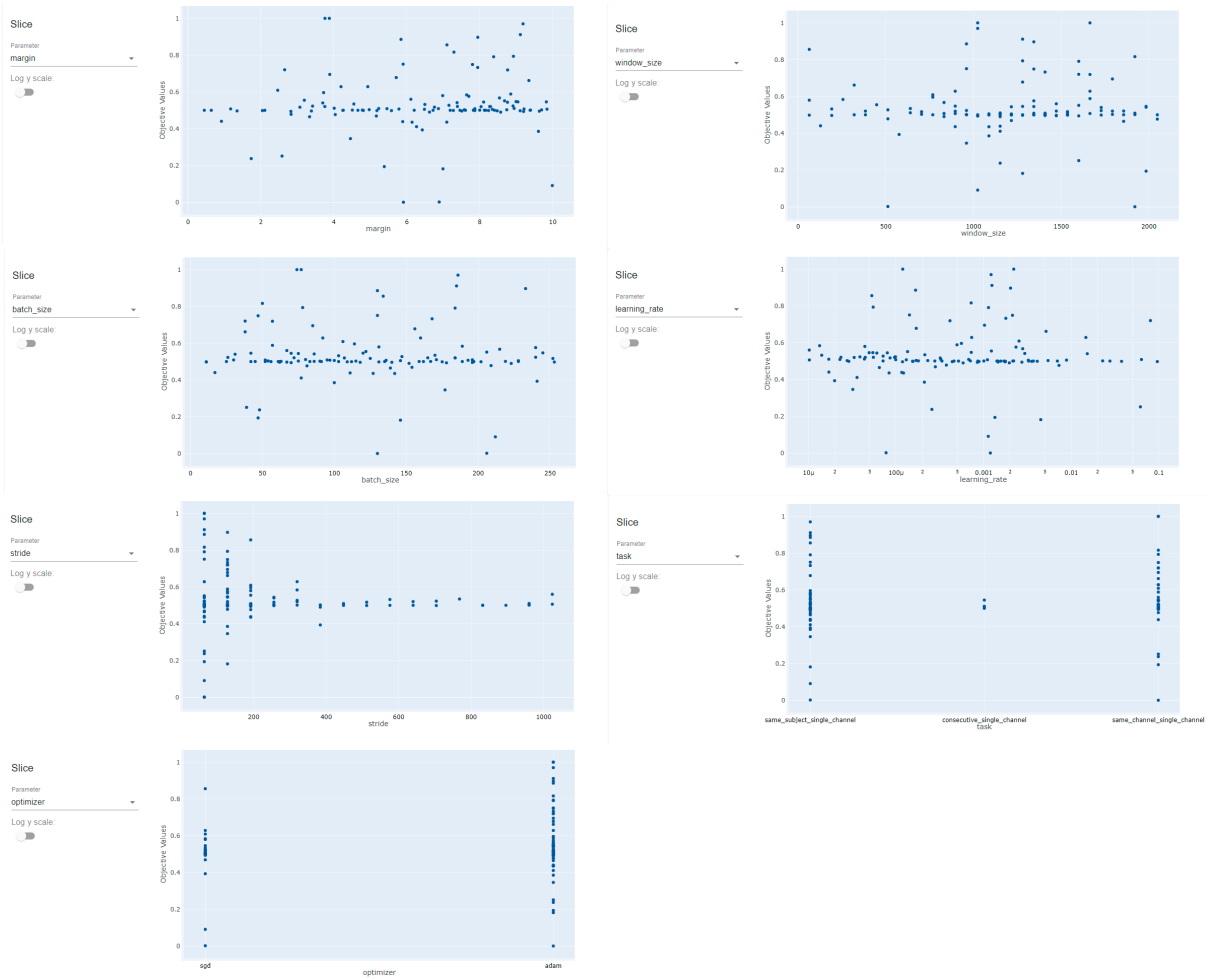


Figura 4.34: Gráfica del *accuracy* respecto al valor de los hiperparámetros.

