



UNIVERSIDAD DE GRANADA

Facultad de Ciencias

GRADO EN INGENIERÍA
ELECTRÓNICA INDUSTRIAL

TRABAJO FIN DE GRADO

**Extracción de
características de
imágenes médicas
cerebrales mediante redes
neuronales profundas.
Aplicación al diagnóstico
de la enfermedad de
Alzheimer**

Presentado por:

D. Ricardo Ignacio Shepstone Aramburu

Tutor:

Prof. D. Fermín Segovia Román

Curso académico 2020/2021



UNIVERSIDAD DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

Extracción de características de imágenes médicas cerebrales mediante redes neuronales profundas. Aplicación al diagnóstico de la enfermedad de Alzheimer.

Autor: Ricardo Ignacio Shepstone Aramburu

Director: Fermín Sevogía Román

Departamento: Teoría de la Señal, Telemática y Comunicaciones.

Palabras clave: Deep Learning, Machine Learning, Autoencoders, imágenes de resonancia magnética, diagnóstico asistido por ordenador.

Resumen: Construcción de estructuras basadas en redes neuronales, para la extracción de características y posterior clasificación mediante algoritmos de aprendizaje supervisado. Aplicado a la enfermedad de Alzheimer.



UNIVERSIDAD DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

AUTORIZACIÓN DE LECTURA DE TRABAJO FIN DE CARRERA

D. Fermín Segovia Román profesor del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada, como director del Trabajo Fin de Grado titulado “Extracción de características de imágenes médicas cerebrales mediante redes neuronales profundas. Aplicación al diagnóstico de la enfermedad de Alzheimer” y realizado por el alumno D. Ricardo Ignacio Shepstone Aramburu

CERTIFICA: que el citado Trabajo Fin de Grado, ha sido realizado y redactado por dicho alumno y autorizan su presentación.

Granada,

Fdo. Prof. Fermín Segovia Román.

Firma (1): FERMÍN SEGOVIA ROMÁN
En calidad de: Personal Docente e Investigador UGR





UNIVERSIDAD DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

AUTORIZACIÓN DE DEPÓSITO EN LA BIBLIOTECA

Yo, D/Dña. Ricardo Ignacio Shepstone Aramburu con DNI 77553533V, autor del Trabajo Fin de Grado titulado “Extracción de características de imágenes médicas cerebrales mediante redes neuronales profundas. Aplicación al diagnóstico de la enfermedad de Alzheimer.” realizado en la Universidad de Granada

DECLARO: explícitamente que asumo la originalidad del trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente.

AUTORIZO: al depósito de dicho Trabajo en la Biblioteca de la Universidad de Granada, y de la visualización a través de Internet.

Granada, 15 de Junio de 2021.

Fdo. D.

Extracción de características de imágenes médicas cerebrales mediante redes neuronales profundas. Aplicación al diagnóstico de la enfermedad de Alzheimer.

Resumen

La llegada del Deep Learning ha supuesto una revolución en la sociedad, permitiendo la resolución de problemas complejos gracias a la gran capacidad que tienen estos algoritmos de aprender patrones en grandes volúmenes de datos.

Por otro lado, los avances en el campo de las imágenes médicas, han mejorado el diagnóstico y el estudio de ciertas patologías que, anteriormente, producían muchas incógnitas.

En este trabajo se busca combinar las grandes destrezas del Deep Learning para el reconocimiento de patrones, con la valiosa información concedida por las imágenes médicas, para construir una herramienta que facilite el diagnóstico de una de las principales enfermedades neurodegenerativas, la enfermedad de Alzheimer.

Concretamente, se utilizarán estructuras de redes neuronales convolucionales, que destacan por su aplicabilidad a imágenes (tanto de dos dimensiones, como de tres), combinadas con algoritmos de aprendizaje supervisado; con el fin de realizar un diagnóstico de imágenes estructurales de resonancia magnética y clasificarlas debidamente.

En este proyecto se pretende también introducir los conceptos fundamentales sobre el aprendizaje automático. Con el fin de facilitar la comprensión del funcionamiento del modelo.

Por último este modelo se utilizará para realizar predicciones sobre un conjunto de datos, cuyos resultados son conocidos, con el objetivo de evaluarlo. Para su evaluación se utilizarán ciertas medidas características usadas en la clasificación, además de técnicas que permiten estimar su capacidad de generalización.

