Instituto Tecnológico de Buenos Aires

Escuela de ingeniería y tecnología

MODELO BASADO EN REDES NEURONALES PARA LA CLASIFICACIÓN DE LA ENFERMEDAD DE ALZHEIMER A PARTIR DE RESONANCIAS MAGNÉTICAS Y BIOMARCADORES

Alumno: Hasbani Jonathan Eliel (Leg: 56695)

Tutor: Ing. Tablón Alberto

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE BIOINGENIERO

BUENOS AIRES

1º CUATRIMESTRE 2020

Dedicatoria

A todos los que están, y a los que se fueron.

Agradecimientos

Quisiera agradecer en primer lugar a mi familia por todo el apoyo que me brindó en todos estos años. A mi mamá por siempre cuidarme y aconsejarme, por darme todo lo que tengo y por hacer posible, a través de su gran esfuerzo, que yo pudiera estudiar esta carrera y llegar a ser quien soy hoy. A mis abuelos, por brindarme todo su apoyo a lo largo de toda mi vida, por siempre estar para mí, y por las innumerables cosas que hicieron para que yo pueda llegar a este momento. A mi hermana, Yael, que siempre me brindó su cariño y apoyo. A mi hermano, José, que fue un gran referente para mí a la hora de transitar este camino arduo que es la carrera de Ingeniería. A Víctor, que vivió todos estos años de esfuerzo junto a mí y siempre me brindó su consejo y apoyo. A Agustina, mi novia, que me ayudó a hacer un gráfico esencial para este trabajo, y además soportó mis monólogos extensos y mi malhumor cuando algo andaba mal, pero también celebró conmigo cuando las cosas salieron bien y me acompaño durante todo este proyecto.

Por otro lado, quiero agradecer a mi tutor, el Ing. Alberto Tablón, por brindarme su tiempo, su conocimiento y dedicación, por siempre tener una visión constructiva de mi trabajo que me ayudó a avanzar, y por ayudarme a cerrar este proyecto de la mejor manera posible.

Quiero agradecer a todos mis compañeros y amigos que me dio esta carrera, con quienes transité a la par este camino, con quienes viví el día a día por varios años, con quienes festejé las buenas notas y en quienes me apoyé para superar los obstáculos. También quiero agradecer a mis amigos de toda la vida, ellos a quienes conozco desde chiquitito y siempre me apoyaron en todo este proceso.

Por último, quiero agradecer al ITBA, a mis profesores, y a todos aquellos que en todos estos años formaron parte de mi carrera, y me ayudaron a llegar hasta aquí.

Resumen

La enfermedad de Alzheimer es una enfermedad neurodegenerativa que se destaca por conducir a la persona que la padece irreversible y gradualmente hacia la demencia y eventualmente la muerte. A lo largo de los años se ha adquirido un entendimiento cada vez más integral acerca de la patogénesis de la enfermedad, e incluso se desarrollaron diversos modelos que la describen, de los cuales el modelo llamado "Sistema A/T/N" es ampliamente aceptado en la actualidad. Este modelo patogénico describe el avance de la enfermedad de Alzheimer a partir del análisis de ciertos biomarcadores que describen los tres aspectos fundamentales de la patología: deposición amiloide, nudos neurofibrilares y neurodegeneración.

Por otro lado, los métodos de aprendizaje automático o *machine learning* se han destacado por sus notables resultados en problemas de clasificación en diversas áreas, entre ellas la medicina. El desarrollo de modelos basados en métodos de aprendizaje automático aplicados a la enfermedad de Alzheimer es de vital importancia para lograr generar herramientas de asistencia a la toma de decisiones médicas. A lo largo de los años, este problema fue abordado por diversos autores, los cuales introdujeron modelos con muy buenos desempeños pero con la limitación de sólo explorar un aspecto de la patogénesis. En este trabajo se propone un modelo que analiza los tres aspectos fundamentales de la patogénesis de la enfermedad de Alzheimer según el Sistema A/T/N y utiliza la combinación de datos obtenidos de resonancias magnéticas y otros biomarcadores relevantes para generar una clasificación conjunta. Los resultados obtenidos reflejan un excelente desempeño del modelo (habiendo obtenido métricas de evaluación como ROC-AUC con valores superiores al 92 %) y sugieren que el uso de todas la variables descritas por el Sistema A/T/N, modelo patogénico tomado como hipótesis fundamental, genera mejores resultados que utilizar las variables por separado, lo cual marca una tendencia para futuras líneas de investigación cuya orientación sea la creación de herramientas de asistencia a la toma de decisiones médicas.

Índice general

Capítulo 1

Introducción

1.1. Enfermedad de Alzheimer

La enfermedad de Alzheimer es una enfermedad neurodegenerativa que ocasiona demencia, lo cual implica la pérdida gradual de la memoria, el juicio y la habilidad para llevar a cabo las actividades de la vida cotidiana. Se trata de una enfermedad que tiene efectos catastróficos para la persona que la padece y para su entorno familiar. Se presenta usualmente en personas mayores a los 65 años de edad, aunque también se presenta en personas más jóvenes de manera más inusual.

La pérdida de memoria es la síntoma característico más común de la enfermedad, manifestándose sutilmente al principio pero empeorando con el tiempo hasta que interfiere con la mayor parte de los aspectos de la vida cotidiana de la persona. Incluso estando en ambientes familiares, la persona que padece de esta enfermedad puede perderse, confundirse, no reconocer a las personas de su entorno y hasta perder la habilidad para nombrar objetos. Otros aspectos en los que las personas afectadas por la enfermedad se ven perjudicados son en su capacidad para vestirse, comer y cuidar de su higiene personal.

A medida que la enfermedad avanza, algunas personas sufren de cambios en su personalidad y comportamiento, lo cual les dificulta interactuar de manera apropiada con otras personas. Debido al gradual incremento en la desorientación que experimentan las personas con el avance de la enfermedad, otros síntomas comunes que se presentan son agitación, inquietud y apartamiento. La pérdida del lenguaje también se encuentra entre los síntomas comunes del avance de la enfermedad.

En las etapas más avanzadas, la desorientación y alienación que tiene la persona que padece la enfermedad de Alzheimer llegan a dificultar su alimentación, higiene y cuidado, lo cual tiene como consecuencia la muerte del paciente.

1.1.1. Historia

En el año 1907, Aloysius Alzheimer describió los síntomas presentados por una mujer de 51 años de edad llamada Auguste Deter, quien estaba bajo su cuidado en la institución estatal psiquiátrica en la ciudad alemana de Frankfurt. La descripción de los síntomas hecha por Alzheimer fue la primera caracterización neuropsicológica de la enfermedad. Cuando Auguste Deter murió, Alzheimer examinó la histología de su cerebro microscópicamente. Al

hacer eso, observó las placas neuríticas, los nudos neurofibrilares y la angiopatía amiloide, que se irían a convertir en las características distintivas de la enfermedad.

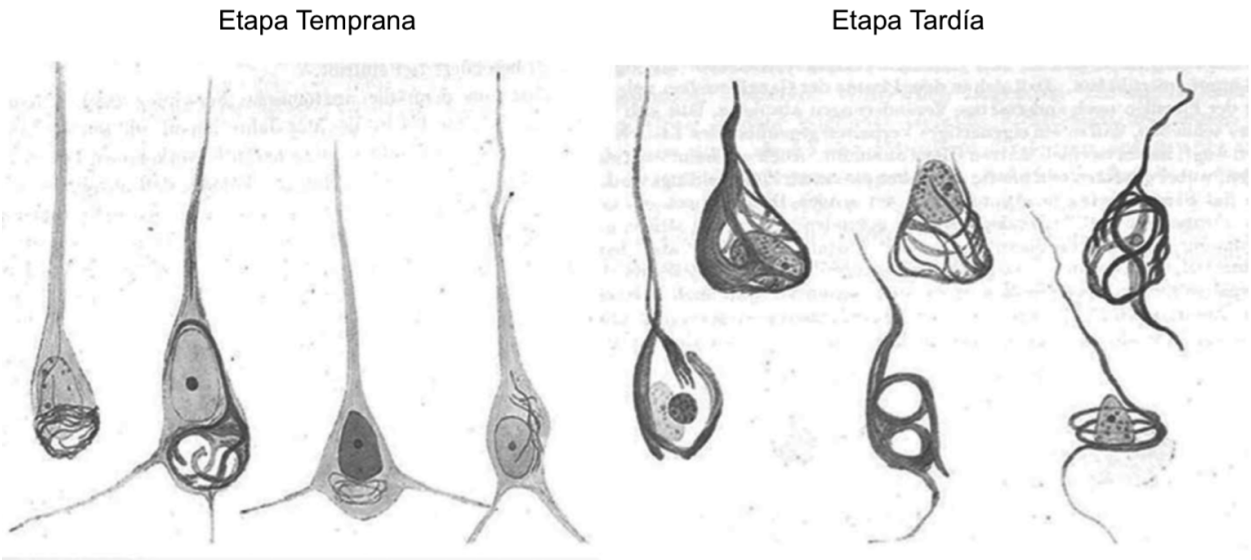


Figura 1.1: Dibujos hechos por Alzheimer a partir de la observación de los preparados histológicos de Auguste Deter. Se pueden ver los nudos neurofibrilares patológicos. [1]

En el año 1911, Alzheimer publicó sus hallazgos en un artículo llamado "Über eigenartige Krankheitsfälle des späteren Alters" y a partir de entonces la comunidad médica comenzó a utilizar sus descripciones para diagnosticar pacientes en Europa y Estados Unidos. Fue también en esta época que Eugen Blueler adoptó el término "psicosíndrome orgánico" en sus estudios de la esquizofrenia para referirse a disminuciones de la memoria, juicio, discriminación perceptiva, labilidad emocional y control deficiente de impulsos asociados con daño cortical difuso crónico. Además, esta descripción fue adoptada por la APA (American Psychiatric Association) para definir demencia en su manual *Diagnostic and Statistical Manual of Mental Disorders* (Manual de diagnóstico y estadísticas de desórdenes mentales), cuya segunda edición fue publicada el año 1968. Más específicamente, el manual de la APA define al "síndrome cerebral orgánico" como "una condición mental básica que resulta característicamente de una incapacidad difusa del tejido cerebral para funcionar por alguna causa", y que es manifestada en el comportamiento como incapacidad en la orientación, memoria y funciones intelectuales y de juicio [1].

Esta introducción de un criterio uniforme, junto con el desarrollo nuevas pruebas cognitivas estandarizadas permitieron la investigación de demencia particularmente enfocada en la enfermedad de Alzheimer (AD - Alzheimer's disease). A pesar de que la demencia está asociada con mas de 70 causas diferentes de disfunción cerebral, AD es la causa más común [1]. Uno de los estudios más importantes en este período mostró que el grado de AD presente en el cerebro estaba significativamente correlacionado con el desempeño del paciente en pruebas cognitivas estandarizadas [2]. Este fue el primer trabajo que unió fuertemente las características clínicas de AD con los cambios patológicos del cerebro que describió Alzheimer.

En los años siguientes, los estudios neuropsicológicos relacionados con la enfermedad de Alzheimer fueron pocos y mayormente limitados a la demencia presenil con inicio anterior a los 65 años de edad. Sin embargo, un cambio importante en el estudio de la demencia ocurrió en el año 1976 cuando Robert Katzman presentó datos que mostraban que la enfermedad de Alzheimer senil y presenil son histopatológicamente idénticas y sugirió que, basándose en datos epidemológicos, dicha enfermedad era la cuarta causa de muerte en los ancianos. Con esta información, la AD pasó de ser considerada una enfermedad relativamente rara a un asunto de salud pública importante. Esto otorgó a la enfermedad más atención del público y de las instituciones que pertenecientes al NIH (National Institutes of Health) que llevó a la creación del programa National Alzheimer's Disease Research Center para estudiar la causa, neuropatología y características clínicas de la enfermedad de Alzheimer. También, en el año 1980 la APA refinó el criterio para el diagnóstico de la enfermedad y en el año 1992 se establecieron criterios específicos para la investigación y diagnóstico de AD por la Organización Mundial de la Salud en la décima revisión del "International Statistical Classification of Diseases and Related Health Problems".

En esta época de mayor exposición de la enfermedad, también fue creciendo la comprensión de que varios trastornos demenciales están asociados con patrones de habilidades cognitivas relativamente preservadas y deterioradas que varían según la etiología y la neuropatología de la enfermedad subyacente. Más adelante, en varios estudios, se delimitaron las diferencias cualitativas en los deficits cognitivos denominados transtornos demenciales corticales, como lo es la enfermedad de Alzheimer, y los subcorticales, y también muchos investigadores sugirieron que ambos tipos de demencia deberían ser reconocidos como síndromes distintos.

La década de 1990 introdujo mucha más profundización en el entendimiento de los síntomas neuropsicológicos de la enfermedad, tales como que el descubrimiento de que los signos patológicos más tempranos de AD ocurren extensamente en la zona del lóbulo temporal medial (hipocampo, amígdala cerebral, corteza entorrinal, corteza perirrinal y corteza parahipocampal posterior). Pero los avances más disruptivos que fueron introducidos en esa década tuvieron que ver con el aspecto genético de la enfermedad. Se identificaron mutaciones de tres genes diferentes en grandes familias que demostraron tener un patrón de herencia autosómica dominante de una forma de AD de inicio temprano (generalmente antes de los 60 años de edad): el gen de la proteína precursora amiloide (APP) en el cromosoma 21, el gen PSEN-1 (preselinina 1) en el cromosoma 14, y el gen PSEN-2 en el cromosoma 1. Aunque estas formas de AD hereditario son raras (componen aproximadamente el 1 o 2 % del total de los casos de la enfermedad), la investigación alrededor de los aspectos genéticos de la enfermedad es notable. Siguiendo esta línea de investigación, se identificó al alelo $\epsilon 4$ del gen de la Apolipoproteína E (APOE) del cromosoma 19 como un factor de riesgo genético más común para AD de inicio tardío. El alelo $\epsilon 4$ del gen de APOE está presente en el 50-60 % de los pacientes con AD (comparado con su prevalencia del 20-25 % en ancianos saludables), independientemente de si existe un historial de demencia familiar o no. A diferencia de los genes asociados con AD de inicio temprano, el alelo $\epsilon 4$ del APOE no es determinístico, pero proporciona un factor riesgo que triplica la probabilidad de desarrollar AD si una copia del alelo $\epsilon 4$, y la aumenta 15 veces si hay dos copias presentes [3]. . A pesar de que el APOE-$\epsilon 4$ continúa siendo el gen de susceptibilidad más potente, los estudios de asociación genómica han identificado más de 25 loci asociados con AD de inicio tardío, y el advenimiento de fac-

tores de riesgo poligénico contribuyen al entendimiento del aspecto genético de la progresión de la enfermedad de Alzheimer.

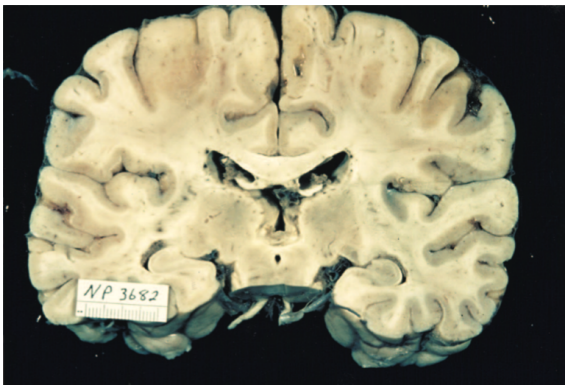
En el año 1999 Ron Petersen, Glenn Smith, y sus colegas de la Mayo Clinic introdujeron el concepto de "Mild Cognitive Impairment" (MCI) lo cual se traduce como deterioro cognitivo leve, y hace referencia al deterioro cognitivo del paciente que padece de la enfermedad de Alzheimer antes de ingresar en la demencia degenerativa severa. Se definió al MCI como la etapa de la enfermedad en la que los sujetos presentan pérdidas de memoria mayores que la uno podría esperar para su edad, pero que aún no cumplen con los requerimientos para tener demencia. Con la introducción de este nuevo concepto, la investigación y el estudio del MCI se difundió sustancialmente en la década del 2000, pasando de haber menos de 50 artículos del tema en el año 1999 a tener más de 900 artículos para el año 2007. En el año 2011, el National Institute on Aging and Alzheimer's Association (NIA-AA) publicó los lineamientos actualizados para el diagnóstico de MCI. A pesar de que el criterio para diagnosticar MCI había sido ampliamente adoptado, se ha demostrado que dichos criterios fueron puestos en práctica de manera limitada y por eso los resultados obtenidos tienen una alta tendencia a arrojar una gran tasa de falsos positivos en sus resultados. Por estas limitaciones, se adoptó un criterio más complejo para el diagnóstico de MCI en el que se utilizan varios tests para obtener un resultado combinado que proporciona más estabilidad diagnóstica. Tener un diagnóstico más certero de MCI genera que haya un seguimiento temprano del paciente previo a que se desencadene la demencia, si es que se desencadena, y permite adoptar nuevas líneas de investigación para estudiar la evolución de la conversión de un paciente con MCI a AD.

En el transcurso de los últimos 20 años se hicieron grandes avances en la identificación de marcadores biológicos in vivo de la enfermedad de Alzheimer. Se refinaron los métodos para medir los niveles de la proteína amiloide β y de la proteína tau en el líquido cefalorraquídeo, los cuales son la principal constituyente de la placa amiloide característica descrita por Alzheimer y de los nudos neurofibrilares, respectivamente. También, se desarrollaron métodos que permiten visualizar la ubicación de dichas placas amiloides y nudos neurofibrilares en imágenes de PET. Otros avances incluyen métodos para medir la atrofia hipocampal, cortical y del cerebro en general a partir de imágenes de resonancia magnética y el uso de la resonancia magnética funcional para detectar cambios patológicos asociados a la enfermedad de Alzheimer. La finalidad de todas estas técnicas es ayudar a comprender la enfermedad y desarrollar métodos que permitan clasificar AD y MCI. Todos estos biomarcadores han contribuido al incremento en la exactitud en la que que se detecta la enfermedad de Alzheimer antes del inicio de los síntomas cognitivos, y mejoraron la habilidad para diferenciar AD de otras patologías que llevan a la demencia.

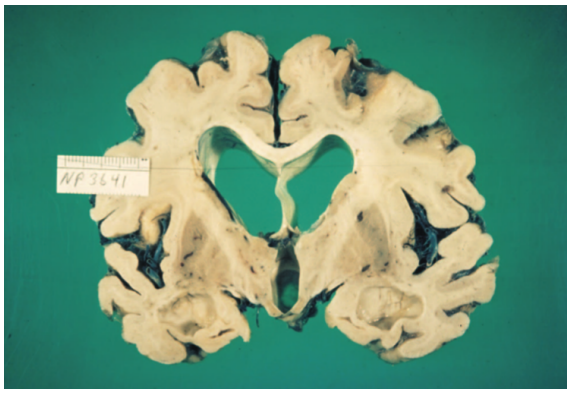
En la última década, muchos estudios examinaron la relación de los biomarcadores con el declive cognitivo y la demencia (ADNI, Australian Imaging, Biomarkers, Lifestyle study, NIAGADS, entre otros). Basándose en los resultados y datos que proporcionan dichos estudios, Jack et. al. [4] propuso un modelo hipotético de cambios dinámicos en los biomarcadores durante el desarrollo de AD. Su modelo, el cual era consistente con la *hipótesis de la cascada amiloide* [5], proponía que la deposición de proteína amiloide, relacionada con el procesamiento anormal de la APP (amiloidosis), impulsa la formación conglomerados anormales de proteína tau. Esto, a su vez, lleva a la lesión y degeneración neuronal mediada por los nudos neurofibrilares formados por la proteína tau. Sin embargo, a pesar de su amplia influencia,

la hipótesis de la cascada amiloide es cuestionada debido a la secuencia temporal de los eventos patológicos que ésta propone. Varios estudios muestran que la atrofia neurológica puede suceder antes que la amiloidosis en sujetos con AD en etapa prodrómica [1]. Además, muchos estudios han demostrado que el desarrollo de los biomarcadores en la mayoría de los sujetos con AD en etapa preclínica/prodrómica no sigue el orden temporal propuesto por la hipótesis de la cascada amiloide [1].

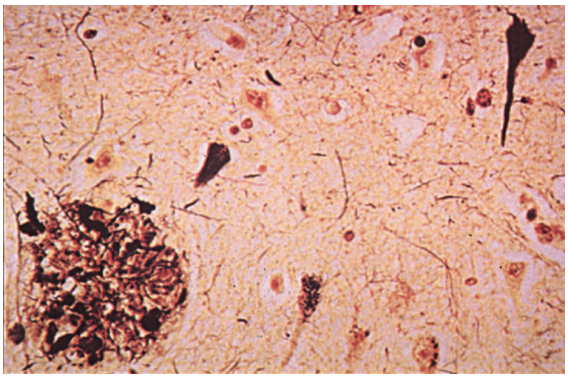
En el año 2016, Jack y Bennet et al en [6] reconocen estos hallazgos y proponen un esquema más descriptivo para el desarrollo de los biomarcadores en AD que es agnóstico al mecanismo temporal subyacente en la patogénesis de la enfermedad. Este nuevo modelo, conocido como *Sistema A/T/N* ("A" por Aβ(deposición amiloide medida con PET o Aβ-42 en líquido cefalorraquídeo) , "T" por Tau (detección de proteína tau en PET o p-tau en líquido cefalorraquídeo) , y "N" haciendo referencia a un biomarcador cuantitativo o topográfico de la Neurodegeneración (t-tau en líquido cefalorraquídeo, PET con FDG o Resonancia Magnética estructural)) propone una conceptualización más equipotencial del riesgo asociado a los biomarcadores en la enfermedad de AD. A pesar de la controversia y discusión alrededor de este modelo patogénico, cada vez más estudios se están enfocando en su confirmación. Un trabajo realizado en Argentina y publicado en diciembre del 2019 muestra de manera acotada la potencialidad de los biomarcadores mencionados en la prognosis de AD [7]. Evidentemente, el foco de la investigación en la enfermedad de Alzheimer en los próximos años va a estar situado en la profundización del rol de los biomarcadores y su correcta aplicación clínica para el diagnóstico temprano de AD.



(a) Corte sagital en la autopsia de un cerebro de un paciente sano.



(b) Corte sagital en la autopsia de un cerebro de un paciente con AD en el que se muestra la atrofia cerebral de la enfermedad



(c) Visualización microscópica post-mortem de la neuropatología de la enfermedad en la que se puede ver la placa neurítica (esquina inferior izquierda) y un nudo neurofibrilar (esquina superior derecha).

Figura 1.2: Visualización de la información de los sujetos seleccionados para el análisis

1.1.2. Diagnóstico clínico de Alzheimer

El diagnóstico clínico de la enfermedad de Alzheimer suele hacerse de acuerdo con los criterios estipulados por NINCDS-ADRDA, los cuales se basan principalmente en la exclusión de otros síndromes sistémicos o cerebrales que puedan ser causantes de la deterioración neurológica confinados al estadío de demencia y resultan en el mejor de los casos en un diagnóstico de AD posible o probable, mientras que un diagnóstico definitivo de la enfermedad de Alzheimer se obtiene a partir de una autopsia del tejido cerebral post-mortem. Un diagnóstico clínico de probable AD basado en estos criterios ha llegado a obtener en promedio 81 % de sensibilidad y 73 % de especificidad [8]. Los criterios de NINCDS-ADRDA fueron establecidos en el año 1984, y desde entonces fueron ampliamente adoptados en la práctica clínica durante 27 años. En el año 2011 dichos criterios fueron revisados y ampliados por el National Institute of Aging y el Alzheimer Association Workgroups (NIA-AA) [9], [10]. Esta revisión de los criterios incluyó, por ejemplo, el uso de los biomarcadores en la práctica

clínica.

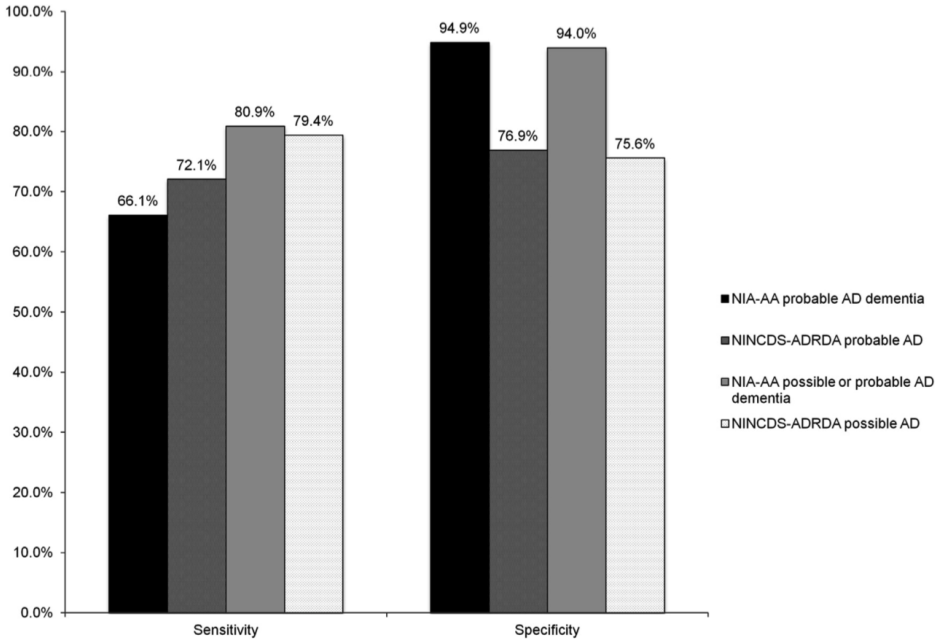


Figura 1.3: Comparación del criterio NINCDS-ADRDA(1984) con NIA-AA2011) [11]. Cabe aclarar que los resultados de NINCDS-ADRDA mostrados en este gráfico no son del todo compatibles con los de [8].

En la figura 1.3 podemos ver un gráfico de comparativa entre los criterios de diagnóstico NINCDS-ADRDA y la versión ampliada y revisada de NIA-AA. Se puede ver que el criterio de NIA-AA muestra un mejor desempeño en general que el NINCDS-ADRDA, aunque también tiene la peor sensibilidad para el diagnóstico de AD probable. A partir de este gráfico podemos establecer que el diagnóstico clínico de AD posible o probable utilizando el criterio de NIA-AA tiene una sensibilidad de 80.9 % y una especificidad de 94.0 % [11].

1.2. Deep Learning

1.2.1. Introducción

Los métodos de aprendizaje automático clásicos, o métodos de *machine learning*, son el motor de muchos aspectos de la tecnología moderna [12]. Sin embargo, estos métodos se encuentran limitados por su inhabilidad de procesar los datos en su forma cruda. Durante décadas, para construir un sistema de aprendizaje automático se requería de un diseño e ingeniería cuidadosos, y un manejo del área considerable para poder construir un extractor de características que transforme los datos crudos en representaciones adecuadas para que el sistema de machine learning, por ejemplo un clasificador, pueda realizar su tarea de detección o clasificación. Para sobreponerse a dicha limitación, los métodos aprendizaje de representaciones permiten que la "máquina" reciba datos crudos y descubra automáticamente las representaciones necesarias para realizar la tarea de detección o clasificación prevista. Los

métodos de aprendizaje profundo, o *deep learning*, son métodos de aprendizaje de representaciones que obtienen múltiples niveles de representación al combinar módulos simples pero no lineales, de los cuales cada uno transforma la representación de los datos en un nivel dado (empezando por los datos crudos) en una representación en un nivel más abstracto. Con la combinación de suficientes transformaciones de ese estilo, se pueden aprender funciones muy complejas. El aspecto clave del deep learning es que los mapas de características, obtenidos mediante cada transformación empleada, no son diseñados por un humano, sino que son aprendidos a partir de los datos usando un algoritmo de aprendizaje general. Debido a esto, existe un interés en particular en el diseño, desarrollo e investigación de modelos de deep learning en el ámbito de las ciencias médicas. Uno de los métodos más establecidos entre la gran variedad de métodos de deep learning es la red neuronal convolucional (CNN - Convolutional Neural Network), que ha conseguido resultados sobresalientes en varios campos, incluídos en la investigación médica [13].

Las redes neuronales forman parte del amplio grupo de métodos de aprendizaje supervisado. En estos métodos, durante la etapa de entrenamiento se le proporciona al modelo un conjunto de datos, a partir de los cuales el modelo produce una salida en la forma de un vector de probabilidades o "scores", uno por cada categoría posible dentro del conjunto de datos. El resultado que se espera obtener es que la categoría correcta para cada instancia de los datos sea la de mayor probabilidad en el vector de salida, lo cual no es probable que suceda antes de que el modelo sea entrenado. Para entrenar el modelo se computa una función de costo que mide el error (o la distancia) entre los scores de la salida y los scores deseados. Luego, el modelo modifica sus parámetros internos de manera que ese error se reduzca. Estos parámetros internos y modificables definen la función entrada-salida del modelo. Para ajustar adecuadamente el vector de parámetros, el modelo calcula un vector gradiente de la función de costo, el cual indica en qué magnitud el error aumentaría o disminuiría si el parámetro fuese incrementado en una pequeña cantidad. Luego, el vector de pesos es aumentado en la dirección que el gradiente disminuye. Una manera de interpretar esto es que la función de costo representa una superficie en el espacio de los parámetros, y que al modificar los parámetros en la dirección negativa del gradiente se busca llegar a un mínimo en dicha superficie, donde el error es mínimo.

1.2.2. Función de Costo

La función de costo mide el error a la salida del modelo en la tarea para la cual fue diseñado, y es vital en el entrenamiento del mismo ya que determina el sentido en el que los parámetros serán modificados. Por eso, debe ser cuidadosamente elegida para cada problema en particular. Para el problema de clasificación binaria, el cual se trata en este trabajo, la función de costo más utilizada en modelos de machine learning y deep learning es la *Entropía Cruzada Binaria* ó Binary Cross-Entropy.

La función de costo de entropía cruzada está definida por:

$$CE = -\sum_{i}^{C} t_i log(s_i)$$

donde t_i corresponde al valor real de la clase correspondiente, y s_i es el score del modelo

para cada clase i. En el problema de clasificación binaria en el que $C = 2$, la función de costo binaria quedaría definida por la ecuación:

$$BCE = -\sum_{i=1}^{2} t_i log(s_i) = -t_1 log(s_1) - (1 - t_1) log(1 - s_1)$$

donde $t_2 = 1 - t_1$ y $s_2 = 1 - s_1$. La entropía cruzada da una medida de la disimilaridad entre los valores reales de la clase y el valor estimado, por lo cual es de particular interés para el entrenamiento de modelos de machine learning y deep learning, ya que al minimizar la entropía cruzada se maximiza para probabilidad de que la clasificación realizada sea igual a la verdadera.

1.2.3. Overfitting

El desafío central en los problemas de machine learning y deep learning es que el método debe tener un buen desempeño al ser utilizado con datos nuevos, que no fueron vistos previamente, no solamente aquellos con los que el modelo fue entrenado. La habilidad de desempeñarse bien con datos nunca antes vistos se denomina **generalización**. El error que el modelo comete al ser probado con un conjunto de datos que no fueron utilizados para nada en el entrenamiento (conjunto de prueba o *test set*) se denomina error de generalización, y buscamos que sea lo menor posible. En el proceso de entrenamiento existen dos principales situaciones problemáticas que pueden ocurrir, las cuales se denominan **Underfitting** y **Overfitting**. Underfitting ocurre cuando el modelo no puede obtener un error lo suficientemente bajo en el conjunto de datos utilizados para entrenarlo (conjunto de entrenamiento o *training set*), es decir que el modelo no puede aprender las carácterísticas de los datos necesarias. Overfitting ocurre cuando la diferencia entre el error del entrenamiento y el error de prueba es demasiado grande [14], es decir que el modelo aprende en exceso las características de los datos de entrada y por eso no se desempeña bien sobre los datos de prueba.

Al principio del entrenamiento del modelo, la optimización y la generalización están correlacionadas: mientras más bajo sea el error en los datos de entrenamiento, también lo será el error en los datos de entrenamiento. Mientras esto sucede, el modelo se dice que tiene *underfitting*, y el modelo todavía tiene que aprender los patrones relevantes en los datos de entrenamiento. Pero luego de un número de de iteraciones sobre los datos de entrenamiento, la generalización deja de mejorar, y las métricas de validación comienzan a degradarse. Aquí el modelo comienza a sobreajustarse o manifestar *overfitting*, lo cual significa que comienza a aprender patrones que son específicos de los datos de entrenamiento pero que son inútiles para su utilización sobre datos nuevos.

1.2.4. Regularización

Para prevenir el overfitting en un modelo, la mejor solución es conseguir más datos [15]. Cuando eso no es posible, la siguiente mejor solución es modular la cantidad de información que el modelo tiene permitido aprender. Si una red sólo puede aprender un número pequeño de patrones, el proceso de optimización la va a forzar a enfocarse en los patrones

más importantes, lo cual le da la oportunidad de generalizar bien. El proceso de mitigación del overfitting de esta manera se denomina regularización.

La manera más simple de prevenir el overfitting es **reducir el tamaño del modelo**, lo cual reduce la cantidad de parámetros entrenables (o pesos) en el modelo o la *capacidad del modelo*. Un modelo con más parámetros tiene más capacidad para memorizar y por lo tanto puede aprender un mapeo perfecto entre los datos de entrenamiento y sus categorías, lo cual no tiene ninguna capacidad de generalización.

Otra técnica de regularización es agregar *regularización a los pesos*. Esta técnica de regularización se basa en la suposición que dados algunos datos de entrenamiento y una arquitectura de red, existen múltiples conjuntos de pesos (múltiples modelos) que pueden explicar los datos. Los modelos más simples son menos propensos a tener overfitting que los más complejos. Un modelo simple en este contexto es un modelo en el que la distribución de parámetros tiene menor entropía (o un modelo con menos parámetros). De esta forma, una manera común para mitigar el overfitting es restringir la complejidad del modelo al forzar que los pesos puedan tomar valores pequeños, lo cual genera que la distribución de pesos sea más regular. Esto se logra agregando a la función de costo otro costo asociado a tener grandes pesos. El costo agregado puede uno de los siguientes:

- *Regularización Lasso*: El costo agregado es proporcional a la suma del valor absoluto de los pesos (la norma l_1 de los pesos)
- *Regularización Ridge*: El costo agregado es proporcional a la suma del cuadrado de los pesos (la norma l_2 de los pesos)
- *Regularización Elastic-Net*: Combinación del uso de las normas l_1 y l_2

La técnica de regularización más utilizada es *Dropout* [16], y su aplicación a una capa de una red neuronal consiste en dejar de lado aleatoriamente algunas salidas de la capa (definir su valor como cero) durante el entrenamiento. La tasa de dropout es la fracción de salidas de la capa que son puestas en cero, y suele tener valores entre 0.2 y 0.5. La idea central de esta técnica es que al introducir ruido a la salida de las capas se pueden romper patrones que se forman de casualidad, no son significativos y que pueden llevar a la red a memorizar si no existiera tal ruido.

Además de las técnicas mencionadas, [17] introduce la técnica de Batch Normalization que fue originalmente pensada para acelerar el proceso de entrenamiento pero resultó mostrar efectos regularizadores que, en algunos casos, pueden reemplazar al dropout. Se trata de un método de reparametrización adaptativa de los datos a la salida de cada capa motivado por la dificultad de entrenar modelos muy profundos [14]. Lo que el método hace es normalizar adaptativamente los datos incluso mientras la media y el desvío estándar cambian en el tiempo durante el entrenamiento. El efecto principal del uso de batch normalization es que ayuda con la propagación del gradiente. En cuanto a los efectos de regularización de Batch Normalization, en [18] se explora una explicación analítica de el efecto regularizador de la técnica y se extiende dicho a efecto a redes neuronales convolucionales.

1.2.5. Evaluación del modelo

Para evaluar modelos basados en redes neuronales se separa una cantidad de los datos dedicada exclusivamente a probar el modelo luego del entrenamiento del mismo; si la cantidad de datos disponibles es grande se puede separar un 30 % para la prueba, pero si la cantidad de datos es poca se suele separar un 10 % o 20 % para este propósito. Los datos restantes se destinan al proceso de entrenamiento del modelo. En el caso de las redes neuronales, se suele utilizar una porción de los datos de entrenamiento para detectar si se comienza a manifestar overfitting. Para eso, también se toma alrededor del 10 % o 20 % de los datos de entrenamiento. Finalmente, los datos disponibles quedan divididos en tres particiones: La partición de entrenamiento, que es utilizada para que el modelo realice el proceso de entrenamiento y optimización de los parámetros entrenables; la partición de validación, que es utilizada para monitorear el desempeño del modelo durante el entrenamiento en datos no utilizados para optimizar los parámetros del mismo; y finalmente, la partición de prueba se utiliza cuando el proceso de entrenamiento terminó para evaluar el desempeño del modelo sobre datos nunca antes vistos, lo cual sirve para determinar si el modelo funciona bien con datos nuevos y puede generalizar bien.

La evaluación de un modelo, entonces, suele reducirse a la división de los datos en los sets de entrenamiento, validación y prueba, y su uso explicado. Esta división de los datos puede parecer simple, pero cuando hay pocos datos disponibles (como en el caso de problemas que utilizan datos médicos) los modelos entrenados pueden sufrir de algunos problemas relacionados al sesgo ocasionado por dicha división de datos, que pueden llevar a que el modelo no se desempeñe correctamente o a que los resultados que se obtienen no sean válidos; por ejemplo, si se disponen de muy pocos datos, en el orden de los cientos de muestras (para entrenar un modelo clasificación de imágenes de deep learning se suelen utilizar millones de imágenes), el modelo que resulta del entrenamiento puede haber sido sesgado por la división de datos que se utilizó para entrenarlo, ya que la división de datos utilizada puede no ser representativa del conjunto de datos. Por eso, se puede sacar mucho provecho de algunas formas más complejas de división de datos para poder evaluar el modelo.

Una modalidad de evaluación de modelos muy utilizada es *K-Fold Cross Validation* y *K-Fold Cross Validation iterada con mezcla de los datos*. La evaluación mediante K-Fold Cross Validation es más ampliamente utilizada en modelos pequeños de machine learning como regresión logísitica, SVM, u otros modelos que tengan hiperparámetros que ajustar para que el modelo pueda funcionar correctamente. Con este enfoque se dividen los datos en K particiones (los más utilizados son $K = 5$ o $K = 10$), y en cada iteración se entrena el modelo con $K - 1$ particiones, y se evalúa con la partición restante. El "score" final es el promedio de los K resultados obtenidos. Lo que se hace para ajustar los hiperparámetros de un modelo es definir un rango de valores para los mismos y realizar K-Fold Cross Validation para cada una de las posibles combinaciones. Finalmente, se elige el conjunto de hiperparámetros que haya obtenido mayor score de Cross Validation. Sin embargo, en modelos más grandes que tengan muchos parámetros para entrenar, este enfoque es poco práctico y se suele utilzar solo en modelos y redes neuronales pequeñas. Debido a que la búsqueda de hiperparámetros por lo general ya fue hecha para modelos tomados como inspiración, y lo que se busca es adaptarlos a otro problema o mejorar los resultados obtenidos, el Cross Validation en redes neuronales para la búsqueda de hiperparámetros no suele ser adoptado. A pesar de esto, se

puede utilizar K-Fold Cross Validation para evaluar el impacto de la división de los datos en el desempeño del modelo, lo cual es muy útil en los casos donde hay pocos datos. En cuando a la otra variante, consiste en realizar la partición de los datos en K una vez por iteración mezclando los datos cada vez, y realizar la K-Fold Cross Validation de manera iterativa. Se suele tomar 50 iteraciones como un número estándar. Este enfoque, al ser demasiado costoso computacionalmente, se usa cuando los datos son escasos y se busca obtener una validación de resultados lo más general posible.

Métricas de evaluación

Es necesario definir qué métricas se utilizarán para poder evaluar si el modelo seleccionado tiene un buen desempeño en el problema que se está analizando, además de cómo se obtienen dichas métricas. Algunas de las métricas más utilizadas en problemas de clasificación son las siguientes:

Matriz de confusión: La matriz de confusión es una tabla en la que se pueden ver los resultados de la clasificación realizada por el modelo en comparación con los valores verdaderos. La matriz de confusión se define de la siguiente manera:

		N'	**P'**
N		Verdadero Negativo	Falso Positivo
P		Falso Negativo	Verdadero Positivo

Tabla 1.1: Matriz de confusión. N (Negativo) y P(Positivo) corresponden a la clasificación real de los valores como, mientras que N' y P' corresponden a la clasificación otorgada por el método.

Siendo *Verdadero Negativo* los valores que son clasificados como negativos y son realmente negativos, *Falso Negativo* son los valores clasificados como negativos pero que en realidad son positivos, *Falso Positivo* son valores clasificados como positivos que en realidad son negativos, y *Verdadero Positivo* que son valores clasificados como positivos que realmente son positivos. A partir de los valores representados en la matriz se pueden derivar varias métricas importantes para el análisis del desempeño del modelo.

Exactitud: También llamada *accuracy*, indica en rasgos generales que tan correcto es el método. Se define como:

$$\frac{VP + VN}{VP + VN + FP + FN}$$

Es una métrica que puede indicar si un modelo está siendo entrenado correctamente y da una medida de cómo se desempeña en general. Sin embargo, no otorga información detallada acerca de su aplicación al problema, y por lo tanto no es recomendable utilizarla

como principal métrica de desempeño ya que si hay desbalance de clases sus resultados no son confiables.

Sensibilidad: La sensibilidad, también llamada *Recall*, indica la capacidad del método para clasificar como positivos a los sujetos que realmente portan la enfermedad. Es importante para determinar la capacidad diagnóstica del método ya que si no lograra detectar la mayoría de los sujetos que portan la enfermedad, el método no tendría aplicación. Se define como:

$$\frac{VP}{VP + FN}$$

La sensibilidad es una métrica que debe monitorearse y evaluarse cuando el costo de los falsos negativos es alto, es decir cuando la clasificación de sujetos enfermos como si fueran sanos tiene un impacto muy negativo para los pacientes. Al buscar un valor alto de esta métrica se está minimizando la cantidad de falsos negativos.

Especificidad: La especificidad del método indica la capacidad del método para clasificar como negativo a los sujetos que realmente son sanos. La importancia de esta métrica yace en que si se detecta como sano un sujeto enfermo, el sujeto podría estar en peligro dependiendo de la enfermedad que se tratara. Por eso es importante tener una especificidad lo más alta posible. Se define como:

$$\frac{VN}{FP + VN}$$

Tanto la sensibilidad como la especificidad son métricas intrínsecas al modelo, lo cual significa que son valores teóricos que sirven para dar cuenta de la validez del modelo utilizado.

Valor predictivo positivo: También denominado *Precisión*, esta métrica indica cuántas veces el modelo está en lo correcto cuando su resultado es positivo. Se define como:

$$\frac{VP}{VP + FP}$$

El VPP es de ayuda cuando el costo de los falsos positivos es alto, como por ejemplo cuando la detección incorrecta de una enfermedad pudiera traer problemas al paciente. En el caso del Alzheimer, la detección incorrecta genera problemas para el paciente y su familia. De esta forma, al buscar la precisión lo más alta posible se está minimizando la cantidad de falsos positivos.

Valor predictivo negativo: Esta métrica indica la probabilidad de que el resultado predicho como negativo sea realmente negativo. Se define como:

$$\frac{VN}{VN + FN}$$

Los valores predictivos positivo y negativo indican el comportamiento del modelo de clasificación en una población con cierta proporción de clases a ser evaluada, y dan una medida de la relevancia de la sensibilidad y especificidad en una determinada población. Es

decir, que en el caso de una prueba diagnóstica, los valores predictivos reflejan la utilidad diagnóstica del modelo.

F1-Score: Es una métrica general de la exactitud del modelo que combina precisión y recall. Un F1 cercano a 1 significa que se obtienen pocos falsos positivos y pocos falsos negativos, de manera tal que se están identificando correctamente casos reales y no se tienen falsas alarmas; un valor cercano a 0 indica que el modelo no es confiable. Se define como:

$$F1 - Score = 2 \times \frac{precision \times recall}{precision + recall} = \frac{2 \times TP}{FN + 2 \times TP + FP}$$

Curva ROC y AUC score: La curva ROC (Receiver Operating Characteristic) nos dice qué tan bien el modelo puede distinguir entre dos situaciones, por ejemplo si un paciente tiene o no una enfermedad. La curva ROC surge al hacer un gráfico de TFP vs. TVP, siendo TVP la tasa de verdaderos positivos (o la sensibilidad) y TFP la tasa de falsos positivos la cual se define como TFP = 1 - especificidad.

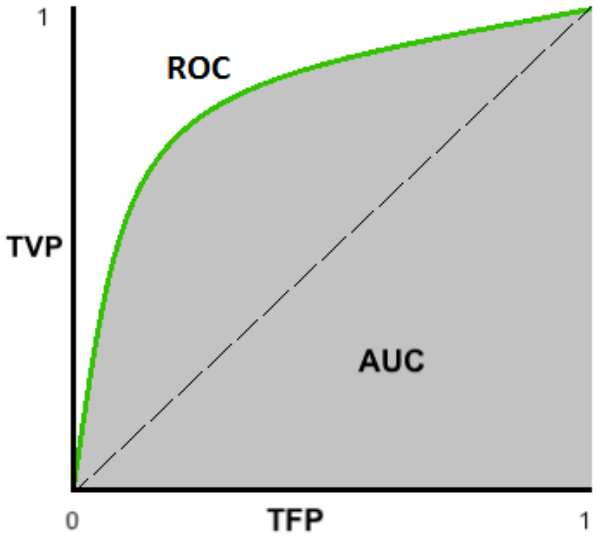


Figura 1.4: Ejemplo de curva ROC y su comparación con la línea punteada que indica un clasificador sin capacidad de distinción entre clases. Visualización de la AUC (área gris).

La métrica asociada a la curva ROC es el área debajo de la curva (AUC), la cual mide la relación entre TFP y TVP de manera cuantitativa. Un AUC = 1 indica que el modelo identifica de manera perfecta todos los pacientes que tienen la enfermedad y a los que no, mientras que un AUC = 0.5 indica que el modelo no funciona mejor que la elección al azar. Por lo tanto, valores de AUC cercanos a 1 (por ejemplo mayores a 0.8) indican que el modelo tiene buena capacidad para distinguir pacientes enfermos de pacientes sanos, mientras que valores cercanos a 0.5 (menores a 0.7) indica que el modelo no tiene buenas capacidades para realizar dicha tarea. Valores entre 0.7 y 0.8 son intermedios, y habría que analizar con mayor detenimiento otras métricas para determinar qué tan bien se desempeña el modelo.

1.3. Problemática a tratar

En este trabajo se propone un método de clasificación de pacientes que sea capaz de distinguir aquellos que tienen la enfermedad (AD) de aquellos que no la tienen (CN - Controles Normales), sin incluir los pacientes MCI en el análsis.

La principal hipótesis tomada en cuenta es que la enfermedad de Alzheimer sigue el modelo patogénico del Sistema A/T/N y, por lo tanto, se eligieron los datos a analizar acorde a lo propuesto en dicho modelo. Se utilizaron imágenes de resonancia magnética estructural y mediciones de las concentraciones en el líquido cefalorraquídeo de las proteínas Aβ, Tau y p-Tau. Además, se incluyeron en el análisis los resultados del genotipado de APOE y de un test cognitivo como factores de riesgo para complementar los datos y generar una visión más integral del paciente introduciendo datos genéticos y cognitivos.

El método de análisis elegido para obtener la clasificación de los pacientes a partir de los datos propuestos es una red neuronal compuesta de dos "subredes" que analizan, por un lado, las resonancias magnéticas, y por el otro los valores de concentración de proteína amioide β, Tau, p-Tau en el líquido cefalorraquídeo, el genotipado del gen APOE y los resultados de un test cognitivo. Los resultados obtenidos por dichas subredes se unifican y procesan para obtener una clasificación final.

1.4. Estructura del trabajo

En los siguientes capítulos se explicará los aspectos teóricos de cada uno de los modelos utilizados en el trabajo, el proceso de desarrollo empleado para cada uno, los datos utilizados y los resultados obtenidos.

En el **capítulo 2** se introduce la red neuronal que fue utilizada para el análisis de las imágenes de resonancia magnéticas. En este capítulo se explica con más detalle el procedimiento de adquisición de imágenes de resonancia magnética, su relación y uso en la enfermedad de Alzheimer, los métodos utilizados para generar un conjunto de datos analizables por la red neuronal, y finalmente se introduce la red neuronal utilizada explicando cada componente y mostrando los resultados obtenidos.

En el **capítulo 3** se introduce la red que analiza los biomarcadores. Se sigue una estructura de análisis similar a la del capítulo anterior: se da una explicación de la adquisición de los biomarcadores, el gen APOE y el test cognitivo utilizado, se explica su importancia en el diagnóstico de la enfermedad de Alzheimer, se exploran las relaciones entre las variables descritas por los biomarcadores, se realiza la selección de modelo y se muestran los resultados obtenidos.

En el **capítulo 4** se explica cómo se unifican los modelos desarrollados en los capítulos anteriores para obtener un resultado final; se introduce el modelo final y se muestran los resultados.

En el **capítulo 5** se exponen las conclusiones extraídas a partir del análisis de los resultados obtenidos en el trabajo.

En el **capítulo 6** se introducen posibles mejoras, cambios y trabajo a realizar en el futuro para avanzar aún más hacia un método de diagnóstico de la enfermedad de Alzheimer que pueda ser utilizado en el ámbito clínico.

Capítulo 2

Resonancia Magnética Estructural

2.1. Imágenes de resonancia magnética

La resonancia magnética (MRI - Magnetic Resonance Imaging) es una técnica de adquisición de imágenes no ionizante que utiliza pulsos de radiofrencuencia (con frecuencias en el rango de 200 MHz a 2 GHz) y grandes campos magnéticos (1-2 T) para obtener una imagen a partir de la reacción de los tejidos estimulados por estas señales.

Las MRI proporcionan detalles anatómicos y fisiológicos a partir de descripciones volumétricas con excelente visualización de tejidos blandos y gran resolución espacial ($\sim$1mm). El tiempo de adquisición puede ser de varios minutos, y por lo tanto, el movimiento del paciente puede resultar en artefactos en la imagen final.

2.1.1. Resonancia magnética nuclear

La obtención de las imágenes de resonancia magnética está basada en el principio físico de la *resonancia magnética nuclear*. Los núcleos de los átomos están compuestos por protones y neutrones, partículas que también se denominan nucleones. Un núcleo con nucleones no emparejados se comporta como un pequeño imán, con su momento magnético asociado. El núcleo del átomo de hidrógeno, que contiene sólo un protón, es de particular importancia para las imágenes de resonancia magnética por su abundancia en el tejido biológico.

La dirección del momento magnético suele ser aleatoria, pero al entrar en contacto con un campo magnético externo, el momento magnético de los núcleos se alinea con el campo y realiza un movimiento de precesión alrededor del mismo con una frecuencia proporcional a la intensidad del campo externo $\boldsymbol{B_0}$, conocida como la frecuencia de Larmor. Los momentos magnéticos se pueden alinear de forma paralela al campo, en el estado de menor energía, o de forma antiparalela, el cual es un estado de mayor energía. Los protones se encuentran continuamente pasando de un estado al otro, pero en cualquier momento habrá siempre una pequeña mayoría de protones alineados paralelos al campo, para disminuir la energía total. Mientras más grande sea el campo externo, mayor será la diferencia en los niveles de energía entre los estados y por lo tanto, serán más los protones alineados paralelos al campo. Los resonadores magnéticos que usen campos de mayor intensidad producen imágenes con una relación señal-ruido más alta, porque es el número de protones el que determina la señal de la resonancia magnética.