Con pocos datos (100,000 por experimento) se logra un *accuracy* en test muy alto llegando hasta 1 (100 %) pero utilizando el *dataset* completo solo se alcanza un valor de 0.79101 así como se muestra en la figura 4.31.

Por otro lado la duración de los experimentos al tener un *dataset* más amplio se dispara la duración lo que hace más complicado dejar corriendo el optimizador, en la figura 4.32 se logra ver un cambio al utilizar más datos.

Optuna da una gráfica mostrando los hiperparámetros más importantes según su variación en los experimentos. Se muestra que debido a la arquitectura el margen es el parámetro más importante seguido del tamaño de la ventana y el tamaño del *batch*, estos valores nos indican que elegir estos parámetros es muy importante para obtener un buen desempeño. De forma contraria, la tarea y el optimizador resultan ser poco significativos en esta arquitectura por lo que elegir «Mismo Canal» o «Mismo Sujeto» no cambia drásticamente el resultado asi como el optimizador utilizado. En la figura 4.34 se observan los valores de *accuracy* obtenidos por cada hiperparámetro. Se puede observar que hay regiones en donde el margen, el tamaño de la ventana,

el tamaño del *batch* y el *learning rate* da mejores resultados. El *stride* tiene una tendencia a dar mejores resultados teniendo un valor bajo. La tarea y el optimizador tienen una importancia baja por lo que elegir cualquier opción puede dar un buen resultado.

4.4.2. Convolucional 1D

Se realizaron 80 experimentos con 10 fallidos, utilizando el *dataset* ampliado de 700,000 datos. El resumen de los resultados se pueden ver en las figuras 4.35, 4.36, 4.37 y 4.38.

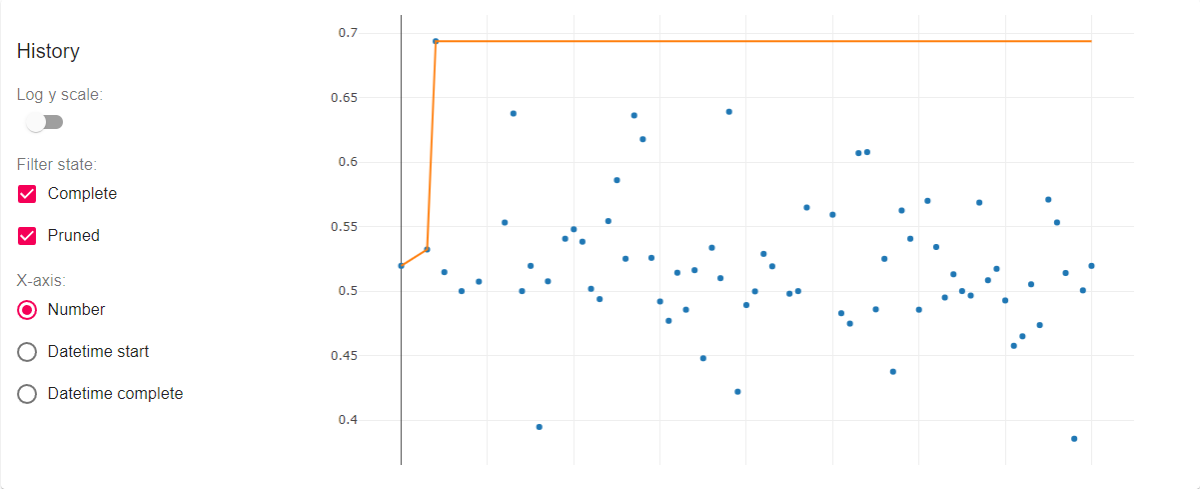


Figura 4.35: Gráfica del *accuracy* por cada experimento.