Feature extraction from medical brain images using deep neural networks.
Applied to Alzheimer's disease.

Abstract

The arrival of Deep Learning has meant a revolution in society, allowing complex problem solving thanks to the great capabilities that these algorithms offer on pattern recognition in huge amounts of data.

On another note, the advancements in the field of medical imaging have improved the diagnosis and research of certain pathologies which previously posed lot of questions.

In this work we look to combine the great advantages of Deep Learning for pattern recognition, with the priceless information granted by medical images, to build a tool that facilitates one of the main neurodegenerative diseases, Alzheimer's disease.

Specifically, convolutional neural network structures will be used. These are structures that shine when applied to images (in two and three dimensions) are combined with supervised training algorithms to perform structural magnetic resonance image diagnosis and to classify the images accordingly.

In this work, we aim to introduce fundamental supervised learning concepts. All to ease the understanding of how the model Works.

Lastly, this model will be used to make predictions on a set of data whose results are known, in order to evaluate it. For the model's evaluation, we will use certain measures used in classification problems, also we will use some techniques specialised in estimating the model's generalizability.

Índice general

1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
2. Enfermedad de Alzheimer	3
2.1. Etiología	4
2.2. Anatomía patológica	5
2.3. Cuadro clínico	7
2.4. Diagnóstico	8
2.5. Tratamiento	8
2.6. Imágenes estructurales por resonancia magnética	9
2.6.1. Comportamiento de un núcleo en un campo magnético	9
2.6.2. Excitación del núcleo	12
2.6.3. Relajación del núcleo	13
2.6.4. Formación de una imagen	15
3. Segmentación mediante SPM y preprocesado de imágenes	19
3.1. Fundamento teórico de la segmentación	19
3.1.1. Intensidad del vóxel según el tipo de tejido	20
3.1.2. Modelado por mezcla de Gaussianas y obtención de la función objetivo	21
3.1.3. Falta de uniformidad en la intensidad	23
3.1.4. Mapas de probabilidad deformables	24
3.1.5. Optimización de la función objetivo	25
3.2. Preprocesado de datos	28
4. Redes neuronales artificiales	31
4.1. Machine Learning y Deep Learning	31
4.2. El perceptrón	34
4.3. Perceptrón Multicapa	35
4.4. Entrenamiento de una red neuronal: forwardpropagation, back- propagation y función de loss	36
4.5. Redes neuronales convolucionales	38

4.6. Hiperparámetros	43
4.7. Autoencoders	46
5. Clasificadores estadísticos	51
5.1. Definición	51
5.2. Árboles de decisión	52
5.3. Maquinas de vectores de soporte	54
5.3.1. SVM para clases linealmente separables	54
5.3.2. SVM para clases no linealmente separables	57
5.3.3. SVM no lineales	59
5.4. Medidas de rendimiento	60
5.5. Validación cruzada	62
5.5.1. Validación cruzada k-fold	62
6. Implementación	65
6.1. Bases de datos	65
6.2. Carga de datos y preprocesado	66
6.3. Construcción e implementación del modelo	67
6.3.1. Contrucción del Autoencoder	67
6.3.2. Evaluación del modelo	70
6.3.3. Funcionamiento del programa principal	70
7. Análisis de los resultados	75
7.1. Clasificación multiclase	75
7.2. Clasificación binaria	79
8. Conclusiones y trabajo futuro	83
8.1. Conclusiones	83
8.2. Proyección de futuro	84