Para poder detectar la diferencia de energía entre los estados de los protones se aplica un pulso de radiación electromagnética. De esta forma, algunos protones alineados paralelos al campo externo pasan a estar alineados antiparalelos al campo, con un estado de energía mayor. La energía del pulso empleado es igual a la diferencia de energía entre los estados y la frecuencia del pulso es la frecuencia de Larmor, ninguna otra frecuencia puede generar la transición del estado paralelo al antiparalelo, por lo tanto, se trata de una frecuencia de *resonancia*. La frecuencia resonante para el núcleo de hidrógeno en campos de 1-2 T corresponde al rango de las radiofrecuencias (RF). Estas transiciones reducen el número de protones alineados al campo externo, representado como el eje z en la figura 2.1, y resulta en la magnetización neta, $\boldsymbol{M_0}$, que tiende a descender en espiral al plano xy o debajo del mismo, dependiendo de la intensidad y duración del pulso. Esta es la vista desde el sistema de referencia fijo en el laboratorio; desde el punto de vista del vector giratorio de magnetización, el sistema de referencia giratorio, la visualización es más simple con M_0 inclinándose suavemente con un ángulo, α, que es función de la intensidad y duración del pulso de RF. El campo magnético del pulso de RF es denominado $\boldsymbol{B_1}$; oscila a la frecuencia de Larmor y forma un ángulo recto con B_0. La interacción de B_1 con M_0 causa su inclinación hacia el plano xy.

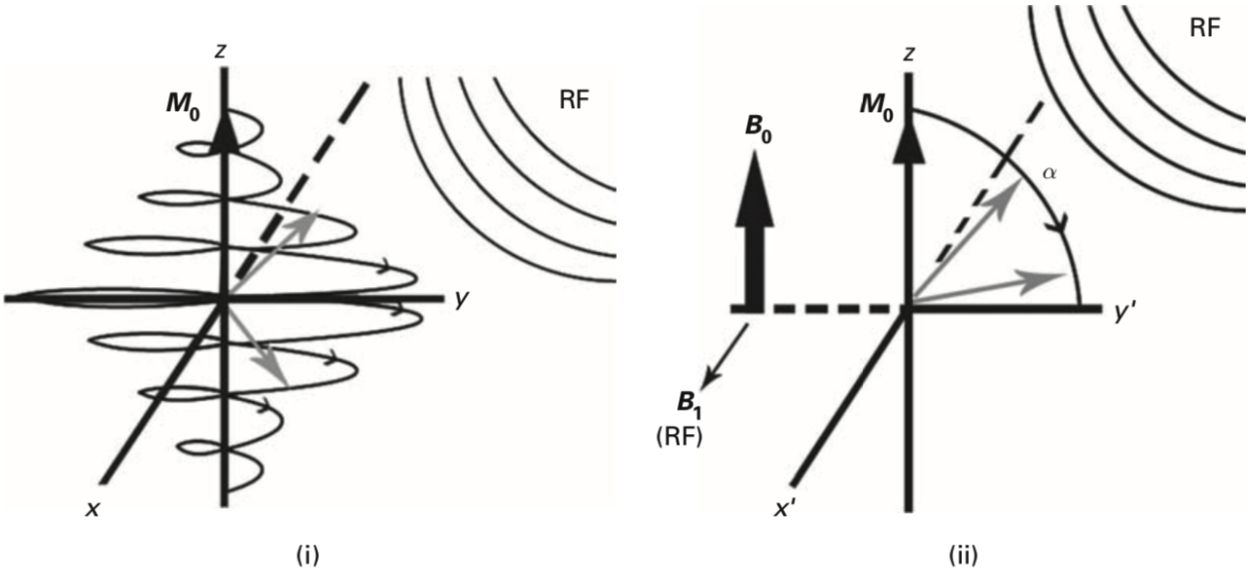


Figura 2.1: Efecto del pulso de RF en la magnetización neta visto desde (i) sistema de referencia del laboratorio, y (ii) sistema de referencia giratorio [19]

Cuando la longitud de onda del pulso es tal que produce una inclinación de 90°, se lo denomina pulso de 90°, y produce la máxima magnetización en el plano xy. En cambio, si el pulso fuera de 180°, la magnetización se inclina hacia la dirección z y no se produce magnetización transversal.

Una vez que el pulso de 90° se apaga, M_0 induce una corriente en la bobina que produjo el pulso de RF, y se relaja lentamente a su orientación original a lo largo del eje z. Esta señal inducida se conoce como la *caída libre de la inducción* (Free Induction Decay - FID). El proceso de relajación está compuesto por dos partes, cada una caracterizada por sus

tiempos característicos. Una parte de los procesos de relajación es la componente *longitudinal*, caracterizada por el tiempo característico T_1, y se la denomina relajación de **espín-red**. Esta componente de la relajación causa que el vector de magnetización neta vuelva a su posición original en z. Esto se produce como resultado de las interacciones entre los espines nucleares y las moléculas circundantes (la red). Este proceso puede ser descrito por:

$$M_z = M_0 \left(1 - \mathrm{e}^{-t/T_1}\right) \tag{2.1}$$

con un tiempo característico T_1, que depende del tipo de tejido.

Al mismo tiempo, la otra componente del proceso de relajación es la transversal, caracterizada por el tiempo característico T_2, y denominada relajación **espín-espín**. En esta componente, la magnetización transversal decae a cero porque los espines individuales giran a diferentes frecuencias y quedan desfasados. El proceso está descrito por:

$$M_{xy} = M_0 \mathrm{e}^{-t/T_2} \tag{2.2}$$

Estos dos procesos ocurren juntos. Sin embargo, la componente transversal es más rápida que el crecimiento de la componente longitudinal de la magnetización; por lo tanto, se cumple siempre que T_2 es menor a T_1 para cada tipo de tejido en particular. En la práctica, el desfasaje también sucede debido a variaciones locales en el campo magnético y la falta de uniformidad dentro del tejido, de manera que T_2 debe ser reemplazado por ${T^*}_2$, donde $\frac{1}{{T^*}_2} = \frac{1}{T_2} + \frac{1}{{T'}_2}$, y ${T'}_2$ caracteriza las variaciones locales y puede llegar a ser 100 veces más corto que T_2. Se suele utilizar una secuencia de pulsos denominada *spin-echo* para obtener la verdadera caída caracterizada por T_2, en lugar la caída más rápida dada por ${T^*}_2$.

La señal producida por la caída de la magnetización transversal es la señal FID, nombrada anteriormente. Dado que no todos los espines se encuentran en condiciones ambientales químicas y magnéticas idénticas, no todos precesionan a exactamente la misma frecuencia y la señal FID detectada es una superposición de las señales individuales. La Transformada de Fourier de la señal de FID nos da el espectro de resonancia magnética nuclear, el cual proporciona información de la señal en el espacio de las frecuencias:

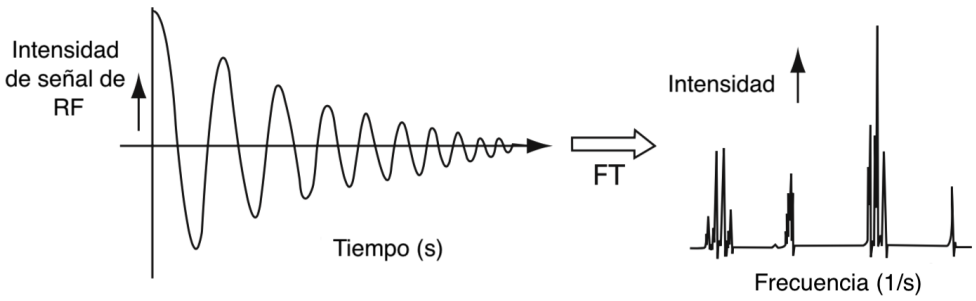


Figura 2.2: Señal de FID y su Transformada de Fourier [19]

2.1.2. Obtención de la imagen

El espectro de resonancia magnética nuclear sólo produce un pico por tipo de tejido, lo cual significa que si el mismo tejido estuviera en dos posiciones diferentes no se podría

distinguir uno del otro a partir del espectro:

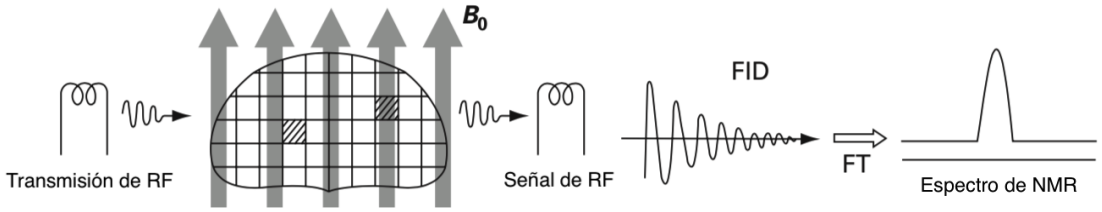


Figura 2.3: Espectro de resonancia magnética nuclear de dos protones en distintas posiciones [19]

Si se agregara un *gradiente lineal de campo magnético*, $\boldsymbol{B_x}$, entonces el campo magnético total incrementaría a lo largo del paciente y por lo tanto, como la frecuencia de Larmor es proporcional al campo magnético aplicado, la señal de FID tendría no sólo variabilidad tisular sino que se le agregaría variabilidad espacial. Para el caso de un tipo de tejido en dos posiciones diferentes, el espectro de NMR ahora tendría dos picos y se los podría diferenciar o ubicar espacialmente:

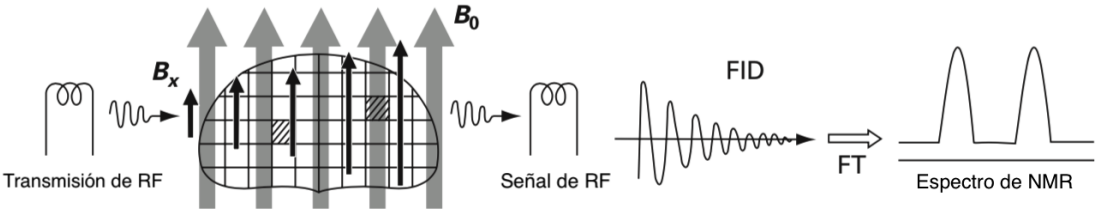


Figura 2.4: Espectro de resonancia magnética nuclear de dos protones en distintas posiciones [19]

De esta forma, el espectro resultante contiene información espacial y puede ser considerado como una proyección. Como sabemos, B_0 se encuentra siempre prendido durante la adquisición de la imagen; por otro lado, el gradiente de campo magnético se aplica transitoriamente durante la recolección de información. Si se obtuvieran las múltiples proyecciones mencionadas alrededor del paciente, entonces se podría reconstruir una imagen axial.

La selección del corte se logra usando un pulso de radiofrecuencia con un ancho de banda seleccionado, y un gradiente de campo magnético. El gradiente de campo magnético utilizado (x, y o z) nos permite seleccionar la orientación (sagital, coronal o transversal) de la imagen. El grosor del corte puede variarse usando diferentes intensidades del gradiente o anchos de banda del pulso de radiofrecuencia. Cambiando la frecuencia central del pulso se puede mover el corte a distintas posiciones dentro del paciente.

Luego de seleccionar un corte, se deben codificar las dos dimensiones restantes que definen el plano para producir la imagen bidimensional. Una de las dimensiones se codifica a partir de cambios en la frecuencia de resonancia durante la adquisición mediante la aplicación de otro gradiente de campo magnético en la dirección a codificar; esto se denomina *codificación de frecuencia*. Para la dimensión restante se utiliza *codificación de fase*, la cual se logra aplicando varios gradientes de campo magnético para generar cambios de fase en la oscilación de los protones.

Se requieren tres bobinas diferentes para codificar las tres dimensiones espaciales unívocamente. Al rotar los gradientes de campo magnético alrededor del paciente se pueden obtener múltiples proyecciones. La Transformada de Fourier de las señales de eco recibidas producen los espectros que son efectivamente la proyección de los objetos en ciertas direcciones. Estas proyecciones unidimensionales pueden ser unidas para formar la transformada bidimensional del objeto, que a la vez se le puede aplicar la Transformada Inversa de Fourier para obtener la imagen axial. Esta es la técnica de *Reconstrucción Directa de Fourier* que se usa casi universalmente en la reconstrucción de imágenes de resonancia magnética.

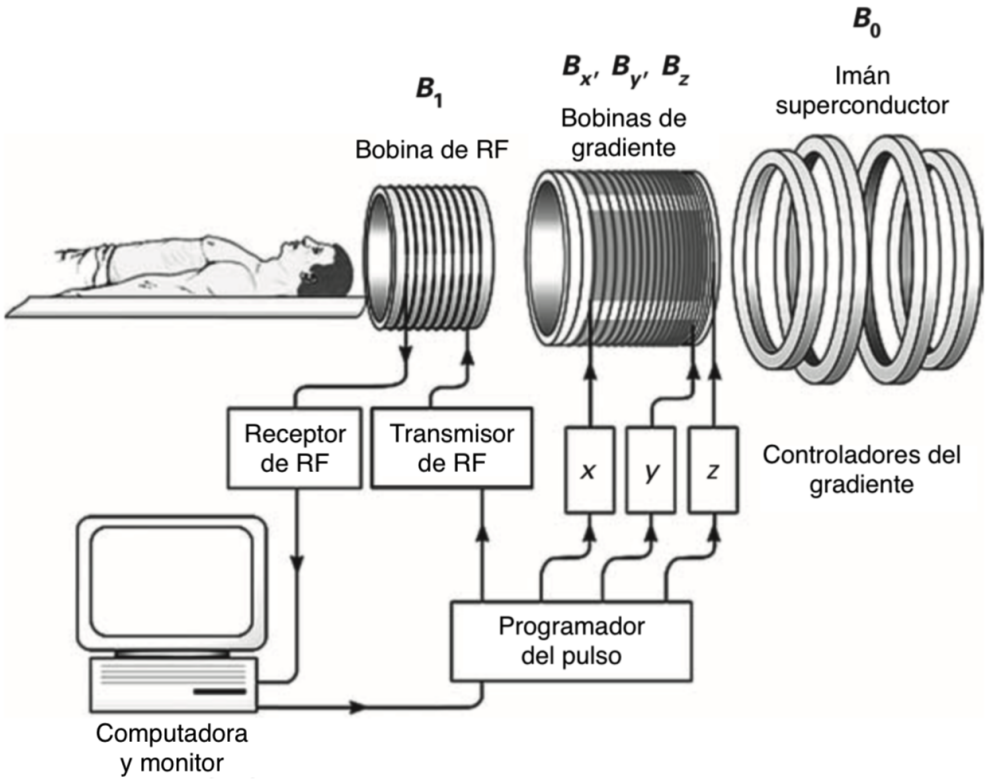


Figura 2.5: Componentes básicas de un resonador magnético [19]

En cuanto a las componentes básicas del resonador, se pueden ver en la figura 2.5. El imán principal polariza los protones del paciente. Las bobinas de gradiente, las cuales se encuentran permanentemente fijas dentro del imán principal, producen variaciones lineales en el campo magnético para que las frecuencias de resonancia de los protones dentro del paciente sean espacialmente dependientes. La bobina de RF produce el campo magnético oscilante necesario para crear la coherencia de fase entre los protones, y recibe la señal de FID por inducción magnética. Dicha bobina se encuentra ubicada rodeando la parte del cuerpo del paciente que se debe visualizar con la imagen, con una geometría optimizada para esa parte específica.

2.1.3. Secuencias del pulso

La secuencia de pulsos spin-echo es el pilar fundamental de la obtención de imágenes por resonancia magnética en el ámbito clínico por su simplicidad y flexibilidad en el sentido de que permite obtener imágenes cuyo contraste esté dominado por T_1 o por T_2.

En esta secuencia de pulsos se emplea en primer lugar un pulso de 90° que mueve la magnetización neta al plano transversal. Cuando el pulso de 90° se apaga, las fases de los espines afectados comienzan a cambiar, lo cual resulta en relajación $T^*{}_2$, como ya fue explicado. Sin embargo, si se aplica un pulso de 180° antes de que la componente transversal desaparezca completamente, los espines se dan vuelta y comienzan a desfasarse nuevamente, como se puede ver en la figura 2.6

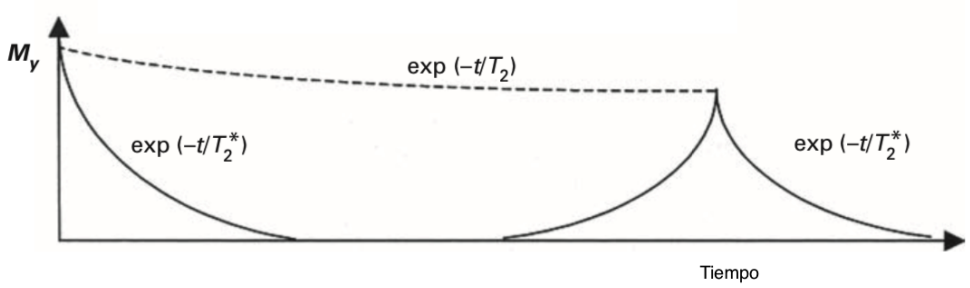


Figura 2.6: Decaimiento después de la finalización del pulso de 90° y subsiguiente aplicación de un pulso de 180° [19]

Se puede ver que a partir de esta combinación de pulsos surge un nuevo eco, denominado *eco de espín* o spin-echo. La aplicación de un tren sucesivo de pulsos de 180° causa este efecto de "re-fasaje"de los espines. Como el valor máximo del spin-echo que le sigue a cada aplicación del pulso de 180° es un punto en la exponencial decreciente que describe el proceso de relajación espín-espín caracterizado por $T^*{}_2$, entonces un pulso de 90° seguido por un tren de pulsos de 180° cuidadosamente cronometrado da una envolvente del proceso de relajación caracterizado por T_2 (figura 2.7)

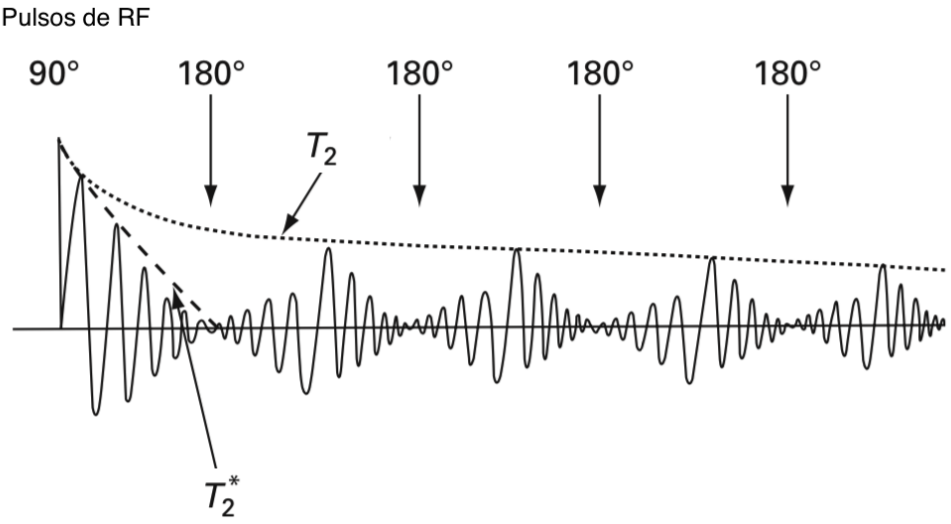


Figura 2.7: Secuencia de pulsos spin-echo, compuesta por un pulso de 90° seguida por un tren de pulsos de 180° cuidadosamente cronometrado produce una serie de ecos de espín de amplitudes decrecientes. Aunque cada eco decrece según $T^*{}_2$, la envolvente decrece según T_2. [19]

Se define el tiempo desde el centro del pulso de 90° al centro del pulso de eco de espín, como *tiempo de eco*, TE. TE/2 es el tiempo desde el centro del pulso de 90° al centro del pulso de re-fasaje de 180°. La secuencia de pulsos se aplica varias veces, dependiendo del tamaño de la imagen y de los requerimientos de la relación señal-ruido. El tiempo de repetición, TR, se define como el tiempo desde el centro del pulso de 90° al centro del siguiente pulso de 90°.

Un problema asociado a la secuencia de pulsos de spin-echo es que los tiempos de adquisición de las imágenes suelen ser relativamente largos. Para superar esta dificultad, se introdujo la secuencia de pulsos de eco de gradiente o *gradient-echo*. En esta secuencia no se aplican pulsos de 180° para que los tiempos de adquisición de la imagen sean más rápidos. En consecuencia, estas imágenes se encontrarán influenciadas por $T^*{}_2$ en lugar de T_2, y por lo tanto son susceptibles a la aparición de artefactos. El uso de secuencias de gradient-echo se da para imágenes con TR cortos.

Existen muchas secuencias de pulsos diferentes, en su mayoría variantes de spin-echo o de gradient-echo, cada una tiene distintas aplicaciones y características que son reflejadas en la imagen final.

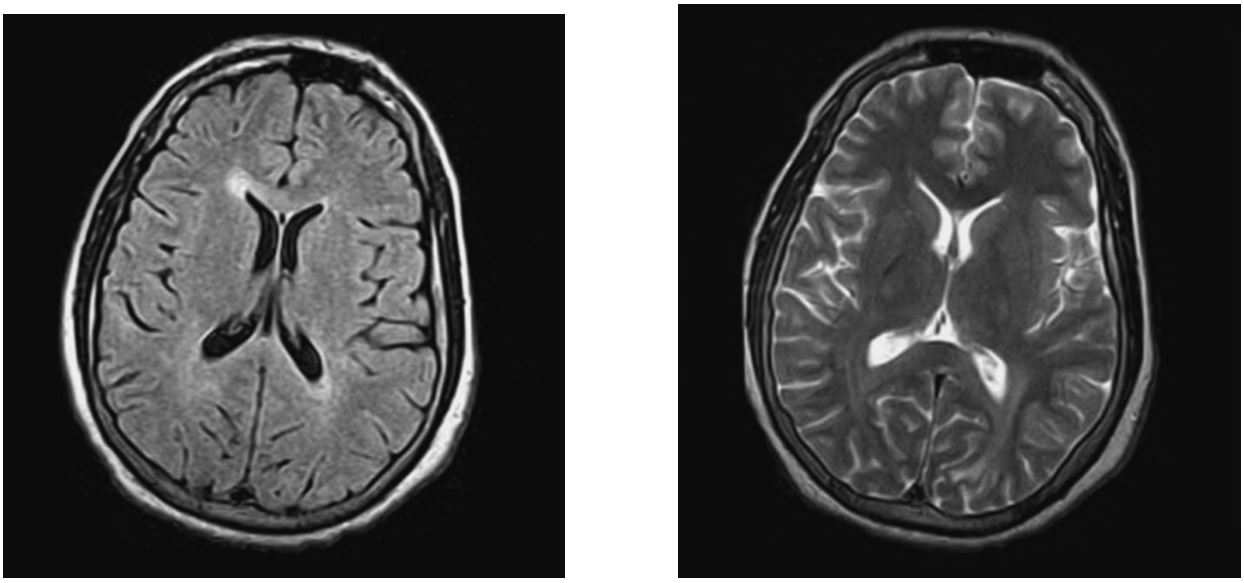
2.1.4. Imágenes ponderadas en T_1 y T_2

La imagen de resonancia magnética es un mapeo de las distintas intensidad de la señal de NMR que se origina en los distintos vóxeles, y depende de la densidad de protones y los valores de T_1 y T_2, los cuales son consecuencia de las inmediaciones de los protones circundantes. Los distintos protocolos de adquisición de imágenes de resonancia magnética usan secuencias de pulsos de diferentes longitudes y separaciones, lo cual puede ser utilizado para mejorar el contraste de la imagen. Las imágenes producidas de forma tal que se reflejen las diferencias en los tejidos con el decaimiento de la magnetización caracterizado por T_1 se denominan "ponderadas en T_1" o T_1-*weighted*, otras imágenes pueden ser "ponderadas en

T_2" o T_2-*weighted*, "ponderadas por la densidad de protones" o *Proton density weighted*, etc.

Para lograr la ponderación en T_1 se utiliza una secuencia de pulsos spin-echo que reduzca las contribuciones de T_2 y de la densidad de protones; por eso, TR y TE deben ser cortos ($\leq 500 - 600$ ms y ≤ 20 ms, respectivamente). Utilizar un TR corto asegura que haya diferencias máximas en la recuperación de la magnetización longitudinal entre los tejidos con diferentes tiempos de relajación T_1; y la utilización de un TE corto asegura que la magnetización transversal no se desfase de forma apreciable y no se pierdan estas diferencias. En una imagen ponderada en T_1 de la cabeza, el tejido adiposo se ve brillante y el líquido cefalorraquídeo se ve oscuro (figura 2.8a).

En las imágenes ponderadas en T_2, se utilizan TR largos (2000-4000 ms) para reducir los efectos de T_1, y se utilizan TE largos (80-150 ms) para producir diferencias de contraste dependientes de T_2. En este tipo de imágenes, el líquido cefalorraquídeo y las áreas con un contenido de agua anormalmente alto (como aquellos afectados por un tumor, infección o un derrame) son brillantes (figura 2.8b). Un problema asociado con las imágenes ponderadas en T_2 es que si el TE se extiende excesivamente el contraste de la imagen mejora, pero la relación señal-ruido decrece.



(a) Imagen ponderada en T_1 de la cabeza.

(b) Imagen ponderada en T_2 de la cabeza.

Figura 2.8: Comparación de imágenes de resonancia magnética de la cabeza ponderadas en T_1 y T_2.[19]

2.2. MRI y la enfermedad de Alzheimer

Debido a que el diagnóstico certero de la enfermedad de Alzheimer se realiza a partir de una autopsia post-mortem, a lo largo de los años se han buscado diversos biomarcadores que puedan ser analizados para realizar una detección y seguimiento de las enfermedad en los pacientes in-vivo. Como ya se explicó anteriormente, el Sistema A/T/N es un modelo patogénico ampliamente aceptado en la actualidad, el cual permite definir un seguimiento

de la enfermedad de Alzheimer a partir de biomarcadores que caracterizan los tres pilares fundamentales de la patología. La neurorodegeneración es uno de estos pilares y puede ser analizado a partir de imágenes médicas para el diagnóstico de la enfermedad [20]. El rol de las imágenes es esencial para que los médicos puedan ver la progresión del estado neurodegenerativo del cerebro del paciente a lo largo de los años.

Existen diversas modalidades de adquisición de imágenes, como CT, MRI, fMRI, PET, por sólo nombrar algunas de las más conocidas y utilizadas. Cada modalidad de adquisición tiene sus fortalezas y sus debilidades, y aportan información diferente. Por ejemplo, las imágenes de CT se utilizan para excluir posibles motivos de demencia que sean quirúrgicamente removibles; las fMRI aportan información acerca de la cognición del paciente, y las imágenes de PET para asesorar el metabolismo cerebral y la presencia de placas amiloides. La resonancia magnética estructural permite visualizar la atrofia cerebral progresiva característica de las patologías neurodegenerativas como la enfermedad de Alzheimer. Las MRI ponderadas en T_1 permiten la visualización de la atrofia de las estructuras del cerebro (figura 2.9, la cual caracteriza al Alzheimer, mientras que, por otro lado, las MRI ponderadas en T_2 muestran hiperintensidades en la sustancia blanca, que son características del daño vascular, al cual no se lo asocia con la enfermedad pero que puede servir para determinar si la causa de la demencia que el paciente presenta es distinta la demencia causada por la enfermedad de Alzheimer.

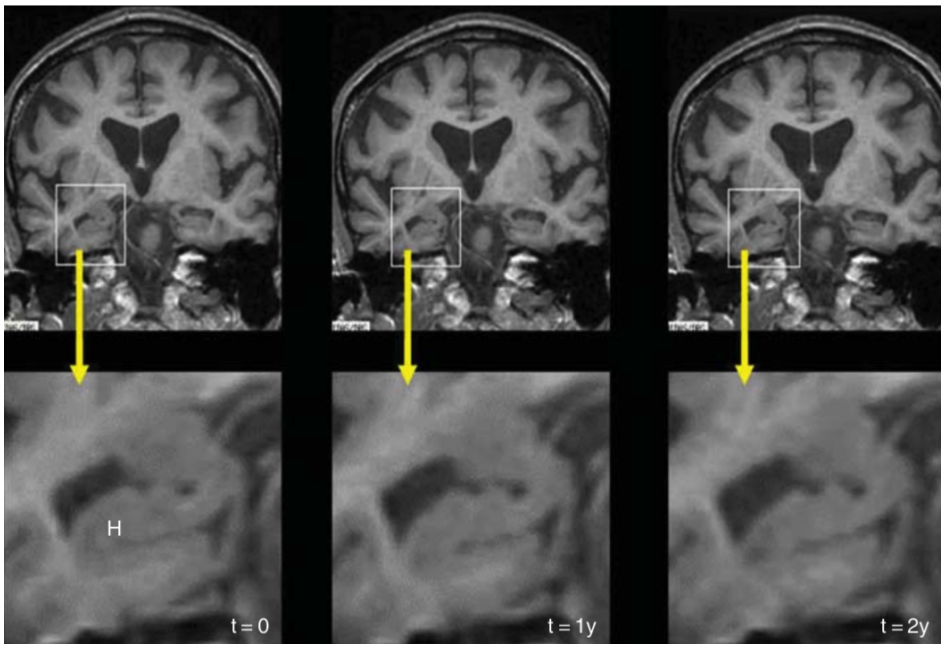


Figura 2.9: Progresión de la atrofia del hipocampo, característica de la enfermedad de Alzheimer. [20]

Por lo tanto, dado que en este trabajo se propone un modelo que detecte la enfermedad de Alzheimer a partir de los biomarcadores característicos de la misma, se utilizarán resonancias magnéticas estructurales ponderadas en T_1 que representan el estado neurodegenerativo de las estructuras del cerebro del paciente.

2.2.1. Secuencia MPRAGE

Las imágenes de resonancia magnética más utilizadas para analizar las estructuras anatómicas del cerebro solían ser obtenidas con secuencias spin-echo (SE) y ponderadas en T_1. Sin embargo, se probó que la utilización de la secuencia **MPRAGE** (Magnetization-Preprared Rapid Gradient Echo) [21] provee imágenes de mejor calidad y definición, con mejor relación de contraste entre materia gris y sustancia blanca, y no requiere grandes tiempos de adquisición ([21], [22]). La secuencia consiste en la preparación magnética al aplicar un pulso de inversión de 180°, lo cual produce una ponderación fuerte en T_1, para luego utilizar una adquisición de gradient-echo rápida. Actualmente, la secuencia MPRAGE es una de las más populares para la adquisición de imágenes de resonancia magnética del cerebro [22].

Por lo tanto, se decidió utilizar imágenes con esta secuencia de adquisición ya que son imágenes que tienen mejor calidad y definición de las estructuras y presenta un alto contraste entre las estructuras del cerebro, lo cual es un beneficio para el modelo propuesto en este trabajo.

2.3. Base de datos

2.3.1. ADNI

Para este trabajo se utilizaron los datos disponibles en la base de datos de *Alzheimer's Disease Neuroimaging Initiative (ADNI)*. ADNI es un proyecto de recolección de imágenes y datos médicos para el estudio de la demencia de Alzheimer dentro del Image and Data Archive (IDA) del Laboratory of Neuroimaging (LONI) de la Universidad de California del Sur (University of Southern California - USC). Se trata de un estudio observacional, longitudinal y de sitios múltiples en el que se incluyen individuos ancianos con condiciones congnitivas normales (CN), deterioro cognitivo leve (MCI) o demencia de Alzheimer (AD), cuyo propósito es el de evaluar el uso de la información brindada por imágenes de resonancia magnética, tomografías por emisión de positrones, orina, sangre, biomarcadores de líquido cefalorraquídeo, como también evaluaciones clínicas y psicométricas, para medir el progreso de la enfermedad en los tres grupos de sujetos ancianos nombrados anteriormente [23].

Para acceder a los datos disponibles en ADNI es necesario hacer un pedido formal y que, luego de la evaluación corresponiente del pedido, se otorgue acceso. Una vez que el acceso sea concedido se puede disponer de la totalidad de la información disponible sin restricciones.

2.3.2. Distribución de sujetos

Los pacientes que se presentan en la base de datos pueden ordenarse según su edad, género, y la condición diagnosticada. A continuación podemos ver cómo se distribuyen los sujetos bajo esos distintos criterios:

Franja etaria	**Numero de sujetos**
50-59	97
60-69	718
70-79	1226
80-89	457
>89	26
Total	**2524**

Tabla 2.1: Distribución de edades de los pacientes.

Franja etaria	**Sin diagnóstico**	**CN**	**MCI**	**EMCI**	**LMCI**	**AD**	**SMC**
50-59	1	19	26	18	10	22	1
60-69	16	243	142	133	54	82	48
70-79	21	407	307	137	95	206	53
80-89	8	110	138	51	24	115	11
>89	2	5	3	1	2	11	2
Total	**48**	**784**	**616**	**30**	**185**	**436**	**115**

Tabla 2.2: Distribución de diagnósticos por edad de los pacientes

Franja etaria	**Femenino**	**Masculino**	**Desconocido**
50-59	62	35	0
60-69	395	321	2
70-79	546	679	1
80-89	179	277	1
>89	11	15	0
Total	**1193**	**1327**	**4**

Tabla 2.3: Distribución de género por edad de los pacientes

Podemos ver de estas tablas que la población de sujetos es más densa en el intervalo de edades de 70-79 años, y que la población esta equitativamente distribuida en términos de género, habiendo 1193 pacientes femeninos y 1327 pacientes masculinos.

El criterio de inclusión de sujetos que se adoptó en este trabajo fue el de considerar sujetos según su ultima visita clínica. De esta forma, se incluyeron solamente los sujetos que hayan sido declarados como cognitivamente normales (CN) o que hayan sido diagnosticados con demencia por la enfermedad de Alzheimer (AD) en su última visita. Los sujetos que hayan sido diagnosticados con MCI no fueron incluidos en este análisis. Así nos queda la siguiente distribución por edades de sujetos:

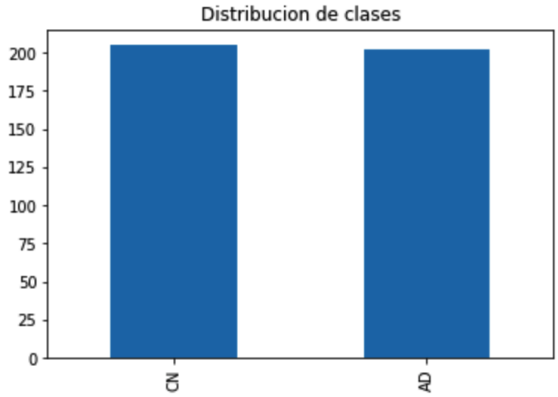
Franja etaria	**CN**	**AD**	**Total**
50-59	19	22	41
60-69	243	82	325
70-79	407	206	613
80-89	110	115	225
>89	5	11	16

Tabla 2.4: Distribución de edades de los pacientes que son CN y AD

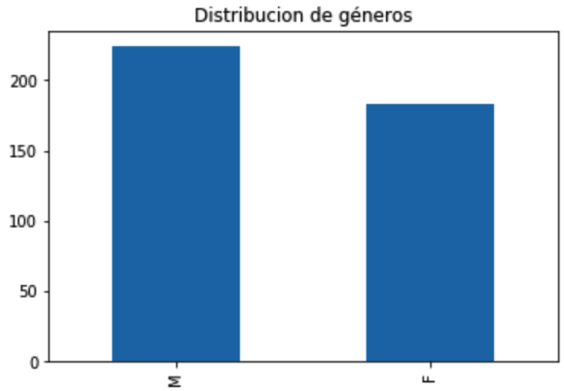
2.3.3. Datos disponibles

Las imágenes presentes en la base de datos de ADNI provienen de distintas modalidades de adquisición, y esto brinda una gran variedad de enfoques para investigar el desarrollo de la enfermedad de Alzheimer. En este trabajo se utilizaron resonancias magnéticas estructurales (MRI) para el desarrollo propuesto debido a su amplio uso en el diagnóstico de la enfermedad de Alzheimer, como fue explicado anteriormente. Las imágenes de resonancia magnética estructural que se pueden obtener de ADNI, además, tienen una variedad de secuencias de adquisición. Sin embargo, por los motivos explicados en las secciones anteriores, el enfoque de este trabajo será puesto en MRI obtenidas con la secuencia MPRAGE.

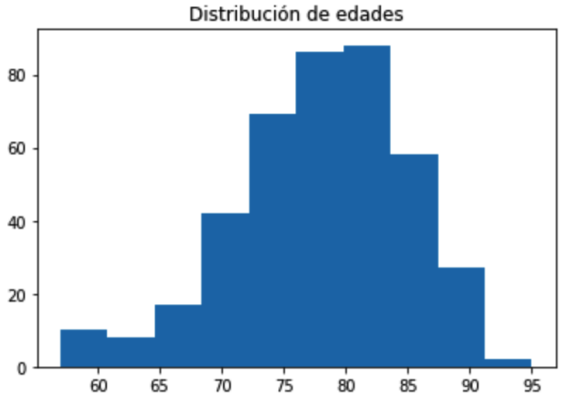
Al aplicar el criterio de selección de sujetos, se obtuvieron un total de 407 resonancias magnéticas estructurales para utilizar en el análisis. Como se puede ver en la figura 2.10, podemos analizar la población de sujetos que fueron seleccionados, por clase, género y edad. En cuanto a las clases, hay 204 sujetos que son CN, y 203 sujetos que son AD. En cuanto al género, se tienen 223 pacientes masculinos y 184 pacientes femeninos. Por último, se puede ver que la distribución de edades tiene su mayoría en el rango de 75-85 años.



(a) Distribución de clases de los sujetos.



(b) Distribución de género de los sujetos



(c) Distribución de edades de los sujetos.

Figura 2.10: Visualización de la información de los sujetos seleccionados para el análisis

Veamos algunos ejemplos de las imágenes que están disponibles para su análisis:

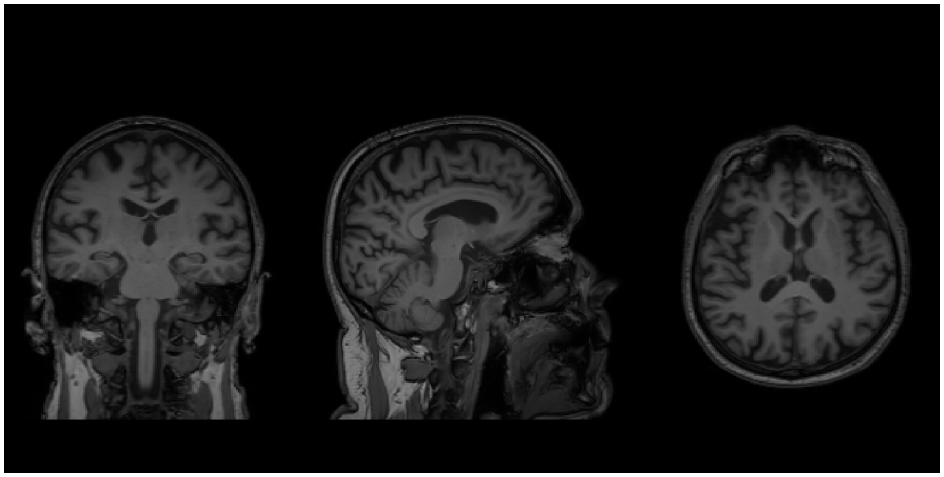


Figura 2.11: Sujeto CN, femenino, 76 años.

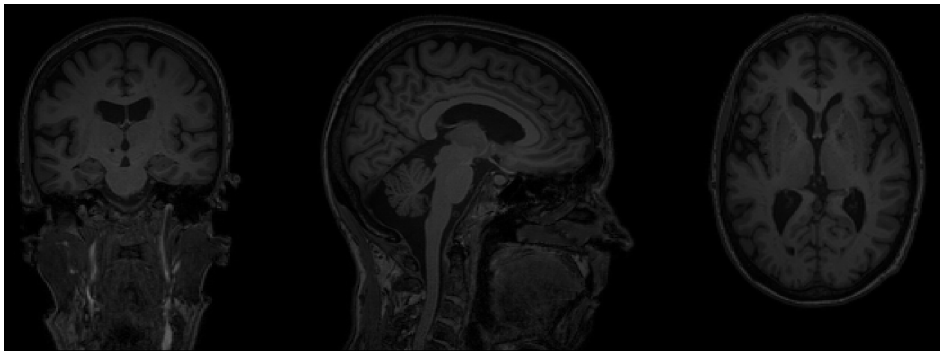


Figura 2.12: Sujeto CN, masculino, 75 años.

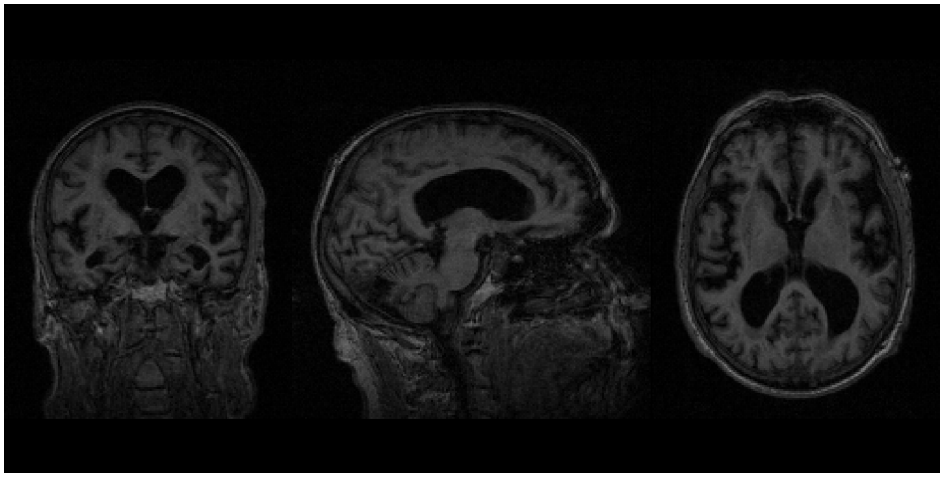


Figura 2.13: Sujeto AD, femenino, 74 años.

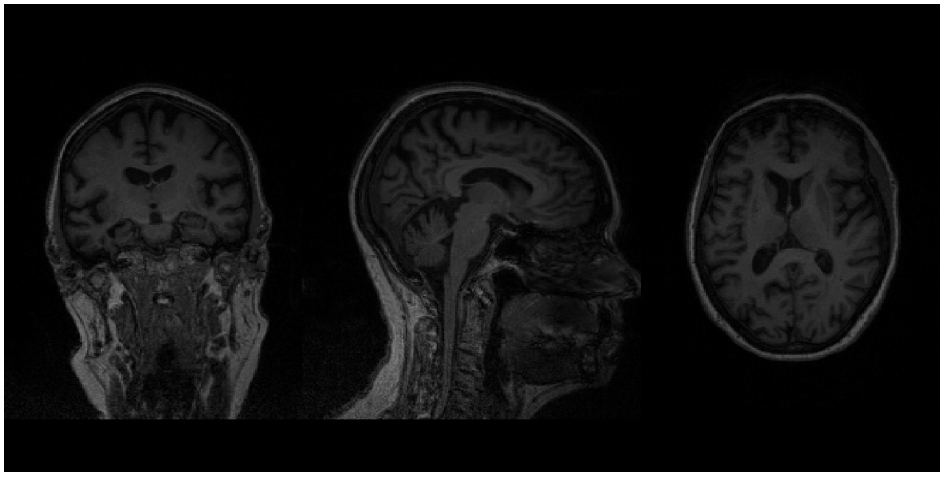


Figura 2.14: Sujeto AD, masculino, 73 años.

Podemos ver en estas imágenes que las imágenes contienen diversas orientaciones del paciente, la adquisición tiene diversas distribuciones de intensidades y que existen estructuras

que no aportan información a un análisis del cerebro, tales como los ojos, el cráneo, el cuello, etc.

Por lo tanto, en la siguiente sección se proponen y exploran una serie de pasos de procesamiento de las imágenes previos al análisis, o pasos de *preprocesamiento*, a partir del modelo que se propone.

2.4. Preprocesamiento

El preprocesamiento es esencial a la hora de utilizar resonancias magnéticas para entrenar un modelo de clasificación como el propuesto en este trabajo. Las resonancias magnéticas generalmente contienen tejido no cerebral, como ojos, grasa, médula espinal o el cráneo; también, suelen tener inhomogeneidades de intensidad por diversas causas, ó podrían tener diferentes resoluciones espaciales. Estas particularidades que suelen acompañar a los volúmenes que se quieren analizar deben ser resueltas. Por eso, para poder utilizar la información proporcionada por las resonancias magnéticas, es necesario definir una estrategia de preprocesamiento que permita estandarizar la información lo más posible, y así lograr el armado de un conjunto de datos a partir del cual se pueda extraer información útil para su análisis con el modelo propuesto. Existen varias estrategias, también llamadas *pipelines*, de preprocesamiento que han sido exploradas en distintos trabajos en los que se utilizan imágenes de MRI ([24], [25], [26], [27], [28]), los cuales varían según el problema que se desee resolver.

El pipeline de preprocesamiento que se siguió en este trabajo es el siguiente:

1. Estandarización de la resolución espacial
2. Corrección de inhomogeneidades de intensidad
3. Registración
4. Extracción de cráneo
5. Normalización de intensidades
6. Extracción de imágenes 2.5D

En los siguientes apartados se desarrollará cada paso del pipeline individualmente y se destacará su importancia para el problema tratado en este trabajo.

2.4.1. Estandarización de la resolución espacial

La resolución espacial de la resonancia magnética representa el volumen del espacio contenido en un vóxel de la misma. El tamaño del vóxel está definido por la modalidad de adquisición de la resonancia y el método de reconstrucción empleado. Por lo tanto, la resolución espacial puede variar entre resonancias tomadas por diferentes resonadores en diferentes centros médicos, como es el caso de las resonancias obtenidas de ADNI. Al tener resonancias con diferentes resoluciones espaciales, es posible que haya diferencias en lo que una misma coordenada o vóxel representa en el espacio, lo cual puede traer problemas a la

hora de implementar un modelo de aprendizaje automático que utilice dichas resonancias. Por eso, es necesario asegurar que todas las resonancias magnéticas que integren el dataset tengan la misma resolución espacial. Esto se logra mediante el remuestreo de las resonancias magnéticas a una resolución espacial estándar. Las dimensiones de las resonancias utilizadas son $(166, 256, 256)$ y tienen una resolución espacial de $1.2\times0.9375\times0.9375$ mm^3. Se tomó la resolución espacial estándar de $1\times1\times1$ mm^3, ya que es la resolución espacial del template que se utilizará para la registración en el tercer paso del presente pipeline de preprocesamiento.

2.4.2. Corrección de inhomogeneidades de intensidad

El método adquisición de resonancias magnéticas genera señales cuya intensidad medida a partir de tejidos homogéneos es raramente uniforme, sino que varía suavemente en la imagen. Esta variación se debe a diversos motivos, que incluyen poca uniformidad en las bobinas de radiofrecuencia del resonador, corrientes de eddy generadas por cualquier medio conductor dentro del resonador que producen variaciones en el campo magnético, y la anatomía del paciente. Estas variaciones de intensidad tienen poco impacto en el análisis visual de las resonancias magnéticas, pero afectan de forma considerable el desempeño de métodos de automáticos de segmentación que asumen uniformidad de intensidad a lo largo de los diferentes tejidos.

Para solucionar este problema, en [29] se introduce el método automático N3 para la corrección de las inhomogeneidades de intensidades en imágenes de resonancia magnética. En este método la inhomogeneidad de intensidad se modela como un campo multiplicativo, y considera la formación de imágenes de resonancia magnética de la siguiente forma:

$$v(x) = u(\mathbf{x})f(\mathbf{x}) + n(\mathbf{x}) \tag{2.3}$$

donde en el punto $\mathbf{x}$, v es la señal medida, u es la señal real emitida por el tejido, f es un "bias field" o un *campo de inhomogeneidad de intensidad*, al cual se lo denomina campo IIH (Intensity InHomogeneity field), desconocido y suavemente variante, y n es ruido blanco Gaussiano que se asume independiente de u. Sin embargo, se asume, como simplificación, un caso en el que no hay ruido y que u en cada punto de $\mathbf{x}$ son variables independientes idénticamente distribuidas, y usando la notación $\hat{u}(\mathbf{x}) = log[u(\mathbf{x})]$, el modelo de formación de la imagen de resonancia magnética se convierte en:

$$\hat{v}(\mathbf{x}) = \hat{u}(\mathbf{x}) + \hat{f}(\mathbf{x}) \tag{2.4}$$

Se considera que la distribución de valores que $\hat{f}$ toma sobre la región de interés como la distribución de probabilidades de una variable aleatoria. Así, se denomina U, V, y F las funciones densidades de probabilidad de $\hat{u}$, $\hat{v}$ y $\hat{f}$, respectivamente. Luego, haciendo la aproximación de que $\hat{u}$ y $\hat{f}$ son variables aleatorias independientes y no correlacionadas, la distribución de su suma puede hallarse a partir la siguiente convolución:

$$V(\hat{v}) = (F * U)(\hat{v}) = \int F\left(\hat{v} - \hat{f}\right) U\left(\hat{f}\right) d\hat{f} \tag{2.5}$$

La distribución de no uniformidad F se puede interpretar como si fuera una difuminación de la distribución de intensidad U. Desde una perspectiva de procesamiento de señales, esta

difuminación debida al campo IIH reduce las componentes de alta frecuencia de U. Entonces, para corregir la inhomogeneidad de intensidades, se debe restaurar el contenido de frecuencia de U. Por lo tanto, el enfoque del método es encontrar el campo multiplicativo suave, de variación lenta que maximiza el contenido en frecuencia de U, y para eso se propone una distribución para U "afilando"la distribución de V, y así estimar de manera iterativa el campo IIH que produce una distribución de U cercana a la propuesta. Una vez estimado el campo de inhomogeneidad, la corrección se produce al desconvolucionar la distribución F estimada de V.

El método N3 para la corrección del campo IIH tuvo mucho éxito, y fue ampliamente adoptado para el preprocesamiento de resonancias magnéticas debido a su buen desempeño y, además, debido a su disponibilidad como código abierto y gratuito. En este trabajo se utiliza la implementación propuesta en [30], llamada "N4ITK", que introduce mejoras en la estimación del campo IIH y en la estrategia iterativa de aproximación de dicho campo. Estos cambios introducidos generan mejoras en la convergencia del método. La variante N4ITK también se encuentra disponible como software de código abierto y por eso, además de mejorar el desempeño del N3, es ampliamente utilizado.

2.4.3. Registración

Introducción

Las resonancias magnéticas representan un volumen en el espacio a partir de vóxeles, pero no siempre todas las resonancias describen la misma disposición espacial, y por lo tanto, una misma coordenada de vóxel en dos resonancias diferentes puede no representar la misma estructura anatómica. Por lo tanto, es necesario asegurar que todas las resonancias magnéticas que se vayan a utilizar estén representadas en el mismo sistema de coordenadas espaciales, y que, en consecuencia, cada vóxel de cada resonancia represente la misma estructura anatómica. Para lograr esto, se utiliza la **registración**.

La registración de dos imágenes implica su alineación espacial, de manera que sus características en común se superpongan y sus diferencias se acentúen. Al registrar dos imágenes se define una transformación geométrica que alinea una imagen (denominada imagen móvil) para que se ajuste a otra (denominada imagen fija).

Podemos clasificar los métodos de registración basándos en la categorización hecha en [31], en la cual los autores utilizan nueve categorías para describir y clasificar las posibles variantes de los métodos de registración existentes. La categorización descrita puede resumirse en la siguiente lista:

1. **Dimensionalidad**: Espacial (2D/2D, 2D/3D, 3D,3D), o también serie temporal (más de dos imágenes con las dimensiones espaciales descriptas).

2. **Naturaleza de la registración**: Puede ser intrínseca, es decir que toma en cuenta características propias de las imágenes como zonas, puntos o intensidades, o puede ser extrínseca, que tiene en cuenta estructuras externas a la imagen

3. **Naturaleza de la transformación**: Puede ser rígida, afín, proyectiva o curva.

4. **Dominio de la transformación**: Local, si la transformación es aplicada a un conjunto de píxeles/vóxeles confinados a un área particular, o global si es aplicada a toda la imagen.

5. **Interacción**: Interactiva, semi automática, o automática.

6. **Procedimiento de optimización**: Según si los parámetros son computados o buscados.

7. **Modalidades**: Monomodal, multimodal, modalidad a paciente o paciente a modalidad.