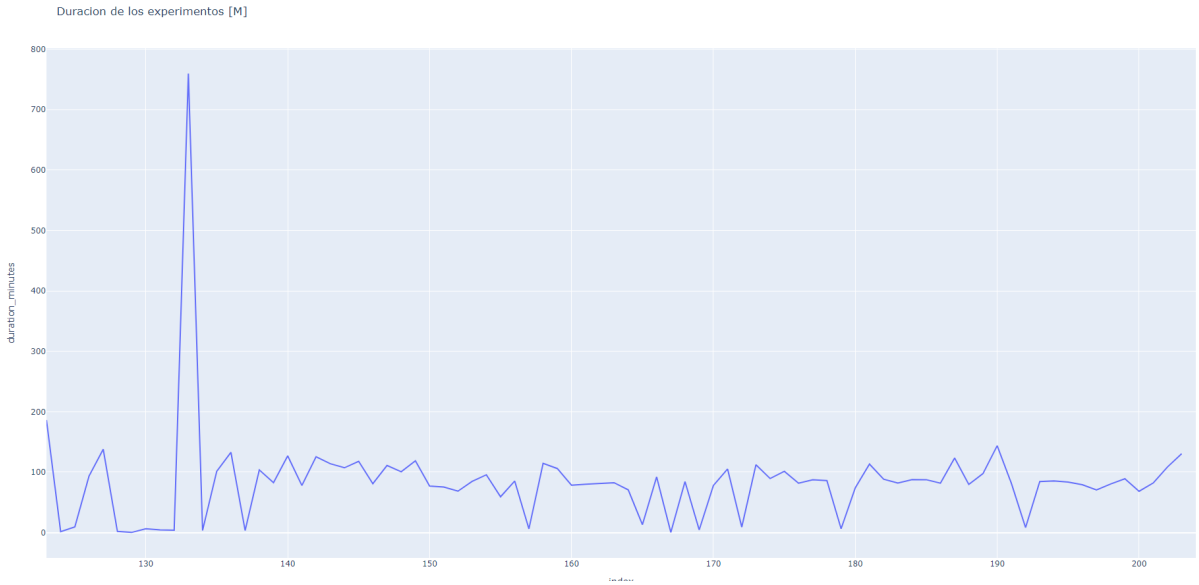


Figura 4.36: Gráfica de la duración de los experimentos.

Con el *dataset* ampliado se alcanza un *accuracy* en test de 0.69 con algunos resultados cercanos de 0.63 así como se observa en la figura 4.35. La duración promedio es de aproximadamente 100 minutos por lo que resulta más complicado ejecutar más experimentos con el optimizador.

Los parámetros más importantes difieren con respecto a la arquitectura *Fully Connected*, el margen en esta arquitectura parece ser poco relevante aunque esto puede deberse a un aumento en el rango de valores del margen llegando hasta el valor de 50. En la figura 4.38 se observan los valores de *accuracy* con respecto a cada parámetro. Se puede concluir que los mejores resultados se obtienen con ciertos valores bajos de *learning rate* pero con valores altos de *batch*. Los valores de margen, resultan ser mejores en el rango de 0 a 10. El tamaño de la ventana obtiene mejores resultados con valores altos mientras que el valor de *stride* es mejor si es bajo. Por último el optimizador en este caso resultó mejor *SGD* y las tareas resultan muy similares en desempeño siendo mejor la tarea «Mismo Sujeto»

4.4.3. Convolucional 2D

Debido a la gran cantidad de memoria VRAM consumida por el modelo más los datos y a la cantidad excesiva de tiempo por experimento (8 y 22 horas) resulta difícil continuar con la búsqueda de hiperparámetros con «*Hardware* casero». Los primeros dos experimentos arrojaron 0.507 y 0.5 de *accuracy* respectivamente.