Índice de figuras

2.1. Fotografía de Alois Alzheimer [1].	3
2.2. Diferencias a nivel celular (A y B) y de volumen (C), entre un cerebro sano y uno afectado por Alzheimer[2].	6
2.3. Momento magnético producido por el espín del núcleo, análogo a un imán de barra. Las flechas contenidas en el núcleo indican la dirección del espín, mientras que las de fuera indican la dirección del campo magnético [3].	10
2.4. Movimiento de precesión de un núcleo en presencia de un campo magnético externo, similar al de una peonza [3]. . . .	10
2.5. Un conjunto de núcleos sometidos a un campo magnético B_0 , con una magnetización neta resultante M_0 debido al mayor número con espín positivo [3].	11
2.6. El efecto Zeeman para un espín nuclear de $I = \frac{1}{2}$, se separa en dos niveles de energía en presencia de un campo magnético [4].	12
2.7. Representación de la variación del vector de magnetización nuclear \vec{M} al someterlo a un campo magnético giratorio; a la derecha se muestra la variación del ángulo con el eje longitudinal [3].	13
2.8. A la izquierda, el movimiento que describe el vector de magnetización nuclear \vec{M} cuando cesa el campo rotatorio y vuelve al estado de equilibrio. A la derecha, la señal que produce el vector de magnetización en la componente del plano transversal si se sitúa una bobina en uno de los ejes (x o y) [3]. . .	14
2.9. Representación del método utilizado para evaluar una sección. El campo \vec{B}_0 varía según el gradiente aplicado, haciendo que sólo la sección de interés cumpla la condición de resonancia, permitiendo así su estudio [3].	16
2.10. El campo \vec{B}_0 varía según los gradientes para realizar la codificación en frecuencia en uno de los ejes y la codificación en fase en el otro [3].	16
2.11. A la izquierda, la señal obtenida en el espacio K. A la derecha, la reconstrucción de la imagen de la sección bajo estudio [3]. .	17

3.1.	Variación de la intensidad del vóxel debido al volumen de cada tipo de tejido (materia gris o blanca) contenido en él [5].	20
3.2.	Gráfica mostrando la distribución de vóxeles según sus intensidades y la clase de tejido a la que pertenecen, en una función de densidad de probabilidad [6].	21
3.3.	A la izquierda, la imagen original afectada por <i>bias</i> . En el centro, el campo de corrección. A la derecha, la imagen corregida [7].	23
3.4.	Mapas de probabilidad de distintos tejidos, provenientes del ICBM (<i>International Consortium for Brain Mapping</i>) [7]. . .	24
3.5.	Imagen médica tipo T1 a segmentar.	26
3.6.	Resultado de la segmentación para materia blanca.	27
3.7.	Resultado de la segmentación para materia gris.	27
4.1.	Esquema de la jerarquización del Machine Learning y Deep Learning [8].	33
4.2.	Esquema simple de modelo de Deep Learning [8].	33
4.3.	Esquema de un perceptrón.	34
4.4.	Funciones de activación para una red neuronal.	35
4.5.	Esquema simple de optimización por gradiente descendiente [8].	38
4.6.	Diagrama de operación de convolución en una imagen.	40
4.7.	Esquema representando el cambio de dimensión y el número de salidas en una capa convolucional.	40
4.8.	Esquema representando el cambio de tamaño tras una capa de <i>pooling</i>	41
4.9.	Esquema indicando el número de salidas en función del número de entradas, tras una operación de <i>pooling</i>	42
4.10.	Representación de convolución en tres dimensiones con un <i>kernel</i> cúbico.	42
4.11.	Gráficas que representan distintos <i>learning rates</i> para el mismo problema [9].	44
4.12.	Gráfica de una función de <i>loss</i> con mínimos locales [8].	45
4.13.	Representación de una matriz de números a la que se la ha aplicado <i>padding</i> [8].	46
4.14.	Arquitectura de un <i>autoencoder</i> [10].	47
4.15.	Ejemplo de un <i>autoencoder</i> constituido por varias capas de neuronas densamente conectadas.	48
4.16.	Representación por pasos de una operación de convolución transpuesta.	49
4.17.	Ejemplo sencillo de un <i>upsampling</i>	49