8. **Sujeto**: Intrasujeto, intersujeto, Atlas.

9. **Objeto**: Áreas del cuerpo que la imagen captura.

Podemos ver a partir de esta clasificación la gran cantidad de variantes que pueden existir a la hora de resolver un problema de registración de imágenes, y por eso es muy importante definir correctamente el problema que se quiere resolver para poder tener en claro qué método de registración será más útil y eficaz. Además de la cantidad de variantes que puede haber, el problema de la registración puede ser dividido en cuatro componentes fundamentales: el espacio de características, el espacio de búsqueda, la estrategia de búsqueda, la métrica de similitud.

El **espacio de características** describe qué información de las imágenes será utilizada para alinearlas, y será lo que decida qué algoritmo se usará para realizar la registración. El **espacio de búsqueda** describe qué transformaciones serán utilizadas para alinear las imágenes, y depende de si la alineación será global o local, y de si se requiere que la alineación sea solamente posicional o si también se desea alinear las estructuras representadas de manera tal que se tengan en cuenta las deformaciones no uniformes presentes. De cualquier manera, podemos ver cómo el espacio de búsqueda es determinante a la hora de decidir qué algoritmo de registración se utilizará. La **estrategia de búsqueda** es el criterio que utilizamos para decidir cómo el método de registración elegido debe modificar la transformación para poder llegar al ajuste óptimo. Por último, la **métrica de similitud** irá de la mano con la estrategia de búsqueda ya que es la medida que será utilizada para decidir cómo modificar la transformación del método.

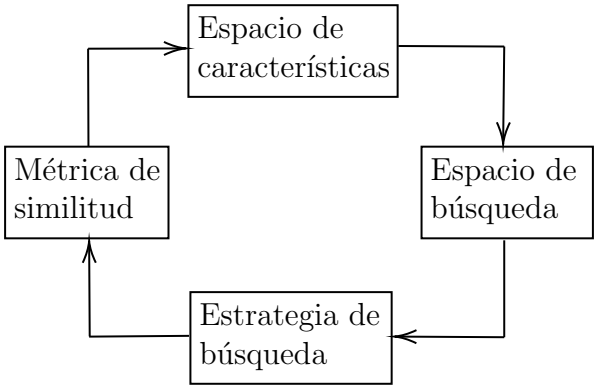


Figura 2.15: Componentes fundamentales de la registración.

Una vez definido el problema que se quiere resolver con la registración, y cuáles van a ser sus componentes fundamentales, podemos describir el proceso que se lleva a cabo. Dicho proceso está conformado por la imagen fija, la imagen móvil, la transformación, la métrica, y el método de optimización, como se puede ver en el siguiente diagrama:

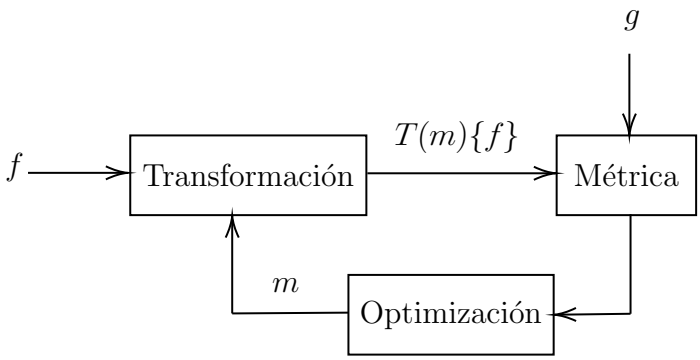


Figura 2.16: Proceso iterativo de la registración

Se aplica una transformación T con parámetros m a la imagen móvil f, se obtiene la imagen transformada $\hat{f} = T(m)\{f\}$. La naturaleza de la transformación puede ser lineal, que incluye operaciones como la traslación, rotación, escalamiento y otras transformaciones afines, que son de naturaleza global y no son capaces de modelar diferencias locales entre las imágenes. Por otro lado, las transformaciones pueden ser no lineales, o "elásticas", y son capaces de deformar localmente la imagen móvil. Usando la imagen fija, g y la imagen móvil transformada $\hat{f}$ se computa la métrica de similitud. Finalmente, utilizando algún algoritmo de optimización se calculan los nuevos parámetros m para redefinir la transformación y repetir el proceso descrito hasta que se llegue a un número máximo de iteraciones, o se logre maximizar o minimizar la métrica, según corresponda al caso.

Atlas

En este trabajo se utiliza la registración de imágenes con el objetivo de lograr la estandarización de resonancias magnética obtenidas de múltiples sujetos en distintos centros médicos, con distintos resonadores y bajo distintas condiciones. Por eso, lo que se necesita es que todas las resonancias compartan el mismo espacio, lo cual se logra registrando cada una de las resonancias a un atlas. Un atlas del cerebro es una representación específica que describe con gran detalle las estructuras anatómicas del mismo [32]. El primer atlas fue introducido por Talairach y Tournoux, era impreso, y describió por primera vez un sistema de coordenadas, conocido como el *Espacio de Talaraich*, que permitía identificar y etiquetar diferentes regiones del cerebro y fue ampliamente usado como estándar para la identificación de estructuras cerebrales, aunque tenía ciertas limitaciones al estar hecho usando un solo cerebro post-mortem de una mujer francesa de 60 años.

En los años siguientes se crearon muchos más atlas que utilizaban una gran cantidad de resonancias magnéticas para lograr un descripción estándar específica para población. Entre ellos, el Montreal Neurological Institute (MNI) desarrolló varios atlas que se basaban en el de Talairach and Tournoux, y uno de ellos, el MNI-152 (Figura 2.18) fue adoptado

por el International Consortium for Brain Mapping (ICBM) como su atlas estándar, y es el atlas utilizado en este trabajo para realizar la registración de las resonancias magnéticas utilizadas.

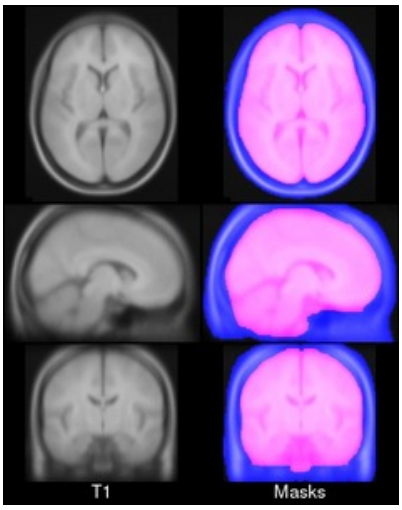


Figura 2.17: Atlas MNI-305.

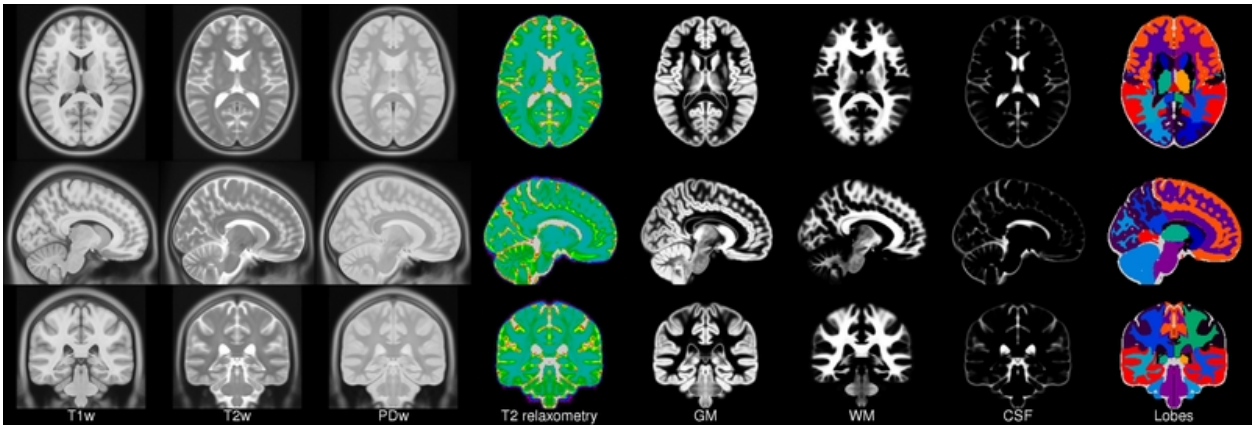


Figura 2.18: Atlas ICBM MNI-152.

Método de SyN

En este trabajo se utilizó el método de registración semi-automático llamado *SyN*, o Symmetric Normalization, propuesto en [33] e implementado en [34]. En este caso se utiliza con imágenes de monomodales, aunque también funciona bien con imágenes multimodales; su naturaleza es intrínseca, y emplea una combinación de transformación afín y deformable utilizando Información Mutua como métrica. Se elige este método en particular debido a que fue diseñado para ser utilizado en estudios que analicen imágenes que no sean estándar, como por ejemplo resonancias magnéticas de pacientes con enfermedades neurodegenerativas, ya que este tipo de imágenes violan las suposiciones básicas de pequeñas deformaciones y/o relaciones básicas de intensidad que se emplean en muchos métodos de registración

deformable. Los autores del método superan estas limitaciones utilizando una formulación de deformación deformable basada en difeomorfismos simétricos.

Un difeomorfismo es un isomorfismo diferenciable, lo cual, en otras palabras, significa que es una transformación biyectiva continua y con inversa continua, lo cual garantiza que la transformación definida por el método tenga inversa y que dicha inversa sea diferenciable. La registración de imágenes usando difeomorfismos sirve para tratar con pequeñas y grandes deformaciones y, además, al ser un método simétrico, el método SyN es invariante al orden en que se ingresen las imágenes que se quieren registrar. Es decir, que no importa el orden en el que se ingresen las imágenes fija y móvil, el resultado será el mismo. Además, la formulación de este método permite utilizar métricas de similitud probabilísticas, mientras que otros métodos no garantizan simetría para métricas que no utilicen diferencias de intensidades entre los píxeles de las imágenes.

Los autores en [33] proponen la implementación del método utilizando Correlación Cruzada como la métrica a optimizar en la registración, argumentando que el problema que surge al usar Información Mutua, una métrica ampliamente usada para problemas similares al tratado en [33], es que la convergencia de la registración se vuelve susceptible a cambios locales en las intensidades de las imágenes, como lo es el campo IIH que fue mencionado anteriormente. Esto sucede debido a que esta métrica estima la alineación óptima de las imágenes a partir de la cantidad de información global compartida como es visto en el histograma conjunto de las mismas. Se comprobó que realizar la registración usando Correlación cruzada tarda alrededor de 2 horas por resonancia, mientras que aplicar el algoritmo N4 para corregir la inhomogeneidad de intensidades y registrar usando Información Mutua tarda alrededor de 7 minutos por imagen. Por lo tanto, se decidió utilizar Información Mutua como la métrica de la registración ya que se corrigen las inhomogeneidades de intensidad en un paso anterior del pipeline de preprocesamiento y, de esa manera, el problema que mencionan los autores de [33] queda solucionado.

2.4.4. Extracción de cerebro

Introducción

La extracción de cerebro es un paso ampliamente adoptado en los pipelines de preprocesamiento de resonancias magnéticas de cerebro que consiste en la segmentación estructural de la resonancia en tejido cerebral y tejido no cerebral. Existen muchos enfoques para lograr esta segmentación:

- **Segmentación Manual:** Al realizar la segmentación manual de las estructuras se puede obtener muy buena precisión debido a que lo realiza una persona con un profundo conocimiento estructural y, a partir de este conocimiento, puede lograr identificar la superficie cerebral correctamente. Las desventajas de este enfoque son el costo temporal que conlleva a una persona llevar a cabo la segmentación, la imposibilidad de paralelizar el proceso y la necesidad de entrenamiento para poder detectar las estructuras pertenecientes al cerebro más difíciles de distinguir.
- **Umbralización con morfología matemática:** En esta clase de método se utiliza una umbralización de intensidades básica para distinguir las zonas con intensidades

más elevadas (que suele corresponder con los ojos y partes del pericráneo), las zonas con intensidades intermedias (tejido cerebral), y las partes más oscuras (que pueden ser aire y el cráneo). Cabe aclarar que la definición de los umbrales requiere que el usuario se involucre, por lo cual esta clase de método resulta ser semi-automático. Luego de la umbralización, se crea una máscara binaria que determine la forma del cerebro y se aplique a la imagen original con el objetivo de realizar la segmentación. Sin embargo antes de realizar la segmentación final, debido a que la máscara suele contener vóxeles de intensidades elevadas pertenecientes a estructuras no cerebrales, se debe erosionar la máscara hasta que estas uniones entre tejido cerebral y tejido no cerebral sean eliminadas y se elige el conjunto de mayor tamaño obtenido, el cual se dilata tanto como se erosionó para que resulte en una máscara del cerebro lo más precisa posible. Las limitaciones que tienen esta clase de métodos son que los umbrales de la segmentación inicial deben ser definidos por el usuario, y por eso es necesario probar el algoritmo varias veces para obtener una buena segmentación; otro problema es que la etapa de morfología matemática es sumamente difícil de generalizar, y por lo tanto el algoritmo varía dependiendo de la resonancia magnética que sea utilizada.

- **Modelos de superficies deformables:** Esta clase de métodos utiliza la definición de modelos superficiales que se ajustan a la superficie cerebral de la imagen con ciertas restricciones que se relacionan con las características de la superficie que se ajustará al cerebro y con el ajuste de la misma a la parte correcta de la imagen. El ajuste se hace deformando la superficie iterativamente desde su posición inicial hasta que se encuentra una solución óptima. Este enfoque de segmentación es por lo general más robusta y más fácilmente automatizable que los métodos de umbralización con morforlogía matemática.

BET - Brain Extraction Tool

En este trabajo se utiliza el método de segmentación de tejido cerebral y tejido no cerebral, llamado *Brain Extraction Tool*, introducido en [35], el cual implementa un modelo de superficie deformable.

En rasgos generales, el método utiliza el histograma de intensidades de la resonancia magnética para encontrar un umbral aproximado que separe tejido cerebral de tejido no cerebral. Luego, se encuentra el centro de gravedad de la cabeza, junto con el tamaño aproximado de la cabeza en la imagen. La superficie deformable se inicia mediante una teselación de una superficie esférica en el centro de gravedad de la cabeza y se la dejorma iterativamente siguiendo restricciones que mantienen la superficie bien espaciada y suave, mientras se intenta hacerla crecer hasta el borde del cerebro. Si no se llega a una solución satisfactoria, el proceso se vuelve a correr con restricciones para la suavidad de la superficie mayores.

En la siguiente figura se puede ver la evolución de la superficie modelada a lo largo de las iteraciones del método:

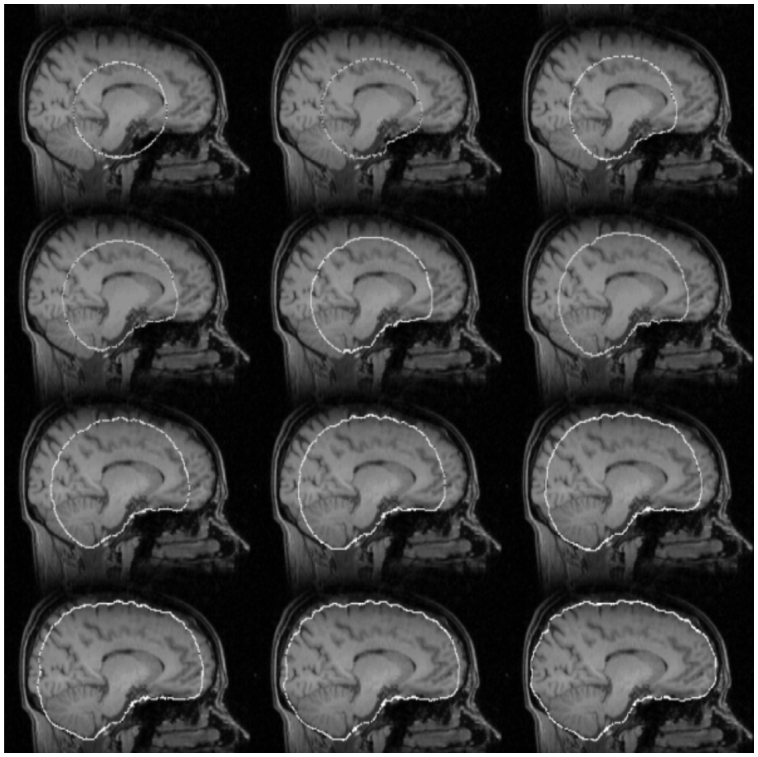


Figura 2.19: Ejemplo del desarrollo de la superficie modelada. Imagen obtenida de [35]

Este método fue elegido debido a que es completamente automático, y se encuentra disponible como software de código abierto y gratuito que se puede usar a través de la línea de comando o como parte de la FMRIB Software Library (FSL) [36] en la forma de FSL-BET.

Al usar FSL-BET, se distinguen dos parámetros principales que se deben establecer correctamente para el uso del método: el parámetro *f*, el cual es el umbral de intensidad fraccional utilizado determinar la ubicación en intensidad del cerebro segmentado final. El valor por defecto es 0.5, y el rango válido es entre 0 y 1. Un valor pequeño va a causar que el cerebro segmentado sea de mayor tamaño, mientras que un valor más grande tendrá el efecto opuesto; y las opciones adicionales que ofrece el método, como por ejemplo B (Bias field correction o Corrección de inhomogeneidades de intensidad), R (estimación robusta del centro del cerebro), y S (extracción del ojo y nervio óptico).

Utilizando los resultados de las pruebas hechas en [37] para encontrar la combinación óptima de los parámetros mencionados, se probaron distintas combinaciones de parámetros y, por prueba y error, se llegó a la conclusión que la combinación que mejor funciona para las resonancias utilizadas, luego de haber realizado los pasos previos a éste en el pipeline, es $f = 0.4$ y R.

2.4.5. Normalización de intensidades

Introducción

Además de la estandarización espacial de las resonancias magnéticas, que fue tratada hasta el momento en los pasos del pipeline de preprocesamiento descritos, la estandarización

de intensidades juega un rol fundamental a la hora de realizar un análisis cuantitativo de las estructuras representadas en ellas. Las intensidades de las resonancias magnéticas no tienen un significado fijo, ni siquiera utilizando un mismo protocolo de adquisición para la misma región del cuerpo para el mismo paciente en el mismo resonador ya que dependen de la composición de los tejidos del paciente. Por lo tanto, es necesario asegurar que todas las resonancias que se utilizan en el trabajo tengan distribuciones de intensidades similares, y en consecuencia que los tejidos representados tengan un rango de intensidades similar para que el modelo implementado pueda realizar un análisis cuantitativo significativo.

Por otro lado, el método de normalización empleado debe poder lograr una estandarización que pueda sea aplicable a cualquier resonancia que integre el análisis. Para lograr eso, se utilizó la estrategia de normalización por estandarización del histograma. La idea de esta estrategia de normalización es deformar los histogramas de las resonancias magnéticas para que coincidan con un histograma estándar. De esta manera, todas las distribuciones de intensidades de las resonancias magnéticas son mapeadas a una distribución estándar y en consecuencia todas las resonancias terminan teniendo una distribución similar.

Método de Nyul y Udupa

El método de normalización que se utilizó en este trabajo es el propuesto por Nyul y Udupa en [38] y [39]. El enfoque de normalización propuesto por los autores consiste en dos etapas. En la primera etapa, la etapa de "entrenamiento", se utiliza un conjunto de resonancias magnéticas de la misma región corporal y protocolo de adquisición como entrada al método. A partir de estas imágenes se estiman los puntos de referencia de un histograma "estándar". Esta etapa debe ser ejecutada sólo una vez para la región del cuerpo y protocolo de adquisición de las resonancias magnéticas. La segunda etapa es la etapa de "transformación", en la que cualquier resonancia de la misma región corporal y que haya sido adquirida con el mismo protocolo que las resonancias a partir de las cuales se aprendió la escala estándar es transformada para que los parámetros de sus histogramas coincidan con los del histograma estándar. De esta manera, el histograma de cada resonancia magnética es deformado para que coincida con el histograma estándar. Las transformaciones resultan en imágenes que mantienen las relaciones entre las intensidades de los tejidos, y por lo tanto, pueden ser utilizadas para realizar comparaciones y análisis con dichas intensidades.

Para explicar más en detalle el método, adoptaremos la notación de los autores: Se denomina al conjunto de protocolos de adquisición de resonancias magnéticas $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ y el conjunto de las regiones corporales $\mathcal{D} = \{D_1, D_2, ..., D_m\}$. Se representa una imagen volumétrica a partir del par $\mathcal{V} = (V, g)$ donde V es el arreglo tridimensional de vóxeles del paciente en particular cuya resonancia magnética fue adquirida, y g es un función llamada *función de intensidad*, que asigna un valor de intensidad a cada vóxel $v \in \mathcal{V}$. También, se asume que $g(v) \geqslant 0, v \in \mathcal{V}$ y que $g(v) = 0$ sólo si no hay información medida y computada para el vóxel v. Además, se llama $\mathcal{V}_{PD}$ al conjunto de todas las imágenes que puedan ser generadas por un protocolo $P \in \mathcal{P}$ para la región corporal $D \in \mathcal{D}$. Luego, se define el histograma de cualquier imagen $\mathcal{V}$ como el par $\mathcal{H} = (G, h)$ donde G es el conjunto de todos los posibles valores de intensidades de $\mathcal{V}$ y h es una función cuyo dominio es G y cuyo valor para cada $x \in G$ es la cantidad de vóxeles $v \in \mathcal{V}$ para los cuales $g(v) = x$. Se definen $m_1 = min\{g(v)|v \in \mathcal{V} \text{ y } g(v) > 0\}$ y $m_2 = max\{g(v)|v \in \mathcal{V} \text{ y } g(v) > 0\}$, como los valores

de intensidad mínimo y máximo, respectivamente, en $\mathcal{V}$. Los autores consideran deseable no incluir las "colas"del histograma. ya que suelen ser problemáticos; la cola de alta intensidad suele corresponder a artefactos y a valores de intensidad aislados que causan variaciones considerables entre los pacientes y resonadores. Por eso, se definen pc_1 y pc_2 como los valores percentiles mínimo y máximo, respectivamente, que se utilizan para definir un rango de intensidades de interés (IOI). Luego, para definir los puntos de interés en el histograma veamos a modo ilustrativo un histograma de las resonancias que, luego de una búsqueda extensa de combinaciones entre protocolo de adquisición y región corporal, los autores afirman es el más representativo de las combinaciones observadas:

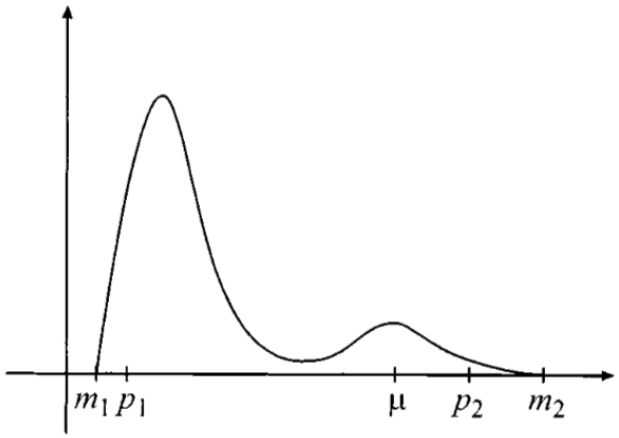


Figura 2.20: Ubicación de los puntos de interés en el histograma [39]

En la figura 2.20 se pueden ver que el histograma es bimodal y los parámetros que se ven ubicados son m_1 y m_2, previamente definidos, p_1 y p_2 que corresponden a pc_1 y pc_2 en el histograma modelo, y por último, μ que corresponde al segundo modo y representa el objeto principal de la resonancia.

Se considera la configuración de puntos de interés como la siguiente:

$$L = \{pc_1, pc_2, \mu_l\}$$

Donde pc_1 y pc_2 son los percentiles definidos anteriormente, y μ_l es el l-ésimo percentil en el histograma asociado con el objeto principal de una imagen. De esta forma, para cualquier imagen dada $\mathcal{V}_i$ se usa la notación p_{1i} y p_{2i} para denominar las intensidades correspondientes a pc_1 y pc_2, y se utiliza μ_{1i}, μ_{2i},..., μ_{li} para representar las intensidades en $\mathcal{V}_i$ que corresponden a puntos de interés distintos de pc_1 y pc_2. Por ejemplo, si se toman la configuración de puntos de interés $L_1 = \{pc_1, pc_2, \mu_{25}, \mu_{50}, \mu_{75}\}$, entonces μ_{1i}, μ_{2i} y μ_{3i} corresponden a los percentiles 25, 50 y 75, respectivamente.

Habiendo establecido las definiciones necesarias, se pueden describir los dos pasos del método. Primero, se definen las intensidades máximas y mínimas de la escala estándar para el rango de intensidades de interés, las cuales se llamarán s_1 y s_2. En la etapa de entrenamiento, los puntos de interés $(p_{1j}, p_{2j}, \mu_{1j}, \mu_{2j}, ..., \mu_{lj})$ obtenidos de cada imagen $\mathcal{V}_j$ de un conjunto de imágenes son mapeados linealmente desde $[p_{1j}, p_{2j}]$ a $[s_1, s_2]$ como se ve en la figura 2.21. Luego, para cada k tal que $1 \leqslant k \leqslant l$, se calcula la media μ_{ks} de los μ_{kj} mapeados de cada

imagen $\mathcal{V}_j$. En la etapa de transformación, para cada imagen $\mathcal{V}_i$, se obtienen los puntos de interés μ_{ki} de su histograma y se corresponen con μ_{ks} haciendo varios mapeos por separado: el primero desde $[p_{1i}, \mu_{1i}]$ a $[s_1, \mu_{1s}]$; el segundo desde $[\mu_{1i}, \mu_{2i}]$ a $[\mu_{1s}, \mu_{2s}]$, y de esa manera hasta la última que es desde $[\mu_{li}, p_{2i}]$ a $[\mu_{ls}, s_2]$ como se puede ver en la figura 2.22.

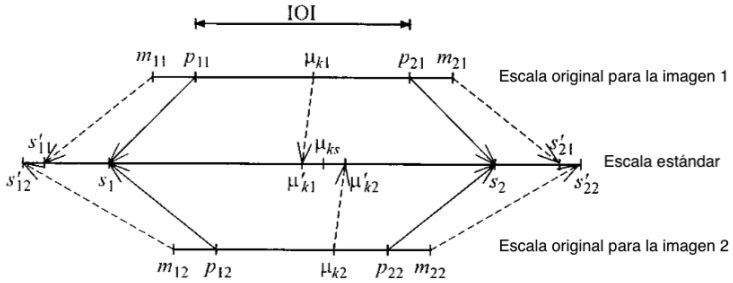


Figura 2.21: Mapeo de intensidades para la etapa de entrenamiento de dos imágenes como ejemplo. [39]

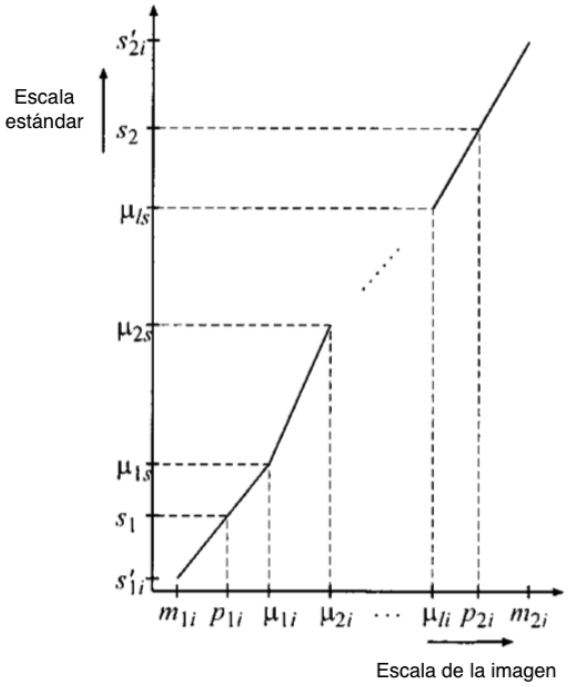


Figura 2.22: Mapeo de intensidades para la etapa de transformación. [39]

Las figuras 2.21 y 2.22 muestran los valores s'_{1i} y s'_{2i} los cuales representan los valores extremos en la escala estándar, y son determinados por el mapeo de los segmentos extremos: $[m_{1i}, p_{1i}]$ a $[s'_{1i}, s_1]$ y $[p_{2i}, m_{2i}]$ a $[s_2, s'_{2i}]$. Se denomina el mapeo general de las intensidades $[m_{1i}, m_{2i}]$ de $\mathcal{V}_i$ a $[s'_{1i}, s'_{2i}]$ de la escala estándar, el *mapeo estandarizador de* $\mathcal{V}_i$, $\tau_{\mathcal{V}_i}$. Vale remarcar que los valores de s'_{1i} y s'_{2i} de la escala estándar dependen de cada imagen $\mathcal{V}_i$. Es decir, que el rango $[s'_{1i}, s'_{2i}]$ puede variar de imagen a imagen. Sin embargo, el rango

$[s_1, s_2]$ es independiente de la imagen, y éste es el intervalo dentro del cual se adquiere la uniformidad de intensidades.

Debido a que todas la imágenes usadas en este trabajo pertenecen a la misma región corporal (D) y fueron adquiridas con el mismo protocolo (P), se asume que las imágenes forman un conjunto representativo de $\mathcal{V}_{PD}$, y por lo tanto la escala estándar aprendida sirve para normalizar las intensidades de cualquier resonancia magnética $\mathcal{V}_i \in \mathcal{V}_{PD}$.

2.4.6. Extracción de imágenes 2.5D

Luego de preprocesar las resonancias magnéticas con los pasos explicados anteriormente, se debe definir una estrategia para extraer de ellas la información a analizar. Un posible enfoque podría ser analizar el volumen completo. Sin embargo, el análisis de imágenes 3D, como lo son los volúmenes descritos por las resonancias magnéticas, requiere de modelos de muy complejos y capacidades computacionales que exceden los recursos disponibles para este trabajo. Además, la cantidad de resonancias requeridas para realizar un modelo apropiado es del orden de las miles de resonancias, lo cual supera la cantidad disponible para este trabajo. Otro enfoque que se podría tomar sería el de seleccionar cuidadosamente un corte de uno de los planos de la resonancia, tal que la imagen seleccionada contenga la mayor cantidad de información posible que permita la resolución del problema. De esta manera obtendríamos una imagen 2D de cada resonancia, lo cual nos llevaría a utilizar modelos de menor dimensionalidad y que no requieran tantos recursos computacionales como los modelos necesarios para analizar imagenes 3D, pero no superaríamos la limitación de la falta de resonancias.

Como alternativa para superar estas limitaciones, se siguió el enfoque usado en [24], y propuesto en [25], en el que se extraen cortes de 32×32 de los planos transversal, sagital y coronal, centrados en un punto, y se los combina en una imagen RGB 2D, denominada **patch 2.5D**. De esta forma, se extraen los tres cortes de varios vóxeles de una región de interés y se combinan en los tres canales de una imagen a color que contiene información del volúmen 3D en esos puntos.

Los vóxeles elegidos para formar los patches 2.5D son aquellos puntos pertenecientes al hipocampo, debido a que la atrofia del hipocampo está altamente relacionada con la Enfermedad de Alzheimer y es un marcador temprano de la misma que ayuda a diferenciar individuos afectados de aquellos controles saludables [40],[41],[42],[43],[44]. Luego, para elegir los vóxeles de los cuales se extraerán los patches 2.5D se definen tres restricciones:

1. Los vóxeles deben proceder del hipocampo izquierdo o derecho.
2. Debe haber una distancia de al menos dos vóxeles entre cada punto elegido.
3. Todos los puntos son elegidos aleatoriamente.

Siguiendo estas restricciones se pueden extraer los patches de la siguiente forma:

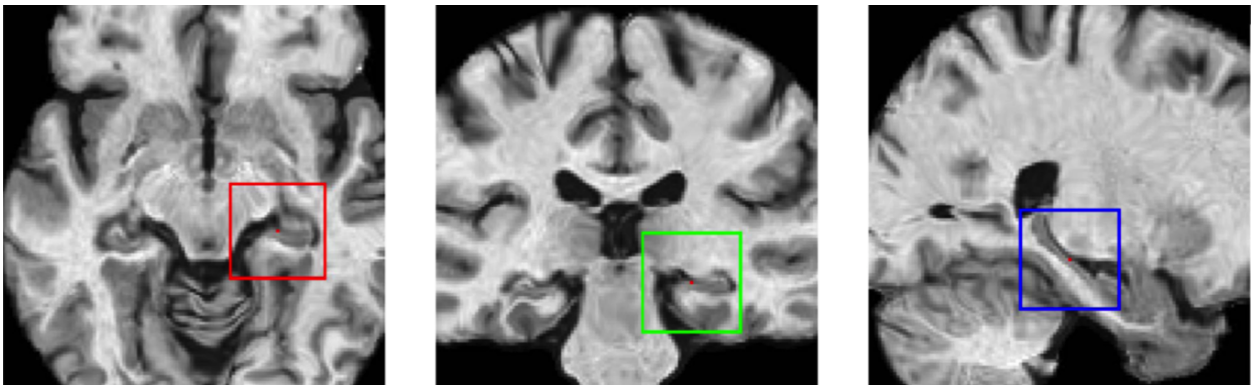
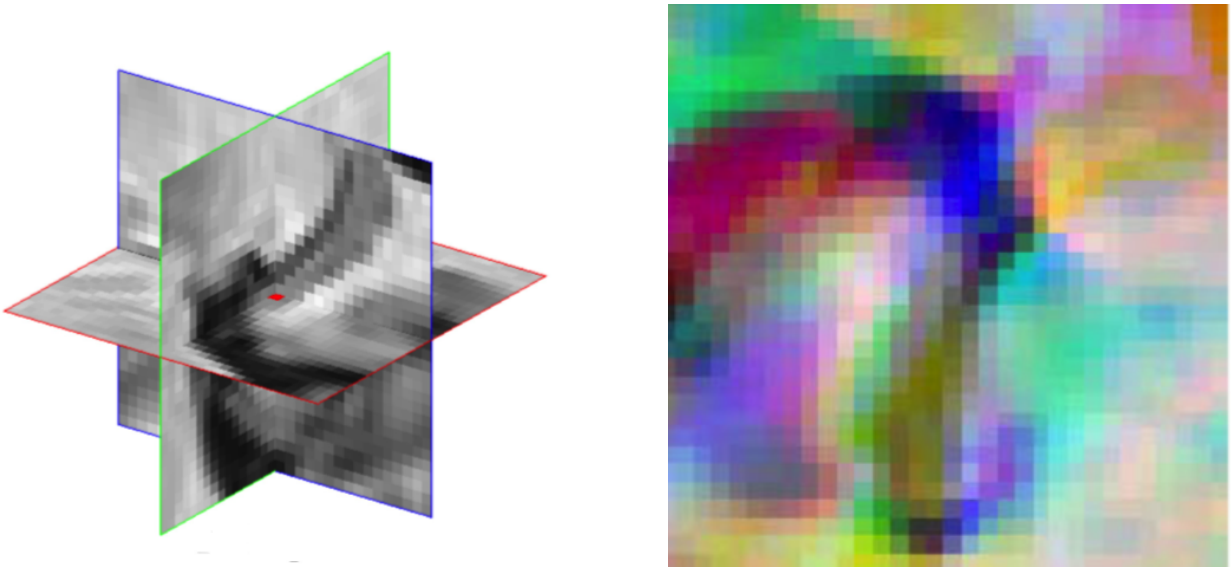


Figura 2.23: Extracción de los cortes 2D de cada plano de la resonancia magnética. En las cajas de colores se ven las imágenes de 32×32 que serán extraídas. [24]



(a) Visualización de los cortes formando el volumen 3D descrito.

(b) Patch 2.5D formado.

Figura 2.24: Combinación de las imágenes de 32×32 para formar el patch 2.5D. [24]

Los patches extraídos contienen información acerca de la atrofia del hipocampo del paciente, mostrando una diferencia entre pacientes pertenecientes al grupo AD y los pertenecientes al grupo CN como se puede ver en la figura 2.25. En esta figura se se ve cómo la atrofia es representada como espacios oscuros en las imágenes de los planos, y al unir dichos planos para formar el patch RGB se puede ver un sombreado distintivo.

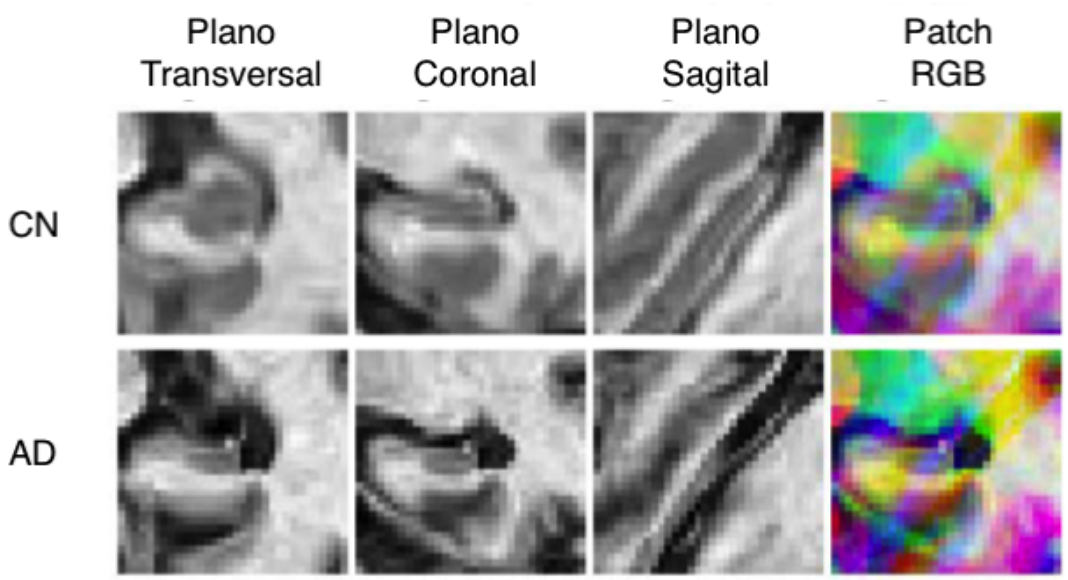


Figura 2.25: Comparación de los patches extraídos de un sujeto del grupo AD y del grupo CN en la que se ve la atrofia del hipocampo. [24]

En este trabajo se utilizaron vóxeles procedentes del hipocampo izquierdo, y siguiendo las demás restricciones se obtuvo un total del 113 patches 2.5D por resonancia magnética. En cuanto a la adquisición de los puntos del hipocampo, se optó por delimitar manualmente un volumen que incluya al hipocampo izquierdo en el template utilizado para la registración. A partir de la ubicación de éste volumen se obtuvo el conjunto de puntos que lo definen. Luego, una vez obtenidos los puntos que definen la región de interés, se chequeó sobre 10 resonancias elegidas al azar que la región definida contenga el hipocampo izquierdo.

2.4.7. Implementación del pipeline

La implementación del pipeline fue realizada con una MacBook Pro de 13" del año 2015 con el sistema operativo macOS Mojave versión 10.14.6 (18G95), procesador Intel Core i5 de 2,7 GHz, memoria RAM de 8 GB 1867 MHz DDR3 y 256 GB de memoria SSD.

Herramientas utilizadas

El pipeline de preprocesamiento fue programado en Python usando un entorno de trabajo con la versión de Python que asegura la compatibilidad entre los paquetes necesarios.

Para los pasos de estandarización espacial, aplicación del método N4ITK para la corrección del campo de bias, y la registración mediante el método SyN se utilizó el paquete ANTsPy [45], el cual es una implementación que engloba todos los métodos de procesamiento de imágenes biomédicas de la librería de C++, ANTs [34] [46]. Además, permite la integración con los paquetes de manejo de datos mas ampliamente utilizados en Python, como numpy y scikit-learn.

Luego, para la extracción del cerebro se utilizó el paquete Nipype [47], el cual es un proyecto de código abierto con el objetivo de ofrecer una interfaz unificada para varias

herramientas de procesamiento de imágenes biomédicas en el área neurológica. En particular, se utilizó la integración con FSL que Nipype ofrece para poder utilizar el método BET.

Finalmente, para el paso de normalización de intensidades se utilizó la implementación del método propuesto por Nyul y Udupa en [38] y [39] hecha en [48]. Dado que el paquete Nipype requiere que la versión de Python sea 2.6, el entorno de trabajo que se utilizó tenía esta versión del lenguaje y se aseguró que los demás paquetes puedan ser utilizados de esta manera.

Implementación en paralelo

Teniendo en cuenta el hardware disponible, se deben considerar los recursos que el pipeline de preprocesamiento implementado requiere para su funcionamiento, y los tiempos que toma su ejecución. En la figura 2.26 podemos ver un diagrama de flujo ilustrativo del pipeline propuesto. Esta implementación está dividida en tres etapas: La primera etapa consiste en los pasos de estandarización espacial, corrección de inhomogeneidades, registración y extracción de cerebro; la segunda etapa es el paso de normalización de intensidades; y la tercera etapa es la extracción de patches 2.5D. La división del pipeline se hace de esta forma debido a que la primera etapa está compuesta por pasos que se deben hacer de a una resonancia a la vez, mientras que la segunda etapa debe hacerse con un conjunto de resonancias, o con todas.

La primera etapa, al tener la restricción de que cada paso procesa una resonancia por vez, se implementa de forma iterativa con un bucle *for* que itera sobre los archivos de resonancias magnéticas crudas y termina al procesar el último archivo presente. Teniendo en cuenta que Python por defecto no distribuye sus tareas en varios núcleos, entonces el pipeline se ejecutará de forma secuencial utilizando un solo núcleo de la computadora y con los recursos de memoria que se le hayan asignado al mismo. Luego de varias pruebas, se estimó que el tiempo medio de ejecución de la primera etapa toma alrededor de 2.5 minutos por resonancia magnética. Teniendo 407 resonancias, solamente la primera etapa tomaría alrededor de 17 horas. Sin embargo, Python ofrece la posibilidad de ejecutar tareas en paralelo de forma simple utilizando el paquete *concurrent*. De esta forma, se pueden utilizar los recursos de la computadora de forma más eficiente.

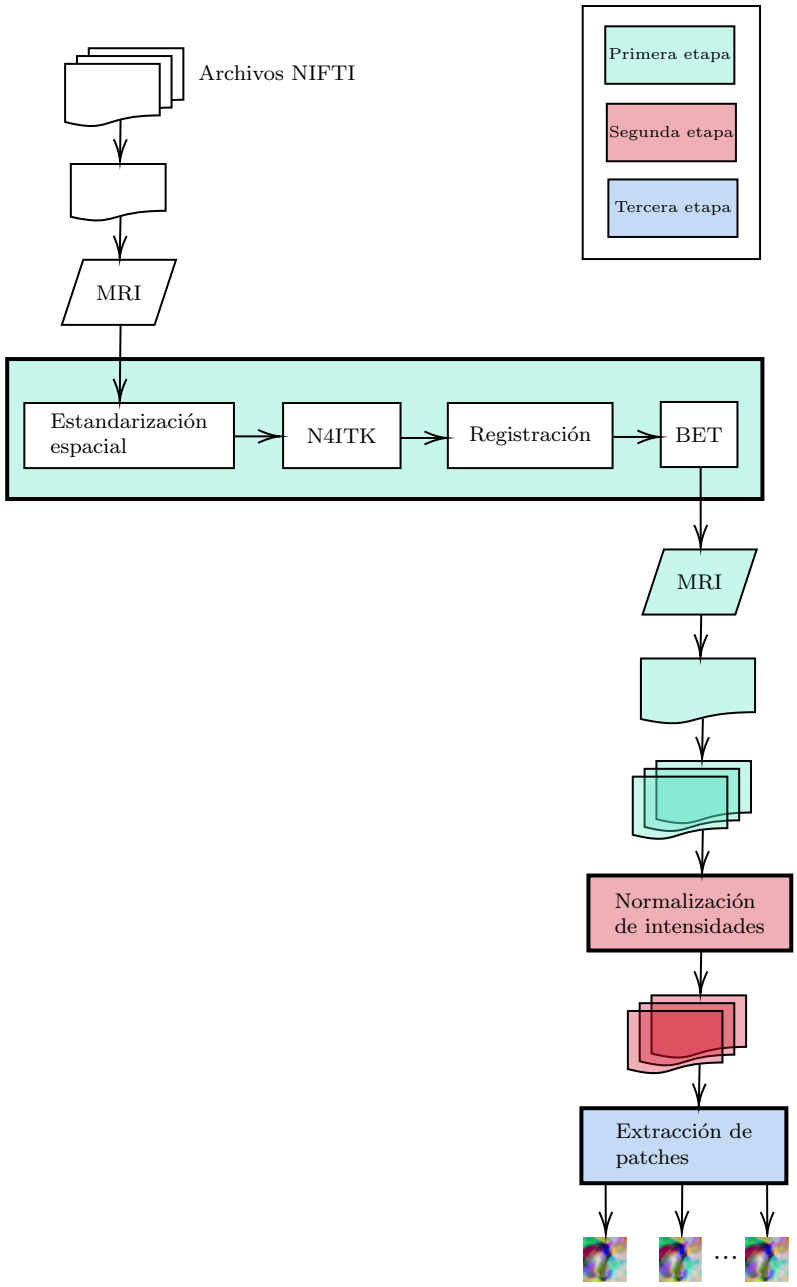


Figura 2.26: Diagrama de flujo del pipeline de preprocesamiento.

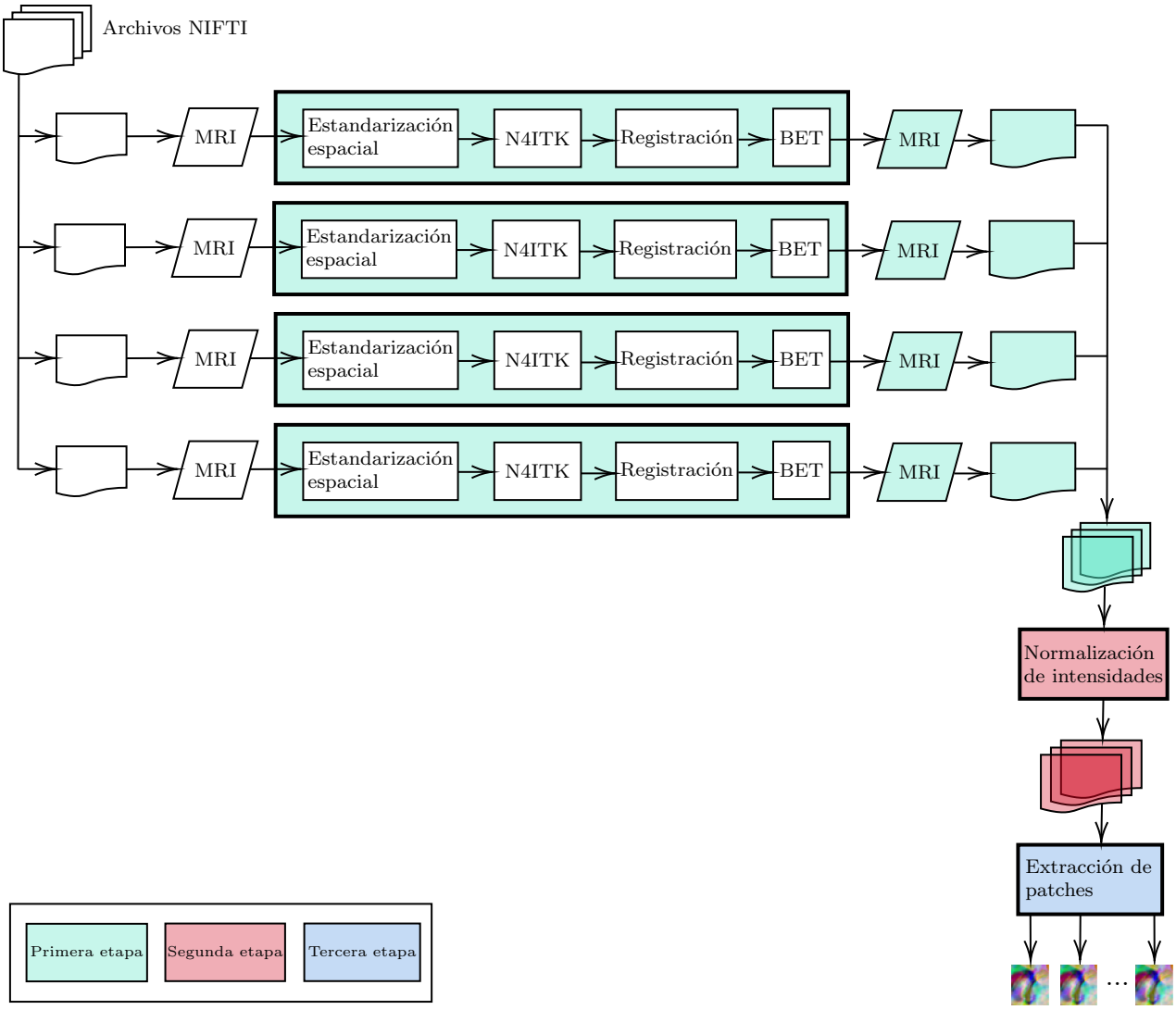


Figura 2.27: Diagrama de flujo del pipeline de preprocesamiento con la primera etapa implementada en paralelo.

Por lo tanto, se decidió implementar la primera etapa en paralelo, como se puede ver en la figura 2.27, de forma tal que se puedan utilizar los cuatro núcleos disponibles en la computadora para poder disminuir los tiempos de procesamiento de la etapa. Lo primero que se hizo fue probar la hipótesis de que la implementación en paralelo efectivamente genera una mejora significativa de los tiempos de preprocesamiento. Para eso se probó preprocesar 8 resonancias de forma secuencial y en paralelo. Esa prueba mostró que al preprocesar las 8 resonancias de manera secuencial se tardó 21 minutos, mientras que al preprocesar las mismas 8 resonancias en paralelo se tardó 8 minutos. Por lo tanto, se puede ver una gran mejora en los tiempos la cual se verá reflejada al ejecutar el pipeline sobre la totalidad de las resonancias. Finalmente al realizar la primera etapa para la totalidad de las resonancias magnéticas se tardó 7 horas, demostrando la eficiencia de la implementación realizada en contraste a las estimaciones de la implementación secuencial.

La segunda etapa del pipeline de preprocesamiento tardó un total de 43 minutos, di-

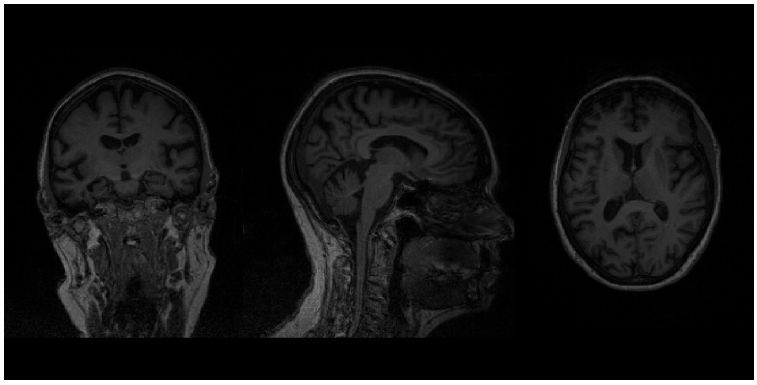
vididos en 12 minutos para el entrenamiento y 31 minutos para la transformación de las resonancias. No se implementó la paralelización de la transformación, la cual se aplica a cada imagen individualmente, debido a que se tendría que haber modificado la implementación del método de normalización para ello y no se consideró que la ganancia de tiempo que esto traería habría sido de tal magnitud como para justificar tal modificación.

En la tercera etapa, la extracción de patches tardó 17 minutos y tampoco se la implementó en paralelo debido a que la aceleración que esto traería no es significativa. Finalmente, para las 407 resonancias magnéticas presentes, el pipeline de preprocesamiento completo tomó aproximadamente 8 horas.