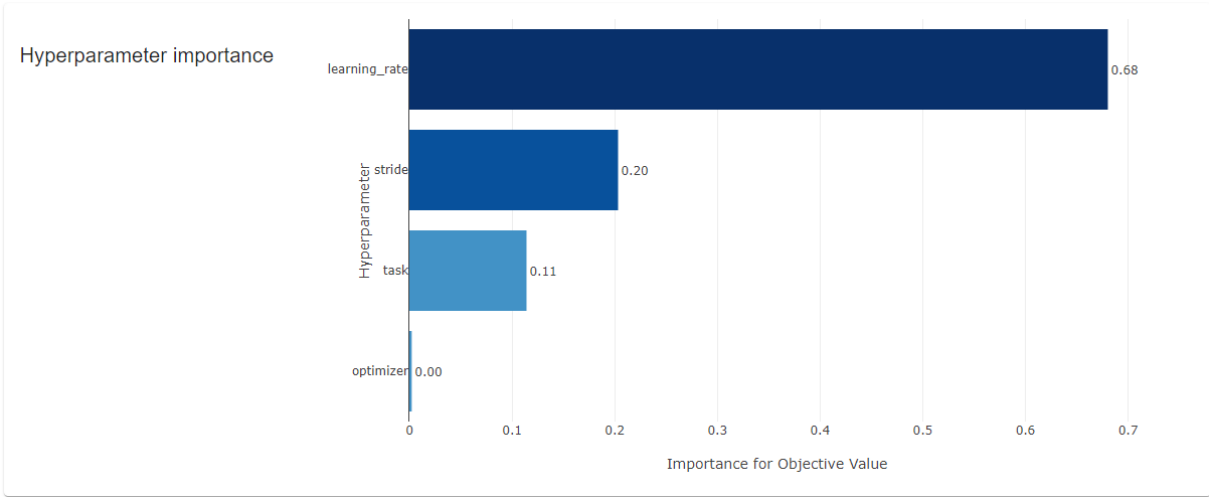


Figura 4.37: Gráfica de importancia de los hiperparámetros.