5.1. Dos distribuciones binarias en un espacio de características bidimensional. A la izquierda se han separado las dos clases mediante una recta. A la derecha se han separado mediante un polinomio [11].	53
5.2. Máquina de vectores de soporte separando elementos de dos clases linealmente separables [12].	55
5.3. Valores de ξ según la posición de un dato con respecto a la recta clasificadora para una distribución bidimensional de clases no linealmente separables [13].	58
5.4. Validación cruzada para $k=5$	63
6.1. Diagrama que muestra la estructura capa por capa de AE-1	67
6.2. Diagrama que muestra la estructura capa por capa de AE-2	68
6.3. Diagrama que muestra la estructura capa por capa de AE-3	68
6.4. Diagrama de flujo del módulo principal.	73
7.1. Gráfica mostrando precisión media según el número de neuronas de la capa central del <i>autoencoder-3</i>	75
7.2. Gráfica que muestra el error cuadrático medio del <i>autoencoder-3</i> para la clasificación con árboles de decisión.	76
7.3. Gráfica que muestra el error cuadrático medio del <i>autoencoder-3</i> para la clasificación con máquinas de vectores de soporte.	77
7.4. Matriz de confusión del clasificador basado en árboles de decisión.	78
7.5. Matriz de confusión del clasificador basado en máquinas de vectores de soporte.	78
7.6. Gráfica que muestra el error cuadrático medio del <i>autoencoder-3</i> en uno de los folds, para la clasificación AD vs Control.	80
7.7. Gráfica que muestra el error cuadrático medio del <i>autoencoder-3</i> en uno de los folds, para la clasificación MCI vs Control.	80
7.8. Gráfica que muestra el error cuadrático medio del <i>autoencoder-3</i> en uno de los folds, para la clasificación AD vs MCI.	81

Índice de cuadros

5.1. Tabla resumen de resultados posibles de la predicción de un clasificador en función de la etiqueta original.	60
6.1. Tabla que muestra el número de imágenes pertenecientes a cada clase.	66
7.1. Resultados para la clasificación multiclase.	77
7.2. Resultados de la precisión obtenida para las distintas estructuras.	79
7.3. Resultados de las clasificaciones binarias utilizando un clasificador SVM.	81

Capítulo 1

Introducción

1.1. Motivación

Los avances en el campo de la medicina desde sus orígenes han tenido una estrecha relación con el descubrimiento de nuevas tecnologías. Inventos como el microscópio o el desfibrilador, entre otros, supusieron enormes cambios para la sociedad con mejores diagnósticos, tratamientos eficaces y una infraestructura sanitaria como nunca antes vista. Estos avances permitieron al ser humano mejorar su esperanza y calidad de vida. Aunque todavía quedan áreas de la medicina de las que aún queda por investigar, entre ellas están las enfermedades neurodegenerativas.

Las enfermedades neurodegenerativas son uno de los grandes males que azotan actualmente a la humanidad suponiendo uno de los grandes retos del siglo XXI. Entre ellas se puede destacar la demencia con el mayor responsable, la enfermedad de Alzheimer, no solamente por la ausencia de una cura, si no por el carácter degenerativo de esta, suponiendo una gran carga emocional tanto para el paciente como para sus familiares. Afortunadamente, avances tecnológicos en el campo de las neuroimágenes han permitido una mejor comprensión de este tipo de patologías y su evolución de una forma no invasiva.

La llegada del Deep Learning y su gran capacidad de reconocimiento de patrones, gracias en gran medida a las redes neuronales convolucionales, ha supuesto una revolución en la rama de la inteligencia artificial. Debido a estas cualidades, cada vez se proponen más modelos basados en este tipo de técnicas para el diagnóstico asistido por ordenador. Es aquí donde surge una gran oportunidad de combinar estos algoritmos con el campo de la neuroimagen para el diagnóstico de enfermedades neurodegenerativas, no sólo con el fin de ayudar a los profesionales médicos a realizar su labor, sino también con el objetivo de darle un nuevo enfoque a la investigación de estas patologías neurológicas.

1.2. Objetivos

El objetivo principal de este proyecto será la creación de un modelo clasificador para el diagnóstico asistido por ordenador aplicado a la enfermedad de Alzheimer. A partir de imágenes estructurales de resonancia magnética ya diagnosticadas, el modelo deberá aprender la relación que existe entre los píxeles de esas imágenes y la categoría a la que pertenece el paciente; con el fin de poder realizar predicciones correctas sobre imágenes nuevas.

El primer paso por lo tanto es obtener los datos necesarios para introducir al modelo. Partiendo de imágenes médicas de resonancia magnética, se deberán obtener los tejidos de interés para el análisis mediante un procesamiento de imágenes.

El modelo utilizará diferentes técnicas de Deep Learning y algoritmos de Machine Learning para aprender las relaciones de los datos proporcionados y las clases a las que pertenecen. Para ello se utilizarán capas de convolución en combinación con otro tipo de capas para la extracción de características a partir de las imágenes médicas, estas características serán empleadas para el aprendizaje de un clasificador, que deberá establecer la relación entre las características sacadas de las imágenes y el diagnóstico asociado a ese dato.