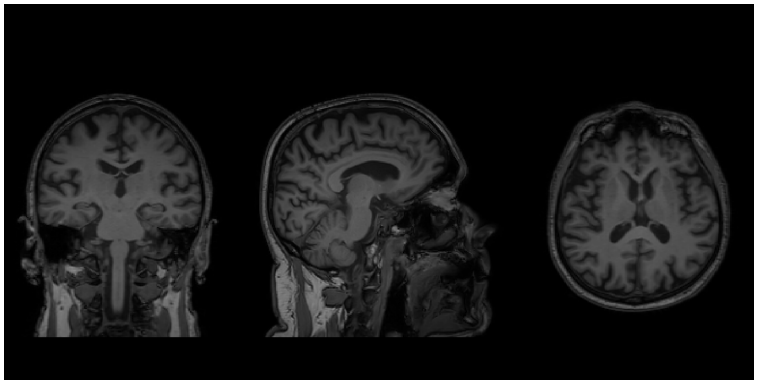
2.4.8. Resultados

Veamos las salida por cada paso de un sujeto AD y otro CN para poder visualizar en concreto el efecto de cada uno de los métodos aplicados.

Inicialmente tenemos las imágenes crudas, como se puede ver en la figura 2.28:



(a) Sujeto AD



(b) Sujeto CN

Figura 2.28: Imágenes de un sujeto perteneciente a la clase AD y a la clase CN antes de iniciar el pipeline de preprocesamiento.

Primera etapa

En esta etapa la imagen cruda pasa por un remuestreo que estandariza la resolución espacial a $1\times1\times1$ mm^3 y las dimensiones se transforman a $(199, 240, 240)$. Luego, se le aplica el método N4ITK para la corrección de inhomogeneidades de intensidad. También, se realiza la registración al template (figura 2.29) con el método SyN, y finalmente se realiza la extracción de cráneo con el método BET.

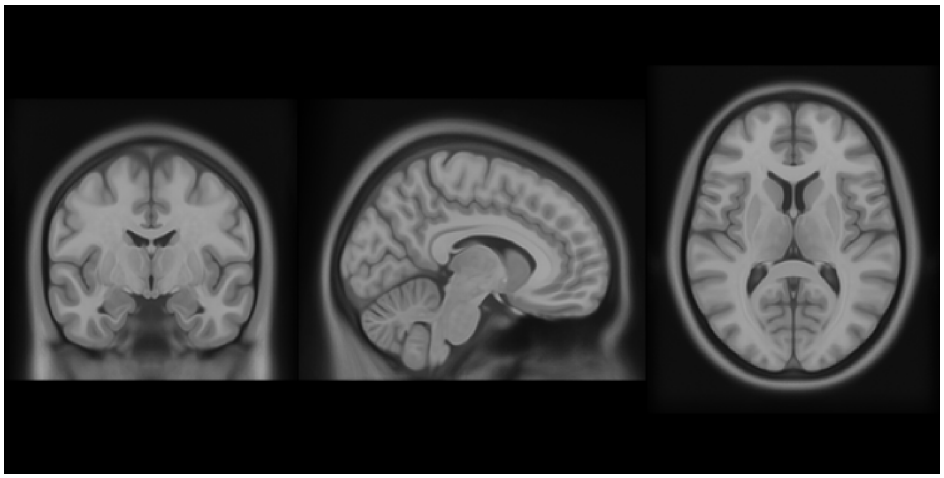
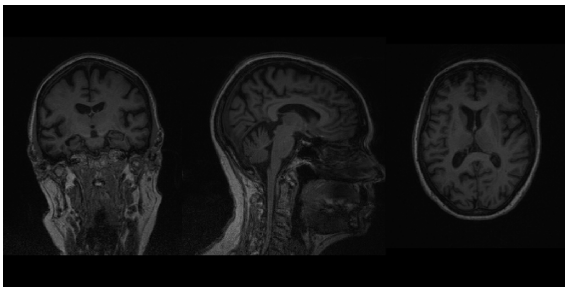
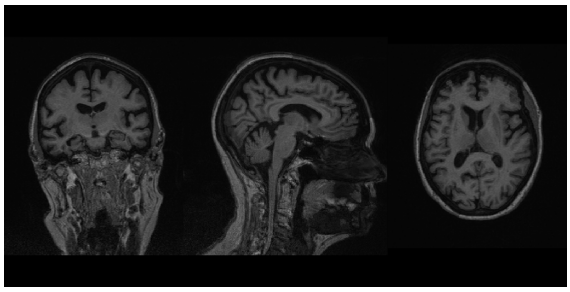


Figura 2.29: Template ICBM-MNI 152.

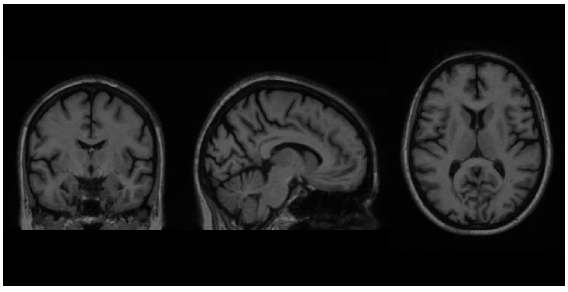
Veamos la salida de los pasos de esta etapa para el sujeto con Alzheimer:



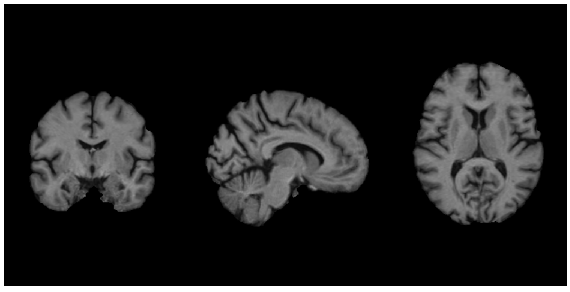
(a) Remuestreo



(b) Corrección de inhomogeneidades de intensidad



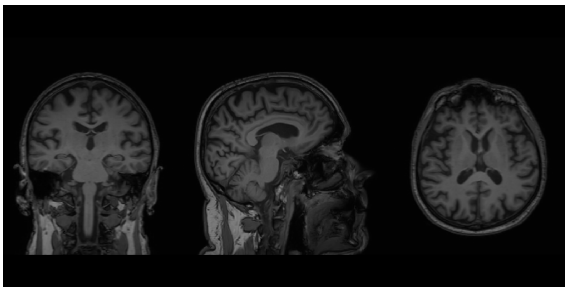
(c) Registración



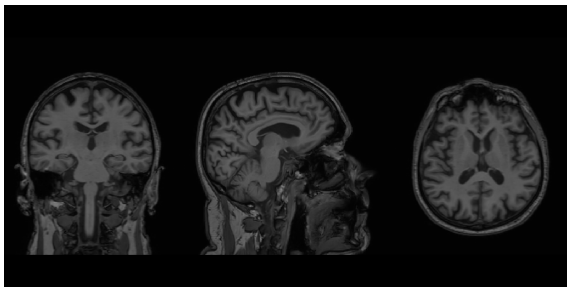
(d) Extracción de cráneo

Figura 2.30: Salida de cada paso de la primera etapa del pipeline de preprocesamiento para un sujeto perteneciente a la clase AD.

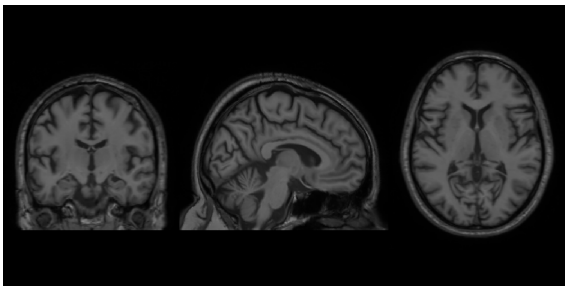
También, veamos la salida de los pasos de esta etapa para el sujeto de control:



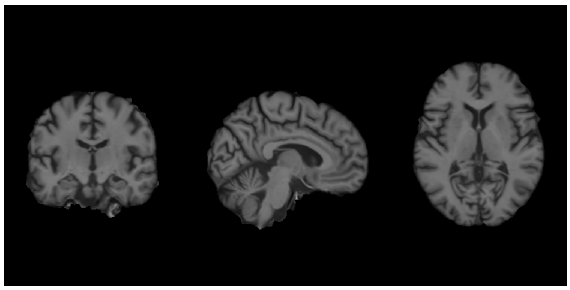
(a) Remuestreo



(b) Corrección de inhomogeneidades de intensidad



(c) Registración



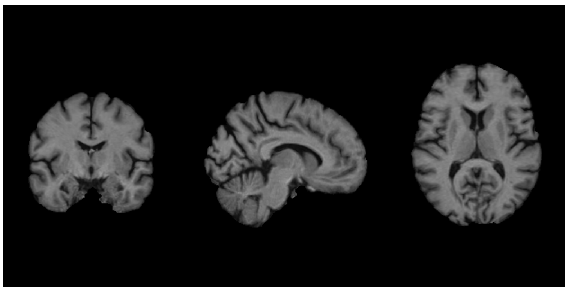
(d) Extracción de cráneo

Figura 2.31: Salida de cada paso de la primera etapa del pipeline de preprocesamiento para un sujeto perteneciente a la clase CN.

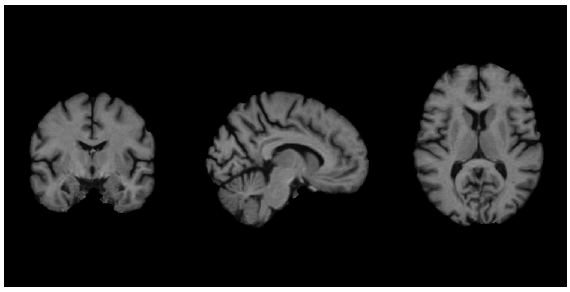
Como se puede ver en las figuras 2.30a y 2.31a, el paso de remuestreo no tiene repercusiones visuales apreciables, sino que genera que las imágenes tengan dimensiones estándar y que cada vóxel represente el mismo volumen en el espacio. En cambio, el paso de corrección de inhomogeneidades de intensidad tiene efectos visibles que son más apreciables en 2.30b, aunque también, en menor medida en 2.31b. Luego, se observa que la registración es el paso que produce los cambios más notorios en las imágenes hasta el momento, como es visible en las figuras 2.30c y 2.31c. El método SyN traslada, rota, escala y deforma las imágenes para alinearlas con el template pero manteniendo las características particulares de las mismas. Luego de ser registradas, se procede a la extracción del cerebro utilizando BET la cual resulta en la eliminación de las estructuras que no son relevantes para el análisis. Aunque, como es visible en la figura 2.31d, el método no siempre logra una extracción completamente limpia. Para lograr un resultado más preciso sería necesario configurar los parámetros discutidos anteriormente de manera particular para cada caso. Sin embargo, se optó por lograr una extracción satisfactoria en general, y tolerar pequeñas imperfecciones como las que se pueden ver, para no perder el grado de automatización logrado.

Segunda etapa

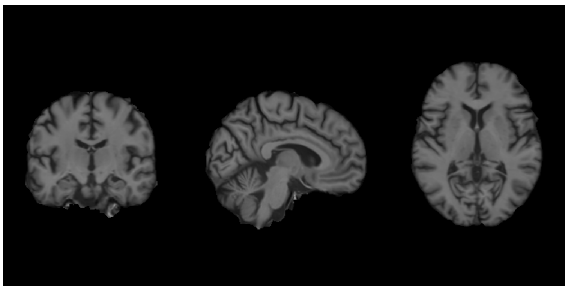
La segunda etapa del pipeline es la normalización de intensidades para la cual se utilizó el método de Nyul y Udupa que transforma los histogramas de las resonancias a un histograma estándar aprendido.



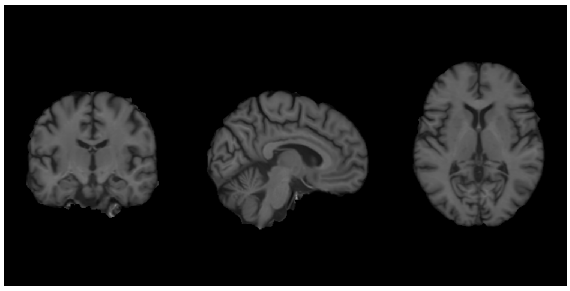
(a) AD sin normalización de intensidades



(b) AD luego de la normalización de intensidades



(c) CN sin normalización de intensidades



(d) CN luego de la normalización de intensidades

Figura 2.32: Imágenes del sujeto perteneciente a la clase AD y del sujeto perteneciente a la clase CN antes y después de la normalización de intensidades.

Debido a la naturaleza del mapeo empleado por el método de normalización que se utilizó, los cambios realizados no son apreciables a nivel de visualización de las imágenes, sino que es más ilustrativo verlos a partir de los histogramas de intensidad de las resonancias.

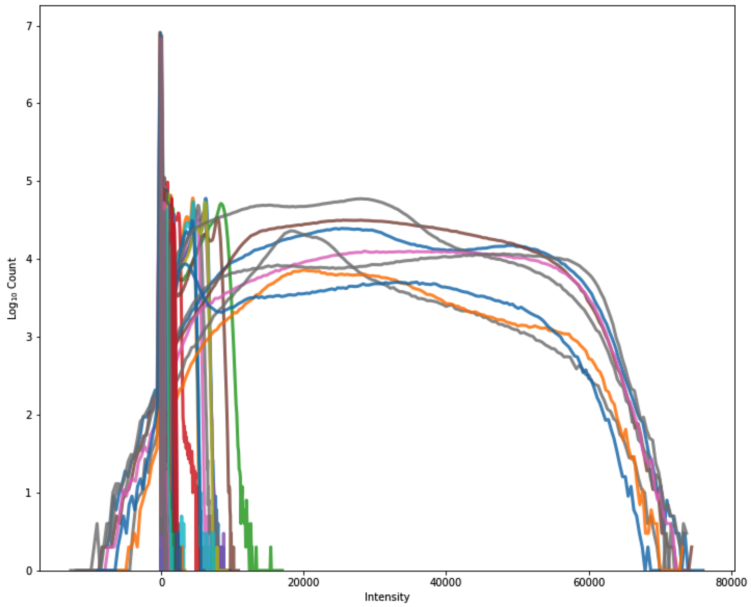


Figura 2.33: Histogramas de intensidad de todas las MRI antes de aplicar el método de normalización.

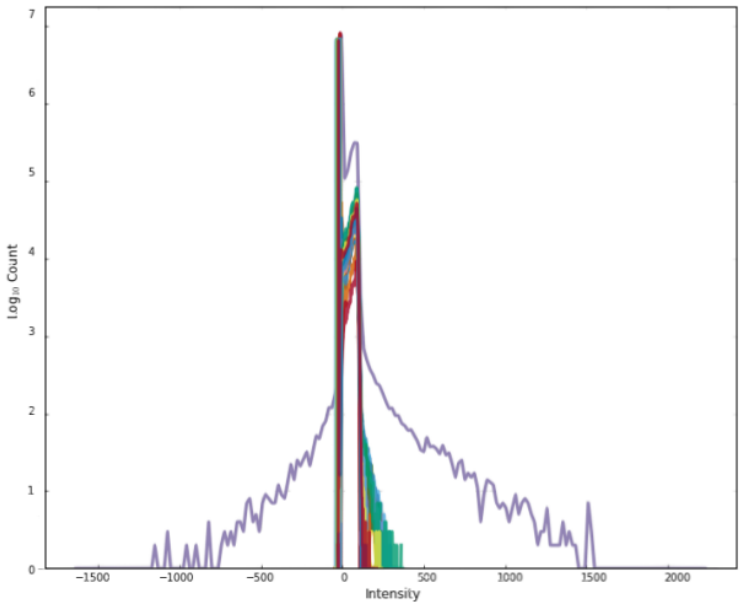


Figura 2.34: Histogramas de intensidad de todas las MRI luego de aplicar el método de normalización. Se ve que hay un histograma que no fue correctamente normalizado. Se revisará en el control de calidad.

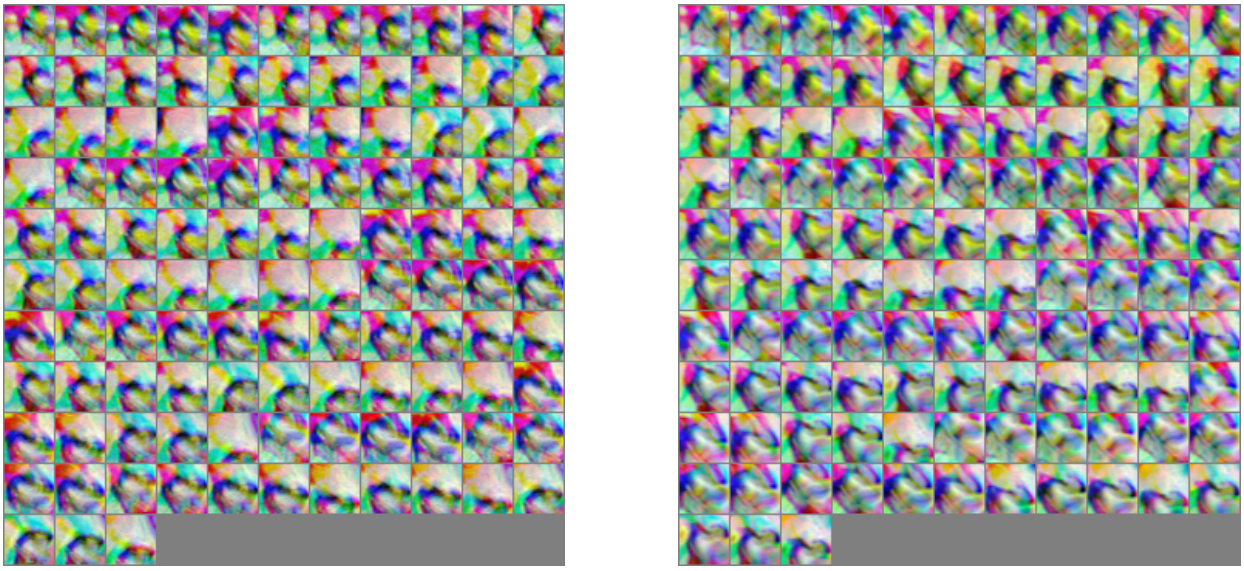
Antes de aplicar el método de normalización de intensidades, los histogramas de las resonancias magnéticas tienen distribuciones de intensidad muy diferentes entre sí, como se puede ver en la figura 2.33. Sin embargo, luego de aplicar el método, los histogramas se deforman para tener distribuciones similares, como se muestra en la figura 2.34, salvo por una imagen (figura 2.35) que muestra tener un histograma distinto.

Figura 2.35: Resonancia magnética que fue preprocesada de manera incorrecta antes de ser preprocesada. La estructura cerebral se ve de color rojo debido a que en escala de grises no es posible distinguirla. Se puede notar que la resonancia tuvo errores en su adquisición que resultaron en la imagen corrupta, la cual no puede ser preprocesada

Tercera etapa

En la tercera, y final, etapa del pipeline se extraen los patches 2.5D del hipocampo sujeto a las restricciones descritas anteriormente. Como ya se dijo, se extraen 113 patches por paciente. Cada patch es una imágen RGB de $32 \times 32 \times 3$ con valores de punto flotante

de 32 bits en el rango $[0, 1]$. En la figura 2.36 podemos ver un mosaico con los 113 patches de los sujetos que fueron tomados como ejemplo en todos los pasos ya vistos:



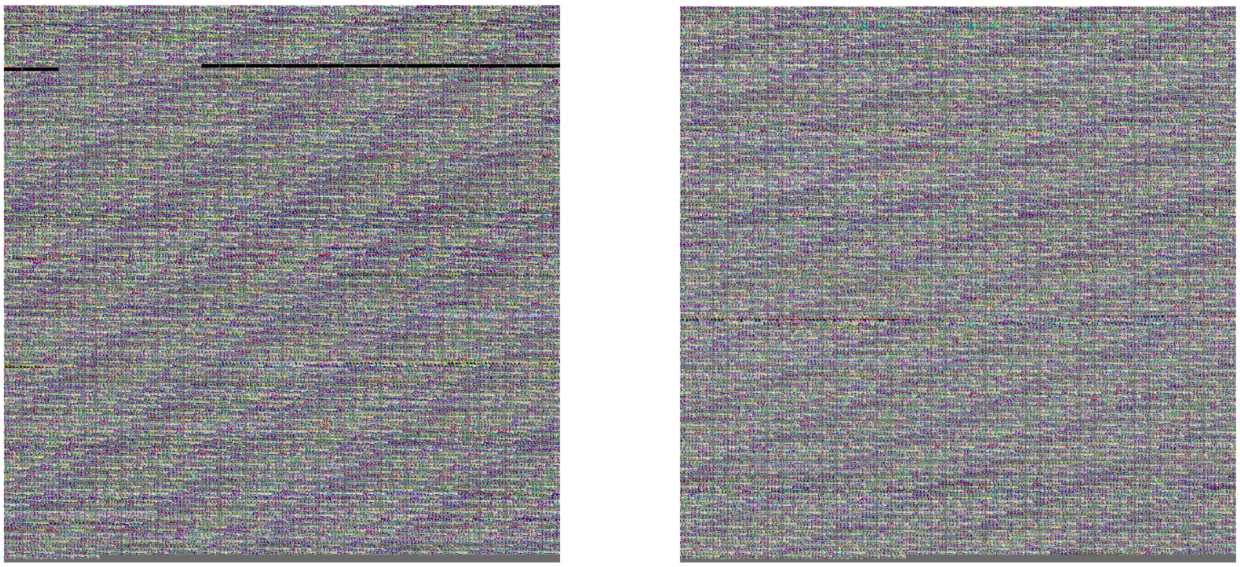
(a) Sujeto del grupo AD (b) Sujeto del grupo CN

Figura 2.36: Patches 2.5D extraídos de un paciente del grupo con Alzheimer y otro control saludable.

Es posible distinguir, si se comparan uno a uno los patches, que los del sujeto perteneciente al grupo con Alzheimer tienen diferencias de colores con respecto a la del sujeto con cognición normal. Estas diferencias de colores son compatibles con las vistas en [24] y en la figura 2.25.

Control de calidad

Se realizó un control de calidad visual de los patches obtenidos para asegurar que el preprocesamiento resultó en imágenes correctas y utilizables, y para detectar los resultados de la imagen que no fue correctamente preprocesada. Para eso, se observan todos los patches de los sujetos pertenecientes a la clase AD y los pertenecientes a la clase CN:



(a) Patches de los sujetos del grupo AD

(b) Patches de los sujetos del grupo CN

Figura 2.37: Patches 2.5D de todos los pacientes del grupo con Alzheimer y otro control saludable.

Podemos ver que en los patches de la clase AD hay una cantidad considerable de imágenes que se encuentran completamente negras. Luego de realizar una inspección se encontró que son 113 imágenes provenientes de un sujeto, cuya resonancia no fue preprocesada bien debido a que presentaba fallas previas a ser ingresada en el pipeline (figura 2.35). Por lo tanto, se eliminó dicha resonancia y de esa forma nos quedan 406 sujetos, de los cuales se generan un total de 45878 patches, los cuales podemos ver en la siguiente figura:

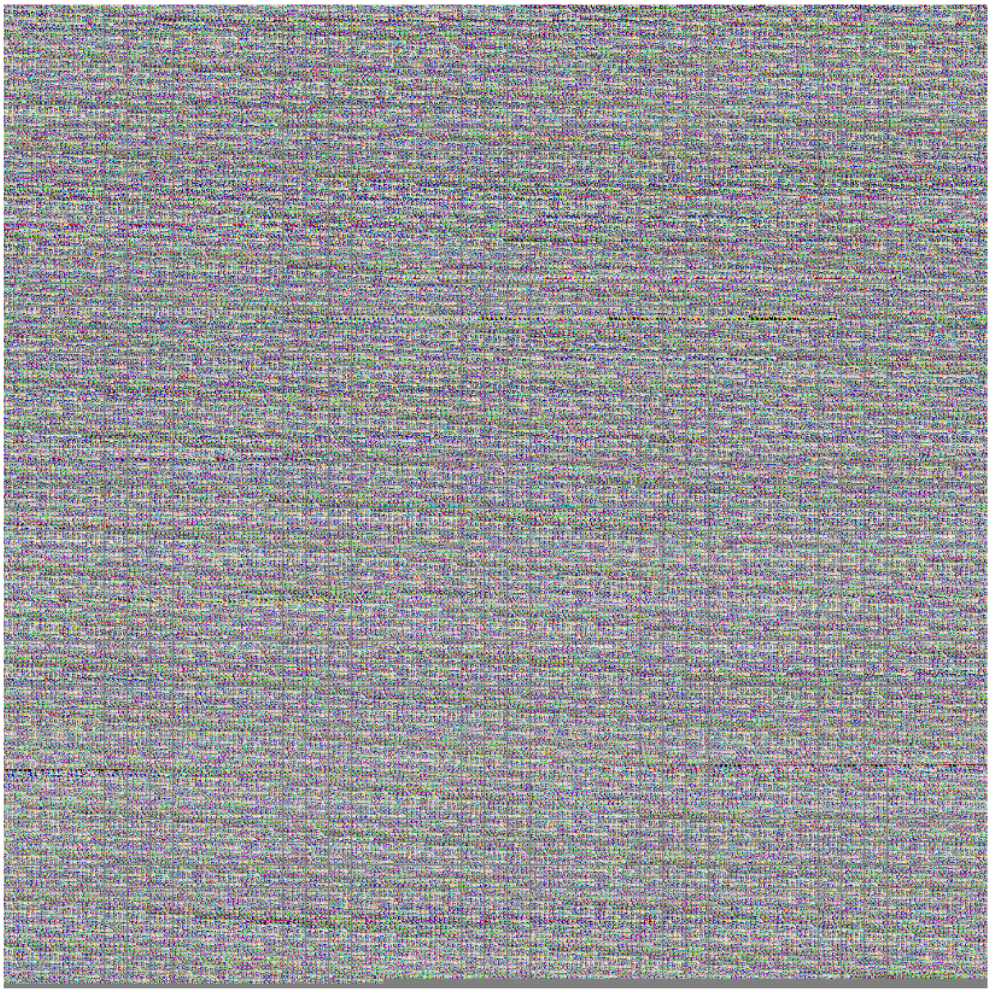


Figura 2.38: Patches 2.5D de todos los pacientes disponibles. Se tiene un total de 406 pacientes, de los cuales 202 pertenecen a la clase AD y 204 a la clase CN; se tiene un total de 45878 imágenes RGB.

y por último, veamos que los histogramas normalizados de todas las resonancias quedaron todos con distribuciones similares.

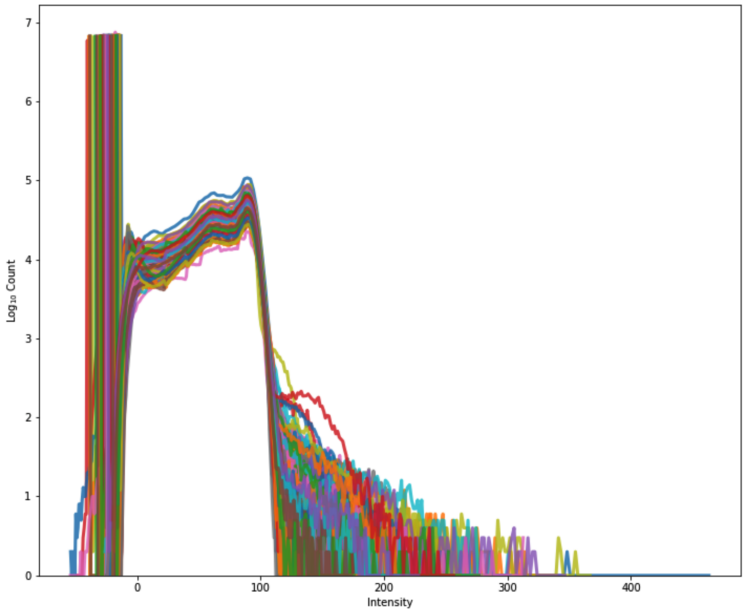


Figura 2.39: Histogramas de intensidad de todas las MRI luego del control de calidad.

2.5. Análisis

Luego de realizar el preprocesamiento de las imágenes y la extracción de los patches 2.5D, se procede a definir un modelo que permita el análisis de los datos. Debido al auge en su adopción en varias áreas de investigación, incluyendo las ciencias médicas, se optó por utilizar un modelo de análisis basado en redes neuronales. Tomando como inspiración el modelo descrito en [24] para el análisis de los patches 2.5D en un problema similar al tratado aquí, se define y se entrena una *Red Neuronal Convolucional* (CNN - Convolutional Neural Network) para clasificar si un sujeto pertenece al grupo AD o CN.

2.5.1. Red Neuronal Convolucional

Las redes neuronales convolucionales son un tipo de modelos de deep learning que están diseñados para el procesamiento de datos matriciales, o en forma de grilla numérica, como lo son las imágenes, y a partir de dichas imágenes aprenden de manera automática y adaptativa las jerarquías espaciales de las representaciones de características en dichas imágenes. En otras palabras, las redes neuronales convolucionales son construcciones matemáticas típicamente compuestas por tres tipos de capas (o bloques fundamentales): capas de convolución, capas de pooling y capas totalmente conectadas (FC - Fully Connected) (Figura 2.40). Las capas de convolución y las de pooling se encargan de la extracción de características, empleando las transformaciones mencionadas anteriormente a los datos crudos, mientras que la capa FC mapea las características extraídas a la salida final, como lo es la clasificación.

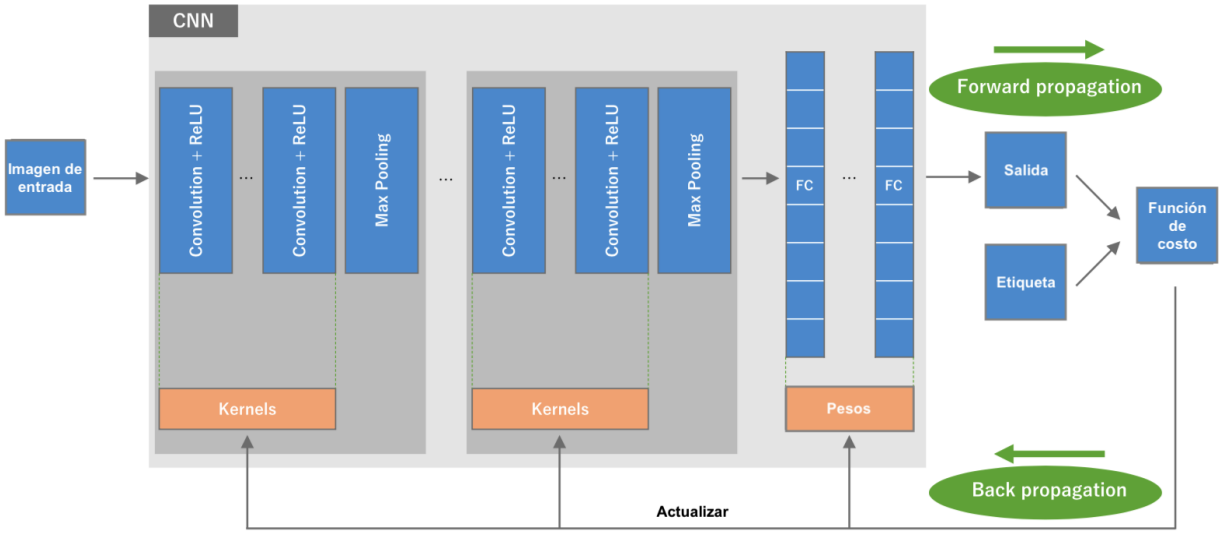


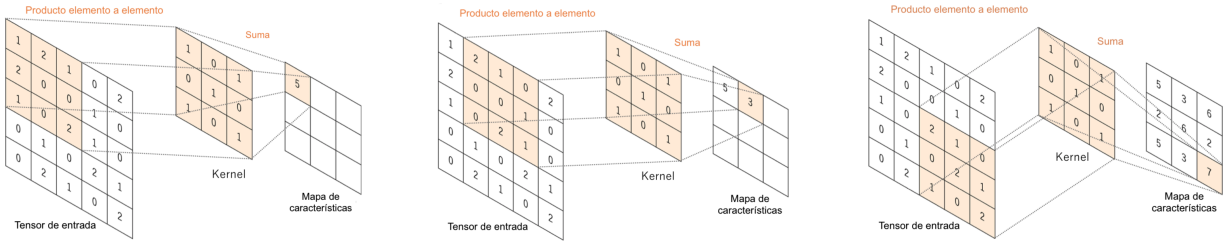
Figura 2.40: Visualización general de una CNN y su proceso de entrenamiento. [13]

En cuanto al entrenamiento de la CNN, como ya fue mencionado anteriormente, se obtiene la salida del modelo a partir de una imagen (o tensor) de entrada, se la compara con un valor, llamado etiqueta o valor verdadero, y se computa la función de costo. Estos pasos componen el "Forward pass" o "Forward Propagation". Para optimizar los parámetros de la CNN, que para la capa convolucional son llamados kernels y para la capa FC son llamados pesos, se utiliza el algoritmo de **Back Propagation** el cual utiliza la regla de la cadena de la derivación para calcular el gradiente de la función de costo con respecto de los parámetros del conjunto de capas que componen la red. Para poder decidir cómo modificar los parámetros en el sentido que minimiza la función de costo se utilizan *optimizadores*, los cuales son algoritmos que dada la función de costo y su gradiente deciden qué cambios se les deben hacer a los parámetros de la red para reducir el error. Algunos de los optimizadores más conocidos y utilizados son SGD (Stochastic Gradient Descent), Adagrad, RMSProp y Adam. Un hiperparámetro muy importante a la hora de definir el optimizador que se utilizará es el "learning rate" o taza de aprendizaje, que suele ser un número muy pequeño (como 0.001) y se utiliza para multiplicarlo al gradiente de la función de costo para controlar la magnitud del cambio que se hará a los pesos. Un cambio muy rápido o grande puede dificultar la convergencia del método y en consecuencia la optimización de la función de costo. Además del learning rate, se pueden especificar otros hiperparámetros como "weight decay", el cual es el valor por el cual se multiplica la suma del cuadrado de los parámetros entrenables cuando se agregan a la función de costo para sumar penalización de la complejidad del modelo; o "momentum", que ayuda acelerar al optimizador para que vaya a en la dirección adecuada y no se quede oscilando alrededor de un valle de la función de costo. Como regla general, se recomienda empezar con Adam como optimizador por defecto ya que suele funcionar mejor que varios algoritmos [49].

Capa Convolucional

La capa convolucional es una de las capas que se encarga de realizar la extracción de características en la CNN. Está compuesta por kernels (filtros cuyos parámetros son entrenables), los cuales son matrices con cierta dimensión que se utilizan para realizar una convolución sobre la entrada. En la convolución se realiza un producto elemento a elemento entre cada elemento del kernel y el tensor de entrada, y se lo suma para obtener el resultado en cada posición del tensor de salida, el cual es denominado mapa de características. Esta operación se repite aplicando múltiples kernels para obtener un número de mapas de características, que representan diferentes características de los tensores de entrada. La convolución está definida por dos hiperparámetros fundamentales, que son el tamaño (o dimensión) del kernel, y la cantidad de kernels. Estas variables son hiperparámetros porque la CNN no los aprende al ser entrenada, sino que deben ser especificados previos al entrenamiento. La dimensión del kernel dará un sentido de qué características serán extraídas de los tensores, por lo que un kernel de dimensión más pequeña obtendrá características más pequeñas contenidas en la imagen de entrada, y por el contrario un kernel de mayor dimensión extraerá características más globales de la imagen; los valores típicos que se utilizan son 3×3, 5×5 o 7×7.

Un problema con la convolución descrita es que el centro del kernel no pasa por todos los elementos del tensor de entrada, sino que los elementos del borde del tensor quedan afuera y el tensor de salida es un tensor de menor dimensión que el de entrada (Figura 2.41). Para solucionar esto se utiliza la técnica de Padding, tipicamente Zero-Padding, en el que se agregan filas y columnas de ceros en los bordes del tensor de entrada, logrando que de esa manera el kernel pueda ubicar su centro en los elementos del borde del tensor y así lograr un mapa de características de las mismas dimensiones que el tensor de entrada (Figura 2.42). El uso del zero-padding es una práctica común en los modelos de CNN para poder utilizar múltiples capas, ya que sin Zero-Padding, cada mapa de características sucesivo se haría más pequeño luego de la convolución.



(a) (b) (c)

Figura 2.41: Ejemplo del proceso de la convolución [13]

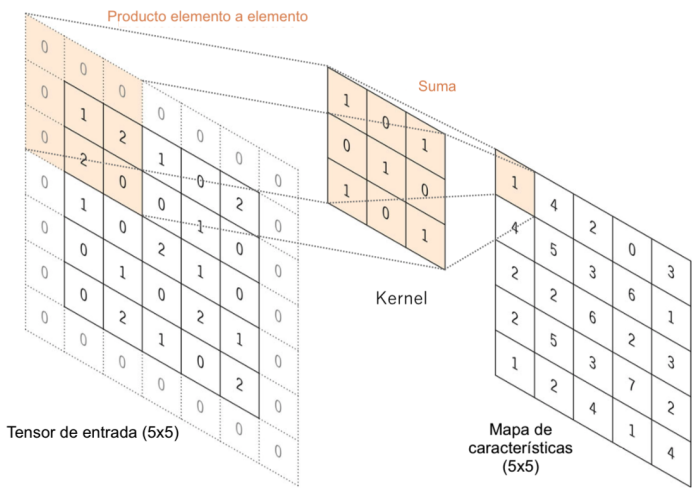


Figura 2.42: Ejemplo de convolución con Zero-Padding [13]

La distancia entre dos posiciones sucesivas del kernel se llama *paso*, y es un hiperparámetro que también define a la convolución. Se suele usar un paso de 1, aunque se podrían usar pasos más grandes para lograr un sub-muestreo (downsampling) del mapa de características. Sin embargo, como se utiliza la capa de pooling para realizar dicho sub-muestreo, la elección general del paso es que sea igual a 1.

Función de activación no lineal

La salida de la capa convolucional, la cual utiliza la operación de convolución (que es lineal), pasa a través de una función de activación no lineal. De esa manera, la CNN puede introducir no linealidad en la salida de cada capa convolucional y generar representaciones más complejas. Existen varias funciones de activación (escalón, lineal, tanh, sigmoidea, etc.) (Figura 2.43) que son utilizadas en diferentes aplicaciones y modelos.

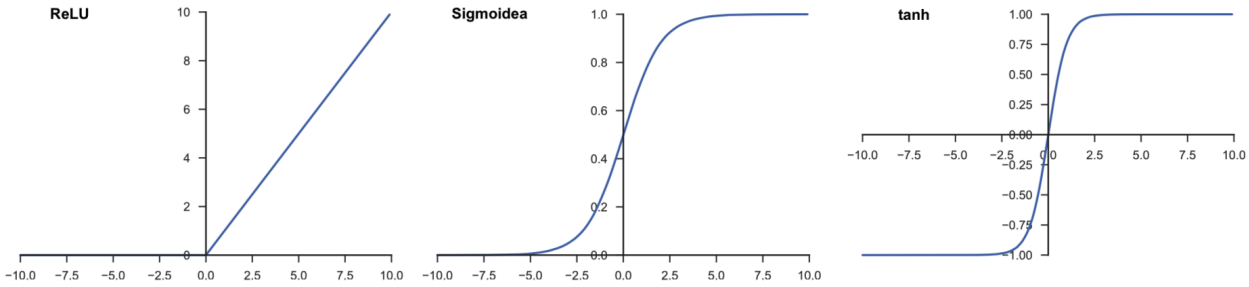


Figura 2.43: Ejemplos de funciones de activación. [13]

La elección de la función de activación a la salida de las capas convolucionales es otro hiperparámetro que se debe elegir previo al entrenamiento del modelo. Sin embargo, la función de activación más utilizada en CNN para activar la salida de las capas convolucionales es la función ReLU (Rectified Linear Unit) que está definida por:

$$f(x) = \begin{cases} max(0, x) & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}$$

En otras palabras, la función trunca a 0 los valores negativos de su entrada, y deja como están los valores mayores a 0. El uso de ReLU fue mayormente adoptado en CNN debido a que acelera los tiempos de entrenamiento de la red al generar tensores de salida esparsos, los cuales son más eficientes para ser manejados computacionalmente.

Capa de pooling

La capa de pooling realiza la operación de sub-muestreo que reduce la dimensión del mapa de características obtenido a la salida de la capa convolucional para introducir una traducción que sea invariante a pequeños cambios y distorsiones, y para reducir la cantidad de parámetros entrenables. En esta capa no hay parámetros entrenables, sino que sólo tiene hiperparámetros tales como el tamaño del filtro, el paso y el padding.

La forma más popular de pooling es el *max pooling*, que extrae parches del mapa de características y devuelve el valor máximo de cada parche descartando el resto (Figura 2.44). Esto genera un sub-muestreo del mapa de características por un factor de 2, es decir que reduce las dimensiones de altura y ancho a la mitad, pero la profundidad no cambia lo cual es útil en imágenes RGB o volúmenes.

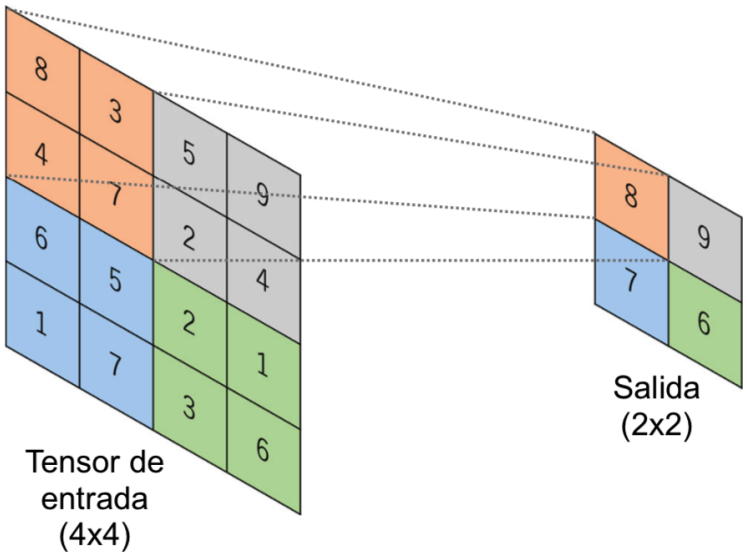


Figura 2.44: Ejemplo de max pooling con un filtro de 2×2 , sin padding y paso de 2. [13]

Otra forma de pooling es el *global average pooling*. Este tipo de pooling lleva a cabo un sub-muestreo extremo, en el que un mapa de características de tamaño alto×ancho es reducido a un arreglo de 1×1, mediante el cálculo del promedio de todos los elementos de mapa de características manteniendo la profundidad. Esta forma de pooling se suele aplicar sólo una vez antes de las capas FC.

Capa Fully Connected

El mapa de características saliente de la última convolución se suele aplanar para transformarlo en un arreglo unidimensional de números, un vector, y se lo conecta con una o más capas Fully Connected, también conocidas como capas densas, en las que cada entrada se conecta con cada salida mediante un peso (parámetro entrenable). Las capas FC mapean la salida del conjunto de capas convolucionales y de pooling a las salidas finales de la red, como lo son las probabilidades de cada clase en un problema de clasificación. La última capa FC suele tener como número de nodos la cantidad de clases. Cada capa FC es seguida por una función de activación no lineal, como la ReLU. Los parámetros entrenables son los pesos, y como hiperparámetros se tienen la cantidad de pesos y la función de activación.

La función de activación aplicada en la última capa FC suele ser diferente a la demás, ya que será la función de activación que devuelva la salida de la red. La función debe ser apropiadamente elegida según la tarea. Para el caso de clasificación binaria, como en este trabajo, se utiliza la función sigmoidea también conocida como función logística, la cual está definida como:

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

la cual transforma cada valor aplicado al rango $(0, 1)$.

2.5.2. Armado del dataset

El primer paso en el proceso de análisis es el armado del dataset que será utilizado para entrenar y probar el modelo propuesto. Para eso hay que definir concretamente la tarea que se desea que el modelo realice con los datos. El problema que fue planteado en [24], utilizado como inspiración para esta parte del trabajo, era el de clasificación individual de patches 2.5D en la clase AD o CN. Debido a que el objetivo final de este trabajo es la clasificación de sujetos, no sólo de un patch, se tendrá en cuenta qué sujetos fueron incluídos en cada partición del dataset para los desarrollos de las siguientes secciones. Sin embargo, en este caso la meta es implementar un modelo que logre clasificar correctamente si un patch individual pertenece a la clase AD o CN.

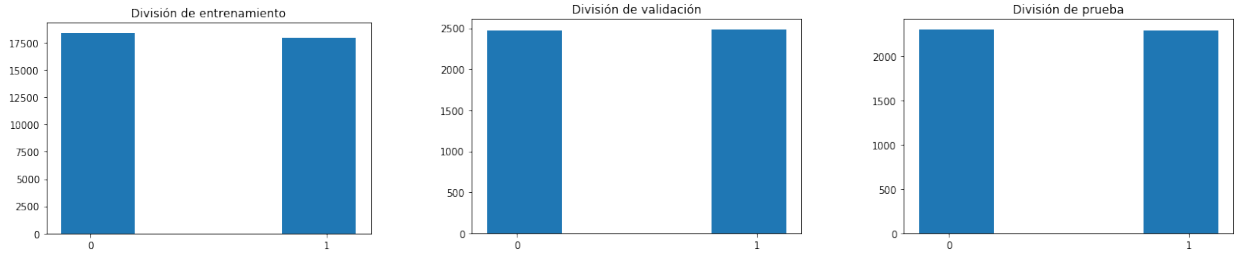
Es importante tener en cuenta en qué forma se guardará la información utilizada por el modelo (en este caso imágenes) y cómo será procesada. En este trabajo, se utilizó Google Colab para la implementación de los modelos en esta sección y en las siguientes debido a que ofrece el uso en la nube de una unidad de procesamiento de gráficos (GPU) de manera gratuita, lo cual es ventajoso para entrenar modelos basados en redes neuronales ya que acelera dicho proceso de entrenamiento. Sin embargo, para usar Google Colab se suele necesitar la conexión con las carpetas de Google Drive donde se encuentran presentes los archivos que serán utilizados. Esto se logra "montando" la carpeta de Google Drive en Colab, lo cual tiene asociado un tiempo límite o "timeout" que al ser superado interrumpe la operación de montado de la carpeta. El timeout suele ser superado en casos donde la carpeta de Google Drive que se necesita montar contiene una gran cantidad de archivos, aunque la documentación de Google Colab no especifica cuántos. Como ya vimos, en esta etapa del trabajo se disponen de 406 resonancias magnéticas, de las cuales se obtienen 113 patches por

resonancia, generando un total de 45878 imágenes RGB que deben ser guardadas en Google Drive. Debido a que cada patch es una imagen $32 \times 32 \times 3$ con valores de punto flotante de 32 bits en el rango $[0, 1]$, no es posible guardarlas como imágenes .jpg o .png, sino que es necesario guardarlas de otra manera. En principio se decidió guardar cada imagen RGB en un archivo de extensión .npy, propia de la librería Numpy de Python que permite guardar arreglos numéricos multidimensionales, llamados Numpy Arrays o *ndarrays*, de manera que pueda ser posible cargar posteriormente el arreglo. Al guardar las imágenes en este formato se generaron 45878 archivos .npy que fueron guardados en Google Drive para luego poder ser utilizados en Colab. Sin embargo, al intentar montar la carpeta de Google Drive el proceso falló debido a que se excedió el timeout mencionado anteriormente. Por lo tanto, fue necesario guardar la información de las imágenes de otra forma.

La forma que se adoptó para guardar la información fue el uso de archivos HDF5, que tienen extensión .h5. Un archivo HDF5 es un contenedor de información para dos tipos de objetos: *datasets*, que son colecciones de datos en forma de arreglos; y *grupos*, que son contenedores del estilo de carpetas que sirven para guardar datasets y otros grupos. Al ser contenedores de información, los archivos HDF5 permiten guardar de manera estructurada y comprimida grandes volúmenes de información. Los grupos funcionan como si fueran diccionarios y los datasets son similares a los ndarrays. De esta manera, se pueden guardar todas las imágenes preprocesadas dentro de un mismo archivo, sin perder la precisión original con la que fueron generadas y sin la necesidad de tener un archivo por imágen, mitigando de esta manera el problema del timeout de Google Colab al montar la carpeta de Google Drive.

Para aprovechar la capacidad de almacenamiento de los datos en los mencionados datasets, se decidió dividir las imágenes en tres grupos importantes para el entrenamiento, monitoreo y prueba del modelo: el conjunto de entrenamiento o *training set*, conjunto de validación o *validation set*, y conjunto de prueba o *test set*.

Debido a que se dispone de las resonancias magnéticas de 406 sujetos, de los cuales 202 pertenecen a la clase AD y 204 a la clase CN, vemos que la proporción de clases no muestra un desbalance de clases que pueda llegar a sesgar el análisis teniendo 49.63 % de los sujetos de la clase AD y 50.36 % de la clase CN. Como ya se vio, por cada sujeto se generan 113 patches RGB lo cual significa que se disponen de 45878 imágenes en total. Se dividió dichas imágenes en 36335 (AD:17941, CN:18394) para el entrenamiento, 4955 (AD:2487, CN:2468) para validación y 4588 (AD:2285, CN:2303) para prueba, de forma tal que se utilizó 80 % para entrenar, 10 % para el conjunto de validación y otro 10 % para el conjunto de prueba. Esta división se hizo teniendo en cuenta que el dato principal que se va a utilizar para esta etapa del trabajo es el patch 2.5D y no el conjunto de datos que corresponden a un sujeto. Cabe aclarar que no se utilizaron datos de los sujetos como su edad o género a la hora de realizar la división en los conjuntos mencionados.



(a) Set de entrenamiento (b) Set de validación (c) Set de prueba

Figura 2.45: División de sujetos en los conjuntos de entrenamiento, validación y prueba.

Por último, es importante mencionar que debido a que el modelo utilizado en esta sección forma parte del modelo final que se presenta en este trabajo, la división de datos para el modelo final deberá contener los mismos sujetos utilizados para entrenar, validar y probar el modelo de esta sección. Por eso, a partir de ahora y en las siguientes secciones se incluirán los mismos sujetos, según la disponibilidad de datos lo permita, para la división en los conjuntos.

2.5.3. Arquitectura del modelo

En esta sección, la arquitectura que se propone para el análisis de los patches 2.5D fue basada en la estructura propuesta por [24]:

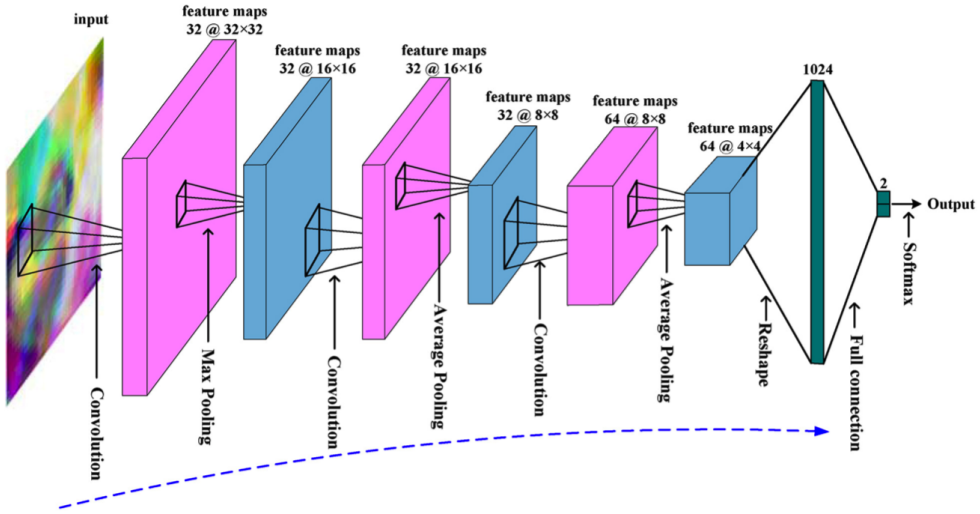
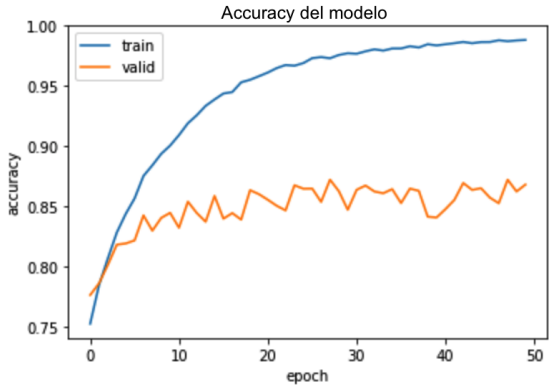


Figura 2.46: Arquitectura propuesta en [24], y usada como inspiración inicial para este trabajo

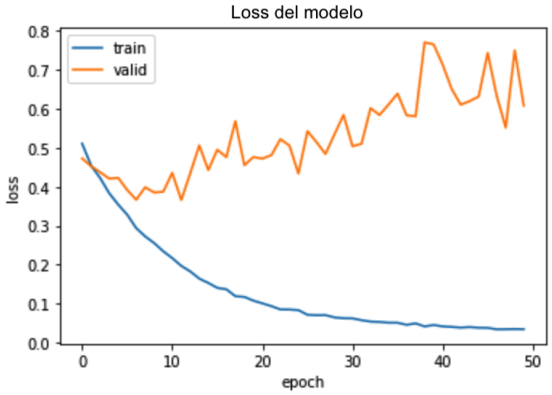
La arquitectura que los autores en [24] utiliza como imagen de entrada los patches 2.5D de 32×32×3; tiene tres capas convolucionales y de max pooling. El tamaño del kernel de la capa convolucional es de 5×5 con padding de 2 píxels, y el filtro de la capa de max pooling es de tamaño 3×3 y tiene un paso de 2. Luego, la salida de las capas convolucionales y de pooling son pasadas por una capa FC de 1024 nodos y por una última capa de activación.