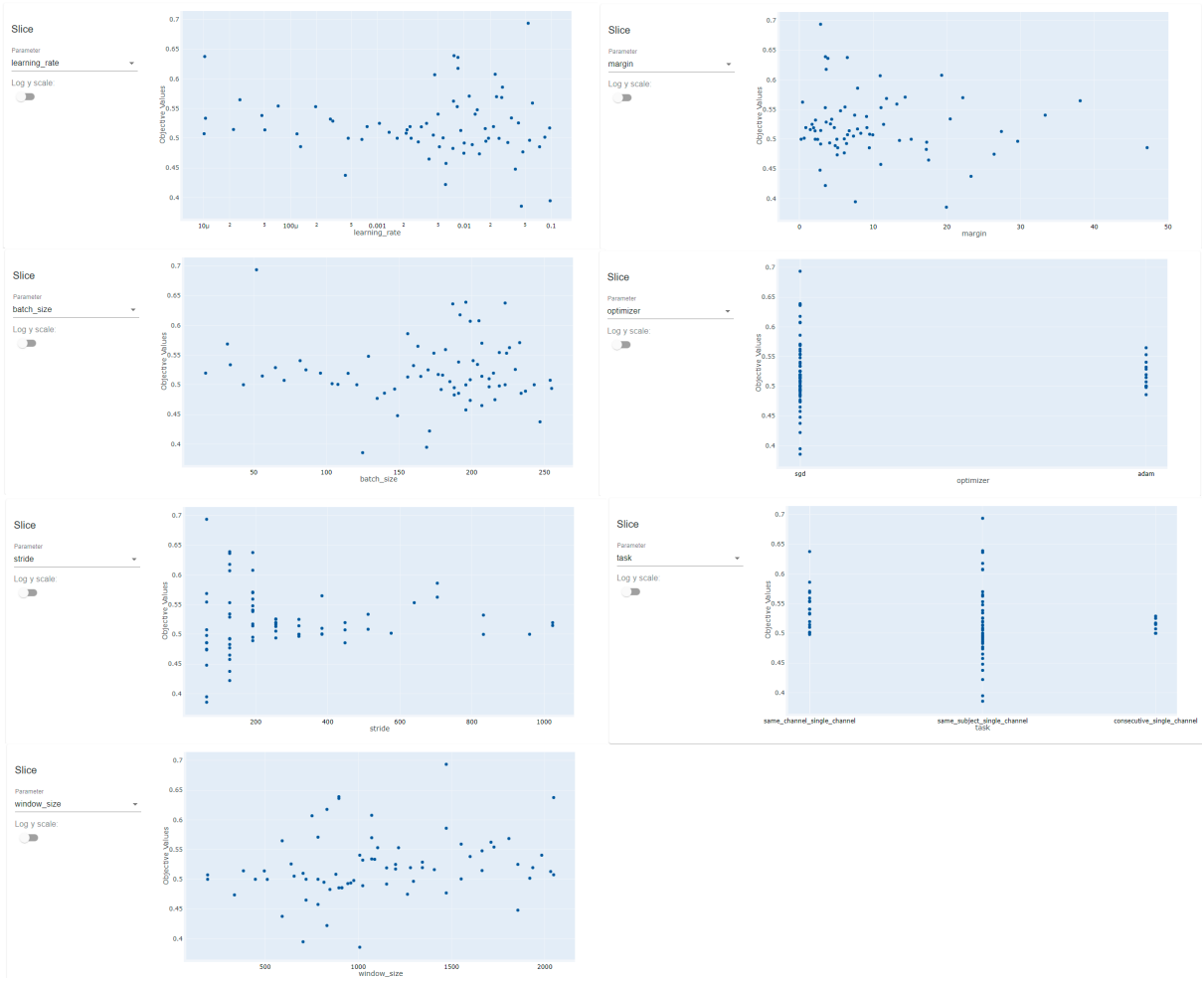


Figura 4.38: Gráfica del *accuracy* respecto al valor de los hiperparámetros.

5 Conclusiones

El uso de *machine learning* para resolver problemas da una gran ventaja al ser un sistema *end to end*. Debido a la complejidad del problema el uso de *self supervised* es una solución interesante y que se usa en otras áreas de la inteligencia artificial como en modelos de lenguaje y en modelos de visión computacional dando como resultado modelos robustos. Además de que el uso de representaciones vectoriales hace mas fácil que el modelo pueda adaptarse a otros dominios en los que no fue entrenado.

Hacer uso de *deep learning* aún se encuentra en exploración para muchas áreas diferentes a las ya establecidas como los es el procesamiento del lenguaje natural o visión computacional. Explorar el área de las neurociencias utilizando *deep learning* es un reto debido a la poca cantidad de investigación comparado con otras áreas ya mencionadas.

Estas técnicas recientemente han abordado problemas importantes como la predicción de estructuras en proteínas utilizando *deep learning*. Un campo en el que el uso y conocimiento de esta área permitió obtener un resultado impresionantes de la mano de expertos en *deep learning* como es la compañía *Deep Mind* con su modelo AlphaFold [37]. Esto motiva el interés por esta área y demuestra que estos modelos y arquitecturas pueden ser muy útiles.

Para manejar el problema de la codificación de señales EEG, debido a la limitada cantidad de datos y al objetivo principal de generar modelos que permitan generalizar el conocimiento se hizo uso de aprendizaje *self supervised.Self supervised* va acompañado de otras técnicas utilizadas en el área como lo es *contrastive learning* que sale del mundo del aprendizaje supervisado y pretende no solo crear un modelo para ejecutar una tarea en especifico si no obtener un modelo que permita captar propiedades generales de los datos y que para su aplicación posterior tenga «conocimiento». La generalización de los modelos de *machine learning* es un problema ya que aun se tienen modelos enfocados a una o varias tareas pero al cambiar el dominio del problema el modelo se vuelve ineficiente. A pesar de los avances en esta área llamada *continual learning* aún se tienen límites para lograr crear la inteligencia artificial general que permita generalizar el conocimiento así como lo hacemos los humanos. Por estas razones se propone el uso de *self supervised* y *contrastive learning*.

Contrastive Learning da un paso hacia la generalización y como se ha demostrado en modelos como SimCLR permite obtener mejores representaciones de imágenes por lo que puede ser fácilmente adaptado a otras tareas posteriores o *downstream tasks* pero conservando esa el conocimiento adquirido en el entrenamiento del modelo.

Para esta investigación el uso de estas técnicas fue relevante para crear un *framework* que permita crear modelos independientes de los datos además de proponer una forma fácil de incluir más tareas espurias o arquitecturas. Este *framework* puede permitir la investigación de otros modelos no presentados en este documento y que mejoren los resultados obtenidos proponiendo otros elementos como tareas, funciones de *loss* y arquitecturas pero manteniendo el objetivo principal ya mencionado.