Se harán evaluaciones para clasificaciones multiclase (múltiples clases a la vez) y para clasificaciones binarias (solamente dos clases). Se utilizarán medidas de rendimiento con el objetivo de cuantificar la validez de la clasificación y técnicas de validación cruzada para medir la capacidad de generalización del modelo.

Capítulo 2

Enfermedad de Alzheimer

Descubierta por Alois Alzheimer y reportado por primera vez en 1907 (Alzheimer, 1907 [14]), al hacer un seguimiento de uno de sus pacientes que al ser hospitalizado ya mostraba síntomas avanzados de la enfermedad: pérdida de orientación, memoria reducida, etc. Tras su fallecimiento en Frankfurt en 1906, al analizar sus tejidos cerebrales se encontraron placas de amiloide y ovillos neurofibrilares.

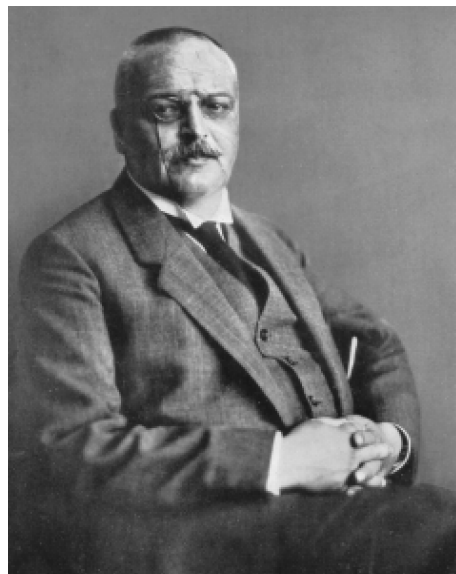


Figura 2.1: Fotografía de Alois Alzheimer [1].

Se trata de un trastorno neurológico que afecta progresivamente al paciente que la padece, destruyendo las células nerviosas del cerebro. Comienza con síntomas que se pueden atribuir a la edad, como olvidos comunes; degenerando a deterioros en la capacidad cognitiva, afectando por ejemplo a la toma de decisiones; acabando en daños cerebrales severos e incluso la muerte

del paciente.

Se define la demencia como el deterioro adquirido de forma progresiva en las capacidades cognitivas hasta el punto que resulta un impedimento para la realización de tareas y actividades cotidianas [15]. Existen varios trastornos y enfermedades que pueden causar demencia mediante procesos que llevan a la destrucción de células cerebrales, entre ellos están la enfermedad de Parkinson, demencia vascular, demencia con cuerpos de Lewy, etc. Algunas condiciones como problemas con la glándula tiroides o ciertos efectos producidos por fármacos o drogas, pueden llevar a síntomas parecidos a los que produce la demencia, aunque con un diagnóstico temprano y consecuente tratamiento estos efectos pueden ser incluso revertidos. Pero la causa más común de demencia es debido a la enfermedad de Alzheimer; según la OMS, se estima que entre 60 % y un 70 % de los casos de demencia son causados por esta enfermedad. Es por ello que no sólo el diagnóstico de la demencia es importante, también es de vital importancia distinguir entre las condiciones que pueden haberla producido en un paciente para su correcto tratamiento.

La enfermedad de Alzheimer se ha convertido en la sexta causa de mayor mortalidad y la quinta para personas mayores de 65 años en Estados Unidos [16][2020 Alzheimer's disease facts and figures]. Siendo esta condición es más usual en personas mayores de 65 años, aumentando el riesgo con la edad, acorta la esperanza de vida además de ser una de las causas principales de discapacidad física. Diversos estudios indican que de media una persona con Alzheimer vive de cuatro a ocho años tras ser diagnosticada, aunque existen casos que han vivido hasta veinte años con este tipo de demencia [17]. Sin embargo es su carácter degenerativo e irreversible que convierte a esta enfermedad en un problema grave en los países desarrollados, afectando a la sociedad tanto a nivel de sistema sanitario, como a las familias con miembros que la padecen.