Para implementar el modelo se utilizó el paquete Keras utilizando como framework de backend Tensorflow 1.0, y se utilizó, como ya se dijo, el servicio de Google Colab para correr el entrenamiento y hacer las pruebas haciendo uso de la posibilidad de alocar una GPU de manera gratuita.

Para el entrenamiento se utilizó el algoritmo SGD con un learning rate de 0.001, weight decay de 0.0001 y momentum de 0.9 como optimizador y se entrenó por 30 epochs, siendo cada epoch una pasada por todo el dataset. Utilizando este modelo con los hiperparámetros mencionados los autores obtuvieron una exactitud media de 88,79 % con un desvío estándar de 0,61 % haciendo el promedio de 50 corridas de 10-Fold Cross Validation. Los autores del trabajo señalan que entrenaron su modelo utilizando batches con la cantidad de imágenes RGB obtenidas por sujeto, pero que dichos batches contenían imágenes mezcladas aleatoriamente de todo el dataset. Esta manera de entrenar el modelo, no fue adoptada en esta sección debido a que el objetivo de esta red en este trabajo es el de clasificar correctamente patches 2.5D, y no sujetos, para ser utilizada en el modelo final el cual sí será utilizado para clasificar sujetos. Sin embargo, se toman los resultados obtenidos en [24] como punto de partida para medir el desempeño del modelo y compararlo con algún resultado registrado en una publicación previa. Por lo tanto, como primer paso se intentó replicar los resultados obtenidos por los autores implementando el modelo como fue propuesto (con algunas variaciones por las cuestiones ya explicadas) por los autores, y a partir de ahí intentar realizar los ajustes necesarios para, de ser posible, mejorar el rendimiento del mismo. Al entrenar durante 50 epochs, en lugar de 30, y con batches de 16 imágenes, se observaron las siguientes curvas de entrenamiento:



(a) Accuracy de clasificación del modelo en el training set y el validation set



(b) Loss o error de clasificación del modelo en el training set y el validation set

Figura 2.47: Curvas de entrenamiento del modelo propuesto en [24] utilizando los datos de este trabajo

A partir las curvas en la figura 2.47 se puede ver que la accuracy para los datos de entrenamiento aumenta, y al mismo tiempo el error para dichos datos disminuye. Pero, por otro lado, el accuracy del modelo con los datos de validación y el error no se comportan de la misma manera, sino que a partir de cierto punto dejan de aumentar y disminuir, respectivamente. Este comportamiento es característico del fenómeno de "Overfitting", en el

cual el modelo clasifica muy bien los datos de entrenamiento pero no tiene capacidad para generalizar sobre datos nunca antes vistos.

Mejoras propuestas

Para mitigar el overfitting detectado al entrenar el modelo propuesto se utilizó Batch Normalization luego de cada capa convolucional, y Dropout del 50 % en la capa FC. También, la capa FC se modificó para disminuir el número de parámetros entrenables del modelo, definiéndola con 64 nodos en lugar de 1024. Además, el optimizador se cambió por Adam con los parámetros default de Keras. Dichas modificaciones pueden verse reflejadas en el diagrama de la siguiente figura:

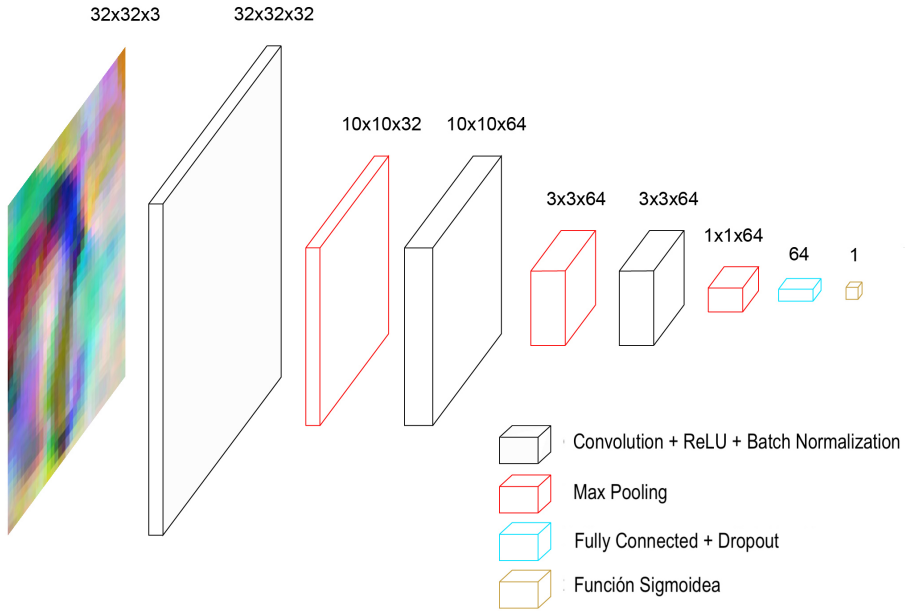
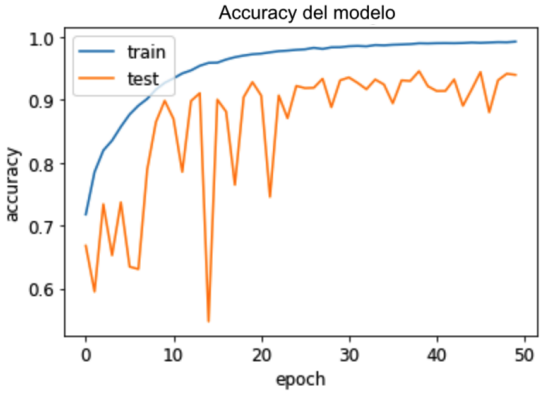
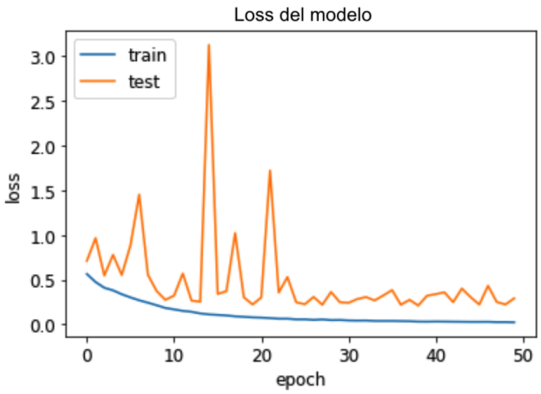


Figura 2.48: Arquitectura con las mejoras propuestas.

Al entrenar esta nueva CNN durante 50 epochs, utilizando batches de 16 imágenes, se obtienen las siguientes curvas de accuracy y loss:



(a) Accuracy de clasificación del modelo en el training set y el validation set



(b) Loss o error de clasificación del modelo en el training set y el validation set

Figura 2.49: Curvas de entrenamiento del modelo con las mejoras propuestas

A partir de estas curvas se puede ver que el overfitting detectado en el modelo propuesto en [24] se ha mitigado con los cambios introducidos.

2.5.4. Resultados

Se procede a mostrar las métricas calculadas utilizando los resultados de clasificación obtenidos a partir del test set como primera instancia de evaluación del modelo. La matriz de confusión calculada es:

$$\begin{bmatrix} 2073 & 230 \\ 91 & 2194 \end{bmatrix}$$

Luego, podemos mostrar las métricas obtenidas a partir de los valores de la matriz de confusión:

	Exactitud	**Sensibilidad**	**Especificidad**	**VPP**	**VPN**	**F1-Score**	**ROC-AUC**
CNN propuesta	93 %	96.02 %	90.01 %	90.51 %	95.79 %	93.18 %	93.01 %

Tabla 2.5: Métricas obtenidas con la CNN al ser probada con el test set.

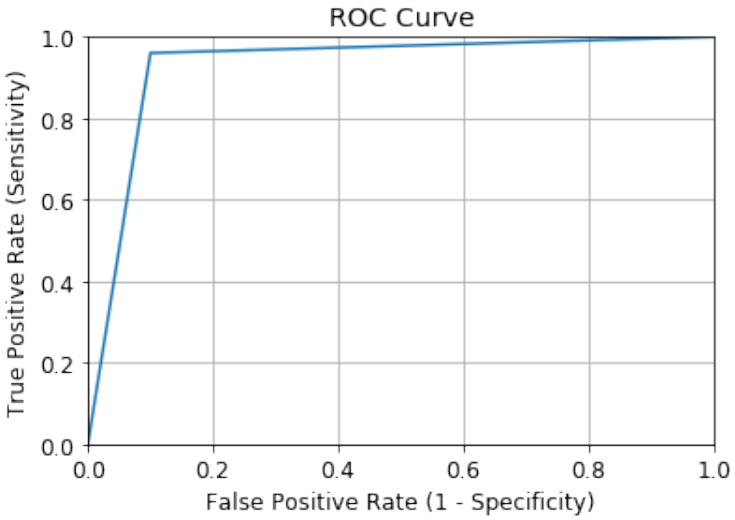


Figura 2.50: Curva ROC obtenida.

Vemos que las métricas obtenidas indican que el modelo en un principio presenta buena capacidad para distinguir entre pacientes enfermos y sanos a partir de los patches 2.5D, aunque con una pequeña tendencia a clasificar erróneamente a pacientes que no tienen la enfermedad como si la tuvieran, es decir que tiene una leve tendencia a la detección de falsos positivos. Un motivo por el que esto podría suceder es debido a que la atrofia cerebral ocasionada por la edad tiene similaridades anatómicas con la que es ocasionada por la enfermedad de Alzheimer. Una manera de evitar esto sería aplicar una técnica de corrección de las resonancias por edad como la que se aplica en [24]. Sin embargo, como se puede ver en la figura 2.50, el clasificador puede, en una primera instancia, catalogarse como muy bueno basado en las métricas obtenidas sin aplicar este método de corrección por edad.

Para validar los resultados obtenidos y poder comparar de manera correctamente el cambio de desempeño ocasionado por las mejoras propuestas con los resultados de [24], se utilizó 10-Fold Cross Validation. En este caso, se utilizó 10-Fold Cross Validation comparar los resultados obtenidos con los de [24], a pesar de que no se realizaron 50 iteraciones del algoritmo debido a que los recursos disponibles no permiten tal ejecución. Además, la evaluación del modelo con 10-Fold Cross Validation sirve para estimar el impacto del entrenamiento del modelo con distintas divisiones de datos para training y validation, utilizando el conjunto de test para evaluar el desempeño de los 10 modelos entrenados con cada división y obtener un valor promedio de las métricas. De esta manera se busca obtener una noción del desempeño promedio del modelo y qué tanto impactan las diferentes divisiones de los datos en el mismo. Los resultados obtenidos fueron:

	Exactitud	**Sensibilidad**	**Especificidad**	**VPP**	**VPN**	**F1-Score**	**ROC-AUC**
CNN de [24]	88.79 (0.61) %	-	-	-	-	-	-
CNN propuesta 10-Fold Cross Validation	92.8 (1.4) %	91.5 (1.7) %	94.1 (2.3) %	93.8 (2.2) %	92.5 (2.1) %	92.6 (1.9) %	92.8 (1.4) %

Tabla 2.6: Métricas obtenidas al hacer 10-Fold Cross Validation con la CNN al ser probada con el test set y comparación con el resultado de [24]. Los resultados se muestran con el formato "media (desvío estándar)".

Se puede ver que las métricas calculadas con el conjunto de prueba luego del 10-Fold Cross Validation cambiaron levemente con respecto a las calculadas anteriormente (Tabla 2.5). Estas nuevas métricas representan una visión más cercana a la realidad que muestra cómo se desempeña el modelo con las mejoras propuestas. Vemos que los resultados obtenidos con la validación hecha contradicen a los obtenidos anteriormente ya que ahora se puede ver que el modelo, en promedio, muestra mayor tendencia a clasificar correctamente los sujetos sanos que los enfermos, sin perder de vista que se obtuvo una sensibilidad por encima del 90 %. Esta tendencia se ve reflejada en el Valor Predictivo Positivo, cuyo alto valor muestra que el modelo tiende a rechazar los falsos positivos de manera correcta, y da a entender que los sujetos que clasifica como positivos tienen más probabilidad de tener la enfermedad. Tener una alta especificidad y un alto rechazo a los falsos positivos en un clasificador de enfermedades es muy importante, ya que indica que un resultado negativo tiene menor probabilidad de ser en realidad positivo y pasar desapercibido. El valor de F1-Score obtenido fue de 92,6 % en promedio, lo cual indica que el modelo tiene una buena capacidad de rechazo de falsos positivos y falsos negativos, y que por lo tanto los sujetos clasificados como enfermos probablemente lo sean realmente, y las falsas alarmas serán pocas. En cuanto al valor de ROC-AUC se ve que, luego de realizar la 10-Fold Cross Validation, sigue siendo muy cercano a 1 y, por lo tanto, la calidad de las predicciones del modelo para la tarea de clasificación planteada es muy alta.

2.6. Conclusiones

Partiendo de las resonancias magnéticas obtenidas de ADNI se implementó un pipeline de preprocesamiento para ellas, el cual genera su estandarización espacial y de intensidades. Luego de preprocesar las resonancias magnéticas, se implementó un enfoque de aumento de datos que permite generar un conjunto de imágenes 113 veces mayor a la cantidad de resonancias disponibles, manteniendo las características espaciales 3D de las mismas generando un enfoque en una región especialmente afectada por la enfermedad de Alzheimer. Además de la implementación del pipeline de preprocesamiento, se logró optimizar la ejecución del mismo para que pueda realizarse en cuatro procesos paralelos y acelerar los tiempos de manera significativa.

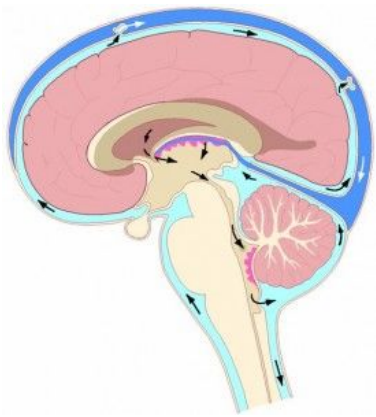
Luego, partiendo del modelo propuesto en [24], que analiza un problema muy similar al tratado en esta sección, se implementaron una serie de cambios que generan una mejora en los resultados publicados en dicho trabajo. El modelo clasificador obtenido en esta sección tiene valores de sensibilidad, especificidad y precisión mayores al 91 %, en promedio y un valor de área debajo de la curva ROC de aproximadamente 93 %. Por lo tanto, la CNN propuesta muestra buena capacidades para detectar y descartar la enfermedad.

Capítulo 3

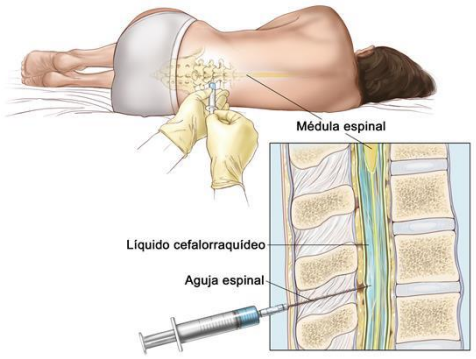
Biomarcadores

3.1. Líquido cefalorraquídeo: Proteínas Amiloide β, Tau y pTau

Los biomarcadores extraídos del líquido cefalorraquídeo (LCR) pueden ser utilizados para complementar y asistir el diagnóstico clínico de ls enfermedad de Alzheimer a partir de una muestra de obtenida con una punción lumbar. Si la punción es hecha correctamente, tiene una baja tasa de complicaciones, un alto rendimiento diagnóstico, y suele ser más tolerable que lo que los pacientes creen. Los biomarcadores que reflejan la patología del AD ya muestran concentraciones anormales en LCR en la etapa prodrómica, permitiendo de esa manera el diagnóstico temprano de la enfermedad, incluso antes del inicio de los síntomas.



(a) Flujo del LCR.



(b) Punción lumbar

Figura 3.1: El LCR fluye desde los ventrículos laterales y penetra en el espacio subaracnoideo, que abarca tanto la corteza como la médula espinal. Su función principal es la de amortiguador de la corteza cerebral y la médula espinal. Otra función del LCR es proporcionar nutrientes al tejido nervioso y eliminar los desechos metabólicos del mismo. La barrera hematoencefálica es una barrera permeable que realiza intercambios en ambas direcciones entre la sangre, LCR y el SNC. El LCR que se recoge en un entorno hospitalario través de una punción suboccipital (directamente por debajo del cráneo) o lumbar (entre la tercera, cuarta y quinta vértebras lumbares).

Los biomarcadores de LCR suelen ser preferidos para reflejar la fisiopatología en el diagnóstico de AD sobre los marcadores bioquímicos presentes en la sangre o en el plasma, ya que el cerebro está en contacto directo con el LCR por un flujo de proteínas bidireccional sin restricciones, y el LCR está recluído del impacto directo del sistema periferal a través del sistema de transporte de moléculas y proteínas por la barrera hematoencefálica. Por eso, el análisis de LCR es importante para la detección de marcadores de AD in vivo. [50].

Se asocia a los biomarcadores de LCR con las tres principales patologías que ocurren en el cerebro en la enfermedad de Alzheimer: Deposición de proteína Amiloide β en placas extracelulares, formación de nudos neurofibrilares intracelulares y degeneración neuronal. El péptido β-amiloide está compuesto por 42 aminoácidos ($A\beta_{1-42}$) y es el resultado de la escisión de la proteína transmembrana precursora de amiloide (APP) por secretasas β y γ. La proteína $A\beta_{1-42}$ es altamente insoluble y se aglomera en depósitos extracelulares sobre el cerebro, lo cual es detectado como una disminución de la concentración de $A\beta_{1-42}$ en el LCR en la patología de AD. La proteína tau es abundante en el citosol de las neuronas, donde su función es la estabilización de los microtúbulos. En AD, un desbalance entre las quinasas y las fosfatasas resulta una hiperfosforilación de la proteína tau, lo cual lleva a la separación de la proteína tau de los microtúbulos y su posterior acumulación en nudos neurofibrilares. Durante el proceso neurodegenerativo, las proteínas tau y tau fosforilada (p-tau) son también liberadas en el espacio extracelular, ocasionando un incremento en su concentración en el LCR en la enfermedad de Alzheimer.

La formación de placas $A\beta$ y nudos neurofibrilares genera lesiones neuronales y la degeneración sináptica presente en AD. Las primeras placas $A\beta$ comienzan a aparecer al menos 10, y probablemente entre 20 y 30 años antes de que comiencen a manifestarse los primeros síntomas, y como tales son detectables en el LCR para el diagnóstico temprano. Los marcadores de la proteína tau en el LCR cambian más tarde en el proceso fisiopatológico comparado con $A\beta_{1-42}$ y la concentración de tau en el LCR se correlaciona más fuertemente con el deterioro cognitivo [50]. Los biomarcadores obtenidos del LCR proporcionan una visión completa de la fisiopatología de la enfermedad de Alzheimer y además, la punción lumbar es altamente accesible y conlleva un costo muy bajo para su realización, en contraste del enfoque para su obtención mediante imágenes, la cual utiliza PET con sustancias de contraste.

La utilidad clínica de los biomarcadores de LCR es de carácter complementario a otras pruebas realizadas por el médico sobre un paciente que es sospechado de padecer AD, y para distinguir si los síntomas de demencia que el paciente presenta tienen más riesgo de estar asociados a la enfermedad de Alzheimer o a otras patologías causantes de demencia. Las pruebas sobre el LCR no son de carácter rutinario en el ámbito del diagnóstico de AD, pero son utilizadas en personas que se sospecha que padecen de demencia causada por la enfermedad de Alzheimer luego de que se hayan descartado otras causas para los síntomas que presenta el paciente.

Mas allá de la utilidad y potencial carácter predictivo de los marcadores de LCR, éstos aún tienen asociados problemas que previenen que sea adoptado de ampliamente en el ámbito clínico y de investigación. El mayor problema asociado a los marcadores de LCR es la falta un umbral de corte global que distinga concentraciones normales de anómalas. En los últimos años se llevaron a cabo múltiples mejoras en los procedimientos asociados a las pruebas de LCR para su uso en AD, y como consecuencia se logró un avance significativo en la

harmonización de las diferentes pruebas comerciales que existen en el mercado [51] . Por lo tanto, se ha reportado que la variabilidad asociada a los resultados de las pruebas de detección de biomarcadores en LCR surge de las diferencias en los procedimientos preanalíticos y, como consecuencia, el establecimiento de umbrales de corte globales dependerá de manera crítica de la definición de un protocolo preanalítico unificado para el tratamiento del LCR, como el que se propone en [51].

3.2. APOE

El gen APOE codifica la Apolipoproteína E, la cual es una glucoproteína polimórfica expresada en el hígado, cerebro, macrófagos y monocitos. El APOE participa en el transporte de colesterol y otros lípidos, y está involucrado en el crecimiento neuronal, la respuesta de reparación ante la lesión de un tejido, regeneración de nervios, regulación inmunológica, y la activación de enzimas lipolíticas. Contiene tres variante alélicas principales en un locus ($\epsilon 2$, $\epsilon 3$ y $\epsilon 4$), las cuales codifican las diferentes isoformas (ApoE2, ApoE3, and ApoE4) de la proteína que difieren en dos sitios de la secuencia de aminoácidos. El alelo $\epsilon 4$ del gen incrementa el riesgo de tener AD, pero no es suficiente para causar la enfermedad. Se estima que el riesgo de tener AD aumenta 3 veces para los portadores de la variante heterocigota APOE $\epsilon 3/4$ y que dicho riesgo aumenta 15 veces para portadores de la variante homocigota APOE $\epsilon 4/4$ comparado con sujetos de edades similares que no tienen la enfermedad, y tiene un efecto de adelantamiento sobre el inicio de la enfermedad. Por otro lado, se considera que el alelo $\epsilon 2$ tiene un efecto de protección que retrasa la edad de inicio de la enfermedad [3]. Solamente el 20-25 % de la población son portadores de uno o más alelos $\epsilon 4$, mientras que el 40-65 % de los pacientes de AD son portadores de $\epsilon 4$. Por lo tanto, no se considera que el genotipado de APOE tiene valor diagnóstico intrínseco, sino que aporta un factor de riesgo que debe usarse como complemento de otros estudios [52].

El genotipado de APOE se ordena cuando el paciente presenta síntomas de demencia progresiva, y fueron descartadas otras causas no relacionadas con AD (como sobremedicación, demencia vascular causada por un derrame, o alguna patología tiroide), para poder determinar la probabilidad de que la demencia sea debida a la enfermedad de Alzheimer. Sin embargo, con el surgimiento de técnicas rápidas y de bajo costo para el genotipado de APOE (como las propuestas en [53] y [54]), que facilitan el acceso a esta prueba, se puede generar que el genotipado de APOE se utilice más ampliamente en el ámbito clínico.

3.3. MMSE

El "Mini Mental State Examination" (MMSE) ([55]) es la evaluación psicométrica del funcionamiento cognitivo más comúnmente administrada, para la cual la literatura ha demostrado que es un marcador relativamente sensible de demencia manifiesta [56]. El MMSE se utiliza para evaluar el deterioro cognitivo, seguir los cambios en el funcionamiento cognitivo en el tiempo, y en algunos casos para evaluar los efectos de agentes terapéuticos sobre la función cognitiva, aunque su utilidad decrece al ser utilizado en pacientes con condiciones psiquiátricas, u otras afecciones que generen un deterioro cognitivo. El puntaje máximo de

la prueba es de 30 puntos, y el resto de los valores tienen asociado un grado de demencia como se muestra en la tabla 3.1.

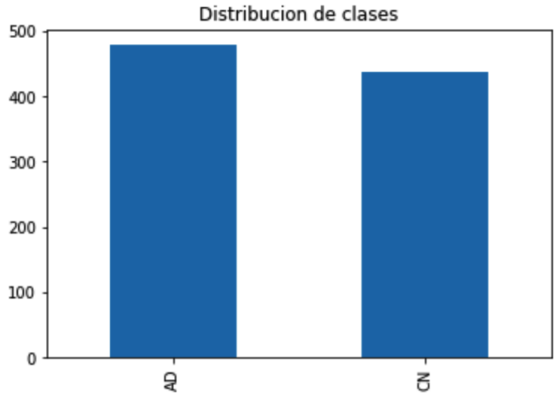
Resultado MMSE	**Significado**
30 - 24	Cognitivamente normal
24 - 20	Demencia leve
20 - 13	Demencia moderada
13 - 0	Demencia severa

Tabla 3.1: Rangos de valores de los resultados del MMSE, y los grados de demencia asociados

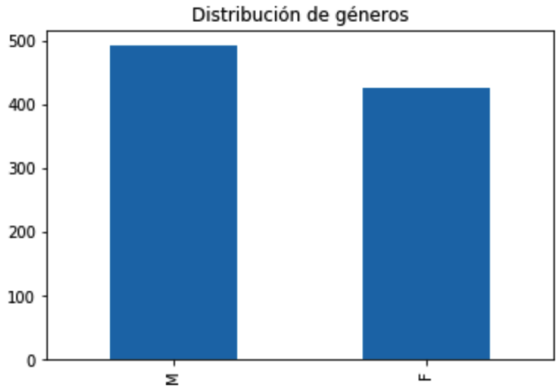
Cabe aclarar que el grado de educación, la demografía y la edad del paciente tienen un impacto en el resultado del MMSE, y estas variables son contempladas a la hora de realizar la prueba [56].

3.4. Datos utilizados

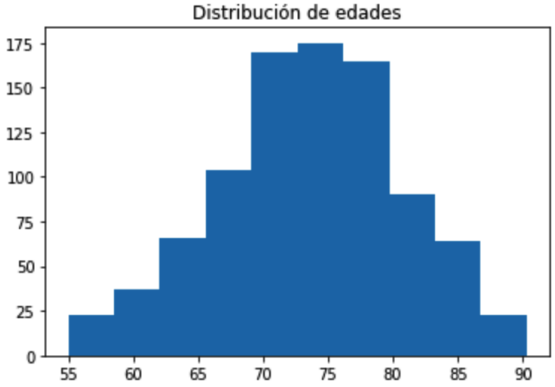
Para esta etapa del proyecto se utilizaron los datos de biomarcadores de LCR, APOE y MMSE disponibles en ADNI. La base de datos ADNI generó una tabla llamada ADNIMERGE en la que unifica todas los estudios hechos para cada paciente en las diferentes visitas registradas. Siguiendo con el criterio de inclusión utilizado previamente, se eligieron los sujetos que tengan todos los datos disponibles en su última visita registrada, la cual coincide con la visita en la que se adquirieron las imágenes de resonancia magnética para los sujetos que las disponen. De esta forma se obtuvo un total de 917 sujetos que califican para el análisis en esta etapa. Como se puede ver en la figura 3.2, se analiza la población de sujetos que fueron seleccionados, por clase, género y edad. En cuanto a las clases, se disponen de 479 sujetos con AD y 438 sujetos CN; en cuanto al género, se tienen 492 sujetos masculinos y 425 femeninos; y en cuanto a edades la mayoría de los sujetos se encuentra en el rango de 70 a 80 años.



(a) Distribución de clases de los sujetos.



(b) Distribución de género de los sujetos



(c) Distribución de edades de los sujetos.

Figura 3.2: Visualización de la información de los sujetos seleccionados para el análisis.

3.5. Análisis

3.5.1. Visualización de los datos

Se transformaron los valores de las variables ABETA (concentración de$A\beta$ en LCR), TAU, PTAU y MMSE al rango $[0, 1]$ dividiendo cada variable por su valor máximo para normalizar las escalas y facilitar la visualización de los datos. En cambio, la variable APOE4 que describe el resultado de genotipado del paciente tiene solo 3 valores posibles: 0, que equivale a que el paciente no presenta ningún alelo $\epsilon 4$ en su genotipado de APOE, 1 que indica que el paciente tiene un alelo $\epsilon 4$, y 2 que indica que el paciente tiene los dos alelos $\epsilon 4$.

	ABETA	TAU	PTAU	MMSE	APOE4
mean	0.5165	0.3468	0.3192	0.8509	-
std	0.2244	0.1576	0.1658	0.1612	-
min	0.1196	0.0919	0.0897	0.0	0.0
25 %	0.3232	0.2342	0.2008	0.7666	0.0
50 %	0.4601	0.3126	0.2813	0.9	0.0
75 %	0.7665	0.4265	0.3922	0.9666	1.0
max	1.0	1.0	1.0	1.0	2.0

Tabla 3.2: Descripción de los datos utilizados.

Como se puede ver en la tabla 3.2, esta descripción no aporta demasiada información acerca de los datos ni de sus características más allá de la verificación de que todas las variables se encuentran en el rango $[0, 1]$. Para poder visualizar mejor las características de los datos y las relaciones entre las variables se mostrarán gráficos de dispersión entre las variables para identificar, si las hubiese, relaciones significativas entre pares de variables y obtener una visión más clara de la interacción de los datos utilizados en el marco del problema propuesto.

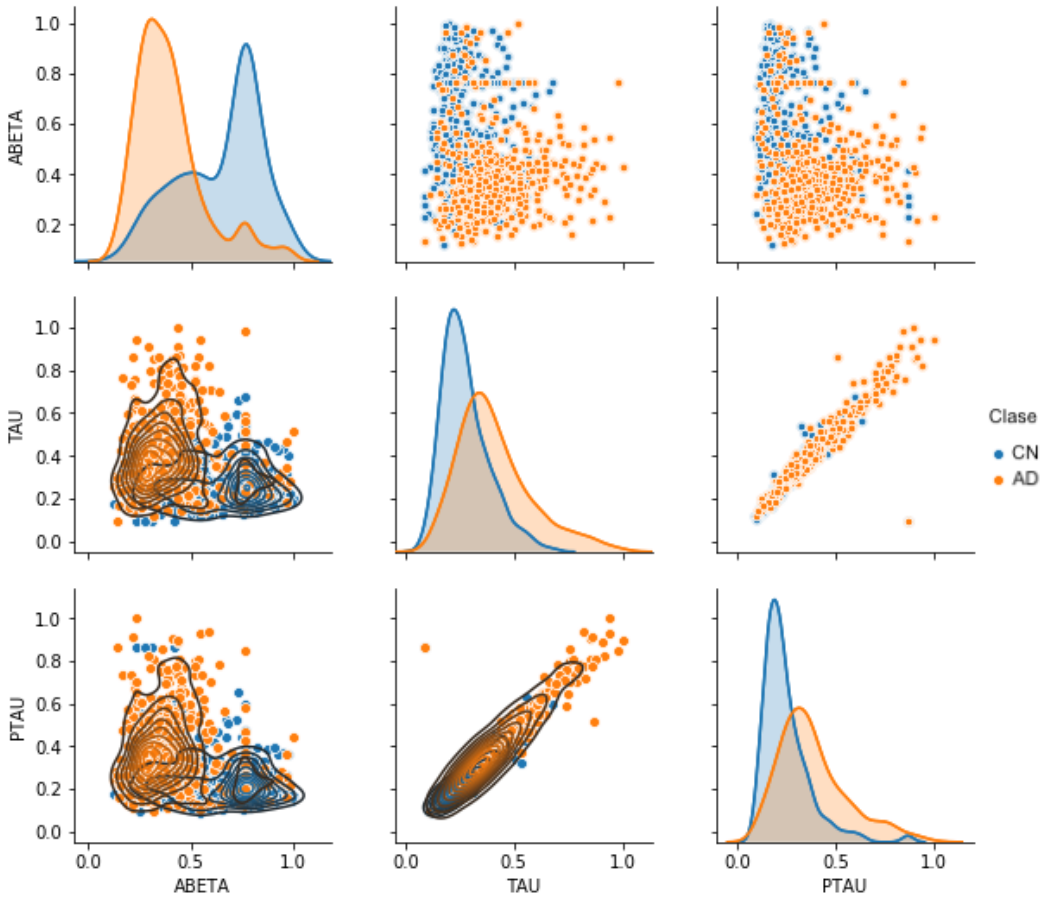


Figura 3.3: Grilla de gráficos (pairplot) que muestra la relación entre los biomarcadores de LCR.

En la figura 3.3 se muestran nueve gráficos que relacionan las variables ABETA, TAU y PTAU para buscar relaciones relevantes entre ellas que puedan ser útiles para selección del modelo de clasificación que será utilizado. En la diagonal de la grilla de gráficos se ve la distribución de probabilidad estimada de cada variable, y se puede ver que las variables por sí mismas no demuestran gran capacidad para diferenciar las clases salvo el caso de ABETA que tiene un corte claro entre AD y CN aunque con cierto solapamiento de las distribuciones. Por otro lado, en estos gráficos se puede observar que los valores de ABETA son en su mayoría más bajos cuando el paciente tiene AD, lo cual se corresponde con el comportamiento de formación de las placas amiloides en la patología que se traduce en una disminución en el nivel de proteína amiloide β en LCR; en cuanto a las proteínas tau y p-tau, se ve que para los pacientes normales los valores de concentración en LCR son en mayor frecuencia bajos, y aumentan para los pacientes con AD, lo cual es el comportamiento esperado por parte de estos biomarcadores.

Al ver los gráficos que relacionan las variables entre sí, se puede ver que al mostrar las distribuciones conjuntas de ABETA con TAU y ABETA con PTAU, se generan focos que permiten diferenciar las clases, aunque con cierto solapamiento entre sí. Estas relaciones mostradas coinciden con los hallazgos explicados anteriormente en los que se explicó que los biomarcadores contienen información relevante para el diagnóstico de AD. Aunque estas relaciones puedan ser útiles, no permiten la generación de un hiperplano que divida a los sujetos, y por lo tanto los datos no son linealmente separables.

En cuanto al valor analítico que aportan los datos de las variables APOE4 y MMSE, se decidió generar gráficos aparte que muestren la relevancia de los datos. En primer lugar, se analiza la distribución de valores para los datos disponibles de genotipado de APOE:

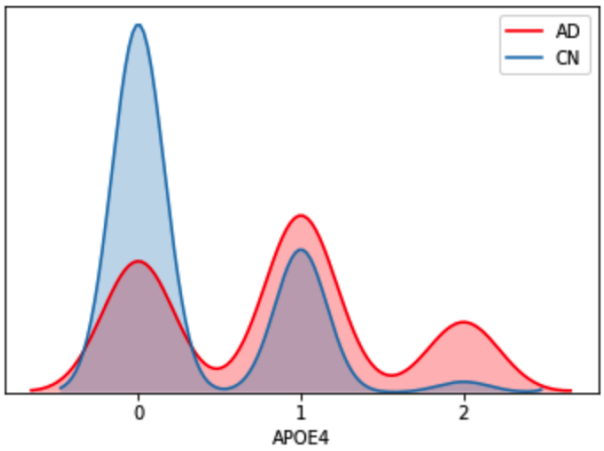
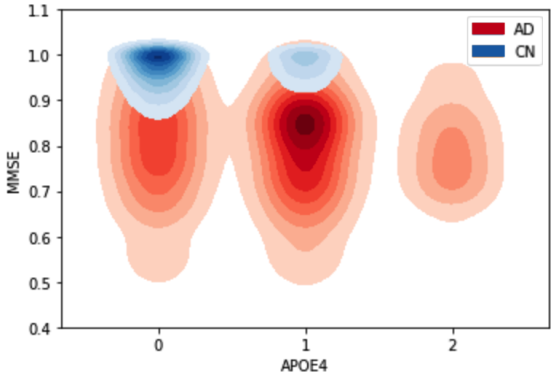


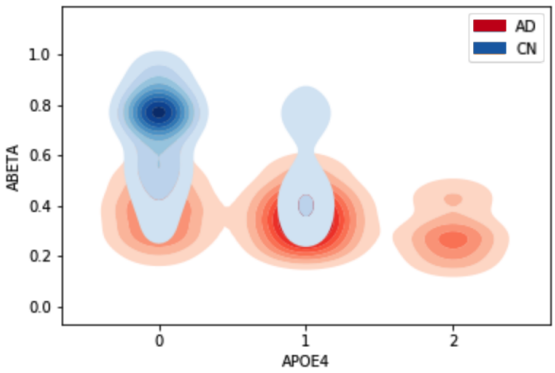
Figura 3.4: Distribución de APOE según diagnóstico de los pacientes. Las distribuciones no tienen la misma escala, pero muestran cómo se distribuyen según el los alelos de $\epsilon 4$ presentes en cada sujeto.

Podemos ver en la figura 3.4 que los sujetos normales tienen su mayor concentración de datos para genotipados de APOE que no contienen ningún alelo $\epsilon 4$, aunque también hay pocos sujetos con AD con dicho resultado de genotipado. Para los sujetos que tienen un solo alelo $\epsilon 4$ comienza a haber más sujetos con AD que CN, y para los sujetos con dos alelos $\epsilon 4$ se

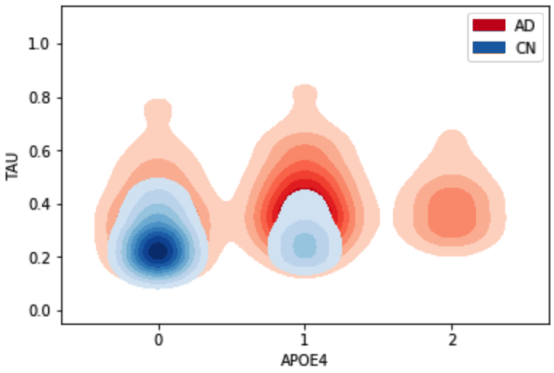
puede ver mucha más prevalencia de sujetos con AD (89 sujetos con AD y sólo 9 CN). Este análisis es consistente con las conclusiones teóricas expuestas anteriormente sobre el uso del gen APOE en el diagnóstico de AD como factor de riesgo, y no como un dato concluyente acerca de si el paciente posee o no la enfermedad. Veamos también la relación de la variable APOE4 con las demás:



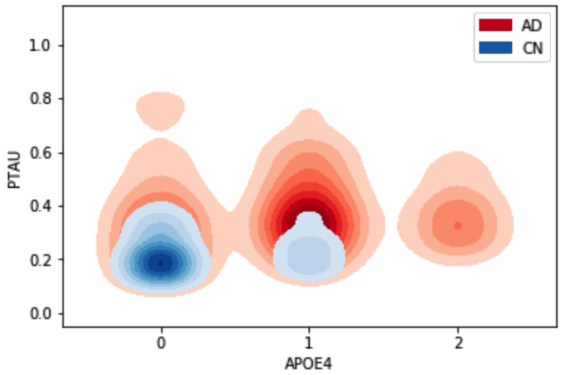
(a) Distribución conjunta de APOE4 con MMSE.



(b) Distribución conjunta de APOE4 con ABETA



(c) Distribución conjunta de APOE4 con TAU



(d) Distribución conjunta de APOE4 con PTAU

Figura 3.5: Distribuciones conjuntas de APOE4 con el resto de las variables.

En la figura 3.5 se pueden cuatro gráficos que muestran las distribuciones conjuntas de la variable APOE4 con ABETA,TAU, PTAU y MMSE con distinción por clase. Podemos notar en el gráfico que muestra APOE4 con MMSE (figura 3.5a) una gran concentración de sujetos normales, representado por la zona de color azul más oscuro, quienes presentan un puntaje de MMSE alto y ningún alelo $\epsilon 4$ en su genotipado de APOE. En cuanto los sujetos que tienen AD y ningún alelo $\epsilon 4$ en su genotipado de APOE, el puntaje de MMSE suele ser menor que el de los sujetos cognitivamente normales, y toma un rango mayor de valores lo cual indica que existe una gran variabilidad de la variable MMSE para los sujetos con AD, mientras que para los CN son más acotados. El amplio rango de posibles valores de la variable MMSE también se puede ver en el caso que los sujetos con AD que tienen un alelo $\epsilon 4$ en su genotipado de APOE, y en ese caso lo sujetos normales también presentan resultados altos en el MMSE. Por último, en el caso de que los sujetos presenten los dos alelos $\epsilon 4$ en su genotipado de APOE, vemos que ya no se distinguen sujetos normales, y que

todos los sujetos con AD tienen valores bajos en sus resultados del MMSE, comparado con los normales de los casos explicados anteriormente, y además los resultados tienen un rango más pequeño de valores.

En el caso de la observación del par de variables APOE4 y ABETA (figura 3.5b), se puede notar que para el caso en que los sujetos no poseen ningún alelo $\epsilon 4$, la mayoría de los sujetos normales poseen valores altos de concentración de la proteína amiloide β en LCR, y que los sujetos que poseen la patología tienen valores bajos de concentración de la proteína en LCR, lo cual coincide con la descripción de la fisiopatología de la proteína amiloide β en AD y con el hecho de que los sujetos que no tienen ningún alelo $\epsilon 4$ en el gen APOE aún pueden tener la enfermedad. En el caso en que los sujetos tienen un alelo $\epsilon 4$ se puede ver que la cantidad de sujetos normales decrece y los valores de ABETA no tienen una tendencia marcada a ser altos o bajos, aunque para los sujetos con AD la tendencia es que los valores de ABETA sean bajos. Finalmente, en el caso que los sujetos tienen dos alelos $\epsilon 4$ en el gen APOE, la mayoría de los sujetos tienen AD y como sólo hay 9 sujetos normales no se pueden ver graficados en la distribución ni sacar conclusiones relevantes acerca de los valores de ABETA que presentan.

Finalmente, al observar el par de variables APOE y TAU (con PTAU los resultados son muy similares), se puede ver un comportamiento similar al visto con la variable ABETA, pero en este caso la concentración de TAU no patológica es baja.

Podemos ver que la combinción de la variable APOE con las demás variables utilizadas para este análisis puede aportar valor analítico significativo para el problema a resolver. Sin embargo, dado que los sujetos que tienen dos alelos $\epsilon 4$ en el gen APOE son en su mayoría pacientes que tienen la enfermedad de Alzheimer, puede existir cierto sesgo por parte de un modelo de aprendizaje automático hacia la identificación de dichos pacientes como enfermos.

Al igual que para APOE4, se hizo un análisis comparativo de MMSE con las demás variables para obtener más relaciones útiles entre ellas. Veamos en principio las distribuciones de MMSE para pacientes con AD y CN

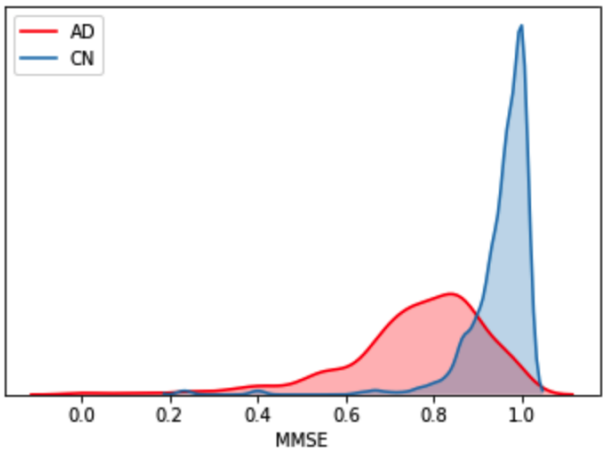
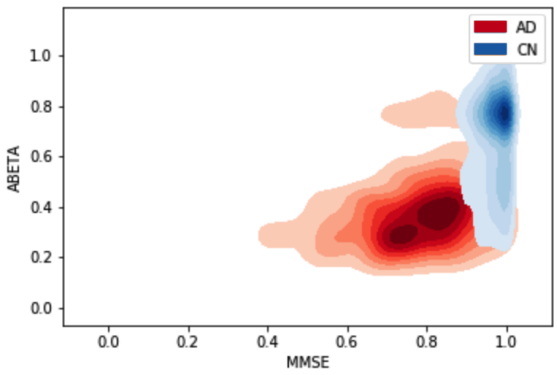


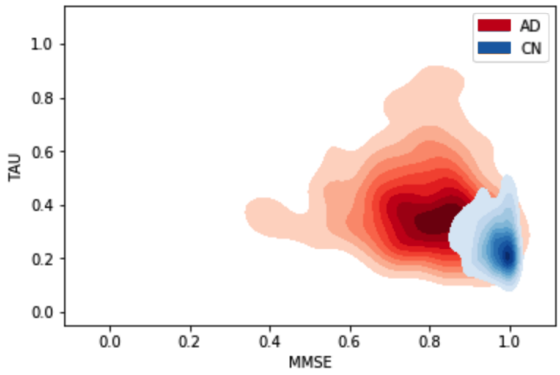
Figura 3.6: Distribución de MMSE según diagnóstico de los pacientes.

Se puede ver en la figura 3.6 que en este caso, la mayor concentración de pacientes normales tienen resultados altos en MMSE, mientras que los pacientes con AD tienen resultados

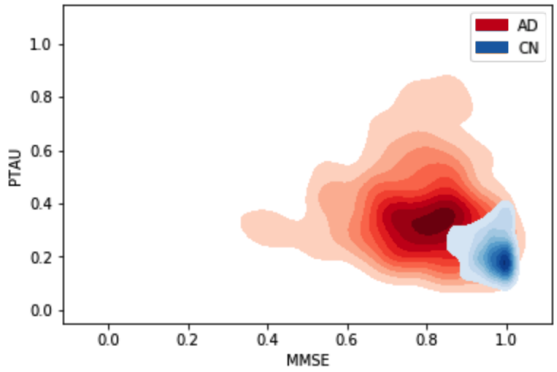
mas variados. En cuanto a la comparación entre MMSE y las demás variables, en la figura 3.7 se muestran las distribuciones conjuntas de MMSE con ABETA, TAU y PTAU:



(a) Distribución conjunta de MMSE con ABETA



(b) Distribución conjunta de MMSE con TAU



(c) Distribución conjunta de MMSE con PTAU

Figura 3.7: Distribuciones conjuntas de MMSE con el resto de las variables.

Se puede ver en los distintos casos un comportamiento similar al mostrado con las distribuciones conjuntas de APOE4 con las demás variables. En la figura 3.7a se puede ver la distribución conjunta de MMSE con ABETA donde se muestra que los sujetos normales tienen una tendencia a tener valores altos en su resultado de MMSE y valores altos de concentración de la proteína amiloide β en LCR, mientras que los sujetos con AD tienden a tener valores más bajos en su MMSE y concentraciones menores de la proteína en LCR, ambos casos con cierta dispersión y coincidiendo con las observaciones hechas sobre los niveles de cognición y su relación con los biomarcadores de LCR. En el caso de las distribuciones conjuntas de MMSE con TAU y PTAU, se puede ver la tendencia de los sujetos normales a tener resultados altos en su MMSE y bajas concentraciones de tau y p-tau en LCR, mientras que para los sujetos con AD sucede lo contrario, es decir, peores resultados en sus MMSE y concentraciones más elevadas de las proteínas en LCR. La conclusión que se podría obtener de estas observaciones es que las variables en conjunto confirman las observaciones teóricas hechas con respecto a la fisiopatología de la enfermedad de Alzheimer, y las fortalezas y debilidades de los biomarcadores analizados para su uso en el ámbito clínico con el objetivo de diagnosticar la enfermedad. Además, se puede ver que las variables seleccionadas para

esta sección del análisis muestran potencialmente una buena capacidad para distinguir los sujetos según sus clases y que un modelo de aprendizaje automático puede explotar las relaciones de las variables para obtener buenos resultados. Por eso, en la siguiente sección se explorarán algunos métodos para la clasificación de estos datos con el objetivo de encontrar el que genere mejores resultados.

3.5.2. Métodos evaluados

A partir de las observaciones hechas sobre los datos disponibles se procede a la definición de un método de análisis que pueda explotar los patrones existentes en ellos. Se eligieron cuatro modelos clásicos y ampliamente utilizados para el análisis de datos estructurados: Clasificador Naïve Bayes, Regresión logística, SVM y Multilayer perceptron. A continuación se realizará una breve introducción para explicar los fundamentos teóricos de cada método, sus ventajas y desventajas, y se hará una comparación de los resultados obtenidos sobre los datos utilizados con cada uno de ellos.

Clasificador Naïve Bayes

Los métodos de Naïve Bayes son un conjunto de métodos de aprendizaje supervisado basados en la aplicación del teorema de Bayes con la suposición de independencia condicional entre las variables dado el valor de la clase. El teorema de Bayes establece la siguiente relación:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

dada la variable y que representa la clase dependiente del vector de datos $\vec{x} = [x_1, \ldots, x_n]$. Utilizando la suposición "naïve" de independencia condicional:

$$P(x_1, \ldots, x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$

para todo i, el teorema de Bayes se simplifica a:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Dado que $P(x_1, \ldots, x_n)$ es constante dados los datos de entrada, y no depende de la clase, se puede tomar que:

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Y por lo tanto la regla de clasificación para obtener la clase predicha $\hat{y}$ a partir de los datos es:

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

lo cual equivale a utilizar la estimación máxima a posteriori tomando la frecuencia relativa de la clase y en los datos de entrenamiento como $P(y)$ y asumiendo que la distribución condicional de $P(x_i \mid y)$, también conocida como *likelihood*, corresponde a alguna distribución conocida. Los diferentes clasificadores de Naïve Bayes difieren en la suposición hecha acerca de la distribución de $P(x_i \mid y)$. Una de las distribuciones más empleadas para asumir sobre el likelihood en problemas de clasificación es la distribución Gaussiana:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

donde σ_y y μ_y son estimadas usando el método de *maximum likelihood estimation*, el cual sirve para estimar los parámetros de una distribución de probabilidad mediante la maximización de una función de likelihood de manera que los datos observados sean los más probables bajo el modelo estadístico asumido.

A pesar de ser métodos que utilizan suposiciones fuertes sobre los datos, los clasificadores Naïve de Bayes suelen funcionar muy bien en problemas reales, requieren pocos datos para estimar los parámetros necesarios y pueden ser extremadamente rápidos comparados con métodos más sofisticados.

Regresión Logística

La regresión logística es un método de aprendizaje supervisado que se utiliza en problemas de clasificación binaria (aunque puede extenderse para ser empleado en un problema multiclase). El método utiliza la función sigmoidea para determinar la probabilidad de que un conjunto de datos describa un elemento perteneciente a una clase. Los datos que se utilizan para entrenar el clasificador pueden ser continuos o discretos, y las características que dichos datos representan pueden ser no lineales.