Se utilizan herramientas *open source* lo que facilita su distribución y el manejo de licencias teniendo como principal lenguaje de programación python y como *framework* de *deep learning Pytorch*. Hacer uso de estas herramientas permite ayudar a los investigadores utilizar el software sin preocuparse por la compatibilidad y reproducibilidad de los experimentos tal como pasa con el software privativo.

Se realizaron múltiples experimentos y se le fueron añadiendo características al software por lo que los resultados mostrados no muestran en su totalidad el trabajo realizado. Sin embargo muestran los resultados más interesantes que tienen el conjunto de características añadidas como las gráficas y la implementación de otras métricas.

El *framework* diseñado permite añadir más funcionalidades y se podrían solucionar o explorar los problemas antes mencionados. En este trabajo solo se utilizó de una función de *loss* por lo que resultaría interesante explorar *Triplet Loss* o *NT-Xent loss* que son versiones mejoradas de la función utilizada en los experimentos. Los nuevos avances en el área utilizan *NT-Xent loss* por lo que se podría implementar y experimentar.

Las arquitecturas utilizadas cubren cierto espectro de los tipos de redes neuronales aunque se podrían implementar arquitecturas más complejas como *Transformers* que poco a poco se han popularizado por obtener buenos resultados en diferentes áreas. Utilizar las últimas arquitecturas requiere de un hardware potente por lo que resulta una limitante explorar su aplicación con estos datos.

Los resultados en las tareas espurias son bajos aunque en algunos casos son mejores que el azar llegando a tener como máximo un 0.79101 de exactitud. Estos bajos resultados se pueden deber al dominio de los datos y a los pocos datos analizados para realizar el entrenamiento y las pruebas ya que a pesar de realizar una fuerte regularización, la red seguía memorizando los datos y había un sobre ajuste. Otro posible causante del bajo rendimiento de la red neuronal es el tipo de tarea espuria

realizada, es probable que las tareas hayan sido muy complicadas para la red neuronal y no haya patrones detectables para «aprender» la tarea. En los modelos que usan *contrastive learning* es importante definir la tarea ya que puede tener mejor o peor rendimiento para problemas específicos.

Otra especulación es que el modelo haya colapsado en varios de los experimentos realizados, el colapso del modelo puede ser debido a varios factores dentro de los datos y del entrenamiento. El problema del colapso en arquitecturas que usan *contrastive self supervised learning* es conocido y hay diferentes formas de evitar el colapso total y colapso dimensional, además de que en trabajos previos se dan recomendaciones como un conjunto de hiperparámetros para evitar el colapso. Una causa probable es el uso de ejemplos negativos sobre los positivos en el entrenamiento, este aumento ha demostrado ser mejor para modelos que usan *self supervised learning* [1].

A pesar de que los resultados obtenidos no son los mejores ocurre algo muy interesante y es que en algunos casos la red neuronal no aprende la tarea propuesta de forma satisfactoria sin embargo produce representaciones vectoriales que de forma visual se distingue la separación entre las diferentes etiquetas así como sucede en la arquitectura con la red convolucional 2D. Las gráficas de reducción de dimensionalidad no implican que haya una agrupación de puntos en el espacio vectorial pertenecientes a la misma categoría sin embargo con la métrica *sillhoete* se puede medir el grado de agrupamiento o *clustering* que tienen las representaciones vectoriales. De forma teórica el mejor valor para ésta métrica es 1 pero en los resultados obtenidos el valor más alto es de 0.0613. Comparando estos resultados con el modelo *SimCLR* la obtención de estos valores coincide con los reportados en la investigación *Less can be more in contrastive learning* [38]. La comparación de los resultados da una perspectiva diferente sobre la realidad de los modelos generados utilizando *contrastive learning*. Permite obtener un valor de referencia para la generación de modelos posteriores.