2.1. Etiología

Las causas de la enfermedad son por ahora desconocidas, solamente se conocen algunos factores de riesgo. Siendo el principal la edad, lo que sugiere que ciertos procesos biológicos relacionados con el envejecimiento pueden llevar a la aparición de la enfermedad. Esta fuerte relación con el envejecimiento también podría reflejar los factores de riesgo y protección que podrían tener un efecto acumulativo a lo largo de una vida. Es por ello que en función de la edad de aparición de los síntomas, la enfermedad se puede clasificar en:

- Enfermedad de Alzheimer de inicio precoz, si comienza antes de los 65 años.
- Enfermedad de Alzheimer de inicio tardío, si empieza después de los

65 años.

Muchos de los casos de inicio precoz están causados por una mutación genética en la familia y avanzan rápidamente. Todas las mutaciones conocidas de este tipo resultan en una sobreproducción de proteína amiloide. Aunque estos casos solamente se considera que ocurren entre el 2 % y 5 % de todos los pacientes con Alzheimer [18]. Existe un riesgo mayor de parecer la enfermedad para familiares de primer grado de un paciente de Alzheimer, ya que tanto factores genéticos como del entorno entran en juego al producir este fenómeno de agregación familiar. Entre los factores genéticos destaca la variante común del gen de apolipoproteína E (apoE) que promueve la sedimentación de la proteína dañina [19]. Otros factores genéticos pueden estar involucrados pero de momento no han sido descubiertos.

Diversos estudios multidisciplinarios recientes apuntan a que existe evidencia, tanto moderada como fuerte, que apoya la relación de factores de riesgo cardiovascular y morbilidad cardiovascular con el riesgo a padecer demencia, e incluso, Alzheimer [20]. Estos factores de riesgo podrían incluir, entre otros, obesidad, ser fumador y niveles de colesterol altos; en cuanto a la morbilidad cardiovascular hace referencia a condiciones o enfermedades que afectan al sistema cardiovascular, como tener presión arterial alta, diabetes y lesiones en la materia blanca.

Una revisión sistemática encontró de que un estilo de vida activo y factores psicosociales podrían reducir el riesgo de la aparición de demencia [21]. Entre estos factores psicosociales se consideraron: un nivel educativo alto en la edad temprana, una vida social rica y estimulante en edades más avanzadas, y alta complejidad en el entorno laboral durante la adultez. En cuanto al estilo de vida activo, hace referencia a la participación en actividades estimulantes tanto físicamente, como mentalmente, a lo largo de la vida.

Otros factores como la dieta, inflamación, exposición a compuestos tóxicos y más factores ambientales siguen bajo estudio por falta de evidencia.

2.2. Anatomía patológica

Aunque las causas que producen la enfermedad son desconocidas, los procesos que tienen lugar a nivel físico y la evolución en el cerebro se comprenden mejor. La enfermedad de Alzheimer se caracteriza por la degeneración de las neuronas en los tejidos cerebrales mediante la formación de placas neuríticas (también conocidas como placas amiloides) y nudos neurofibrilares. Estas lesiones aparecen primero en determinadas zonas y se van expandiendo paulatinamente por el neocórtex, produciendo los síntomas en distintas etapas, tal y como se verá en el siguiente apartado.

Las placas neuríticas son formaciones de la proteína beta-amiloide que aparecen entre las neuronas interfiriendo con la sinapsis, es decir, interfiriendo en la comunicación de neurona a neurona. En el caso de un cerebro sano,

estas proteínas que se producen de forma natural serían descompuestas y eliminadas; en cambio, en el de una persona afectada por esta enfermedad, las proteínas se acumulan formando placas duras e insolubles.

Los nudos neurofibrilares son fibras insolubles entrelazadas, formadas principalmente por proteínas tau, que se encuentran dentro de las neuronas. Estas proteínas forman parte de unas estructuras llamadas microtúbulos, que se encargan del transporte de nutrientes y otras moléculas esenciales dentro de las neuronas. Cuando la proteína tau es excesiva, los microtúbulos se ven comprometidos interfiriendo por lo tanto en su función.

Otros efectos y cambios que experimenta el cerebro son la inflamación y la atrofia. La inflamación es una respuesta por parte del sistema inmunológico cuando las células microgliales (células que forman parte del sistema inmunitario dentro del sistema nervioso central) no pueden lidiar con la limpieza de una gran cantidad de proteínas tóxicas y desechos de células muertas. La atrofia de los tejidos cerebrales se produce por la muerte de las células que los componen.