Para referirnos a los valores de las clases se emplea la variable $y \in \{0, 1\}$, siendo 0 y 1 las etiquetas asignadas a las clases del problema binario. Se define la probabilidad que se debe maximizar con este método de la siguiente manera:

$$h_a(x) = P(y = 1 \mid x : a), 0 \leq h_a(x) \leq 1$$

lo cual significa que *la probabilidad de que* $y = 1$ *dado los valores del vector de características* x *parametrizado por* a. También, la probabilidad de que $y = 0$ está dada por $P(y = 0|x : a) = 1 - P(y = 1|x : a)$. La función que se utiliza para modelar la probabilidad $h_a(x)$ está dada por:

$$h_a(x) = g(a^\intercal x)$$

siendo $g(z)$ la función sigmoidea definida por:

$$g(z) = \frac{1}{1 + e^{-z}}$$

y los vectores a y x:

$$a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} ; x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

los cuales son el vector de coeficientes de parametrización del método y una observación de los datos (se le agrega un 1 para poder utilizar el coeficiente independiente a_0). De esta manera, obtenemos:

$$h_a(x) = \frac{1}{1 + e^{-a^\intercal x}} = g(a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_n x_n)$$

Luego, generalizando para m observaciones de los datos se utiliza la matriz X y $h_a(x)$ se convierte en $h_a(X)$:

$$h_a(X) = g\left(\begin{bmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \right)$$

La función sigmoidea $g(z)$ tiene la particularidad de tender a 1 para valores de z grandes, y a 0 para los valores más negativos, siendo $g(0) = 0{,}5$ el punto de corte de la función ya que para $z \geq 0$, $g(z) \geq 0{,}5$ mientras que para $z < 0$, $g(z) < 0{,}5$.

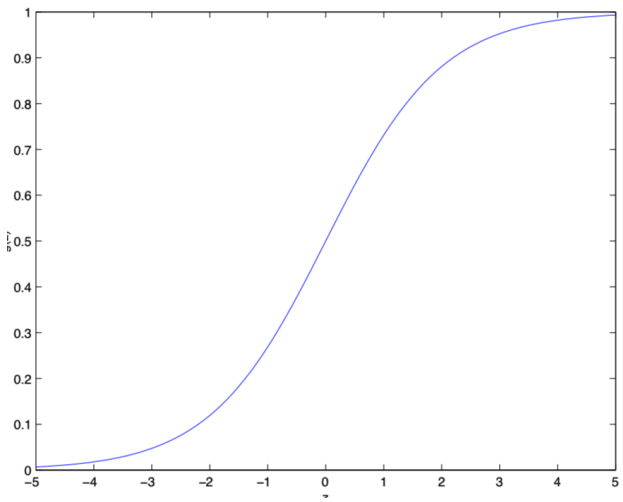


Figura 3.8: Función sigmoidea, también llamada función logística [57].

Como se puede ver, la función sigmoidea tiene un rol fundamental en la definición del umbral de decisión para la asignación de clases. Para el caso del método de regresión logística se considera que si $h_a(x) \geq 0{,}5$ entonces la clase predicha es $\hat{y} = 1$, y si $h_a(x) < 0{,}5$ entonces la clase predicha es $\hat{y} = 0$. Este umbral de decisión definido por la función sigmoidea depende

de la interacción de los coeficientes de parametrización del método en el vector a con las observaciones utilizadas para entrenar el modelo. Por eso, para encontrar los coeficientes que definan el umbral de decisión óptimo se utiliza una función de costo que se obtiene a partir de la función sigmoidea. Una vez que se hallaron los valores óptimos de a la función $h_a(x)$ de qué lado del umbral de decisión queda la predicción basándose en los valores del vector x.

La función de costo, como ya se dijo, se obtiene a partir de la función sigmoidea y para formularla se utiliza la expresión logarítmica, lo cual genera beneficios computacionales al reemplazar el producto de probabilidades por la suma de logaritmos, y genera una función convexa que hace mucho más simple su minimización. La función de costo será dividida en dos casos cada para vector de datos, uno para $y = 1$ y otro para $y = 0$:

$$J^{(i)}_{y=1}(a) = -log(h_a(x^{(i)})); J^{(i)}_{y=0}(a) = -log(1 - h_a(x^{(i)}))$$

Finalmente, estos dos casos de combinan para obtener:

$$J(a) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} log(h_a(x^{(i)})) + (1 - y^{(i)}) log(1 - h_a(x^{(i)}))$$

o en la forma compacta utilizando la notación matricial que se introdujo anteriormente:

$$J(a) = -\frac{1}{m}\left(y^{\intercal} log(h_a(X)) + (1 - y)^{\intercal} log(1 - h_a(X))\right.$$

La función de costo actúa penalizando los valores incorrectos de $h_a(x)$; en el caso $y = 1$, cuando $h_a(x)$ tiende a 0 (lo cual debería ser penalizado), $J(a)$ crece y por lo tanto, se optimizarán los parámetros del vector a de manera tal que $J(a)$ disminuya.

Para minimizar la función de costo se emplea un algoritmo de optimización. El más conocido y básico es el método de descenso por el gradiente o Stochastic Gradient Descent, pero existen métodos más avanzados y eficientes como el algoritmo BFGS(Broyden–Fletcher–Goldfarb–Shanno), el algoritmo L-BFGS(como BFGS pero usa menos memoria) y el algoritmo de Gradiente Conjugado. El tipo de algoritmo utilizado para optimizar la función de costo y la tasa de aprendizaje (learning rate) que se utiliza son dos de los hiperparámetros que deben ser ajustados para obtener el mejor resultado con este clasificador. Como en este método se optimizan los parámetros mediante la minimización de una función de costo, se pueden emplear las técnicas de regularización de pesos mencionadas anteriormente (l_1, l_2 o Elastic-Net), lo cual corresponde a otro hiperparámetro que se debe ajustar.

Support Vector Machine (SVM)

El método de clasificación SVM es uno de los métodos de aprendizaje supervisado más utilizados en los problemas de clasificación binaria. Se trata de un método discriminativo que utiliza el concepto de los márgenes funcionales y geométricos entre clases para calcular el hiperplano que mejor separa los datos y le asigna clases a los valores según de qué lado de dicho hiperplano se encuentran.

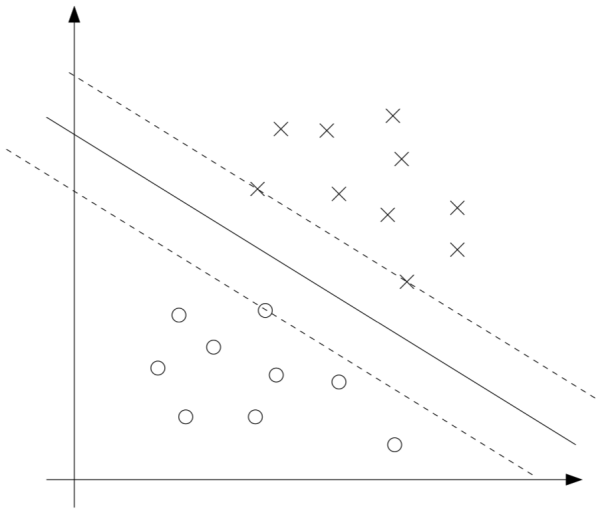


Figura 3.9: Separación de los datos mediante un hiperplano que maximiza el márgen entre clases [57].

Los puntos con los márgenes más pequeños son exactamente aquellos más cercanos al umbral de decisión (los puntos que caen sobre las líneas punteadas en la figura 3.9) y son llamados *vectores de soporte* (support vectors).

Para derivar la formulación de este método los autores de [58] y [57] utilizan la notación $y \in \{-1, 1\}$ para referirse a las clases del problema binario. Dados los puntos de entrenamiento x_i, se define el hiperplano $\mathcal{H}$ tal que:

$$\mathcal{H} : \begin{cases} w^\intercal x_i + b \geq 1 & \text{si} \quad y_i = 1 \\ \\ w^\intercal x_i + b \leq -1 & \text{si} \quad y_i = -1 \end{cases}$$

y además $\mathcal{H}_1 : w^\intercal x_i = 1$ y $\mathcal{H}_2 : w^\intercal x_i = -1$ son los planos sobre los cuales se encuentran los vectores de soporte.

El hiperplano óptimo es aquel que maximiza el margen de separación con el punto de los datos más cercano para los pesos w y el sesgo b, es decir que se halla el hiperplano que maximiza la separación con los vectores de soporte. Para eso, se debe maximizar la distancia normal del hiperplano al vector de soporte más cercano, lo cual equivale a maximizar la distancia:

$$\frac{|w^\intercal x_i + b|}{\|w\|}$$

y el problema se reduce a hallar:

$$\min \|w\|^2 \text{ sujeto a } \begin{cases} w^\intercal x_i + b \geq 1 & \text{si} \quad y_i = 1 \\ \\ w^\intercal x_i + b \leq -1 & \text{si} \quad y_i = -1 \end{cases}$$

lo cual se puede simplificar en:

$$\min\|w\|^2 \text{ sujeto a } y_i\left(w^\intercal x_i + b\right) \geq 1$$

El problema planteado es un problema de optimización convexa que se puede resolver utilizando el método de multiplicadores de Lagrange, cuya formulación para este caso es:

$$\mathcal{L}(w, b, \lambda) = \frac{1}{2} w^\intercal w + \sum_{i=1}^{m} \lambda_i \left[y_i\left(w^\intercal x_i + b\right) - 1\right] \forall\, i, \lambda_i \geq 0$$

lo cual debe cumplir:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{m} \lambda_i y_i x_i \\ \\ \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{i=1}^{m} \lambda_i y_i = 0 \end{cases}$$

Por lo tanto, el problema planteado es un problema dual, lo cual significa que es suficiente resolverlo para una de las condiciones. La solución de este problema de optimización son los multiplicadores de Lagrange λ_i^*, los cuales si son no nulos indican que el correspondiente x_i es un vector de soporte. Con estos valores se pueden calcular:

$$w^* = \sum_{i=1}^{m} \lambda_i^* y_i x_i$$

$$b^* = y_k - w^{*\intercal} x_k \;\forall\, x_k \text{ si } \lambda_k \neq 0$$

Luego, la función de clasificación está dada por:

$$f(x) = \sum_{i=1}^{m} \lambda_i^* y_i x_i^\intercal x + b^*$$

En el caso de que los datos no sean totalmente linealmente separables, es decir que las clases se solapan en algunos puntos, se introducen variables "slack" $\xi_i \geq 0$, considerando:

$$y_i\left(w^\intercal x_i + b\right) \geq 1 - \xi_i$$

donde las variables slack para los puntos x_i indican cuánto viola dicho punto la condición de separabilidad, teniendo en cuenta que un punto no puede estar a distancia menor a $\frac{1}{\|w\|}$ del hiperplano. La introducción de variables slack en la formulación genera que se calculen los hiperplanos de separación óptimos basados en la utilización de márgenes "blandos" que permiten cierta violación de la separabilidad lineal de las clases. Luego, teniendo datos que no son totalmente linealmente separables, el problema de optimización es:

$$\min_{w,b,\xi} \left\{ \frac{\|w\|^2}{2} + C \sum_{i=1}^{m} (\xi_i)^k \right\} \text{ sujeto a } y_i\left(w^\intercal x_i + b\right) \geq 1 - \xi_i \text{ con } \xi_i \geq 0 \,\forall\, i$$

donde C es la *constante de regularización* que se elije empíricamente, el término $\sum_{i=1}^{m} (\xi_i)^k$ da una estimación de cuánto se desvía la clasificación del caso separable (medida de la pérdida de información) y k determina la forma de la pérdida, tomando usualmente los valores 1 ó 2. Luego, se toma el mismo enfoque explicado previamente usando multiplicadores de Lagrange y resolviendo el problema dual.

Por último, se considera el caso de datos que cuya distribución no sea linealmente separable. En este caso se plantea una transformación de los datos a un espacio superior donde sí puedan ser separados mediante un hiperplano. Esta transformación se realiza mediante un *kernel*, el cual dada una función de mapeo de los datos ϕ se define como:

$$K\left(x_i, x_j\right) = \phi\left(x_i\right)^{\intercal} \phi\left(x_j\right)$$

por lo tanto si antes se tenía ${x_i}^{\intercal} x$ en la función de clasificación, se pasa a tener $K\left(x_i, x\right)$, y dicha función resultaría:

$$f(x) = \sum_{i=1}^{m} \lambda_i^* y_i K\left(x_i, x\right) + b^*$$

Tanto la elección del kernel como sus parámetros son hiperparámetros del método que se deben ajustar de manera cuidadosa para obtener los mejores resultados posibles.

Multilayer Perceptron

El Multilayer Perceptron (MLP) es una red neuronal compuesta de capas Fully Connected, o capas densas, las cuales realizan transformaciones sobre la entrada para generar la salida. Este tipo de redes neuronales se utiliza para resolver problemas de clasificación con datos estructurados, y no está restringido a problemas binarios.

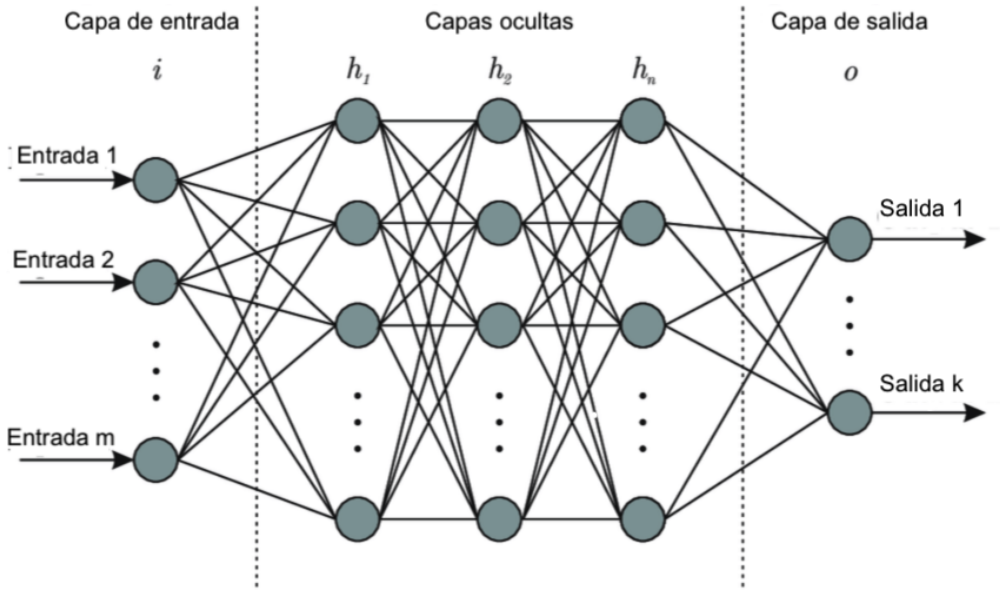


Figura 3.10: Arquitectura típica de una red MLP.

Como se puede ver en la figura 3.10, las capas ocultas son las que transforman los datos de entrada y la capa de salida genera el vector de probabilidades de cada clase. Cada capa oculta define una función que realiza una transformación afín a los datos e introduce no linealidad mediante una función de activación no lineal (típicamente ReLU) [14], y la capa de salida tiene una función de activación sigmoidea para el caso binario o softmax para el

caso multiclase. De esta manera la red MLP aprende una función global, que mapea los datos de entrada a las salidas, mediante la minimización de una función de costo.

Cabe destacar la relación de la MLP con el método de Regresión Logística: Si se tuviera una MLP sin capas ocultas y utilizando como función de activación de la capa de salida la función sigmoidea, se estaría planteando una Regresión Logística. En ese sentido, se podría pensar que la MLP genera transformaciones que optimizan la separabilidad de los datos y mejoran el desempeño de una Regresión Logística común. Sin embargo, al plantear transformaciones más complejas, también se complejiza el entrenamiento del modelo debido a que la optimización de la función de costo se vuelve más compleja y está sujeta a la aparición de mínimos locales que pueden generar un modelo cuyo desempeño no sea el deseado. Como ya se explicó en la introducción, se utilizan optimizadores y técnicas de regularización para intentar asegurar que el proceso de entrenamiento sea lo más exitoso posible. Otros hiperparámetros que se deben especificar son la cantidad de capas y parámetros entrenables que tendrá cada una, la función de activación utilizada, la cantidad de epochs que será entrenada y el tamaño del batch de datos que le será dado en cada iteración del entrenamiento.

3.5.3. Selección del mejor método

Para elegir el mejor modelo para el problema que en esta sección del trabajo, se hará una comparación del desempeño de clasificación de los modelos sobre un conjunto de prueba separado antes del entrenamiento y selección de hiperparámetros de los mismos. El conjunto de entrenamiento utilizado para esta parte contiene 733 sujetos, de los cuales 390 pertenecen a la clase AD (53 %) y 343 a la clase CN (47 %); mientras que el conjunto de prueba contiene 184 sujetos, de los cuales 88 (48 %) pertenecen a la clase AD y 96 (51 %) a la clase CN. Cabe destacar que se tomó el recaudo de asegurar que los sujetos utilizados para el entrenamiento y validación en la sección de imágenes de resonancia magnética estén incluidos en el set de entrenamiento de esta parte, y lo mismo para los sujetos de prueba. De esta manera se puede mantener un paralelismo entre las secciones y de manera preliminar comparar los desempeños de los modelos.

Selección de hiperparámetros

En cuanto al método utilizado para hallar los mejores hiperparámetros de cada modelo y comparar los resultados obtenidos sobre el set de prueba se procedió de la manera indicada en el siguiente esquema:

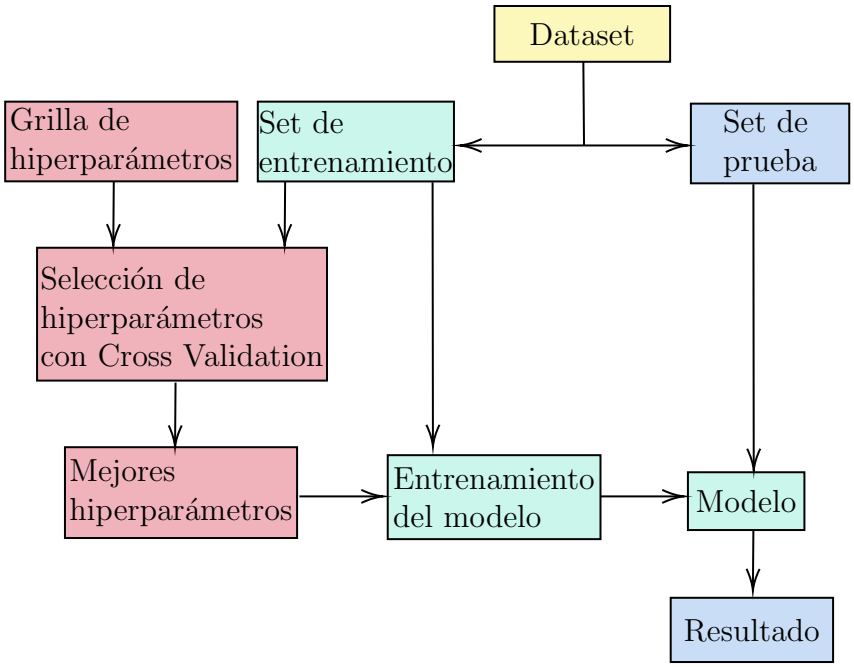


Figura 3.11: Diagrama que muestra el proceso de selección de hiperparámetros y prueba de los modelos obtenidos con cada método de clasificación propuesto. El dataset es dividido en un set de entrenamiento y uno de prueba. Con el set de entrenamiento y una grilla de hiperparámetros determinada se realiza una búsqueda de los mejore hiperparámetros para el modelo utilizando Cross Validation. Luego de obtener los mejores hiperparámetros, se entrena el modelo con la totalidad de los datos del set de entrenamiento y se realiza la prueba final sobre el set de prueba separado antes de iniciar el proceso.

Para el clasificador Naïve de Bayes, no se necesitó la selección de hiperparámetros y por lo tanto solamente se entrenó el modelo y se probó con el set de prueba. En cuando a los modelos de Regresión Logística y SVM se utilzó el método *GridSearchCV* del paquete Sci-Kit Learn [59], el cual realiza el procedimiento de búsqueda en una grilla de hiperparámetros provista manualmente y utiliza Cross-Validation para encontrar el conjunto de hiperparámetros que tienen mejor desempeño. Es un método de fuerza bruta, lo cual significa que prueba todas las combinaciones posibles en la grilla provista, lo cual es útil para métodos que tienen velocidades de entrenamiento altas como Regresión Logística y SVM. Sin embargo, para hallar los mejores hiperparámetros de la MLP se decidió encontrar primero una arquitectura adecuada y luego se utilizó el método *RandomizedSearchCV* de Sci-Kit Learn. Este método es más eficiente para la búsqueda de hiperparámetros en términos de tiempos y recursos utilizados que el método GridSearchCV, lo cual es útil para hallar los mejores hiperparámetros de una red neuronal [60].

Las grillas de hiperparámetros que se utilizaron para la búsqueda fueron:

Modelo	Grilla de hiperparámetros
Regresión Logística	C = {0.001; 0.01; 0.1; 1; 10; 100} Método = {bfgs; lbfgs; newton-cg} Penalización = {l_1; l_2; Elastic-Net}
SVM	C = {0.01; 0.1; 1; 10; 100; 1000; 1×10^4; 1×10^5; 1×10^6; 1×10^7; 1×10^8; 1×10^9; 1×10^{10}} gamma = {1×10^{-9}; 1×10^{-8}; 1×10^{-7}; 1×10^{-6}; 1×10^{-5}; 1×10^{-4}; 1×10^{-3}; 0.01; 0.1; 1; 10; 100; 1000}
MLP	Optimizador = {rmsprop; adam} Regularización l_2 = {0.1; 0.01; 0.001; 0.0001} Dropout = {0; 0.1; 0.25; 0.5} Epochs =,{100; 500; 1000} Tamaño de Batch = {16; 32; 64}

Tabla 3.3: Grillas de hiperparámetros usadas para cada método. Para Regresión Logística y SVM se utilizó Grid Search, mientras que para MLP se utilizó Randomized Search.

Los resultados obtenidos sobre el set de prueba fueron los siguientes:

Modelo	Mejores hiperparámetros	Resultados en el set de prueba
Naïve Bayes	-	81.54 %
Regresión Logística	C = 1 Método = lbfgs Penalización = l_2	86.22 %
SVM	C = 100 gamma = 0.01	85.87 %
MLP	**Optimizador = rmsprop** **Regularización l_2 = 0.001** **Dropout = 0.25** **Epochs =1000** **Tamaño de Batch = 16**	**89.13 %**

Tabla 3.4: Mejores hiperparámetros hallados y resultados de exactitud de clasificación obtenidos a partir de los modelos entrenados con ellos.

Como se puede ver el modelo que mejores resultados obtuvo sobre el set de prueba fue la red neuronal MLP, y por eso se utilizará este modelo para el desarrollo del modelo final de este trabajo. La arquitectura de la red se puede ver en la siguiente figura:

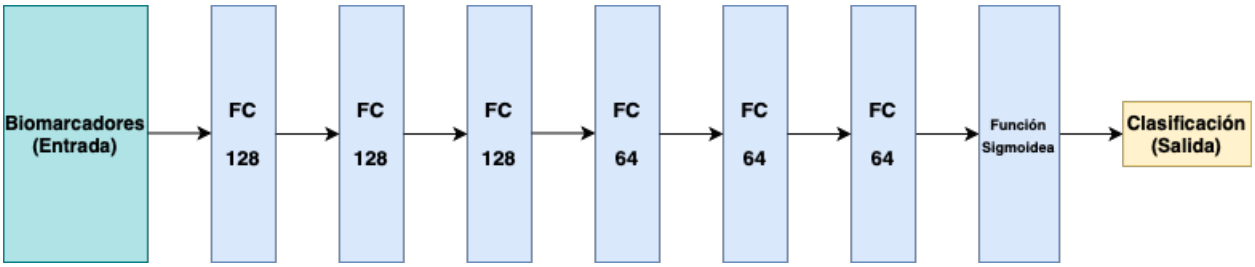


Figura 3.12: Arquitectura de la red MLP propuesta.

La arquitectura está compuesta por 3 capas Fully Connected de 128 unidades y 3 capas Fully Connected de 64 unidades, todas con función activación ReLU , Regularización l_2 de 0.001 y Dropout del 25 % de las unidades (como fue obtenido a partir de la búsqueda de los hiperparámetros). Se eligió la regularización l_2 debido a que se probó con l_1 y $l_1 - l_2$, y fue la que proporcionaba mejor desempeño de regularización.

3.5.4. Resultados

Luego de elegir el modelo que se utilizará, se calcularon las métricas de evaluación del modelo con el set de prueba, y se obtuvo la siguiente matriz de confusión:

$$\begin{bmatrix} 77 & 9 \\ 11 & 87 \end{bmatrix}$$

y las siguientes métricas:

	Exactitud	Sensibilidad	Especificidad	VPP	VPN	F1-Score	ROC-AUC
MLP propuesta	89.13 %	88.77 %	89.53 %	90.62 %	87.5 %	89.7 %	89.15 %

Tabla 3.5: Métricas obtenidas con la MLP al ser probada con el set de prueba.

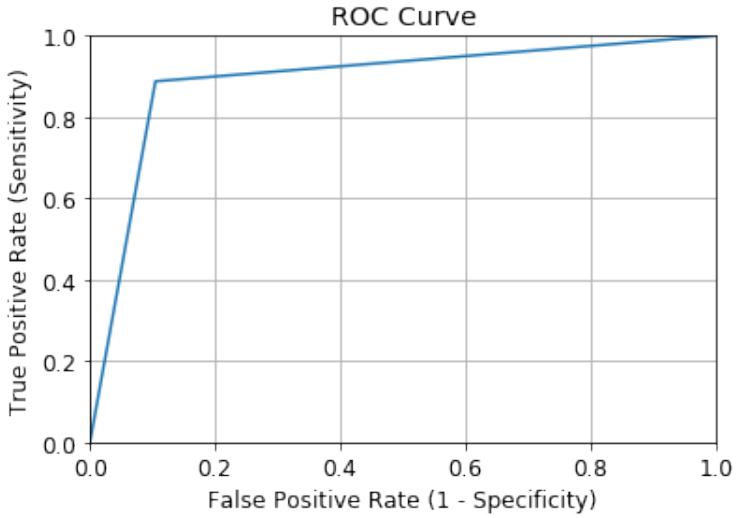


Figura 3.13: Curva ROC obtenida con la red MLP.

Ahora, veamos las métricas obtenidas al realizar 10-Fold Cross-Validation, como se hizo con la CNN en el capítulo anterior:

	Exactitud	Sensibilidad	Especificidad	VPP	VPN	F1-Score	ROC-AUC
CNN propuesta 10-Fold Cross Validation	92.8 (1.4) %	91.5 (1.7) %	94.1 (2.3) %	93.8 (2.2) %	92.5 (2.1) %	92.6 (1.9) %	92.8 (1.4) %
MLP propuesta 10-Fold Cross Validation	88.2 (0.4) %	90.3 (1.7) %	85.8 (2.7) %	87.9 (1.8) %	88.7 (1.5) %	89.1 (0.3) %	88.1 (0.6) %

Tabla 3.6: Métricas obtenidas al hacer 10-Fold Cross Validation con la MLP al ser probada con el test set y comparación con los resultados obtenidos con la CNN propuesta. Los valores se muestran con el formato "media (desvío estándar)".

En el caso de este modelo, al igual que con la CNN, las métricas empeoran levemente después de realizar 10-Fold Cross Validation. Las métricas que más cambiaron fueron la especificidad y el valor predictivo positivo, lo cual tiene sentido al ser ambas métricas impactadas

por el aumento del número de falsos positivos; por lo tanto, se puede concluir que el modelo de MLP propuesto tiene una leve susceptibilidad a clasificar sujetos sanos como enfermos, lo cual concuerda con las conclusiones obtenidas de la visualización de los datos previa a la selección de modelos. Debido a que existe un alto volúmen de datos que se superponen entre clases, las transformaciones aprendidas por la red no generan una separación de los datos suficiente como para superar esta dificultad inherente a los mismos. Una posible solución para este problema sería aumentar la cantidad de observaciones presentes en el dataset para que el modelo pueda generalizar mejor sobre los datos y aprender relaciones más complejas a partir de los datos. También podría ser útil el agregado de nuevas variables que sirvan para obtener más relaciones relevantes entre las variables. Por otro lado, en general se puede ver que las métricas del modelo indican que su desempeño en el problema tratado en la sección es muy bueno, reflejando una buena capacidad de detección de pacientes enfermos y de pacientes sanos (como indica el valor de ROC-AUC), y un buen rechazo a los falsos positivos y falsos negativos (como se puede ver con el valor del F1-Score)

3.6. Conclusiones

A partir de las teorías patogénicas de la enfermedad de Alzheimer se seleccionaron datos obtenidos del líquido cefalorraquídeo, sangre y de un test cognitivo, que contienen información acerca del desarrollo y estado de demencia de los pacientes, e indican su relación con la enfermedad de Alzheimer. Luego de seleccionar las variables relevantes para el estudio del problema de clasificación binario planteado en este trabajo, se buscaron relaciones entre dichas variables que permitan visualizar la capacidad de los datos para describir el problema. El resultado de la visualización de los datos indicó que el problema está bien descrito por los datos, aunque los mismos no sean linealmente separables, y que se puede distinguir entre las clases con cierto solapamiento entre ellas, lo cual podría contribuir al error a la hora de entrenar un modelo de clasificación. Teniendo estos resultados en cuenta se eligieron cuatro modelos de clasificación ampliamente adoptados para los problemas de clasificación binaria: Clasificador Naïve Bayes, Regresión Logística, Support Vector Machine y Multilayer Perceptron. Se identificaron los hiperparámetros fundamentales de dichos modelos y se seleccionó los que generan el modelo con mejor desempeño, y luego se compararon los resultados obtenidos por cada modelo para elegir el mejor y utilizarlo para esta parte del trabajo. El modelo que obtuvo mejores resultados fue la red neuronal MLP, la cual al ser entrenada y probada con el esquema de 10-Fold Cross Validation adoptado para este trabajo mostró tener buenos resultados en la distinción entre pacientes sanos y enfermos, aunque no logró superar los hipotéticos problemas planteados en la visualización de los datos causados por la naturaleza de los datos. De todas maneras, los resultados obtenidos indican que la MLP propuesta tiene un muy buen desempeño como clasificador de pacientes con enfermedad Alzheimer a partir de los biomarcadores utilizados.

Capítulo 4

Modelo Final

Luego de la introducción de los modelos para el análisis de los dos tipos de datos (imágenes y datos numéricos estructurados) utilizados para la resolución del problema planteado en este trabajo, se debe implementar una manera de combinar dichos modelos para obtener un resultado de clasificación global. Debido a que la CNN propuesta para el análisis de las resonancias magnéticas clasifica una imagen por vez, es necesario generar un esquema que permita el análisis de los 113 patches 2.5D obtenidos de cada resonancia a la vez y a partir de dicho análisis poder combinar los resultados obtenidos con los de la MLP de forma que se termine generando una clasificación unificada a partir de todos los datos disponibles.

4.1. Long Short-Term Memory

Las redes LSTM [61] son un tipo de redes neuronales recurrentes (RNN), las cuales son consideradas como el estado del arte para el análisis de datos secuenciales (lenguaje, audio, video, etc.) [62]. Una de las principales ventajas y atractivos de las RNNs es su capacidad para conectar la información previa a la tarea presente, como por ejemplo la información de "frames" anteriores de un video puede ayudar al entendimiento del frame actual. Las redes LSTM tienen esta capacidad al poder aprender dependencias de largo plazo entre los datos.

Todas las RNN tienen una estructura de capa en la que existe una disposición de repetición en cadena de módulos de red neuronal; las LSTM tiene esta estructura en cadena, pero con la particularidad de tener cuatro capas que interactúan de una manera muy particular.

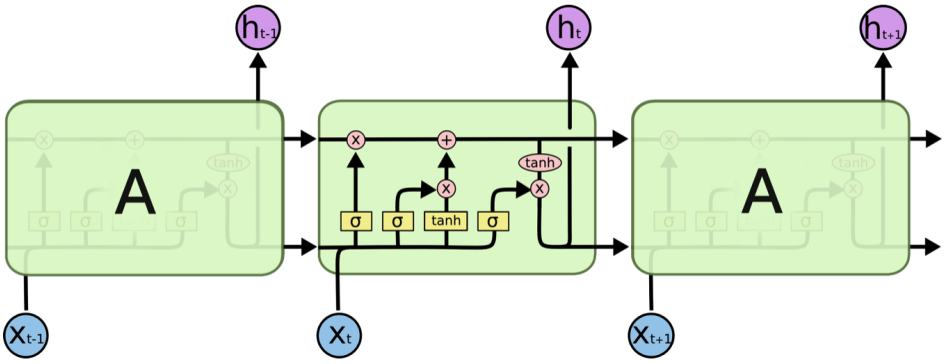


Figura 4.1: Estructura repetitiva de la LSTM con las cuatro capas que interactúan. En este diagrama cada línea conduce un vector completo, desde la salida de un nodo a la entrada de otro. Los círculos rosa representan operaciones punto a punto, como suma de vectores, y las cajas amarillas representan capas de red neuronal entrenables.[63]

La característica clave de las LSTM es el *estado de la celda*, representado en el diagrama como la línea horizontal en la parte superior del diagrama de la figura 4.2.

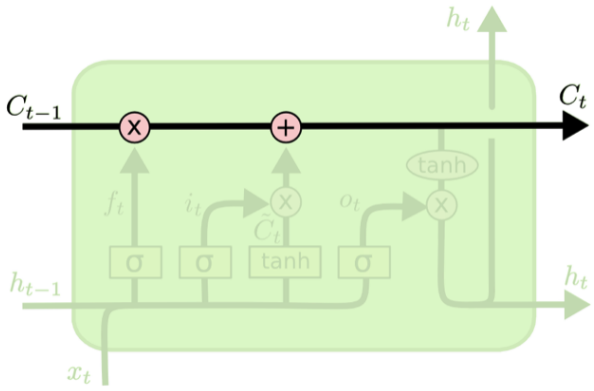


Figura 4.2: Estado de la celda. [63]

El estado de la celda cumple la función de transporte de información, mediante pocas interacciones lineales, de una celda a la otra. La LSTM introduce cambios en la información a través de estructuras denominadas portales o "gates". Los gates regulan qué información es introducida a la celda a partir de unas capas de activación sigmoideas (representadas como las cajas amarillas que contienen una letra σ) y una operación de multiplicación punto a punto. La función sigmoidea, como ya se explicó anteriormente, devuelve valores entre 0 y 1, describiendo así cuánta información se deja pasar al estado de la celda. La LSTM protege y controla el estado de la celda utilizando tres de estos gates.

Para explicar en más profundidad que sucede en cada módulo LSTM de divide el proceso en una serie de pasos. En el primer paso, se decide qué información se conserva o se descarta del estado de la celda. Esta decisión se hace con una capa sigmoidea denominada "forget gate layer" que observa la salida del bloque anterior, h_{t-1} y la entrada del bloque actual, x_t, y devuelve un número entre 0 y 1 (f_t en la figura 4.3) para cada valor en el estado de celda C_{t-1}.

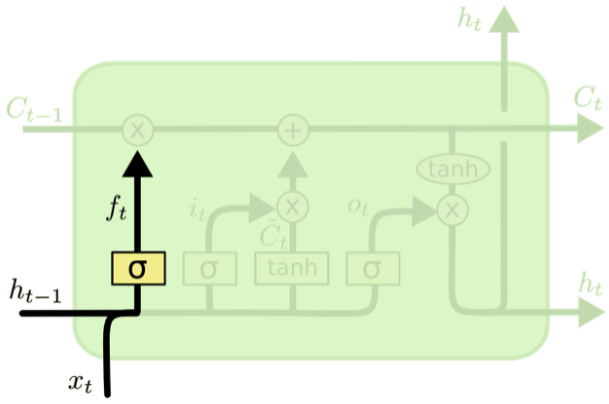


Figura 4.3: Utilización de información del estado de celda pasado. [63]

El siguiente paso consiste en decidir qué información será guardada en el estado actual de la celda, y tiene dos etapas: En primer lugar, una capa sigmoidea denominada "input gate layer" indica qué valores se actualizarán; luego, una capa con activación *tanh* (tangente hiperbólica) crea un vector de nuevos valores candidatos $\tilde{C}_t$ que podrían ser agregados al estado .

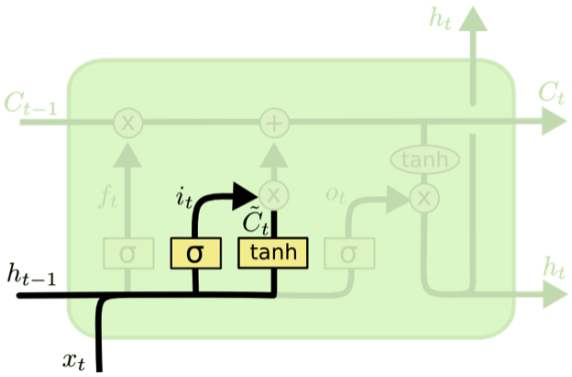


Figura 4.4: Generación de valores para actualizar el estado de la celda. [63]

Luego, se actualiza el viejo estado de celda, C_{t-1}, al nuevo estado C_t. Se multiplica punto a punto C_{t-1} con f_t, descartando la información de la salida anterior, y se suma el producto de punto a punto de i_t con $\tilde{C}_t$, los cuales son los candidatos a nuevos valores escalados por cuánto se decidió actualizar el estado.

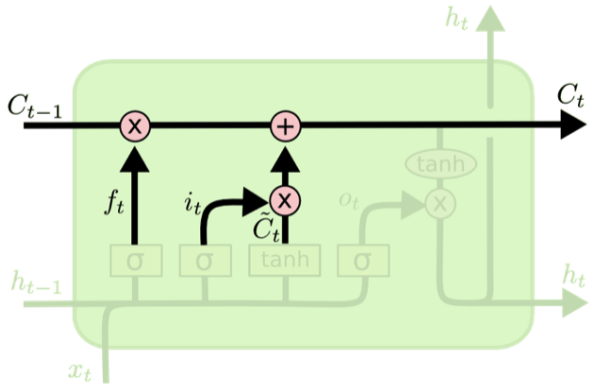


Figura 4.5: Actualización del estado de la celda. [63]

Finalmente, se decide cuál será la salida del módulo h_t. Esta salida estará basada en el estado de la celda, pero será una versión filtrada del mismo. Primero se utiliza una capa sigmoidea para decidir qué partes del estado de la celda serán incluidas en la salida. Luego, se hace pasar el estado de la celda a través de una capa tanh, que mapea los valores al intervalo $[-1, 1]$, y se lo multiplica punto a punto con la salida de la capa sigmoidea para devolver solamente las partes que se decidió incluir.

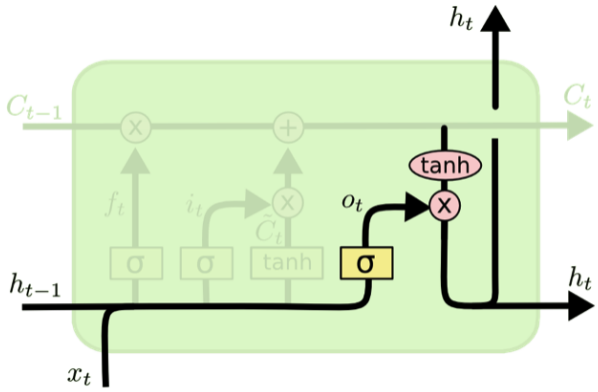


Figura 4.6: Generación de la salida de cada módulo LSTM. [63]

Como se puede ver, las capas LSTM sirven especialmente para el análisis de datos que se relacionan temporalmente entre sí y generan salidas utilizando la influencia de todos los pasos temporales de la entrada.

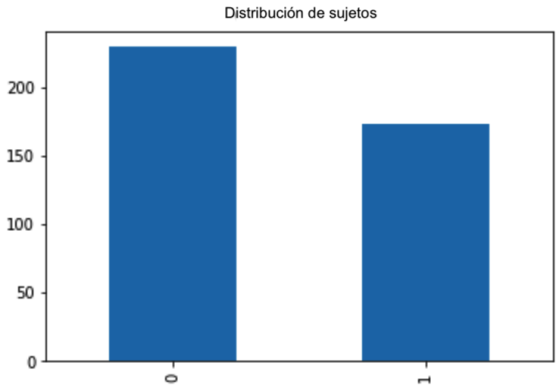
4.1.1. Aplicación en este trabajo

En este trabajo se generan 113 imágenes RGB (Figura 2.36) a partir de una resonancia magnética correspondiente a un paciente. Estas imágenes se obtienen a partir del mismo conjunto de puntos sobre la misma región espacial en todas las resonancias, y por lo tanto todos los conjuntos de patches 2.5D comparten una disposición espacial particular y un orden de generación específico; es decir, que por cada sujeto se genera una secuencia de

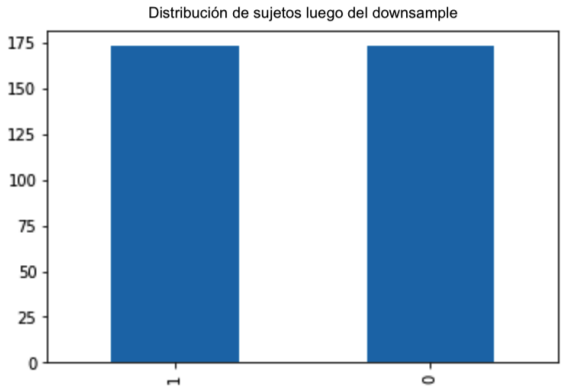
imágenes que describe la región de interés tomada de cada resonancia magnética. Aunque dicha secuencia de imágenes no siga un orden temporal, sino espacial, se busca explotar la relación existente entre ellas. Siguiendo esta línea de razonamiento se puede definir a cada imagen RGB generada a partir de las resonancias magnéticas como un **paso temporal** o **time step** de la secuencia de imágenes, y como tal se justifica el uso de una red LSTM para su análisis en conjunto. De esa manera, se soluciona el problema del uso de todas las resonancias de un sujeto junto con los biomarcadores en modelo que se propone en este trabajo.

4.2. Armado del dataset

Los datos utilizados en esta etapa final del trabajo son una combinación de las imágenes RGB, generadas en la sección de análisis de resonancias magnéticas, y los biomarcadores de líquido cefalorraquídeo, genotipado del gen APOE y el test MMSE. Como ya se explicó anteriormente, se disponen de 406 resonancias magnéticas (una por sujeto) y 917 instancias de los biomarcadores. Por lo tanto, se debe encontrar la intersección de ambos datasets. Al buscar los sujetos que tienen resonancias magnéticas y además todos los biomarcadores considerados en este análisis, se encontraron 402 sujetos de los cuales 229 (57 %) pertenecen a la clase AD y 173 (43 %) a la clase CN. Debido al desbalance de clases presente se decidió utilizar el enfoque de *random downsample* para obtener una distribución de sujetos balanceada. De esta manera, se eliminaron de manera aleatoria 56 sujetos pertenecientes a la clase AD y se obtuvo un total de 346 sujetos (173 de cada clase).



(a) Distribución de sujetos que tienen resonancia magnética y biomarcadores.



(b) Distribución de los sujetos luego de realizar random downsample.

Figura 4.7: Distribución de los sujetos que componen el dataset antes y después de balancear las clases.

Luego, se dividió los 346 sujetos en 279, 35 y 32 para los conjuntos de entrenamiento, validación y prueba cuidando que los sujetos incluidos en el conjunto de prueba hayan sido utilizados también para probar la CNN y la MLP en los capítulos anteriores. Lamentablemente, la poca cantidad de datos es una limitación en el análisis de este trabajo. Sin embargo, esta condición es una constante en los problemas de análisis de datos médicos y

existen trabajos que exploran uso de herramientas basadas en redes neuronales aún en casos con pocos datos obteniendo buenos resultados ([64],[65]).

4.3. Arquitectura del modelo CNN-LSTM-MLP

Para lograr implementar un modelo que pueda analizar la información de diferentes fuentes utilizada en este trabajo, se utilizó la *Functional API* de Keras [66] que permite definir una arquitectura compuesta por varios modelos y otorga flexibilidad para el manejo e interconexión de las capas utilizadas.

En cuanto al análisis de la secuencia de imágenes RGB mediante una RNN que utiliza capas LSTM, primero se debe establecer una forma de definir la secuencia de imágenes como una serie de pasos temporales. El manejo de pasos temporales se logra mediante la utilización de una "capa de envoltura" o *wrapper layer* llamada **Capa Time Distributed**. La capa Time Distributed aplica una capa de red neuronal, o un modelo, a cada paso temporal de la entrada. De esta manera, la entrada debe ser definida como un tensor cuya primera dimensión será considerada la dimensión temporal; es decir que por cada sujeto, la entrada compuesta por las imágenes RGB será un tensor de $113 \times 32 \times 32 \times 3$. Utilizando la capa Time Distributed de esta manera, se puede lograr que se aplique la CNN propuesta en este trabajo a cada paso temporal de la entrada y la salida de esta capa distribuida sea una secuencia numérica compuesta de unos y ceros que será analizada por una RNN compuesta de capas LSTM.

De esta manera, modelo que se plantea para la utilización en conjunto de todos los datos de cada sujeto es el siguiente:

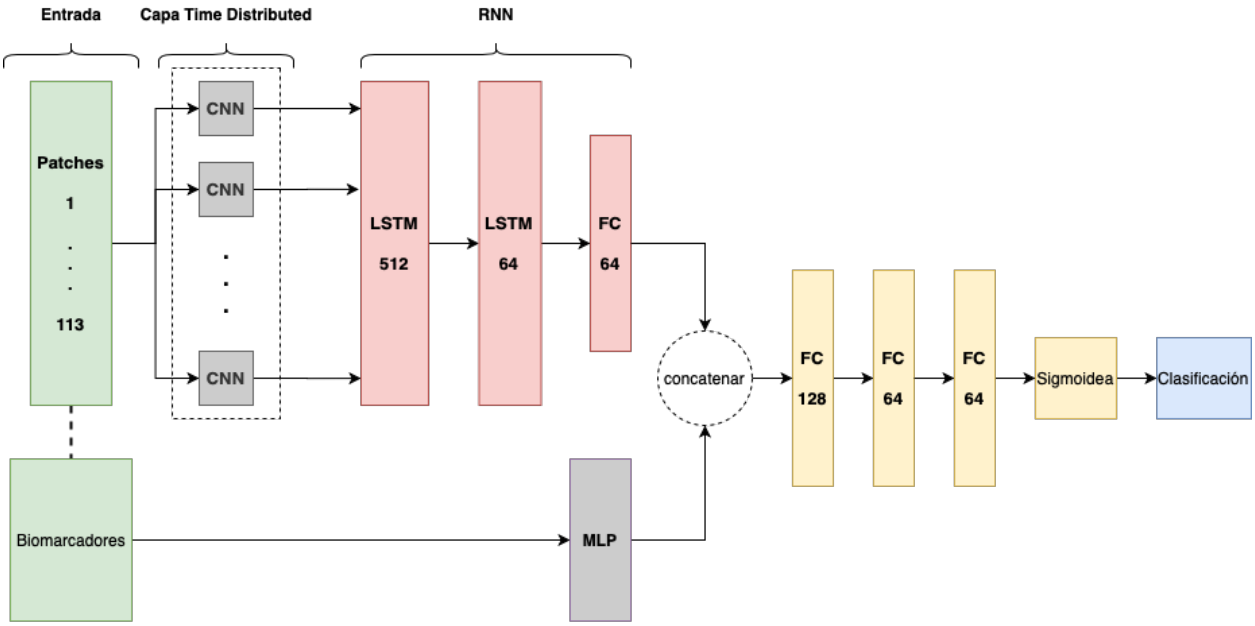


Figura 4.8: Arquitectura del modelo CNN-LSTM-MLP propuesto. Verde: Entradas; Gris: Modelos implementados anteriormente (CNN Time distributed y MLP) y configurados para no ser entrenables en este modelo compuesto; Rojo: RNN formada por dos capas LSTM y una FC; Amarillo: Capas FC anteriores a la salida; Azul: Clasificación

En la figura 4.8 se puede ver que el modelo final tiene dos entradas, las cuales corresponden a los 113 patches 2.5D por un lado y los biomarcadores por el otro. Por lo tanto, quedan definidas dos ramas principales cuyos resultados son finalmente concatenados para obtener un resultado final. Cabe destacar que en el modelo las redes CNN y MLP fueron configuradas para que sus parámetros no sean entrenables, y en el caso de la red MLP se quitó la capa sigmoidea para utilizar el mapa de características que ésta genera y combinarlo con el que se genera en la otra rama.

Este modelo se entrenó por 30 epochs, con batch size de 16 y utilizando como optimizador Adam con los parámetros default de Keras.

4.4. Resultados

Luego de entrenar el modelo propuesto, se realizó una prueba sobre el test set y se obtuvo la siguiente matriz de confusión:

$$\begin{bmatrix} 16 & 0 \\ 1 & 15 \end{bmatrix}$$

y por lo tanto, las métricas de evaluación calculadas fueron:

	Exactitud	**Sensibilidad**	**Especificidad**	**VPP**	**VPN**	**F1-Score**	**ROC-AUC**
Modelo propuesto	96.88 %	93.75 %	100 %	100 %	94.12 %	96.77 %	96.88 %

Tabla 4.1: Métricas obtenidas con el modelo propuesto al ser probada con el set de prueba.

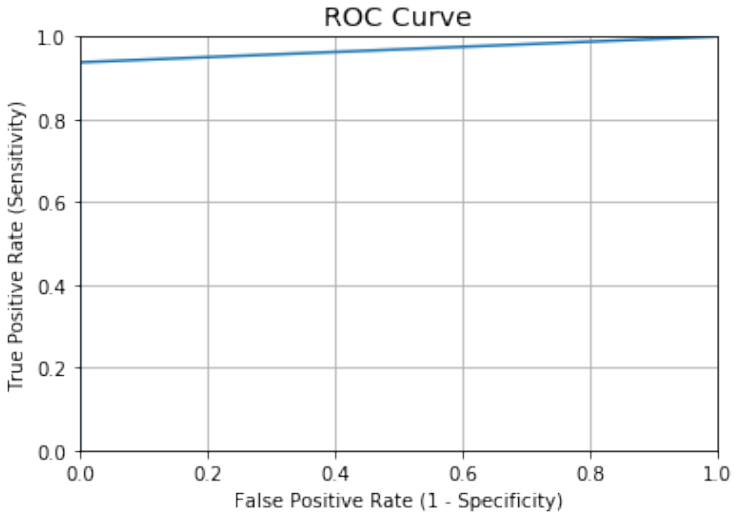


Figura 4.9: Curva ROC obtenida con el modelo final.

Como se puede ver, las métricas obtenidas muestran un gran desempeño del modelo sobre el conjunto de prueba. Sin embargo, al tener tan pocos valores para realizar la prueba, la clasificación errónea tiene una gran influencia sobre los valores de las métricas.

Por lo tanto, para evaluar de manera más confiable el desempeño del modelo, y siguiendo la modalidad utilizada en todo el trabajo, se utiliza el marco de evaluación mediante 10-Fold Cross-Validation que también fue usado en la CNN y la MLP. Además, utilizando esta forma de evaluación se va a poder visualizar de manera más generalizada el desempeño del modelo y el impacto de diferentes divisiones de los datos sobre el mismo, ya que al utilizar diferentes combinaciones de datos para entrenar y validar se va a poder visualizar el desempeño del modelo ante variaciones en los datos utilizados para entrenarlo. Los resultados obtenidos, y su comparación con los resultados de los otros modelos implementados a lo largo de este trabajo, se pueden ver en la siguiente tabla:

	Exactitud	**Sensibilidad**	**Especificidad**	**VPP**	**VPN**	**F1-Score**	**ROC-AUC**
Diagnóstico con el criterio de NIA-AA	-	80.9 %	94.0 %	-	-	-	-
CNN propuesta 10-Fold Cross Validation	92.8 (1.4) %	91.5 (1.7) %	94.1 (2.3) %	93.8 (2.2) %	92.5 (2.1) %	92.6 (1.9) %	92.8 (1.4) %
MLP propuesta 10-Fold Cross Validation	88.2 (0.4) %	90.3 (1.7) %	85.8 (2.7) %	87.9 (1.8) %	88.7 (1.5) %	89.1 (0.3) %	88.1 (0.6) %
CNN-LSTM-MLP 10-Fold Cross Validation	92.8 (2.8) %	94.4 (1.9) %	91.2 (5.7) %	91.8 (5.1) %	94.3 (5.3) %	93 (2.6) %	92.8 (2.8) %

Tabla 4.2: Métricas obtenidas al hacer 10-Fold Cross Validation con el modelo final al ser probado con el test set y comparación con los resultados obtenidos con los modelos CNN y MLP propuestos en el trabajo, y con las métricas publicadas en [11] acerca del criterio de diagnóstico clínico de NIA-AA. Los valores se muestran con el formato "media (desvío estándar)".