Para tareas posteriores o *downstream tasks* es necesario el uso de etiquetas para obtener un clasificador robusto. Estas etiquetas pueden ser relacionadas a cualquier tipo de tarea que se requiera y disminuye considerablemente el uso de datos etiquetados para obtener por ejemplo un clasificador con un desempeño muy bueno. Esta característica de los modelos basados en *contrastive learning* permite trabajar con tareas en las que es muy difícil conseguir datos etiquetados. En el área de las neurociencias se requiere de un experto para identificar por ejemplo secciones donde hay artefactos pero este conocimiento se reduce solo a dichos expertos. La obtención de información en el área es complicada por lo que el uso de estos modelos ayuda a los especialistas en diversas tareas. En este trabajo no hubo exploración de *finetuning* en tareas *downstream tasks* ya que la cantidad de datos etiquetada es limitada y el rendimiento de los modelos no fue el mejor pero en otros trabajos se demuestra la capacidad de estos modelos para tareas de clasificación por ejemplo.

Las tareas no solo se quedan en la detección de artefactos, debido a la naturaleza del *contrastive learning* permite transferir el conocimiento adquirido de los datos a un sin fin de tareas que permitirían a los especialistas la eliminación de ruido en los registros pero aún más importante puede ayudar en la detección y predicción de diversos trastornos o enfermedades mentales. Así como sucede en otras áreas como el procesamiento del lenguaje natural, el uso de *deep learning* puede cambiar el estado actual de la investigación en psicología y neurociencias permitiendo abordar problemas de detección o de predicción de una forma fácil y en el que es complicado dar un diagnostico médico.

Finalmente, estas herramientas pueden ser de gran ayuda para la investigación en el área clínica, existen trastornos y enfermedades mentales que son difíciles de detectar por lo que desarrollar modelos utilizando *deep learning* no solo ayuda a los neurocientíficos si no a la sociedad en general. Cada vez se hace más visible el hecho de que existen las enfermedades mentales y que afectan a millones de personas en todo el mundo. Deben de ser detectadas y tratadas según el criterio de los especialistas, brindar estas herramientas generaría un impacto positivo en la sociedad.

Bibliografía

[1] L. Jing, P. Vincent, Y. LeCun, and Y. Tian, "Understanding dimensional collapse in contrastive self-supervised learning." [Online]. Available: https://arxiv.org/abs/2110.09348v3

[2] Manual sistema 10/20 internacional. [Online]. Available: http://liceaga.facmed.unam.mx/deptos/fisiologia/wp-content/uploads/2019/09/UTI-pr%C3%A1ctica-7-a.-Electroencefalograma.-AnexoManual.pdf

[3] *Neuroscience: science of the brain: an introduction for young students.* British Neuroscience Association, 2003. [Online]. Available: https://www.bna.org.uk/static/uploads/resources/BNA_English.pdf

[4] M. Shaurya. McCulloch-pitts neuron vs perceptron model. [Online]. Available: https://medium.com/@manushaurya/mcculloch-pitts-neuron-vs-perceptron-model-8668ed82c36

[5] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)." [Online]. Available: http://arxiv.org/abs/1606.08415

[6] A. Saxe and S. Salehi, "Deep linear neural networks. tutorial 1: Gradient descent and autograd." [Online]. Available: https://colab.research.google.com/github/NeuromatchAcademy/course-content-dl/blob/main/tutorials/W1D2_LinearDeepLearning/student/W1D2_Tutorial1.ipynb

[7] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations." [Online]. Available: http://arxiv.org/abs/2002.05709

[8] G. H. Ting Chen. Advancing self-supervised and semi-supervised learning with SimCLR. [Online]. Available: http://ai.googleblog.com/2020/04/advancing-self-supervised-and-semi.html

[9] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding." [Online]. Available: http://arxiv.org/abs/1807.03748

[10] M. Bekuzarov. Losses explained: Contrastive loss. [Online]. Available: https://medium.com/@maksym.bekuzarov/losses-explained-contrastive-loss-f8f57fe32246

[11] H. Banville, O. Chehab, A. Hyvärinen, D.-A. Engemann, and A. Gramfort, "Uncovering the structure of clinical EEG signals with self-supervised learning." [Online]. Available: http://arxiv.org/abs/2007.16104

[12] "Apa dictionary of psychology." [Online]. Available: https://dictionary.apa.org/electroencephalography

[13] J. P. O. Gonzáles and M. J. R. Avecillas, "Diseño y construcción de un prototipo de electroencefalógrafo para adquisición de señales cerebrales." Ph.D. dissertation. [Online]. Available: https://dspace.ups.edu.ec/bitstream/123456789/417/15/UPS-CT001873.pdf

[14] F. Wendling, F. Bartolomei, J.-J. Bellanger, J. Bourien, and P. Chauvel, "Epileptic fast intracerebral EEG activity: evidence for spatial decorrelation at seizure onset," vol. 126, pp. 1449–1459. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2040489/

[15] H. Aurlien, I. O. Gjerde, J. H. Aarseth, G. Eldøen, B. Karlsen, H. Skeidsvoll, and N. E. Gilhus, "EEG background activity described by a large computerized database," vol. 115, no. 3, pp. 665–673. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138824570300378X

[16] D. H. RAMÍREZ, "El machine learning a travÉs de los tiempos, y los aportes a la humanidad," 2018. [Online]. Available: https://repository.unilibre.edu.co/bitstream/handle/10901/17289/ELMACHINELEARNING.pdf?sequence=1&isAllowed=y

[17] D. Fumo. Types of machine learning algorithms you should know. [Online]. Available: https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861

[18] Multilayer perceptron. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/multilayer-perceptron

[19] Y. LeCun and A. Canziani, "Deep learning," Jun 2021. [Online]. Available: https://atcold.github.io/pytorch-Deep-Learning/

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[21] E. W. Weisstein, "Convolution." [Online]. Available: https://mathworld.wolfram.com/Convolution.html

[22] J. Brownlee. A gentle introduction to cross-entropy for machine learning. [Online]. Available: https://machinelearningmastery.com/cross-entropy-for-machine-learning/

[23] S. Ruder, "An overview of gradient descent optimization algorithms." [Online]. Available: http://arxiv.org/abs/1609.04747

[24] L. Ungar, "Regularization. tutorial 2: Regularization techniques part 2." [Online]. Available: https://colab.research.google.com/github/NeuromatchAcademy/course-content-dl/blob/main/tutorials/W1D5_Regularization/student/W1D5_Tutorial2.ipynb#scrollTo=4DAEv_YW__qE

[25] Amazon mechanical turk. [Online]. Available: https://www.mturk.com/

[26] Self-supervised learning - pretext tasks · deep learning. [Online]. Available: https://atcold.github.io/pytorch-Deep-Learning/en/week10/10-1/

[27] Energy-based models · deep learning. [Online]. Available: https://atcold.github.io/pytorch-Deep-Learning/en/week07/07-1/

[28] "Artificial neural networks." [Online]. Available: http://link.springer.com/10.1007/978-1-0716-0826-5

[29] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese"time delay neural network," in *Advances in Neural Information Processing Systems*, vol. 6. Morgan-Kaufmann. [Online]. Available: https://papers.nips.cc/paper/1993/hash/288cc0ff022877bd3df94bc9360b9c5d-Abstract.html

[30] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, vol. 2. IEEE, pp. 1735–1742. [Online]. Available: http://ieeexplore.ieee.org/document/1640964/

[31] M. A. Garcia, V. C. Ortiz, and R. H. Martinez.

[32] Classification: Accuracy | machine learning. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/accuracy

[33] T. Wood. F-score. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/f-score

[34] ——. Precision and recall. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall

[35] K. R. Shahapure and C. Nicholas, "Cluster quality analysis using silhouette score," in *2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA)*, 2020, pp. 747–748.

[36] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[37] AlphaFold: Using AI for scientific discovery. [Online]. Available: https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery

[38] J. Mitrovic, B. McWilliams, and M. Rey, "Less can be more in contrastive learning." PMLR, pp. 70–75, ISSN: 2640-3498. [Online]. Available: https://proceedings.mlr.press/v137/mitrovic20a.html