Cabe aclarar que en el análisis de los resultados que se realizará a continuación se reconoce la limitación generada por la diferencia en la cantidad de los datos empleados en cada proceso de entrenamiento y prueba, pero dada la limitada disponibilidad de los datos y la necesidad de extraer conclusiones del análisis realizado, se decide suponer que

los resultados obtenidos con los modelos propuestos en cada etapa son comparables entre sí. De esta manera, podemos ver que el modelo CNN-LSTM-MLP muestra un desempeño levemente mejor a la CNN, con exactitudes y ROC-AUC con valor medio iguales, pero con mayor desvío estándar en el modelo final (lo cual se atribuye a la poca cantidad de datos para la prueba). El modelo final muestra un aumento de la sensibilidad, valor predictivo negativo y mayor score F1, pero los valores de especificidad y valor predictivo positivo decrecen. El aumento en la sensibilidad promedio indica que el modelo CNN-LSTM-MLP tiene una mayor tendencia al rechazo de falsos negativos, lo cual es una ventaja importante para considerar la capacidad diagnóstica del modelo; el aumento en el VPN refleja esta tendencia de igual manera. El aumento en el score F1, pese al decrecimiento en el VPP (indicativo de una mayor susceptibilidad a clasificar falsos positivos), sugiere que el impacto del aumento de los falsos positivos se ve mitigado por el decrecimiento en los falsos negativos. El decrecimiento de la especificidad también indica el aumento en los falsos positivos, pero es necesario aclarar que el hecho de que los falsos positivos hayan aumentado, en el caso de la enfermedad de Alzheimer, es menos grave que si hubieran aumentado los falsos negativos ya que una persona sana que se diagnostica como enferma no tiene demasiadas consecuencias sobre su salud en el caso de la enfermedad de Alzheimer (salvo las cuestiones emocionales que tal diagnóstico implicaría para el paciente y su familia), pero una persona enferma que no se diagnostica como tal tiene asociado el riesgo de no comenzar a tratar la enfermedad a tiempo y perder tiempo valioso para detener o por lo menos intentar contener el deterioro cognitivo inminente. En líneas generales, el modelo final propuesto verifica la hipótesis tomada al principio de este trabajo, en la que se suponía que el uso en conjunto de los datos de diversas fuentes permite tener un panorama más amplio de la patología y generar un modelo que obtenga mejores resultados que aquellos que utilizan los datos de manera individual.

Capítulo 5

Conclusiones

El sistema A/T/N, modelo patogénico de la enfermedad de Alzheimer, describe el mecanismo por el cual la enfermedad se desarrolla a partir de biomarcadores relacionados con la proteína amiloide β, la proteína Tau y la neurodegeneración. La hipótesis principal de este trabajo es que el uso de tales biomarcadores en conjunto puede generar una herramienta de clasificación de sujetos sanos y sujetos enfermos que se nutra de todos los aspectos descritos por este modelo. Para evaluar tal hipótesis se definieron tres principales etapas de análisis en las que se utilizaron los biomarcadores contemplados por el sistema A/T/N.

Por un lado, en la primera etapa de análisis se consideró el uso de resonancias magnéticas estructurales para poder evaluar la neurodegeneración del paciente. Para poder utilizar las resonancias magnéticas de manera eficiente se implementó un procedimiento de preprocesamiento que genera una estandarización de las resonancias tanto espacial como de intensidades y permite una extracción de características de manera generalizada. Las características extraídas de las resonancias fueron en la forma de 113 imágenes RGB, llamadas patches 2.5D, por cada resonancia que contenían la información volumétrica 3D presente en la MRI en la zona del hipocampo izquierdo, zona de particular interés por su degeneración en la enfermedad. Se implementó y mejoró el desempeño la red neuronal convolucional propuesta en [24], la cual era utilizada para el análisis de imágenes RGB extraídas de manera similar a este trabajo. Con las mejoras introducidas en la CNN se obtuvieron valores de sensibilidad, especificidad, valor predictivo positivo y valor predictivo negativo, entre otras métricas, superiores al 90 % en promedio utilizando un esquema de 10-Fold Cross-Validation lo cual demuestró que el modelo obtenido tiene un gran desempeño para la tarea de clasificación planteada.

En la segunda etapa del análisis, se utilizaron un conjunto de biomarcadores cuya relevancia para la asistencia y complemento del diagnóstico clínico de la enfermedad de Alzheimer es ampliamente aceptado. Estos biomarcados son la concentración de la proteína amiloide β, Tau y p-Tau en el líquido cefalorraquídeo, el estatus de los alelos ϵ_4 del gen APOE y el MMSE, el cual es el test cognitivo más adoptado en la práctica para medir el estado de demencia de un paciente. A partir de estos biomarcadores se exploraron las relaciones existentes entre las variables que los describen, obteniendo así información importante para el desarrollo de esta etapa. En primer lugar, se verificaron las suposiciones teóricas acerca de la relación existente entre las proteínas amiloide β, Tau y p-Tau, y su capacidad para dividir a los pacientes en sanos y enfermos, observando que estas variables no generan separabilidad

lineal total entre las clases presentes en el problema. Luego, se analizó la relación de las variables con el estatus de APOE y el test cognitivo, y se consiguió verificar que la capacidad de dichos biomarcadores no es de carácter diagnóstico sino que se trata de biomarcadores que brindan asistencia y complementan al diagnóstico junto con las demás variables. Habiendo obtenido información acerca del comportamiento de las variables a analizar, se decidió buscar un método de clasificación que explote de la mejor manera posible las relaciones entre las variables. Se entrenó, optimizó los hiperparámetros y probó los modelos de clasificación de Naïve Bayes, Support Vector Machine, Regresión logística y una red neuronal multicapa (MLP). El modelo que mejor resultados obtuvo fue la red MLP, y se utilizó el mismo esquema de 10-Fold Cross-Validation para evaluar el desempeño del modelo de forma más rigurosa. Los resultados obtenidos fueron, en cierta medida, no tan buenos comos los obtenidos con la CNN, pero aún así superan al desempeño del diagnóstico clínico de referencia. El hecho de obtener tales resultados de evaluación de los métodos implementados indica que la hipótesis tomada acerca de la patogénesis y el desarrollo de la enfermedad descrito por el Sistema A/T/N tiene un impacto positivo en el problema tratado en este trabajo.

En la tercera, y última, etapa de este análisis se implementó un método que permite explotar los datos de manera conjunta y obtener un resultado de clasificación unificado que refleje el potencial del enfoque tomado en el diagnóstico de la enfermedad de Alzheimer. Para eso se implementó un modelo que, a partir de las 113 imágenes RGB y los biomarcadores, genera una clasificación final. Las imágenes RGB fueron consideradas como una secuencia cuya relación espacial contiene información que puede ser explotada mediante una RNN; se utilizó una capa de envoltura Time Distributed que considera cada imagen RGB como un paso temporal y le aplica la CNN implementada en la primera etapa a cada una para que la secuencia de números obtenida ingrese a una RNN compuesta de capas LSTM que procesan la relación secuencial subyacente. Por otro lado, se le aplica la red MLP (sin su capa de activación sigmoidea final) a los biomarcadores. Luego, las salidas de cada rama del modelo se concatenan y analizan por un conjunto de capas fully connected, y de esa manera se termina llegando a un resultado de clasificación final. Como se puede ver, el modelo propuesto utiliza de manera conjunta todos los datos y emplea los modelos desarrollados en las otras etapas como capas de extracción de características, las cuales son explotadas unificadamente para obtener un resultado final. El modelo fue evaluado con el mismo esquema de validación cruzada que los demás modelos para ser consistente en el análisis. Los resultados obtenidos indican que el modelo propuesto es el que mejor desempeño tiene, demostrando así que la hipótesis rectora del trabajo se cumple.

En cuanto a la comparación del desempeño del modelo propuesto en este trabajo con el desempeño del diagnóstico clínico utilizando el criterio más adoptado, se puede ver que se logró un mejor desempeño en cuanto a las métricas de sensibilidad y especificidad. Aunque el modelo de este trabajo haya demostrado tener una especificidad aproximadamente 3 % menor que el criterio de NIA-AA, es posible notar que en cuanto a la métrica de sensibilidad se obtuvo un desempeño ampliamente mejor. La métrica de sensibilidad del modelo CNN-LSTM-MLP propuesto fue de 94.4 (1.9) %, mientras que la sensibilidad reportada para NIA-AA fue del 80.9 %. Por lo tanto, en términos generales se consiguió que el modelo propuesto tenga un desempeño mejor que el reportado para el criterio de diagnóstico clínico actualmente adoptado.

Finalmente, los resultados obtenidos a lo largo de todo el trabajo sugieren que una

herramienta de diagnóstico de la enfermedad de Alzheimer que explote información de varias fuentes del paciente de manera unificada es posible de realizar, y que con la tendencia de los trabajos de investigación, el descubrimiento de nuevas técnicas de análisis de muestras de líquido cefalorraquídeo y la obtención de más datos, se podría llegar a lograr obtener modelos robustos y validados para su uso en el ámbito clínico.

Capítulo 6

Trabajos a futuro

Una de las principales limitaciones de este trabajo fue la poca disponibilidad de datos que cumplan con los criterios de inclusión adoptados. Esta limitación se debe a que los datos necesarios para el desarrollo del proyecto sólo pudieron ser obtenidos de una sola base de datos, ADNI, ya que a pesar de haber intentado conseguir acceso a otras fuentes, no se pudo utilizar los datos provenientes de ellas ya que las instituciones a las que pertenecen no otorgan acceso sin tener una beca de investigación, o simplemente los protocolos de obtención de los biomarcadores no eran compatibles con los de ADNI. Por lo tanto, como principal mejora a futuro se propone ampliar el dataset. Con el surgimiento de nuevos avances en tecnología médica (como la reciente aprobación por la FDA de un resonador magnético portátil) que simplifican la obtención de datos de manera masiva para su explotación, y con la correspondiente implementación de protocolos de estandarización de las técnicas de laboratorio para el manejo de las muestras de líquido cefalorraquídeo que aseguren la transferibilidad de los resultados obtenidos en diversas pruebas clínicas para la detección de biomarcadores, sería posible generar grandes datasets con información estandarizada obtenida de múltiples fuentes.

Luego de ampliar el dataset, sería posible redefinir los modelos propuestos para que éstos sean más profundos, puedan extraer características más complejas de los datos y obtener resultados más confiables y aplicables en un entorno clínico. Además, se podría explorar la inclusión de más regiones de interés del cerebro para la extracción de patches 2.5D, ampliando así el alcance del análisis de la neurodegeneración del cerebro; por otro lado, se podría incluir sujetos MCI en el análisis e investigar acerca de la aplicabilidad del modelo para el problema de clasificación de sujetos en normales, MCI y con demencia.

Otra mejora a futuro sería la posibilidad de utilizar datos obtenidos en el ámbito hospitalario para probar el funcionamiento del modelo con datos no obtenidos para la investigación, sino para ser utilizados para el diagnóstico de los pacientes. De esa manera, el modelo sería probado en un marco de aplicación real y se podría ver la capacidad diagnóstica verdadera del mismo.

Apéndice A

Código

A.1. Pipeline de preprocesamiento

A.1.1. Paralelización del proceso

A continuación se presenta el código desarrollado para implementar el pipeline de preprocesamiento mostrado en la figura 2.27. El siguiente extracto de código implementa los pasos de estandarización espacial, corrección de inhomogeneidades de intensidad, registración y extracción de cerebro.

```
import os
import time
import glob
import numpy as np
import nibabel as nb
import ants
from nipype.interfaces.fsl import BET
import concurrent.futures


def preprocess_ants(img_path,template_path, transform="SyN"):
    """
    Pasos del pipeline ANTsPy:

        - Resampleo a 1x1x1 mm^3
        - Bias Field Correction usando N4
        - Registracion con el metodo SyN al Template MNI ICBM 152

    Genera un archivo .nii con la salida.

    :param img_path (String). Path de la imagen de entrada.
    :param template_path (String). Path del template utilizado para la registracion.
    :param transform (String). Tipo de transformacion aplicada en la registracion. Default: "SyN"

    :returns img_prep_ants. ANTS image. Salida de estos pasos del pipeline.
```

```
    """

    img = ants.image_read(img_path,reorient=True)
    img = ants.resample_image(img,(1, 1, 1),interp_type=4)
    img = ants.n4_bias_field_correction(img)

    template = ants.image_read(template_path,reorient=True)

    mytx = ants.registration(fixed=template, moving=img, type_of_transform
    =  transform)
    img_prep_ants = mytx["warpedmovout"]

    return img_prep_ants

def skullStripping(img_path,bet_path,robust = True, f = 0.4):

    """
    Paso de extraccion de cerebro del pipeline.

        - Robust
        - f = 0.4

    Genera un archivo .nii con la salida.

    :param img_path(String). Path de la imagen de entrada.
    :param bet_path (String). Path de la imagen de salida.
    :param robust (Boolean). Default = True
    :param f = (Float). Umbral para BET. Default = 0.4

    :returns img_bet:  Path del archivo .nii luego de la extraccion del
    cerebro.
    """

    input_file = img_path
    output_file = bet_path
    bet = BET()
    bet.inputs.robust = robust
    bet.inputs.frac = f
    bet.inputs.in_file = input_file
    bet.inputs.out_file = output_file
    bet.inputs.output_type = "NIFTI"
    res = bet.run()
    res.outputs


def pipeline(img_path):

  """
  Funcion que ejecuta el pipeline de preprocesamiento para una imagen.
  Utilizada para paralelizacion del proceso.

  Genera un archivo .nii con la salida del proceso.

```

```
77    :param img_path: String. Path de la resonancia magnetica a la cual se le
        aplica el pipeline.
78    """
79
80    cwd = "/volumes/JEH/datos-proyecto-final"
81    mri_folder_name = "MRI"
82    mri_path = os.path.join(cwd,mri_folder_name)
83
84    template = "/Users/jonathanhasbani/Documents/proyecto-final-alzheimer/
       data/MNI-ICBM-152/mni_icbm152_t1_tal_nlin_sym_09a.nii"
85    temp_folder_path = os.path.join("/Users/jonathanhasbani/Documents/
       proyecto-final-alzheimer/preprocessing","temp-preprocess")
86    out_folder_path = os.path.join(cwd,"MRI-preprocessed")
87    out_ad_folder_path = os.path.join(out_folder_path, "AD")
88    out_cn_folder_path = os.path.join(out_folder_path, "CN")
89
90    if os.path.isdir(out_folder_path) == False:
91        os.mkdir(out_folder_path)
92        os.mkdir(out_ad_folder_path)
93        os.mkdir(out_cn_folder_path)
94    else:
95        None
96
97    if os.path.isdir(temp_folder_path) == False:
98        os.mkdir(temp_folder_path)
99    else:
100       None
101
102   print("Preprocesando el archivo %s" % img_path[len(mri_path+"/")+3:])
103   img_prep_ants = preprocess_ants(img_path,template, transform = "SyN")
104   img_temp_path = os.path.join(temp_folder_path,"temp_"+img_path[len(
       mri_path+"/")+3:])
105   ants.image_write(img_prep_ants,img_temp_path)
106
107   if "/AD/" in img_path:
108       out_file_path = os.path.join(out_ad_folder_path,img_path[len(
       mri_path+"/")+3:])
109   else:
110       out_file_path = os.path.join(out_cn_folder_path,img_path[len(
       mri_path+"/")+3:])
111
112   skullStripping(img_temp_path,out_file_path)
113   os.unlink(img_temp_path)
114
115   return out_file_path
116
117# Obtencion de los paths de las resonancias.
118cwd = "/volumes/JEH/datos-proyecto-final"
119mri_folder_name = "MRI"
120mri_path = os.path.join(cwd,mri_folder_name)
121files = glob.glob(mri_path+"/*/*.nii")
122
123workers = 4 # Numero de nucleos del procesador utilizados.
124
```

```
# Ejecucion del pipeline en paralelo utilizando concurrencia.
start = time.time()
with concurrent.futures.ProcessPoolExecutor(max_workers = workers) as
    executor:
     print("Preprocesando...")
     executor.map(pipeline, files, chunksize = len(files)//workers)
end = time.time()
time_elapsed = end-start
print("Tiempo total: %.2f segundos" % time_elapsed)
```

A.1.2. Normalización de intensidades

Como paso siguiente, se muestra la implementación del paso de normalización de intensidades del pipeline:

```
from intensity_normalization.normalize import nyul
import nibabel as nb
import os
import glob
import time
import numpy as np
import pandas as pd

# Paths a los archivos
working_dir = "/volumes/JEH/datos-proyecto-final/"
mri_dir = os.path.join(working_dir,"MRI-preprocessed")

AD = os.path.join(mri_dir,"AD")
CN = os.path.join(mri_dir,"CN")

ad_files = glob.glob(AD+"/*.nii")
cn_files = glob.glob(CN+"/*.nii")
files = glob.glob(mri_dir+"/*/*.nii")

# Datos de los ID de cada sujeto
mri_df = pd.read_csv("MRI_subjects_labels.csv")
subjects = mri_df.Subject.values.tolist()

# Entrenamiento del metodo de Nyul y Udupa
start = time.time()
print("Entrenamiento del metodo:")
standard_scale,percs = nyul.train(files)
end = time.time()
train_time = end-start
print("Fin. {} segundos".format(train_time))

# Se guarda la escala estandar luego del entrenamiento para futuros
    trabajos
standard_hist = [standard_scale,percs]
np.save("standard_hist",standard_hist,allow_pickle=True)
standard_hist = np.load("standard_hist.npy")
standard_scale = standard_hist[0]
percs = standard_hist[1]
```

```
38
39 # Proceso de normalizacion
40 start = time.time()
41 norm_out_path = "/volumes/JEH/datos-proyecto-final/MRI-Nyul-Udupa"
42 ad_path = os.path.join(norm_out_path,"AD")
43 cn_path = os.path.join(norm_out_path,"CN")
44 if os.path.isdir(norm_out_path)==False:
45     os.mkdir(norm_out_path)
46     os.mkdir(ad_path)
47     os.mkdir(cn_path)
48 else:
49     None
50
51 for i,file in enumerate(ad_files):
52
53     if i % 50 == 0:
54         print("{}/{}".format(i,len(ad_files)))
55
56     filename = file[61:]
57     subject = file[61:71]
58     if subject in subjects:
59         img = nb.load(file)
60         img_norm = nyul.do_hist_norm(img,percs,standard_scale)
61         nb.save(img_norm,os.path.join(ad_path,filename))
62
63 for file in cn_files:
64
65     if i % 50 == 0:
66         print("{}/{}".format(i,len(cn_files)))
67
68     filename = file[61:]
69     subject = file[61:71]
70     if subject in subjects:
71         img = nb.load(file)
72         img_norm = nyul.do_hist_norm(img,percs,standard_scale)
73         nb.save(img_norm,os.path.join(cn_path,filename))
74
75 end = time.time()
76 time_elapsed = end-start
77 print("Tiempo total: {} segundos".format(time_elapsed))
```

A.1.3. Extracción de patches 2.5D

Luego, se presenta la implementación de la extracción de los patches 2.5D:

```
1 import glob
2 import matplotlib.pyplot as plt
3 import numpy as np
4 import os
5 import time
6 import pandas as pd
7
8 def load_nifti(path):
```

```
    """
    Funcion que carga una resonancia magnetica NIFTI y devuelve un tensor con los datos.

    :param path (String). Path de la resonancia magnetica de entrada.
    :returns data (numpy ndarray). Tensor con los datos de la resonancia magnetica.
    """
    img = nb.load(path)
    data = img.get_fdata()

    if len(data.shape) == 4:
        data = data[:,:,:,0]

    return data

def get_rgb(mri, p):

    """
    Funcion que obtiene el patch 2.5D de la resonancia dado un punto de la region de interes.

    :param mri (numpy ndarray). Tensor con los datos de la resonancia magnetica.
    :param p (list). Punto de la region de interes.

    :returns mri_rgb (numpy ndarray). Patch 2.5D.
    """
    sag = np.rot90(mri[p[0],p[1]-16:p[1]+16,p[2]-16:p[2]+16]).astype("float32")
    cor = np.rot90(mri[p[0]-16:p[0]+16,p[1],p[2]-16:p[2]+16]).astype("float32")
    trv = np.rot90(mri[p[0]-16:p[0]+16,p[1]-16:p[1]+16,p[2]]).astype("float32")

    mri_rgb = np.zeros((32,32,3),np.float32)
    mri_rgb[:,:,0] = norm(trv)
    mri_rgb[:,:,1] = norm(cor)
    mri_rgb[:,:,2] = norm(sag)

    return mri_rgb

# Paths de los archivos
working_dir = "/volumes/JEH/datos-proyecto-final/"
mri_dir = os.path.join(working_dir,"MRI-Nyul-Udupa")

ad_norm_path = os.path.join(mri_dir,"AD")
cn_norm_path = os.path.join(mri_dir,"CN")

ad_files = glob.glob(ad_norm_path+"/*.nii")
cn_files = glob.glob(cn_norm_path+"/*.nii")

#Paths de salida de los patches
out_path = os.path.join(working_dir,"Patches")
```

```
rgb_ad_path = os.path.join(out_path,"AD")
rgb_cn_path = os.path.join(out_path,"CN")

if os.path.isdir(out_path)==False:
    os.mkdir(out_path)
    os.mkdir(rgb_ad_path)
    os.mkdir(rgb_cn_path)
else:
    None

# Cargo los puntos de la region de interes
indexes = np.load("indexes.npy")

# Datos de los sujetos para asociar ID con clase.
mri_df = pd.read_csv("MRI_subjects_labels.csv")
mri_subjects = mri_df.Subject.values.tolist()

# Obtencion de los patches 2.5D y almacenamiento como archivos .npy
start = time.time()
for file in ad_files:
    subject = file[107:117]
    if subject in mri_subjects:
        mri = load_nifti(file)
        for n,ind in enumerate(indexes):
            rgb = get_rgb(mri, ind)
            np.save(rgb_ad_path+"/"+subject+"_patch_"+str(n),rgb,
   allow_pickle=True)
    else:
        print(subject)
end = time.time()
t_el = end - start
print("Tiempo: {}".format(t_el))

start = time.time()
for file in cn_files:
    subject = file[107:117]
    if subject in mri_subjects:
        mri = load_nifti(file)
        for n,ind in enumerate(indexes):
            rgb = get_rgb(mri, ind)
            np.save(rgb_cn_path+"/"+subject+"_patch_"+str(n),rgb,
   allow_pickle=True)
    else:
        print(subject)
end = time.time()
t_el = end - start
print("Tiempo: {}".format(t_el))
```

A.1.4. Formación del dataset en formato HDF5

Finalmente, veamos la formación del dataset en el formato comprimido .h5:

```
import os
```

```
import glob
import time
import h5py
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split


def get_patches(subject,path):
    """
    Funcion que obtiene todos los patches 2.5D para un determinado sujeto.

    :param subject (String). ID del sujeto.
    :param path (list). Lista de paths a los archivos .npy que contienen
   los patches 2.5D del sujeto.

    :return patches (list). Lista con los patches 2.5D del sujeto.
    """
    patches = []
    for p in path:
        if subject in p:
            patches.append(np.load(path))
    return patches

def shuffle(x,y):
    """
    Funcion que dados dos listas las mezcla manteniendo la relacion
   ordinal entre si.

    :param x (list).
    :param y (list).

    :returns x (list). Lista mezclada.
    :returns y (list). Lista mezclada.
    """
    z = list(zip(x,y))
    random.shuffle(z)
    x,y = zip(*z)
    return x,y


# Defino el directorio de trabajo y los paths a los archivos necesarios
working_dir = "/volumes/JEH/datos-proyecto-final"
mri_dir = os.path.join(working_dir,"MRI-Nyul-Udupa")
rgb_dir = os.path.join(working_dir,"Patches")

mri_ad_path = os.path.join(mri_dir,"AD")
mri_cn_path = os.path.join(mri_dir,"CN")

rgb_ad_path = os.path.join(rgb_dir,"AD")
rgb_cn_path = os.path.join(rgb_dir,"CN")

# Creo lista con los paths a los patches
```

```
rgb_patches = glob.glob(rgb_dir+"/*/*.npy")


# Cargo un archivo .csv que contiene informacion acerca del
# ID, clase, edad y genero de los sujetos.
mri_df = pd.read_csv("mri_dataset.csv")
subjects = mri_df["Subject"].values.tolist()
group = mri_df["Group"].values.tolist()
labels = [0 if g == "CN" else 1 for g in group]

# Submuestreo de los sujetos para igualar la cantidad de sujetos en cada clase.
df_dataset = pd.DataFrame(list(zip(subjects,labels)), columns = ["subject","target"])

count_class_0, count_class_1 = df_dataset.target.value_counts()

df_class_0 = df_dataset[df_dataset["target"] == 0]
df_class_1 = df_dataset[df_dataset["target"] == 1]

df_class_0_under = df_class_0.sample(count_class_1)
df_dataset_under = pd.concat([df_class_0_under, df_class_1], axis=0)

print("Train set Random under-sampling:")
print(df_dataset_under.target.value_counts())

subjects_under = df_dataset_under.subject.values.tolist()
labels_under  = df_dataset_under.target.values.tolist()

# Division del dataset en Train_1 y Test
X_train_1,X_test,y_train_1,y_test = train_test_split(subjects_under,
                                                     labels_under,
                                                     stratify = labels_under,
                                                     test_size=0.2,
                                                     random_state=42
                                                     )

# Division de Train_1 en Train y Validation
X_train,X_valid,y_train,y_valid = train_test_split(X_train_1,
                                                   y_train_1,
                                                   stratify = y_train_1,
                                                   test_size=0.1,
                                                   random_state=42
                                                  )


# Obtengo los patches para los sets de entrenamiento, validacion y prueba.
patches_train = []
labels_train = []
for i,subject in enumerate(X_train):
    patches = get_patches(subject,rgb_patches)
    for patch in patches:
        patches_train.append(patch)
```

```
        labels_train.append(y_train[i])

patches_valid = []
labels_valid = []
for i,subject in enumerate(X_valid):
    patches = get_patches(subject,rgb_patches)
    for patch in patches:
        patches_valid.append(patch)
        labels_valid.append(y_valid[i])

patches_test = []
labels_test = []
for i,subject in enumerate(X_test):
    patches = get_patches(subject,rgb_patches)
    for patch in patches:
        patches_test.append(patch)
        labels_test.append(y_test[i])


# Mezclo aleatoriamente los sets
patches_train,labels_train = shuffle(patches_train,labels_train)
patches_valid,labels_valid = shuffle(patches_valid,labels_valid)
patches_test,labels_test = shuffle(patches_test,labels_test)


# Defino las dimensiones de los sets
train_shape = (len(patches_train), 32, 32, 3)
valid_shape = (len(patches_valid), 32, 32, 3)
test_shape = (len(patches_test), 32, 32, 3)

#Creo el archivo h5
hdf5_path = os.path.join(os.getcwd(),"dataset_CNN.h5")
f = h5py.File(hdf5_path, mode="w")

f.create_dataset("X_train", train_shape, np.float32)
f.create_dataset("X_valid", valid_shape, np.float32)
f.create_dataset("X_test", test_shape, np.float32)

f.create_dataset("Y_train", (len(labels_train),), np.uint8)
f["Y_train"][...] = labels_train
f.create_dataset("Y_valid", (len(labels_valid),), np.uint8)
f["Y_valid"][...] = labels_valid
f.create_dataset("Y_test", (len(labels_test),), np.uint8)
f["Y_test"][...] = labels_test

# Creo los sets en el archivo h5
for i,patch in enumerate(patches_train):

    if i % 10000 == 0 and i > 1:
        print ("Train data: {}/{}".format(i, len(patches_train)))

    f["X_train"][i,...] = patch[None]

for i,patch in enumerate(patches_valid):
```

```
159
160     if i % 1000 == 0 and i > 1:
161         print ("Validation data: {}/{}".format(i, len(patches_valid)))
162
163     f["X_valid"][i,...] = patch[None]
164
165 for i,patch in enumerate(patches_test):
166
167     if i % 1000 == 0 and i > 1:
168         print ("Test data: {}/{}".format(i, len(patches_test)))
169
170     f["X_test"][i,...] = patch[None]
171
172
173 f.close()
174
175 print("Done.")
```

A.2. Red neuronal convolucional

A.2.1. Función auxiliar

Función implementada para el cálculo de las métricas de evaluación del modelo. Se utiliza la misma función para la MLP de los biomarcadores y para el modelo final CNN-LSTM-MLP.

```
1  import sklearn.metrics as metrics
2
3  def get_model_metrics(y_true,y_pred):
4    """
5    Funcion que dada la prediccion y los valores verdaderos genera las
      metricas de evaluacion.
6
7    :param y_true (Numpy ndarray). Valores verdaderos.
8    :param y_pred (Numpy ndarray). Valores predichos.
9
10   :return metrics_array (list). Lista de las metricas de evaluacion.
11   """
12
13   # Matriz de confusion
14   confusion = metrics.confusion_matrix(y_true,y_pred)
15
16   TP = confusion[1, 1]
17   TN = confusion[0, 0]
18   FP = confusion[0, 1]
19   FN = confusion[1, 0]
20
21   # Calculo de metricas
22   accuracy_score = (TP + TN) / float(TP + TN + FP + FN)
23   classification_error = (FP + FN) / float(TP + TN + FP + FN)
24   sensitivity = TP / float(FN + TP)
25   specificity = TN / (TN + FP)
```

```
  precision = TP / float(TP + FP)
  vpn = TN / float(TN + FN)
  F1_score = 2*(precision*sensitivity)/(precision+sensitivity)
  ROC_AUC_score = metrics.roc_auc_score(y_true,y_pred)

  metrics_array = list((accuracy_score,classification_error,sensitivity,
   specificity,precision,vpn,F1_score,ROC_AUC_score))
  return metrics_array
```

A.2.2. Prueba del modelo propuesto por Tong et al.

```
import os
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,Activation
from keras.models import Model, Sequential
from keras.optimizers import SGD
from keras.callbacks import ModelCheckpoint
import h5py

# Defino los paths importantes
working_dir = "/content/gdrive/My Drive/Colab/CNN_2D"
dataset_dir = os.path.join(working_dir,"dataset_CNN.h5")

# Cargo el dataset
dataset = h5py.File(dataset_dir, "r")

#Definicion de algunos hiperparametros del modelo
IMAGE_SHAPE = (32,32,3)
DENSE_LAYER_ACTIVATION = "sigmoid"
OBJECTIVE_FUNCTION = "binary_crossentropy"

LOSS_METRICS = ["binary_accuracy"]
sgd = SGD(lr=0.001, decay=0.0001, momentum=0.9, nesterov=True)

BATCH_SIZE = 16
NUM_EPOCHS = 50

#Definicion del modelo
model = Sequential(name="CNN paper")
model.add(Conv2D(32, (5, 5), input_shape=IMAGE_SHAPE, padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, (5, 5),padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, (5, 5),padding = "same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(3, 3)))
```

```
42
43 model.add(Flatten())
44 model.add(Dense(64))
45 model.add(Activation("relu"))
46 model.add(Dense(1))
47 model.add(Activation("sigmoid"))
48
49 # Resumen del modelo definido
50 model.summary()
51
52 # Checkpointing para guardar el mejor modelo
53 checkpoint_dir = os.path.join(working_dir,"cnn_paper")
54 cb_checkpointer = ModelCheckpoint(filepath = os.path.join(checkpoint_dir,"
       {epoch:02d}-{val_loss:.2f}.hdf5"),
55                                   monitor = "val_loss",
56                                   save_best_only = True,
57                                   mode = "auto")
58
59 #Compilacion del modelo
60 model.compile(optimizer =sgd,
61               loss = OBJECTIVE_FUNCTION,
62               metrics = LOSS_METRICS)
63
64 # Entrenamiento del modelo
65 history = model.fit(
66   x = np.array(dataset["X_train"]),
67   y = np.array(dataset["Y_train"]),
68   batch_size = BATCH_SIZE,
69   epochs = NUM_EPOCHS,
70   validation_data = [np.array(dataset["X_valid"]),np.array(dataset["
       Y_valid"])],
71   callbacks=[cb_checkpointer])
72
73 # Curvas de entrenamiento
74 # Accuracy
75 plt.plot(history.history["binary_accuracy"])
76 plt.plot(history.history["val_binary_accuracy"])
77 plt.title("model accuracy")
78 plt.ylabel("accuracy")
79 plt.xlabel("epoch")
80 plt.legend(["train", "valid"], loc="upper left")
81 plt.show()
82 # Loss
83 plt.plot(history.history["loss"])
84 plt.plot(history.history["val_loss"])
85 plt.title("model loss")
86 plt.ylabel("loss")
87 plt.xlabel("epoch")
88 plt.legend(["train", "valid"], loc="upper left")
89 plt.show()
```

A.2.3. CNN con las mejoras propuestas

```
1 import os
```

```
import matplotlib.pyplot as plt
import numpy as np
import keras
from keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,Activation,
    Dropout,BatchNormalization
from keras.models import Model, Sequential
from keras.callbacks import ModelCheckpoint
import h5py


# Defino los paths importantes
working_dir = "/content/gdrive/My Drive/Colab/CNN_2D"
dataset_dir = os.path.join(working_dir,"dataset_CNN.h5")
models_dir = os.path.join(working_dir,"models")

# Cargo el dataset
dataset = h5py.File(dataset_dir, "r")

#Definicion de algunos hiperparametros del modelo
IMAGE_SHAPE = (32,32,3)
DENSE_LAYER_ACTIVATION = "sigmoid"
OBJECTIVE_FUNCTION = "binary_crossentropy"

LOSS_METRICS = ["binary_accuracy"]

BATCH_SIZE = 16
NUM_EPOCHS = 50

#Definicion del modelo con regularizacion (Batch normalization y Dropout)
model = Sequential(name="CNN Propuesta")
model.add(Conv2D(32, (5, 5), input_shape=IMAGE_SHAPE, padding = "same"))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, (5, 5),padding = "same"))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Conv2D(64, (5, 5),padding = "same"))
model.add(Activation("relu"))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(3, 3)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature
    vectors
model.add(Dense(64))
model.add(Activation("relu"))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation("sigmoid"))

checkpoint_dir = os.path.join(working_dir,"cnn_propuesta")
```

```
cb_checkpointer = ModelCheckpoint(filepath = os.path.join(checkpoint_dir,"{epoch:02d}-{val_loss:.2f}.hdf5"),
                                  monitor = "val_loss",
                                  save_best_only = True,
                                  mode = "auto")

#Compilacion del modelo
model.compile(optimizer = "adam",
              loss = OBJECTIVE_FUNCTION,
              metrics = LOSS_METRICS)

# Guardo el modelo sin entrenar
model.save(os.path.join(models_dir,"CNN_empty.h5"))

# Entrenamiento del modelo
history = model.fit(
  x = np.array(dataset["X_train"]),
  y = np.array(dataset["Y_train"]),
  batch_size = BATCH_SIZE,
  epochs = NUM_EPOCHS,
  validation_data = [np.array(dataset["X_valid"]),np.array(dataset["Y_valid"])],
  callbacks=[cb_checkpointer])

#Curvas de entrenamiento
#Accuracy
plt.plot(history.history["binary_accuracy"])
plt.plot(history.history["val_binary_accuracy"])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper left")
plt.show()
# Loss
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper left")
plt.show()

# Evaluacion del modelo con el Test set
y_pred = model.predict(dataset["X_test"],batch_size=1)
y_pred = [np.round(value)[0].astype(np.uint8) for value in y_pred]
evaluation_metrics = model_metrics(dataset["Y_test"],y_pred)

print("Accuracy: {}".format(evaluation_metrics[0]))
print("Error de Clasificacion: {}".format(evaluation_metrics[1]))
print("Sensibilidad: {}".format(evaluation_metrics[2]))
print("Especificidad: {}".format(evaluation_metrics[3]))
print("Precision (VPP): {}".format(evaluation_metrics[4]))
print("VPN: {}".format(evaluation_metrics[5]))
print("F1-Score: {}".format(evaluation_metrics[6]))
```

```
106 print("ROC AUC Score: {}".format(evaluation_metrics[7]))
107
108
109 # Guardo el modelo entrenado
110 model.save(os.path.join(models_dir,"CNN.h5"))
```

A.2.4. 10-Fold Cross Validation

```
1  import os
2  import h5py
3  import keras
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from keras.callbacks import ModelCheckpoint
7  from sklearn.model_selection import StratifiedKFold
8  from keras.models import Model, Sequential, load_model
9  from keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,Activation,
       Dropout,BatchNormalization
10
11 def get_checkpointer(name_weights):
12   """
13   Funcion que crea un checkpointer para cada fold
14   :param name_weights (String). Nombre del fold.
15   :return checkpointer (Object). Callback para hacer checkpoint del modelo
       para un fold.
16   """
17   checkpointer = ModelCheckpoint(filepath = os.path.join(kfold_dir,
       name_weights),
18                                  monitor = "val_loss",
19                                  save_best_only = True,
20                                  mode = "auto")
21   return checkpointer
22
23
24 # Defino los paths importantes
25 working_dir = "/content/gdrive/My Drive/Colab/CNN_2D"
26 dataset_KFold_dir = os.path.join(working_dir,"dataset_KFold.h5")
27 model_dir = os.path.join(working_dir,"cnn_propuesta")
28 kfold_dir = os.path.join(working_dir,"cnn_propuesta_KFold")
29
30 # Dataset generado de la misma manera que el utilizado para el
       entrenamiento
31 # de la CNN, pero sin dividir el Train_1 en Train y Validation.
32 # (ver A.1.4. Formacion del dataset en formato HDF5)
33 dataset_KFold = h5py.File(dataset_KFold_dir, "r")
34
35 # Genero el marco de la 10-Fold Cross Validation
36 BATCH_SIZE = 16
37 NUM_EPOCHS = 50
38 k = 10
39 folds = list(StratifiedKFold(n_splits=k, shuffle=True, random_state=1).
       split(dataset_KFold["X_train"],dataset_KFold["y_train"]))
40 cv_score = list()
41
```

```
for j, (train_idx, val_idx) in enumerate(folds):

    print("Fold {}".format(j+1))
    X_train_cv = np.array(dataset_KFold["X_train"])[train_idx]
    y_train_cv = np.array(dataset_KFold["Y_train"])[train_idx]
    X_valid_cv = np.array(dataset_KFold["X_train"])[val_idx]
    y_valid_cv = np.array(dataset_KFold["Y_train"])[val_idx]

    name_weights = "CNN_model_fold" + str(j+1) + "_weights.h5"
    checkpointer = get_checkpointer(name_weights)
    model.load_weights(os.path.join(model_dir,"CNN_empty.h5"))
    model.fit(
      x = X_train_cv,
      y = y_train_cv,
      batch_size = BATCH_SIZE,
      epochs = NUM_EPOCHS,
      validation_data = [X_valid_cv,y_valid_cv],
      callbacks=[checkpointer])

    val_acc = model.evaluate(X_valid_cv, y_valid_cv)
    cv_score.append(val_acc[1])
    print("Fold {} - CV Accuracy Score: {}".format(j+1,val_acc[1]))

print("Accuracy de Cross Validation %.3f (%.3f)" % (np.mean(cv_score), np.
   std(cv_score)))

# Calculo las metricas sobre el test set para cada fold
accuracy_score_test_cv = list()
classification_error_test_cv = list()
sensitivity_test_cv = list()
specificity_test_cv = list()
precision_test_cv = list()
vpn_test_cv = list()
f1_score_cv = list()
roc_auc_score_test_cv = list()

for name_weights in os.listdir(kfold_dir):
  model = load_model(os.path.join(model_dir,"CNN_empty.h5"))
  model.load_weights(os.path.join(kfold_dir,name_weights))
  y_pred = model.predict(dataset_KFold["X_test"],batch_size=1)
  y_pred = [np.round(value)[0].astype(np.uint8) for value in y_pred]
  metrics_arr = model_metrics(dataset_KFold["Y_test"],y_pred)
  accuracy_score_test_cv.append(metrics_arr[0])
  classification_error_test_cv.append(metrics_arr[1])
  sensitivity_test_cv.append(metrics_arr[2])
  specificity_test_cv.append(metrics_arr[3])
  precision_test_cv.append(metrics_arr[4])
  vpn_test_cv.append(metrics_arr[5])
  f1_score_test_cv.append(metrics_arr[6])
  roc_auc_score_test_cv.append(metrics_arr[7])

print("Accuracy %.3f (%.3f)" % (np.mean(accuracy_score_test_cv), np.std(
   accuracy_score_test_cv)))
print("Error de clasificacion %.3f (%.3f)" % (np.mean(
```

```
    classification_error_test_cv), np.std(classification_error_test_cv)))
print("Sensibilidad %.3f (%.3f)" % (np.mean(sensitivity_test_cv), np.std(
    sensitivity_test_cv)))
print("Especificidad %.3f (%.3f)" % (np.mean(specificity_test_cv), np.std(
    specificity_test_cv)))
print("Precision %.3f (%.3f)" % (np.mean(precision_test_cv), np.std(
    precision_test_cv)))
print("VPN %.3f (%.3f)" % (np.mean(vpn_test_cv), np.std(vpn_test_cv)))
print("F1-Score %.3f (%.3f)" % (np.mean(f1_score_test_cv), np.std(
    f1_score_test_cv)))
print("ROC-AUC Score %.3f (%.3f)" % (np.mean(roc_auc_score_test_cv), np.
    std(roc_auc_score_test_cv)))
```

A.3. Biomarcadores

A.3.1. Armado de dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


def get_matching_values(df,col_name,id_list):
  """
  Funcion que obtiene los valores de la variable especificada
   correspondientes
  a una lista de IDs de pacientes.

  :param df (Pandas DataFrame). Dataframe con la informacion.
  :param col_name (String). Nombre de la variable a encontrar.
  :param id_list (list). Lista con los IDs de los pacientes a encontrar.

  :returns matching_values (list). Lista con los valores de la variable
   que corresponden a cada ID.
  """
    matching_values = []
    for id_ in id_list:
        idx = df.index[df[col_name] == id_]
        matching_values.append(df.loc[idx].values.tolist()[0][1])
    return matching_values


# Cargo el archivo adnimerge que contiene todos los datos utilizados
adnimerge = pd.read_csv("ADNIMERGE.csv",low_memory = False)

# Obtengo el diagnostico junto con dos codigos que identifican al paciente
ptid = adnimerge["PTID"].values.tolist()
dx = adnimerge["DX"].values.tolist()
viscode = adnimerge["VISCODE"].values.tolist()

data_dx = list(zip(ptid,dx,viscode))

dx_df = pd.DataFrame(data_dx, columns = ["PTID","DX","VISCODE"])
```

```
dx_df = dx_df.dropna()
dx_df.drop_duplicates(subset=["PTID","DX"], keep="first", inplace=True)
dx_df = dx_df.query("DX != 'MCI'") # Filtro los MCI

DX = dx_df["DX"].values.tolist()
dx_int =[]
for diag in DX:
    if diag == "CN":
        dx_int.append(0)
    else:
        dx_int.append(1)
dx_df["DX"]=dx_int
dx_df.head()


# Obtengo los datos de la proteina amiloide beta
ptid = adnimerge["PTID"].values.tolist()
abeta = adnimerge["ABETA"].values.tolist()
viscode = adnimerge["VISCODE"].values.tolist()

data = list(zip(ptid,abeta,viscode))

abeta_df = pd.DataFrame(data, columns = ["PTID","ABETA","VISCODE"])
abeta_df = abeta_df.dropna()
abeta_df.drop_duplicates(subset=["PTID"], keep="last", inplace=True)
abeta_df.head()

A = np.array(abeta_df["ABETA"].values.tolist())
for i,a in enumerate(A):
    if "<" in a:
        A[i] = "80"
    elif ">" in a:
        A[i] = "1300"
abeta_df["ABETA"] = list(A)
abeta = [np.float(value) for value in abeta_df["ABETA"].values.tolist()]
abeta_df["ABETA"] = abeta
abeta_df.head()


# Obtengo los datos de la proteina tau
ptid = adnimerge["PTID"].values.tolist()
tau = adnimerge["TAU"].values.tolist()
viscode = adnimerge["VISCODE"].values.tolist()

data_tau = list(zip(ptid,tau,viscode))

tau_df = pd.DataFrame(data_tau, columns = ["PTID","TAU","VISCODE"])
tau_df = tau_df.dropna()
tau_df.drop_duplicates(subset=["PTID"], keep="last", inplace=True)
tau_df.head()

A = np.array(tau_df["TAU"].values.tolist())
for i,a in enumerate(A):
    if "<" in a:
```

```
        A[i] = "80"
    elif ">" in a:
        A[i] = "1300"
tau_df["TAU"] = list(A)
tau = [np.float(value) for value in tau_df["TAU"].values.tolist()]
print(max(tau))
print(min(tau))
tau_df["TAU"] = tau
tau_df.head()

# Obtengo los datos de la proteina tau fosforilada
ptid = adnimerge["PTID"].values.tolist()
ptau = adnimerge["PTAU"].values.tolist()
viscode = adnimerge["VISCODE"].values.tolist()

data_ptau = list(zip(ptid,ptau,viscode))

ptau_df = pd.DataFrame(data_ptau, columns = ["PTID","PTAU","VISCODE"])
ptau_df = ptau_df.dropna()
ptau_df.drop_duplicates(subset=["PTID"], keep="last", inplace=True)

A = np.array(ptau_df["PTAU"].values.tolist())
for i,a in enumerate(A):
    if "<" in a:
        A[i] = "80"
    elif ">" in a:
        A[i] = "1300"
ptau_df["PTAU"] = list(A)
ptau = [np.float(value) for value in ptau_df["PTAU"].values.tolist()]
ptau_df["PTAU"] = ptau
ptau_df.head()


# Obtengo los datos de APOE
ptid = adnimerge["PTID"].values.tolist()
apoe4 = adnimerge["APOE4"].values.tolist()
viscode = adnimerge["VISCODE"].values.tolist()

data_apoe4 = list(zip(ptid,apoe4,viscode))

apoe4_df = pd.DataFrame(data_apoe4, columns = ["PTID","APOE4","VISCODE"])
apoe4_df = apoe4_df.dropna()
apoe4_df.drop_duplicates(subset=["PTID"], keep="first", inplace=True)
apoe4_df.head()

apoe4_int = [np.int(value) for value in apoe4_df["APOE4"].values.tolist()]
apoe4_df["APOE4"] = apoe4_int
apoe4_df.head()

# Obtengo los datos del MMSE
ptid = adnimerge["PTID"].values.tolist()
mmse = adnimerge["MMSE"].values.tolist()
viscode = adnimerge["VISCODE"].values.tolist()

```

```
data_mmse = list(zip(ptid,mmse,viscode))

mmse_df = pd.DataFrame(data_mmse, columns = ["PTID","MMSE","VISCODE"])
mmse_df = mmse_df.dropna()
mmse_df.drop_duplicates(subset=["PTID"], keep="last", inplace=True)
mmse_df

mmse = [np.float(value) for value in mmse_df["MMSE"].values.tolist()]
mmse_df["MMSE"] = mmse
mmse_df.head()


# Armo el dataset

# Obtengo los ID de los pacientes de la proteina amiloide beta
abeta_ptid = abeta_df["PTID"].values.tolist()

# Obtengo los ID de los pacientes que tienen diagnostico
ptid_dataset = [ptid for ptid in abeta_ptid if ptid in dx_df["PTID"].
    values.tolist()]

# Obtengo los datos que corresponden a cada variable para los sujetos
    filtrados.
abeta_dataset = get_matching_values(abeta_df,"PTID",ptid_dataset)
abeta_dataset = [value/max(abeta_dataset) for value in abeta_dataset]

tau_dataset = get_matching_values(tau_df,"PTID",ptid_dataset)
tau_dataset = [value/max(tau_dataset) for value in tau_dataset]

ptau_dataset = get_matching_values(ptau_df,"PTID",ptid_dataset)
ptau_dataset = [value/max(ptau_dataset) for value in ptau_dataset]

apoe4_dataset = get_matching_values(apoe4_df,"PTID",ptid_dataset)

mmse_dataset = get_matching_values(mmse_df,"PTID",ptid_dataset)
mmse_dataset = [value/max(mmse_dataset) for value in mmse_dataset]

dx_dataset = get_matching_values(dx_df,"PTID",ptid_dataset)

cols = ["PTID","ABETA","TAU","PTAU","MMSE","APOE4","DX"]

data_dataset = list(zip(ptid_dataset,abeta_dataset,tau_dataset,
    ptau_dataset,mmse_dataset,apoe4_dataset,dx_dataset))
dataset_df = pd.DataFrame(data_dataset, columns = cols)

# Guardo el dataset
dataset_df.to_csv("dataset_cor.csv",index = False)
```

A.3.2. Comparacion de modelos

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression as LR
from sklearn.model_selection import StratifiedKFold
```

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import StratifiedShuffleSplit
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential,load_model
from keras.utils import np_utils
from keras.layers import Dense, Activation, Dropout, BatchNormalization
from keras.callbacks import EarlyStopping, ModelCheckpoint,
    ReduceLROnPlateau
from keras.optimizers import Adam,RMSprop,SGD
from keras.wrappers.scikit_learn import KerasClassifier
from keras import regularizers
import h5py as h5

def create_model(
  dense_layers=[128,128,128,64,64,64],
  activation="relu",
  optimizer="adam",
  l2_reg = 0.001,
  dropout = 0):

  """
  Funcion que crea un modelo de MLP con los hiperparametros especificados.

  :param dense_layers (array). Vector con los tamanos de las capas densas.
    Default: [128,128,128,64,64,64]
  :param activation (String). Funcion de activacion de las capas densas.
   Default: "relu".
  :param optimizer (String). Algoritmo optimizador del entrenamiento.
   Default: "adam".
  :param l2_reg (float). Regularizacion L2 aplicada. Default: 0.001.
  :param dropout (float). Proporcion de dropout aplicado en las capas.
   Default: 0.

  :return model (Keras Model Object). Modelo MLP.
  """

  model = Sequential()

  for index, lsize in enumerate(dense_layers):
      # Input Layer - includes the input_shape
      if index == 0:
          model.add(Dense(lsize,
                          activation=activation,
                          input_shape=(5,),
                          kernel_regularizer=regularizers.l2(l2_reg)
                          ))
          if dropout != 0:
            model.add(Dropout(dropout))
      else:
```

```
54            model.add(Dense(lsize,
55                            activation=activation,
56                            kernel_regularizer=regularizers.l2(l2_reg)
57                            ))
58            if dropout != 0:
59              model.add(Dropout(dropout))
60
61    model.add(Dense(1,activation="sigmoid"))
62    model.compile(optimizer = optimizer,
63                  loss="binary_crossentropy",
64                  metrics=["accuracy"])
65    return model
66
67
68  # Cargo el dataset
69  dataset_df = pd.read_csv("dataset_cor.csv")
70  cols = ["PTID","ABETA","TAU","PTAU","MMSE","APOE4","DX"]
71
72  X = np.array(dataset_df[cols[1:6]].values.tolist())
73  y = np.array(dataset_df[cols[6]].values.tolist())
74
75  # Divido el dataset en train y test
76  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=42)
77
78  # Regresion logistica #
79  # Busqueda de hiperparametros
80  grid={"C":np.logspace(-3,3,7)}# l1 lasso l2 ridge
81  logreg=LR(solver="lbfgs")
82  logreg_cv=GridSearchCV(logreg,grid,cv=10)
83  logreg_cv.fit(X_train,y_train)
84
85  print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
86  print("accuracy :",logreg_cv.best_score_)
87
88  skf = StratifiedKFold(n_splits=10)
89  skf.get_n_splits(X_train, y_train)
90  CV_score =[]
91  fold = 1
92  for train_index, test_index in (X_train, y_train):
93      X_train_cv = X[train_index]
94      X_test_cv = X[test_index]
95      y_train_cv = y[train_index]
96      y_test_cv = y[test_index]
97
98      clf = LR(penalty = "l2",C = 1.0,random_state=0, solver="lbfgs",
        multi_class="auto").fit(X_train_cv, y_train_cv)
99      CV_score.append(clf.score(X_test_cv, y_test_cv))
100     print("Fold {}: {}".format(fold,clf.score(X_test_cv, y_test_cv)))
101     fold+=1
102
103 print("CV Score LR: {}({})".format(np.array(CV_score).mean(),np.array(
        CV_score).std()))
104
```

```
105 # Pruebo sobre el test set
106 clf = LR(penalty = "l2",C = 1.0,random_state=0, solver="lbfgs",multi_class
       ="auto").fit(X_train, y_train)
107 score = clf.score(X_test, y_test)
108 print("LR score: {}".format(score))
109
110
111 # SVM #
112 # Busqueda de hiperparametros
113 scaler = StandardScaler()
114 X = scaler.fit_transform(X)
115
116 C_range = np.logspace(-2, 10, 13)
117 gamma_range = np.logspace(-9, 3, 13)
118 param_grid = dict(gamma=gamma_range, C=C_range)
119 cv = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
120 grid = GridSearchCV(SVC(), param_grid=param_grid, cv=cv)
121 grid.fit(X, y)
122
123 print("The best parameters are %s with a score of %0.2f"
124       % (grid.best_params_, grid.best_score_))
125
126 # Pruebo los mejores hiperparametros sobre el test set
127 clf = SVC(C = 100.0, gamma =  0.01).fit(X_train, y_train)
128 score = clf.score(X_test, y_test)
129 print("SVM score: {}".format(score))
130
131 # 10-Fold Cross Validation
132 skf = StratifiedKFold(n_splits=10)
133 skf.get_n_splits(X_train, y_train)
134 CV_score =[]
135 fold = 1
136 for train_index, test_index in skf.split(X_train, y_train):
137     X_train_cv = X[train_index]
138     X_test_cv = X[test_index]
139     y_train_cv = y[train_index]
140     y_test_cv = y[test_index]
141
142     clf = SVC(C = 100.0, gamma =  0.01).fit(X_train_cv, y_train)
143     CV_score.append(clf.score(X_test_cv, y_test_cv))
144     print("Fold {}: {}".format(fold,clf.score(X_test_cv, y_test_cv)))
145     fold+=1
146
147 print("CV Score SVM: {}({})".format(np.array(CV_score).mean(),np.array(
       CV_score).std()))
148
149
150 # Naive Bayes #
151 # 10-Fold Cross Validation
152 fold = 1
153 skf = StratifiedKFold(n_splits=10)
154 skf.get_n_splits(X_train, y_train)
155 CV_score =[]
156 for train_index, test_index in (X_train, y_train):
```

```
157     X_train_cv = X[train_index]
158     X_test_cv = X[test_index]
159     y_train_cv = y[train_index]
160     y_test_cv = y[test_index]
161 
162     clf = GaussianNB().fit(X_train_cv,y_train_cv)
163     CV_score.append(clf.score(X_test_cv, y_test_cv))
164     print("Fold {}: {}".format(fold,clf.score(X_test_cv, y_test_cv)))
165     fold+=1
166 
167 print("CV Score GNB: {}({})".format(np.array(CV_score).mean(),np.array(
    CV_score).std()))
168 
169 # Pruebo entrenando con el test set
170 clf = GaussianNB().fit(X_train,y_train)
171 score = clf.score(X_test, y_test)
172 
173 print("GNB score: {}".format(score))
174 
175 
176 # MLP #
177 
178 # Luego de elegir un modelo que obtenga un desempeno comparable con los
    demas modelos
179 # se utiliza la funcion create_model
180 
181 # Especifico el modelo, que va a ser creado con la funcion create_model
182 model = KerasClassifier(build_fn=create_model,
183                         batch_size=16,
184                         verbose=0)
185 
186 # Grilla de hiperparametros
187 param_dist = {"optimizer":("rmsprop","adam"),
188               "l2_reg":[0.1,0.01,0.001,0.0001],
189               "dropout": [0,0.1,0.25,0.5],
190               "epochs": [100,500,1000],
191               "batch_size":[16,32,64]}
192 
193 # Randomized search CV
194 n_iter_search = 50
195 random_search = RandomizedSearchCV(model,
196                                    param_distributions=param_dist,
197                                    n_iter=n_iter_search,
198                                    cv=5,
199                                    iid=False,
200                                    scoring=["neg_log_loss","f1","roc_auc"
    ],
201                                    refit = "roc_auc",
202                                    n_jobs = -1)
203 
204 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
    random_state=seed)
205 
206 random_search.fit(X_train,y_train,verbose = 0)
```

```
207
208 # Guardo los resultados de la busqueda en un DataFrame.
209 cv_results_df = pd.DataFrame(random_search.cv_results_)
210 cv_results_df.to_csv(os.path.join(cwd,"random_search.csv"),sep=",")
211
212 print("Mejores parametros")
213 print(random_search.best_params_)
```

A.3.3. MLP

```
1  import os
2  import pandas as pd
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from sklearn.model_selection import StratifiedKFold
6  from sklearn.model_selection import train_test_split, KFold, GridSearchCV,
       RandomizedSearchCV
7  from sklearn.metrics import confusion_matrix, classification_report,
       accuracy_score, roc_curve,roc_auc_score
8  from sklearn.preprocessing import MinMaxScaler
9  from keras.models import Sequential,load_model
10 from keras.layers import Dense, Activation, Dropout, BatchNormalization
11 from keras.callbacks import EarlyStopping, ModelCheckpoint,
       ReduceLROnPlateau
12 from keras.optimizers import Adam,RMSprop,SGD
13 from keras.wrappers.scikit_learn import KerasClassifier
14 from keras import regularizers
15 import h5py as h5
16
17 # Defino el path al dataset
18 cwd = "/content/gdrive/My Drive/Colab/ADNIMERGE"
19 dataset_df = pd.read_csv(os.path.join(cwd,"dataset.csv"))
20
21 cols = dataset_df.columns
22 X = np.array(dataset_df[cols[1:6]].values.tolist())
23 y = np.array(dataset_df[cols[6]].values.tolist())
24
25 seed = 42
26 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
       random_state=seed)
27 X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
       train_size=0.8, random_state=seed)
28
29
30 random_search_df = pd.read_csv(os.path.join(cwd,"random_search.csv"))
31 random_search_df = random_search_df.sort_values("mean_test_roc_auc",
       ascending=False)
32
33 param_columns = ["param_optimizer", "param_l2_reg", "param_epochs","
       param_dropout", "param_batch_size"]
34
35 best_params = {}
36 for param in param_columns:
37   best_params[param] = random_search_df[param].values[0]
```

```
38
39
40 # Defino la callback para el entrenamiento
41 # Checkpointer
42 cb_checkpointer = ModelCheckpoint(filepath = os.path.join(cwd,"best_model.
       hdf5"),
43                                   monitor = "val_loss",
44                                   save_best_only = True,
45                                   mode = "auto")
46
47 model = create_model(optimizer= best_params["param_optimizer"],
48                      l2_reg = best_params["param_l2_reg"],
49                      dropout = best_params["param_dropout"])
50
51 # Guardo el modelo sin entrenar
52 model.save(os.path.join(cwd,"MLP_empty.h5"))
53
54 ## Entrenamiento
55 history = model.fit(X_train,
56                     y_train,
57                     epochs=params["param_epochs"],
58                     batch_size=params["param_batch_size"],
59                     validation_data=(X_val,y_val),
60                     verbose = 1,
61                     callbacks=[cb_checkpointer])
62
63 # Curvas de entrenamiento
64 # Accuracy
65 plt.plot(history.history["binary_accuracy"])
66 plt.plot(history.history["val_binary_accuracy"])
67 plt.title("model accuracy")
68 plt.ylabel("accuracy")
69 plt.xlabel("epoch")
70 plt.legend(["train", "valid"], loc="upper left")
71 plt.show()
72 # Loss
73 plt.plot(history.history["loss"])
74 plt.plot(history.history["val_loss"])
75 plt.title("model loss")
76 plt.ylabel("loss")
77 plt.xlabel("epoch")
78 plt.legend(["train", "valid"], loc="upper left")
79 plt.show()
80
81 # Evaluacion del modelo con el Test set
82 model.load_weights(os.path.join(cwd,"best_model.hdf5"))
83 y_pred = model.predict_classes(X_test,batch_size=X_test.shape[0])
84 y_pred = [value[0] for value in y_pred]
85 evaluation_metrics = model_metrics(y_pred,y_test)
86
87 print("Accuracy: {}".format(evaluation_metrics[0]))
88 print("Error de Clasificacion: {}".format(evaluation_metrics[1]))
89 print("Sensibilidad: {}".format(evaluation_metrics[2]))
90 print("Especificidad: {}".format(evaluation_metrics[3]))
```

```
print("Precision (VPP): {}".format(evaluation_metrics[4]))
print("VPN: {}".format(evaluation_metrics[5]))
print("F1-Score: {}".format(evaluation_metrics[6]))
print("ROC AUC Score: {}".format(evaluation_metrics[7]))

# Guardo el modelo entrenado
model.save(os.path.join(cwd,"MLP.h5"))
```

A.3.4. 10-Fold Cross Validation

```
import os
import h5py
import keras
import numpy as np
import matplotlib.pyplot as plt
from keras.callbacks import ModelCheckpoint
from sklearn.model_selection import StratifiedKFold
from keras.models import Model, Sequential, load_model

def get_checkpointer(name_weights):
  """
  Funcion que crea un checkpointer para cada fold
  :param name_weights (String). Nombre del fold.
  :return checkpointer (Object). Callback para hacer checkpoint del modelo
    para un fold.
  """
  checkpointer = ModelCheckpoint(filepath = os.path.join(kfold_dir,
   name_weights),
                                 monitor = "val_loss",
                                 save_best_only = True,
                                 mode = "auto")
  return checkpointer


# Defino los paths importantes
cwd = "/content/gdrive/My Drive/Colab/ADNIMERGE"
dataset_df = pd.read_csv(os.path.join(cwd,"dataset.csv"))

cols = dataset_df.columns
X = np.array(dataset_df[cols[1:6]].values.tolist())
y = np.array(dataset_df[cols[6]].values.tolist())

seed = 42
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
   random_state=seed)

# Path al modelo sin entrenar
empty_model = os.path.join(cwd,"MLP_empty.h5")


random_search_df = pd.read_csv(os.path.join(cwd,"random_search.csv"))
random_search_df = random_search_df.sort_values("mean_test_roc_auc",
   ascending=False)

```

```
param_columns = ["param_optimizer", "param_l2_reg", "param_epochs","
    param_dropout", "param_batch_size"]

best_params = {}
for param in param_columns:
  best_params[param] = random_search_df[param].values[0]

# Genero el marco de la 10-Fold Cross Validation
k = 10
folds = list(StratifiedKFold(n_splits=k, shuffle=True, random_state=1).
    split(X_train,y_train))
cv_score = list()

for j, (train_idx, val_idx) in enumerate(folds):

    X_train_cv = X_train[train_idx]
    y_train_cv = y_train[train_idx]
    X_val_cv = X_train[val_idx]
    y_val_cv = y_train[val_idx]

    name_weights = "MLP_model_fold" + str(j+1) + "_weights.h5"
    checkpointer = get_checkpointer(name_weights)

    model = load_model(empty_model)

    model.fit(X_train_cv,
              y_train_cv,
              epochs=best_params["param_epochs"],
              batch_size=best_params["param_batch_size"],
              validation_data=(X_val_cv,y_val_cv),
              verbose = 0,
              callbacks=[checkpointer])
    model.load_weights(os.path.join(kfold_dir,name_weights))
    val_acc = model.evaluate(X_val_cv, y_val_cv,verbose = 0)
    cv_score.append(val_acc[1])

    print("Fold {} - CV Accuracy Score: {}".format(j+1,val_acc[1]))

print("Accuracy de Cross Validation %.3f (%.3f)" % (np.mean(cv_score), np.
    std(cv_score)))

# Calculo las metricas sobre el test set para cada fold
accuracy_score_test_cv = list()
classification_error_test_cv = list()
sensitivity_test_cv = list()
specificity_test_cv = list()
precision_test_cv = list()
vpn_test_cv = list()
f1_score_cv = list()
roc_auc_score_test_cv = list()

for name_weights in os.listdir(kfold_dir):
  model = load_model(os.path.join(model_dir,"MLP_empty.h5"))
  model.load_weights(os.path.join(kfold_dir,name_weights))
```

```
  y_pred = model.predict(X_test,batch_size=1)
  y_pred = [np.round(value)[0].astype(np.uint8) for value in y_pred]
  metrics_arr = model_metrics(y_test,y_pred)
  accuracy_score_test_cv.append(metrics_arr[0])
  classification_error_test_cv.append(metrics_arr[1])
  sensitivity_test_cv.append(metrics_arr[2])
  specificity_test_cv.append(metrics_arr[3])
  precision_test_cv.append(metrics_arr[4])
  vpn_test_cv.append(metrics_arr[5])
  f1_score_test_cv.append(metrics_arr[6])
  roc_auc_score_test_cv.append(metrics_arr[7])

print("Accuracy %.3f (%.3f)" % (np.mean(accuracy_score_test_cv), np.std(
    accuracy_score_test_cv)))
print("Error de clasificacion %.3f (%.3f)" % (np.mean(
    classification_error_test_cv), np.std(classification_error_test_cv)))
print("Sensibilidad %.3f (%.3f)" % (np.mean(sensitivity_test_cv), np.std(
    sensitivity_test_cv)))
print("Especificidad %.3f (%.3f)" % (np.mean(specificity_test_cv), np.std(
    specificity_test_cv)))
print("Precision %.3f (%.3f)" % (np.mean(precision_test_cv), np.std(
    precision_test_cv)))
print("VPN %.3f (%.3f)" % (np.mean(vpn_test_cv), np.std(vpn_test_cv)))
print("F1-Score %.3f (%.3f)" % (np.mean(f1_score_test_cv), np.std(
    f1_score_test_cv)))
print("ROC-AUC Score %.3f (%.3f)" % (np.mean(roc_auc_score_test_cv), np.
    std(roc_auc_score_test_cv)))
```

A.4. Modelo final

A.4.1. Formación del dataset en formato HDF5

```
import os
import glob
import time
import h5py
import numpy as np
import pandas as pd
import random
from sklearn.model_selection import train_test_split

def get_patches(subject,path):
    """
    Funcion que obtiene todos los patches 2.5D para un determinado sujeto.

    :param subject (String). ID del sujeto.
    :param path (list). Lista de paths a los archivos .npy que contienen
   los patches 2.5D del sujeto.

    :return patches (list). Lista con los patches 2.5D del sujeto.
    """
    patches = []
    for p in path:
```

```
        if subject in p:
            patches.append(np.load(path))
    return patches

# Defino el directorio de trabajo y los paths a los archivos necesarios
working_dir = "/volumes/JEH/datos-proyecto-final"
mri_dir = os.path.join(working_dir,"MRI-Nyul-Udupa")
rgb_dir = os.path.join(working_dir,"Patches")

mri_ad_path = os.path.join(mri_dir,"AD")
mri_cn_path = os.path.join(mri_dir,"CN")

rgb_ad_path = os.path.join(rgb_dir,"AD")
rgb_cn_path = os.path.join(rgb_dir,"CN")

rgb_patches = glob.glob(img_dir+'/*/*.npy')

# Cargo el archivo csv con los datos de Biomarcadores
biom_df = pd.read_csv('dataset.csv')

# Cargo un archivo .csv que contiene informacion acerca del
# ID, clase, edad y genero de los sujetos.
mri_df = pd.read_csv("mri_dataset.csv")

subjects_mri = mri_df['Subject'].values.tolist()
labels_mri = mri_df['Label'].values.tolist()
subjects_biom = biom_df['PTID'].values.tolist()
labels_biom = biom_df['DX'].values.tolist()

dataset_subjects = []
dataset_labels = []

# Obtengo los sujetos que tienen MRI y biomarcadores
for i,subject in enumerate(subjects_mri):
    if subject in subjects_biom and labels_mri[i] == biom_df[biom_df['PTID
    ']==subject].values.tolist()[0][6]:
        dataset_subjects.append(subject)
        dataset_labels.append(labels_mri[i])

# Submuestreo de los sujetos para igualar la cantidad de sujetos en cada
   clase.
df_dataset = pd.DataFrame(list(zip(subjects,labels)), columns = ["subject"
   ,"target"])

count_class_0, count_class_1 = df_dataset.target.value_counts()

df_class_0 = df_dataset[df_dataset["target"] == 0]
df_class_1 = df_dataset[df_dataset["target"] == 1]

df_class_0_under = df_class_0.sample(count_class_1)
df_dataset_under = pd.concat([df_class_0_under, df_class_1], axis=0)

print("Train set Random under-sampling:")
print(df_dataset_under.target.value_counts())
```

```
72 print(df_dataset_under.target.value_counts())
73
74 subjects_under = df_dataset_under.subject.values.tolist()
75 labels_under  = df_dataset_under.target.values.tolist()
76
77 X_train_1,X_test,y_train_1,y_test = train_test_split(subjects_under,
78                                                     labels_under,
79                                                     stratify =
    labels_under,
80                                                     test_size=0.2,
81                                                     random_state=42
82                                                     )
83
84 X_train,X_valid,y_train,y_valid = train_test_split(X_train_1,
85                                                   y_train_1,
86                                                   stratify = y_train_1,
87                                                   test_size=0.1,
88                                                   random_state=42
89                                                  )
90
91
92 # Creo DataFrames para cada particion
93 # Train
94 df_train = pd.DataFrame(list(zip(X_train,y_train)), columns = ['subject','
    target'])
95
96 # Validation
97 df_valid = pd.DataFrame(list(zip(X_valid,y_valid)), columns = ['subject','
    target'])
98
99 # Test
100 df_test = pd.DataFrame(list(zip(X_test,y_test)), columns = ['subject','
    target'])
101
102 #Defino las formas de los sets
103 mri_train_shape = (len(X_train), 113, 32, 32, 3)
104 mri_valid_shape = (len(X_valid), 113, 32, 32, 3)
105 mri_test_shape = (len(X_test), 113, 32, 32, 3)
106 biom_train_shape = (len(X_train), 5)
107 biom_valid_shape = (len(X_valid), 5)
108 biom_test_shape = (len(X_test), 5)
109
110 #Creo el archivo hdf5
111 hdf5_path = os.path.join(os.getcwd(),'dataset-CNN-LSTM.h5')
112 f = h5py.File(hdf5_path, mode='w')
113
114 f.create_dataset("X_train_mri", mri_train_shape, np.float32)
115 f.create_dataset("X_valid_mri", mri_valid_shape, np.float32)
116 f.create_dataset("X_test_mri", mri_test_shape, np.float32)
117 f.create_dataset("X_train_biom", biom_train_shape, np.float32)
118 f.create_dataset("X_valid_biom", biom_valid_shape, np.float32)
119 f.create_dataset("X_test_biom", biom_test_shape, np.float32)
120
121 f.create_dataset("Y_train", (len(y_train),), np.uint8)
```

```
f["Y_train"][...] = y_train
f.create_dataset("Y_valid", (len(y_valid),), np.uint8)
f["Y_valid"][...] = y_valid
f.create_dataset("Y_test", (len(y_test),), np.uint8)
f["Y_test"][...] = y_test

# Creo los sets
for i,subject in enumerate(X_train):

    if i % 50 == 0 and i > 1:
        print ('Train data: {}/{}'.format(i, len(X_train)))

    f["X_train_biom"][i,:] = biom_df[biom_df['PTID']==subject].values.
   tolist()[0][1:6]

    patches = get_patches(subject,rgb_patches)
    for j,patch in enumerate(patches):
        f["X_train_mri"][i,j,...] = patch[None]

for i,subject in enumerate(X_valid):

    if i % 10 == 0 and i > 1:
        print ('Validation data: {}/{}'.format(i, len(X_valid)))

    f["X_valid_biom"][i,:] = biom_df[biom_df['PTID']==subject].values.
   tolist()[0][1:6]

    patches = get_patches(subject,rgb_patches)
    for j,patch in enumerate(patches):
        f["X_valid_mri"][i,j,...] = patch[None]

for i,subject in enumerate(X_test):

    if i % 10 == 0 and i > 1:
        print ('Test data: {}/{}'.format(i, len(X_test)))

    f["X_test_biom"][i,:] = biom_df[biom_df['PTID']==subject].values.
   tolist()[0][1:6]

    patches = get_patches(subject,rgb_patches)
    for j,patch in enumerate(patches):
        f["X_test_mri"][i,j,...] = patch[None]

f.close()

print('Done.')
```

A.4.2. Modelo CNN-LSTM-MLP

```
import os
import keras
import h5py
import numpy as np
import matplotlib.pyplot as plt
```

```
import pandas as pd
import sklearn.metrics as metrics
import keras.backend as K
from keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,Activation,
    Dropout,BatchNormalization
from keras.layers import Input, LSTM, TimeDistributed, Layer
from keras.models import Model, Sequential, load_model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.model_selection import StratifiedKFold
from keras import regularizers


def get_model(CNN, MLP, name = 'CNN_LSTM_MLP'):
  """
  Funcion que dados los modelos CNN y MLP genera el modelo final con
    Functional API de Keras.

  :param CNN (Keras Model). Modelo CNN
  :param MLP (Keras Model). Modelo MLP
  :param name (String). Nombre del modelo final. Default = 'CNN_LSTM_MLP'
  :return model (Keras Model). Modelo compuesto.
  """


  # Defino el modelo con la Functional API

  # Rama de MRI
  input_patches = Input(shape=(113,32,32,3),name = 'Patches')
  processed_patches = TimeDistributed(CNN,name = 'TimeDistributed_CNN')(
    input_patches)
  lstm_1 = LSTM(512,name = 'LSTM_1',return_sequences=True)(
    processed_patches)
  lstm_2 = LSTM(64,name = 'LSTM_3',return_sequences=True)(lstm_1)
  flatten = Flatten()(lstm_2)
  dense_1 = Dense(64,activation='relu',name='FC_1',kernel_regularizer=
    regularizers.l2(0.001))(flatten)


  # Rama de Biomarcadores
  input_biom = Input(shape=(5,),name = 'Biomarcadores')
  mlp_out = MLP(input_biom)

  # Concateno
  concat = keras.layers.concatenate([dense_1, mlp_out], name = 'Concatenar
    ')

  # Capas densas
  x = Dense(128, activation='relu',kernel_regularizer=regularizers.l2
    (0.001))(concat)
  x = Dense(64, activation='relu',kernel_regularizer=regularizers.l2
    (0.001))(x)
  x = Dense(64, activation='relu',kernel_regularizer=regularizers.l2
    (0.001))(x)

```

```
  # Capa final
  main_output = Dense(1, activation='sigmoid', name='main_output')(x)

  model = Model(inputs=[input_patches, input_biom], outputs=[main_output],
   name = name)

  model.save(os.path.join(models_dir,name+'.h5'))
  return model

# Defino los paths importantes
working_dir = '/content/drive/My Drive/Colab/final-model'
dataset_dir = os.path.join(working_dir,'dataset-CNN-LSTM.h5')
models_dir = os.path.join(working_dir,'models')

# Datasets
dataset = h5py.File(dataset_dir, "r")

# Defino el train set
X_train = [np.array(dataset["X_train_mri"]),np.array(dataset["X_train_biom
    "])]
y_train = np.array(dataset['Y_train'])

# Defino el validation set
X_valid = [np.array(dataset["X_valid_mri"]),np.array(dataset["X_valid_biom
    "])]
y_valid = np.array(dataset['Y_valid'])

# Defino el test set
X_test = [np.array(dataset["X_test_mri"]),np.array(dataset["X_test_biom"])
    ]
y_test = np.array(dataset["Y_test"])

# Cargo los modelos
CNN = load_model(os.path.join(models_dir,'CNN.h5'))
CNN.load_weights(os.path.join(models_dir,'CNN_weights.h5'))
MLP = load_model(os.path.join(models_dir,'MLP.h5'))
MLP.load_weights(os.path.join(models_dir,'MLP_weights.h5'))

#Quito las capas superiores de la MLP (dropout y sigmoidea)
MLP.pop()
MLP.pop()
MLP.name = 'MLP'

# Especifico que los modelos CNN y MLP no sean entrenables
CNN.trainable = False
MLP.trainable = False

# Obtengo el modelo CNN-LSTM-MLP
model = get_model(CNN,MLP)

# Definicion de algunos hiperparametros del modelo
BATCH_SIZE = 16
NUM_EPOCHS = 30
OBJECTIVE_FUNCTION = 'binary_crossentropy'
```

```
LOSS_METRICS = ['binary_accuracy']

# Compilacion del modelo
model.compile(optimizer='adam', loss=OBJECTIVE_FUNCTION,metrics =
    LOSS_METRICS)

# Model Checkpoint callback
cb_checkpointer = ModelCheckpoint(filepath = os.path.join(models_dir,'
    test_3_{epoch:02d}-{val_loss:.2f}.hdf5'),
                                  monitor = 'val_loss',
                                  save_best_only = True,
                                  mode = 'auto')

# Entrenamiento del modelo
history = model.fit(
    x = X_train,
    y = y_train,
    batch_size = BATCH_SIZE,
    epochs = NUM_EPOCHS,
    validation_data = [X_valid,y_valid],
    callbacks = [cb_checkpointer]
    )

#Curvas de entrenamiento
#Accuracy
plt.plot(history.history["binary_accuracy"])
plt.plot(history.history["val_binary_accuracy"])
plt.title("model accuracy")
plt.ylabel("accuracy")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper left")
plt.show()
# Loss
plt.plot(history.history["loss"])
plt.plot(history.history["val_loss"])
plt.title("model loss")
plt.ylabel("loss")
plt.xlabel("epoch")
plt.legend(["train", "test"], loc="upper left")
plt.show()


# Evaluacion del modelo con el Test set
y_pred = model.predict(X_test,batch_size=1)
y_pred = [np.round(value)[0].astype(np.uint8) for value in y_pred]
evaluation_metrics = model_metrics(dataset["Y_test"],y_pred)

print("Accuracy: {}".format(evaluation_metrics[0]))
print("Error de Clasificacion: {}".format(evaluation_metrics[1]))
print("Sensibilidad: {}".format(evaluation_metrics[2]))
print("Especificidad: {}".format(evaluation_metrics[3]))
print("Precision (VPP): {}".format(evaluation_metrics[4]))
print("VPN: {}".format(evaluation_metrics[5]))
print("F1-Score: {}".format(evaluation_metrics[6]))
```

```
print("ROC AUC Score: {}".format(evaluation_metrics[7]))

# Guardo el modelo entrenado
model.save(os.path.join(models_dir,"CNN_LSTM_MLP.h5"))
```

A.4.3. 10-Fold Cross Validation

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import time
import os
import glob
import keras
from keras.layers import Dense,Conv2D,Flatten,MaxPooling2D,Activation,
    Dropout,BatchNormalization
from keras.layers import Input, Embedding, LSTM, TimeDistributed, Layer
from keras.models import Model, Sequential, load_model
from keras.optimizers import Adam,RMSprop,SGD
from keras.callbacks import EarlyStopping, ModelCheckpoint
import h5py
from sklearn.model_selection import StratifiedKFold
import sklearn.metrics as metrics
import keras.backend as K
from keras import regularizers

def get_model(CNN, MLP, name = 'CNN_LSTM_MLP'):
  """
  Funcion que dados los modelos CNN y MLP genera el modelo final con
    Functional API de Keras.

  :param CNN (Keras Model). Modelo CNN
  :param MLP (Keras Model). Modelo MLP
  :param name (String). Nombre del modelo final. Default = 'CNN_LSTM_MLP'
  :return model (Keras Model). Modelo compuesto.
  """


  # Defino el modelo con la Functional API

  # Rama de MRI
  input_patches = Input(shape=(113,32,32,3),name = 'Patches')
  processed_patches = TimeDistributed(CNN,name = 'TimeDistributed_CNN')(
    input_patches)
  lstm_1 = LSTM(512,name = 'LSTM_1',return_sequences=True)(
    processed_patches)
  lstm_2 = LSTM(64,name = 'LSTM_3',return_sequences=True)(lstm_1)
  flatten = Flatten()(lstm_2)
  dense_1 = Dense(64,activation='relu',name='FC_1',kernel_regularizer=
    regularizers.l2(0.001))(flatten)


  # Rama de Biomarcadores
  input_biom = Input(shape=(5,),name = 'Biomarcadores')
```

```
  mlp_out = MLP(input_biom)

  # Concateno
  concat = keras.layers.concatenate([dense_1, mlp_out], name = 'Concatenar')

  # Capas densas
  x = Dense(128, activation='relu',kernel_regularizer=regularizers.l2(0.001))(concat)
  x = Dense(64, activation='relu',kernel_regularizer=regularizers.l2(0.001))(x)
  x = Dense(64, activation='relu',kernel_regularizer=regularizers.l2(0.001))(x)

  # Capa final
  main_output = Dense(1, activation='sigmoid', name='main_output')(x)

  model = Model(inputs=[input_patches, input_biom], outputs=[main_output], name = name)

  model.save(os.path.join(models_dir,name+'.h5'))
  return model

# Defino los paths importantes
working_dir = "/content/drive/My Drive/Colab/final-model"
dataset_dir = os.path.join(working_dir,"dataset-CNN-LSTM.h5")
test_dir = os.path.join(working_dir,"test.h5")
models_dir = os.path.join(working_dir,"models")

# Cargo elDataset
dataset = h5py.File(dataset_dir, "r")

# Uso el dataset para armar el de CV.
# Junto train y validation.
mri_train_shape = dataset["X_train_mri"].shape
mri_test_CV_shape = dataset["X_valid_mri"].shape
mri_valid_shape = dataset["X_test_mri"].shape

mri_train_CV_shape= (mri_train_shape[0]+mri_valid_shape[0],113, 32, 32, 3)

biom_train_shape = dataset["X_train_biom"].shape
biom_test_CV_shape = dataset["X_valid_biom"].shape
biom_valid_shape = dataset["X_test_biom"].shape

biom_train_CV_shape= (biom_train_shape[0]+biom_valid_shape[0],5)
biom_train_CV_shape

train_label_len_CV = biom_train_CV_shape[0]
test_label_len_CV = biom_test_CV_shape[0]

f.close()

# Guardo la data en el dataset_cv.h5
dataset_cv_path = os.path.join(working_dir,"dataset_cv.h5")
```

```
92 f = h5py.File(dataset_cv_path, "w")
93
94 f.create_dataset("X_train_mri_CV",mri_train_CV_shape,np.float32)
95 f["X_train_mri_CV"][:len(dataset["Y_train"]),...] = dataset["X_train_mri"
       ][...]
96 f["X_train_mri_CV"][len(dataset["Y_train"]):,...] = dataset["X_test_mri"
       ][...]
97
98 f.create_dataset("X_train_biom_CV",biom_train_CV_shape,np.float32)
99 f["X_train_biom_CV"][:len(dataset["Y_train"]),...] = dataset["X_train_biom
       "][...]
100 f["X_train_biom_CV"][len(dataset["Y_train"]):,...] = dataset["X_test_biom"
       ][...]
101
102 f.create_dataset("X_test_mri_CV",mri_test_CV_shape,np.float32)
103 f["X_test_mri_CV"][...] = dataset["X_valid_mri"][...]
104
105 f.create_dataset("X_test_biom_CV",biom_test_CV_shape,np.float32)
106 f["X_test_biom_CV"][...] = dataset["X_valid_biom"][...]
107
108 f.create_dataset("Y_train_CV", (train_label_len_CV,), np.uint8)
109 f["Y_train_CV"][:len(dataset["Y_train"])] = dataset["Y_train"]
110 f["Y_train_CV"][len(dataset["Y_train"]):] = dataset["Y_test"]
111
112 f.create_dataset("Y_test_CV", (test_label_len_CV,), np.uint8)
113 f["Y_test_CV"][...] = dataset["Y_valid"]
114
115 f.close()
116
117 # Cargo los modelos CNN Y MLP
118 CNN = load_model(os.path.join(models_dir,"CNN.h5"))
119 CNN.load_weights(os.path.join(models_dir,"CNN_weights.h5"))
120 MLP = load_model(os.path.join(models_dir,"MLP.h5"))
121 MLP.load_weights(os.path.join(models_dir,"MLP_weights.h5"))
122
123 #Quito las capas superiores de la MLP (dropout y sigmoidea)
124 MLP.pop()
125 MLP.pop()
126 MLP.name = "MLP"
127
128 # Especifico que los modelos CNN y MLP no sean entrenables
129 CNN.trainable = False
130 MLP.trainable = False
131
132
133 # 10 Fold Cross validation
134
135 #Cargo el dataset para Cross Validation
136 dataset_cv_path = os.path.join(working_dir,"dataset_cv.h5")
137 dataset_cv = h5py.File(dataset_cv_path, "r")
138
139 X_train_mri_CV = np.array(dataset_cv["X_train_mri_CV"])
140 X_train_biom_CV = np.array(dataset_cv["X_train_biom_CV"])
141
```

```
Y_train_CV = np.array(dataset_cv["Y_train_CV"])

# Defino el test set
X_test = [np.array(dataset_cv["X_test_mri_CV"]),np.array(dataset_cv["
    X_test_biom_CV"])]
Y_test = np.array(dataset_cv["Y_test_CV"])


# Defino el directorio para guardar los resultados del CV
kfold_dir = os.path.join(working_dir,"KFold_weights")
if not os.path.isdir(kfold_dir):
  os.mkdir(kfold_dir)

# Model Checkpoint callback
def get_checkpointer(name_weights):
  checkpointer = ModelCheckpoint(filepath = os.path.join(kfold_dir,
    name_weights),
                                 monitor = "val_loss",
                                 save_best_only = True,
                                 mode = "auto")
  return checkpointer

# Genero el marco de la 10-Fold Cross Validation
BATCH_SIZE = 16
NUM_EPOCHS = 30
k = 10
folds = list(StratifiedKFold(n_splits=k, shuffle=True, random_state=1).
    split(X_train_mri_CV,Y_train_CV))
cv_score = list()
accuracy_score_cv = list()
classification_error_cv = list()
sensitivity_cv = list()
specificity_cv = list()
precision_cv = list()
f1_cv = list()
roc_auc_score_cv = list()

for j, (train_idx, val_idx) in enumerate(folds):

    X_train_mri = X_train_mri_CV[train_idx]
    X_train_biom = X_train_biom_CV[train_idx]
    y_train = Y_train_CV[train_idx]

    X_val_mri = X_train_mri_CV[val_idx]
    X_val_biom = X_train_biom_CV[val_idx]
    y_val = Y_train_CV[val_idx]

    name_weights = "CNN_LSTM_fold" + str(j+1) + "_weights.h5"
    checkpointer = get_checkpointer(name_weights)

    model = get_model(CNN,MLP)
    model.compile(optimizer="adam", loss=OBJECTIVE_FUNCTION,metrics =
    LOSS_METRICS)

```

```
    model.fit([X_train_mri,X_train_biom],
              y_train,
              epochs=NUM_EPOCHS,
              batch_size=BATCH_SIZE,
              validation_data=([X_val_mri,X_val_biom],y_val),
              verbose = 0,
              callbacks=[checkpointer])

    model.load_weights(os.path.join(kfold_dir,name_weights))
    val_acc = model.evaluate([X_test_mri_CV,X_test_biom_CV], Y_test_CV,
    verbose = 0)
    cv_score.append(val_acc[1])
    print("Fold {} - CV Accuracy Score: {}".format(j+1,val_acc[1]))

print("Estimated Cross Validation Accuracy %.3f (%.3f)" % (np.mean(
    cv_score), np.std(cv_score)))

# Calculo las metricas sobre el test set para cada fold
accuracy_score_test_cv = list()
classification_error_test_cv = list()
sensitivity_test_cv = list()
specificity_test_cv = list()
precision_test_cv = list()
vpn_test_cv = list()
f1_score_cv = list()
roc_auc_score_test_cv = list()

for name_weights in os.listdir(kfold_dir):
  model = get_model(CNN,MLP)
  model.load_weights(os.path.join(kfold_dir,name_weights))
  y_pred = model.predict(X_test,batch_size=1)
  y_pred = [np.round(value)[0].astype(np.uint8) for value in y_pred]
  metrics_arr = model_metrics(Y_test,y_pred)
  accuracy_score_test_cv.append(metrics_arr[0])
  classification_error_test_cv.append(metrics_arr[1])
  sensitivity_test_cv.append(metrics_arr[2])
  specificity_test_cv.append(metrics_arr[3])
  precision_test_cv.append(metrics_arr[4])
  vpn_test_cv.append(metrics_arr[5])
  f1_score_test_cv.append(metrics_arr[6])
  roc_auc_score_test_cv.append(metrics_arr[7])

print("Accuracy %.3f (%.3f)" % (np.mean(accuracy_score_test_cv), np.std(
    accuracy_score_test_cv)))
print("Error de clasificacion %.3f (%.3f)" % (np.mean(
    classification_error_test_cv), np.std(classification_error_test_cv)))
print("Sensibilidad %.3f (%.3f)" % (np.mean(sensitivity_test_cv), np.std(
    sensitivity_test_cv)))
print("Especificidad %.3f (%.3f)" % (np.mean(specificity_test_cv), np.std(
    specificity_test_cv)))
print("Precision %.3f (%.3f)" % (np.mean(precision_test_cv), np.std(
    precision_test_cv)))
print("VPN %.3f (%.3f)" % (np.mean(vpn_test_cv), np.std(vpn_test_cv)))
print("F1-Score %.3f (%.3f)" % (np.mean(f1_score_test_cv), np.std(
```

```
    f1_score_test_cv)))
print("ROC-AUC Score %.3f (%.3f)" % (np.mean(roc_auc_score_test_cv), np.
    std(roc_auc_score_test_cv)))
```

Bibliografía

[1] M. W. Bondi, E. C. Edmonds, and D. P. Salmon, "Alzheimer's disease: past, present, and future," *Journal of the International Neuropsychological Society*, vol. 23, no. 9-10, pp. 818–831, 2017.

[2] G. Blessed, B. E. Tomlinson, and M. Roth, "The association between quantitative measures of dementia and of senile change in the cerebral grey matter of elderly subjects," *British Journal of Psychiatry*, vol. 114, no. 512, p. 797–811, 1968.

[3] C. Van Cauwenberghe, C. Van Broeckhoven, and K. Sleegers, "The genetic landscape of alzheimer disease: clinical implications and perspectives," *Genetics in Medicine*, vol. 18, no. 5, pp. 421–430, 2016.

[4] C. R. Jack Jr, D. S. Knopman, W. J. Jagust, L. M. Shaw, P. S. Aisen, M. W. Weiner, R. C. Petersen, and J. Q. Trojanowski, "Hypothetical model of dynamic biomarkers of the alzheimer's pathological cascade," *The Lancet Neurology*, vol. 9, no. 1, pp. 119–128, 2010.

[5] J. A. Hardy and G. A. Higgins, "Alzheimer's disease: the amyloid cascade hypothesis," *Science*, vol. 256, no. 5054, pp. 184–186, 1992.

[6] C. R. Jack, D. A. Bennett, K. Blennow, M. C. Carrillo, H. H. Feldman, G. B. Frisoni, H. Hampel, W. J. Jagust, K. A. Johnson, D. S. Knopman, *et al.*, "A/t/n: an unbiased descriptive classification scheme for alzheimer disease biomarkers," *Neurology*, vol. 87, no. 5, pp. 539–547, 2016.

[7] R. F. Allegri, L. Pertierra, G. Cohen, P. C. Méndez, M. J. Russo, I. Calandri, P. Bagnati, F. Tapajóz, F. Clarens, J. Campos, *et al.*, "A biological classification for alzheimer's disease-amyloid, tau and neurodegeneration (a/t/n): results from the argentine-alzheimer's disease neuroimaging initiative," *International psychogeriatrics*, vol. 31, no. 12, pp. 1837–1838, 2019.

[8] G. McKhann, D. Drachman, M. Folstein, R. Katzman, D. Price, and E. M. Stadlan, "Clinical diagnosis of alzheimer's disease: Report of the nincds-adrda work group* under the auspices of department of health and human services task force on alzheimer's disease," *Neurology*, vol. 34, no. 7, pp. 939–939, 1984.

[9] C. R. Jack Jr, M. S. Albert, D. S. Knopman, G. M. McKhann, R. A. Sperling, M. C. Carrillo, B. Thies, and C. H. Phelps, "Introduction to the recommendations from the

national institute on aging-alzheimer's association workgroups on diagnostic guidelines for alzheimer's disease," *Alzheimer's & Dementia*, vol. 7, no. 3, pp. 257–262, 2011.

[10] C. R. Jack Jr, M. Albert, D. S. Knopman, G. M. McKhann, R. A. Sperling, M. Carillo, W. Thies, and C. H. Phelps, "Introduction to revised criteria for the diagnosis of alzheimer's disease: National institute on aging and the alzheimer association workgroups," *Alzheimer's & dementia: the journal of the Alzheimer's Association*, vol. 7, no. 3, p. 257, 2011.

[11] J. M. Harris, J. C. Thompson, C. Gall, A. M. Richardson, D. Neary, D. du Plessis, P. Pal, D. M. Mann, J. S. Snowden, and M. Jones, "Do nia-aa criteria distinguish alzheimer's disease from frontotemporal dementia?," *Alzheimer's & Dementia*, vol. 11, no. 2, pp. 207–215, 2015.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[13] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights into imaging*, vol. 9, no. 4, pp. 611–629, 2018.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[15] F. Chollet, *Deep Learning with Python and Keras*. MITP-Verlags GmbH & Co. KG, 2018.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[18] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," *arXiv preprint arXiv:1809.00846*, 2018.

[19] G. Dougherty, *Digital Image Processing for Medical Applications*. Cambridge University Press, 2009.

[20] K. A. Johnson, N. C. Fox, R. A. Sperling, and W. E. Klunk, "Brain imaging in alzheimer disease," *Cold Spring Harbor perspectives in medicine*, vol. 2, no. 4, p. a006213, 2012.

[21] J. P. Mugler III and J. R. Brookeman, "Three-dimensional magnetization-prepared rapid gradient-echo imaging (3d mp rage)," *Magnetic resonance in medicine*, vol. 15, no. 1, pp. 152–157, 1990.

[22] J. Wang, L. He, H. Zheng, and Z.-L. Lu, "Optimizing the magnetization-prepared rapid gradient-echo (mp-rage) sequence," *PLOS ONE*, vol. 9, pp. 1–12, 05 2014.

[23] C. Jack, M. Bernstein, N. Fox, P. Thompson, G. Alexander, D. Harvey, B. Borowski, P. Britson, J. Whitwell, C. Ward, A. Dale, J. Felmlee, J. Gunter, D. Hill, R. Killiany, N. Schuff, S. Fox-Bosetti, C. Lin, C. Studholme, and M. Weiner, "The alzheimer's disease neuroimaging initiative (adni): Mri methods," *Journal of magnetic resonance imaging : JMRI*, vol. 27, pp. 685–91, 05 2008.

[24] W. Lin, T. Tong, Q. Gao, D. Guo, X. Du, Y. Yang, G. Guo, M. Xiao, M. Du, X. Qu, *et al.*, "Convolutional neural networks-based mri image analysis for the alzheimer's disease prediction from mild cognitive impairment," *Frontiers in neuroscience*, vol. 12, 2018.

[25] H. R. Roth, L. Lu, J. Liu, J. Yao, A. Seff, K. Cherry, L. Kim, and R. M. Summers, "Improving computer-aided detection using convolutional neural networks and random view aggregation," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1170–1181, 2015.

[26] H.-C. Shin, H. R. Roth, M. Gao, L. Lu, Z. Xu, I. Nogues, J. Yao, D. Mollura, and R. M. Summers, "Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning," *IEEE transactions on medical imaging*, vol. 35, no. 5, pp. 1285–1298, 2016.

[27] S. Sarraf, G. Tofighi, *et al.*, "Deepad: Alzheimer′ s disease classification via deep convolutional neural networks using mri and fmri," *BioRxiv*, p. 070441, 2016.

[28] J. V. Manjón, "Mri preprocessing," in *Imaging Biomarkers*, pp. 53–63, Springer, 2017.

[29] J. G. Sled, A. P. Zijdenbos, and A. C. Evans, "A nonparametric method for automatic correction of intensity nonuniformity in mri data," *IEEE Transactions on Medical Imaging*, vol. 17, no. 1, p. 87–97, 1998.

[30] N. J. Tustison, B. B. Avants, P. A. Cook, Y. Zheng, A. Egan, P. A. Yushkevich, and J. C. Gee, "N4itk: Improved n3 bias correction," *IEEE Transactions on Medical Imaging*, vol. 29, no. 6, p. 1310–1320, 2010.

[31] J. B. A. Maintz and M. A. Viergever, "An overview of medical image registration methods," tech. rep., In Symposium of the Belgian hospital physicists association (SBPH-BVZF), 1996.

[32] P. K. Mandal, R. Mahajan, and I. D. Dinov, "Structural brain atlases: design, rationale, and applications in normal and pathological cohorts," *Journal of Alzheimer's Disease*, vol. 31, no. s3, pp. S169–S188, 2012.

[33] B. Avants, C. Epstein, M. Grossman, and J. Gee, "Symmetric diffeomorphic image registration with cross-correlation: Evaluating automated labeling of elderly and neurodegenerative brain," *Medical Image Analysis*, vol. 12, no. 1, p. 26–41, 2008.

[34] B. Avants, N. Tustison, and G. Song, "Advanced normalization tools, ants 1.0," *Sourceforge. Jun*, 2009.

[35] S. M. Smith, "Fast robust automated brain extraction," *Human brain mapping*, vol. 17, no. 3, pp. 143–155, 2002.

[36] M. Jenkinson, C. F. Beckmann, T. E. Behrens, M. W. Woolrich, and S. M. Smith, "Fsl," *NeuroImage*, vol. 62, no. 2, pp. 782 – 790, 2012.

[37] V. Popescu, M. Battaglini, W. Hoogstrate, S. C. Verfaillie, I. Sluimer, R. A. van Schijndel, B. W. van Dijk, K. S. Cover, D. L. Knol, M. Jenkinson, *et al.*, "Optimizing parameter choice for fsl-brain extraction tool (bet) on 3d t1 images in multiple sclerosis," *Neuroimage*, vol. 61, no. 4, pp. 1484–1494, 2012.

[38] L. G. Nyúl and J. K. Udupa, "On standardizing the mr image intensity scale," *Magnetic Resonance in Medicine: An Official Journal of the International Society for Magnetic Resonance in Medicine*, vol. 42, no. 6, pp. 1072–1081, 1999.

[39] M. Shah, Y. Xiao, N. Subbanna, S. Francis, D. L. Arnold, D. L. Collins, and T. Arbel, "Evaluating intensity normalization on mris of human brain with multiple sclerosis," *Medical image analysis*, vol. 15, no. 2, pp. 267–282, 2011.

[40] C. Jack, D. W. Dickson, J. E. Parisi, Y. Xu, R. Cha, P. O'brien, S. Edland, G. Smith, B. F. Boeve, E. G. Tangalos, *et al.*, "Antemortem mri findings correlate with hippocampal neuropathology in typical aging and dementia," *Neurology*, vol. 58, no. 5, pp. 750–757, 2002.

[41] C. R. Jack, R. C. Petersen, P. C. O'Brien, and E. G. Tangalos, "Mr-based hippocampal volumetry in the diagnosis of alzheimer's disease," *Neurology*, vol. 42, no. 1, pp. 183–183, 1992.

[42] J. A. Kaye, T. Swihart, D. Howieson, A. Dame, M. Moore, T. Karnos, R. Camicioli, M. Ball, B. Oken, and G. Sexton, "Volume loss of the hippocampus and temporal lobe in healthy elderly persons destined to develop dementia," *Neurology*, vol. 48, no. 5, pp. 1297–1304, 1997.

[43] M. Laakso, H. Soininen, K. Partanen, E.-L. Helkala, P. Hartikainen, P. Vainio, M. Hallikainen, T. Hänninen, and P. Riekkinen Sr, "Volumes of hippocampus, amygdala and frontal lobes in the mri-based diagnosis of early alzheimer's disease: correlation with memory functions," *Journal of Neural Transmission-Parkinson's Disease and Dementia Section*, vol. 9, no. 1, pp. 73–86, 1995.

[44] L. A. van de Pol, A. Hensel, W. M. van der Flier, P. J. Visser, Y. A. Pijnenburg, F. Barkhof, H. J. Gertz, and P. Scheltens, "Hippocampal atrophy on mri in frontotemporal lobar degeneration and alzheimer's disease," *Journal of Neurology, Neurosurgery & Psychiatry*, vol. 77, no. 4, pp. 439–442, 2006.

[45] ANTsX, "Antsx/antspy," Dec 2019.

[46] B. B. Avants, N. J. Tustison, G. Song, P. A. Cook, A. Klein, and J. C. Gee, "A reproducible evaluation of ants similarity metric performance in brain image registration," *Neuroimage*, vol. 54, no. 3, pp. 2033–2044, 2011.

[47] K. Gorgolewski, C. Burns, C. Madison, D. Clark, Y. Halchenko, M. Waskom, and S. Ghosh, "Nipype: A flexible, lightweight and extensible neuroimaging data processing framework in python," *Frontiers in Neuroinformatics*, vol. 5, p. 13, 2011.

[48] J. C. Reinhold, B. E. Dewey, A. Carass, and J. L. Prince, "Evaluating the impact of intensity normalization on mr image synthesis," *Medical Imaging 2019: Image Processing*, Mar 2019.

[49] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization. international conference on learning representations (2015)," 2015.

[50] E. Niemantsverdriet, S. Valckx, M. Bjerke, and S. Engelborghs, "Alzheimer's disease csf biomarkers: Clinical indications and rational use," *Acta Neurologica Belgica*, vol. 117, no. 3, pp. 591–602, 2017.

[51] S. Janelidze, E. Stomrud, B. Brix, and O. Hansson, "Towards a unified protocol for handling of csf before β-amyloid measurements," *Alzheimer's research & therapy*, vol. 11, no. 1, p. 63, 2019.

[52] T. D. Bird, "Genetic aspects of alzheimer disease," *Genetics in Medicine*, vol. 10, no. 4, pp. 231–239, 2008.

[53] L. Zhong, Y.-Z. Xie, T.-T. Cao, Z. Wang, T. Wang, X. Li, R.-C. Shen, H. Xu, G. Bu, and X.-F. Chen, "A rapid and cost-effective method for genotyping apolipoprotein e gene polymorphism," *Molecular neurodegeneration*, vol. 11, no. 1, p. 2, 2016.

[54] O. Calero, L. García-Albert, A. Rodríguez-Martín, S. Veiga, and M. Calero, "A fast and cost-effective method for apolipoprotein e isotyping as an alternative to apoe genotyping for patient screening and stratification," *Scientific reports*, vol. 8, no. 1, pp. 1–8, 2018.

[55] M. F. Folstein, S. E. Folstein, and P. R. McHugh, ""mini-mental state": a practical method for grading the cognitive state of patients for the clinician," *Journal of psychiatric research*, vol. 12, no. 3, pp. 189–198, 1975.

[56] S. E. O'Bryant, J. D. Humphreys, G. E. Smith, R. J. Ivnik, N. R. Graff-Radford, R. C. Petersen, and J. A. Lucas, "Detecting dementia with the mini-mental state examination in highly educated individuals," *Archives of neurology*, vol. 65, no. 7, pp. 963–967, 2008.

[57] A. Ng, "Cs229 lecture notes," *CS229 Lecture notes*, vol. 1, no. 1, pp. 1–3, 2000.

[58] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[60] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

[61] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[62] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," pp. 2625–2634, 2015.

[63] C. Colah, "Understanding lstm networks." `http://colah.github.io/posts/2015-08-Understanding-LSTMs`, 2015.

[64] J. Shu, Z. Xu, and D. Meng, "Small sample learning in big data era," *CoRR*, vol. abs/1808.04572, 2018.

[65] A. Pasini, "Artificial neural networks for small dataset analysis," *Journal of thoracic disease*, vol. 7, no. 5, p. 953, 2015.

[66] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.