En un cerebro afectado en una etapa avanzada de la enfermedad se puede observar una disminución del tamaño de la corteza, afectando a regiones cerebrales encargadas del pensamiento, la planificación y recordar. El hipocampo se ve especialmente afectado por un encogimiento severo, región importante a la hora de la creación de nuevos recuerdos. Debido a la reducción de volumen del tejido, los ventrículos, que son los espacios llenos de líquido cefalorraquídeo dentro del cerebro, son de un tamaño mayor que el de una persona sana.

En la figura 2.2 se puede observar las diferencias descritas anteriormente, entre el cerebro de una persona sana y el de un paciente de Alzheimer.

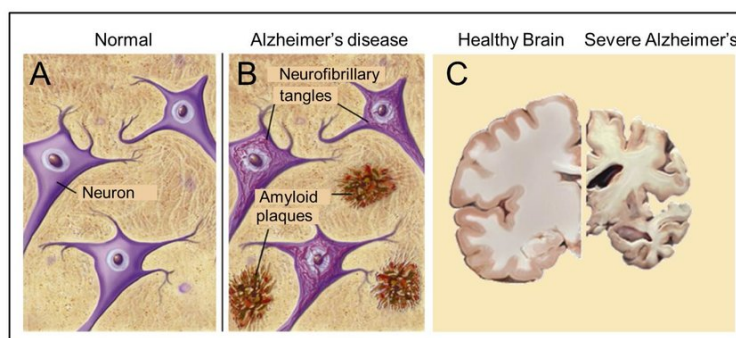


Figura 2.2: Diferencias a nivel celular (A y B) y de volumen (C), entre un cerebro sano y uno afectado por Alzheimer[2].

2.3. Cuadro clínico

La evolución de la enfermedad, desde su comienzo con síntomas casi imperceptibles hasta que causa problemas con la vida diaria del paciente, afecta a cada persona de forma diferente. Esta evolución se puede dividir en tres etapas, aunque la duración de cada etapa varía de paciente a paciente, ya que existen diversos factores que pueden influir en su desarrollo. Las tres etapas se caracterizarán por lo tanto, por sus síntomas particulares.

- En la etapa inicial, el síntoma principal de la enfermedad es el olvido. El paciente colocará objetos en sitios inusuales y olvidará su localización. Su sentido de la orientación y percepción temporal se verán afectados, causando que a menudo el paciente se sienta perdido, especialmente en lugares desconocidos. El hecho de aprender nuevas tareas o realizar tareas complejas, puede resultar problemático; en cambio, la realización de aquellas tareas que son bien conocidas no se ve enormemente afectada, es por ello que el paciente puede mantener cierto grado de autonomía con la supervisión para ciertas actividades. Esta etapa es tan gradual que puede llevar mucho tiempo hasta que se empiecen a notar los síntomas.
- En la etapa intermedia la pérdida de memoria es apreciable, hasta el punto que interfiere con las actividades cotidianas (como vestirse y asearse) y las capacidades comunicativas del paciente, requiriendo por lo tanto la atención de un cuidador. El paciente se suele encontrar desorientado incluso en lugares familiares, además de que su percepción y su criterio se ven gravemente afectados. Es en esta etapa donde los cambios de comportamiento se vuelven aparentes, los pacientes se vuelven desconfiados y alterados. En algunos casos los comportamientos característicos son de agresión, hostilidad y comportamiento inapropiado, mientras que en otros son de apatía. Aparecen ciertos trastornos mentales como alucinaciones y paranoia.
- Una vez se alcanza la etapa avanzada, el paciente sufre de un deterioro severo de funciones cognitivas y corporales, por lo que la dependencia se vuelve total requiriendo asistencia para todas las tareas comunes. Los síntomas se vuelven obvios, entre ellos están: incapacidad de reconocer a familiares y amigos, la capacidad de comunicación verbal disminuye drásticamente, e incluso, se vuelve imposible para algunos pacientes, pérdida de la coordinación y de funciones motrices, lo que dificulta la habilidad para caminar además de causar que algunos pacientes tengan que ser encamados.

2.4. Diagnóstico

Actualmente, no existe una prueba única para el diagnóstico de demencia producida por enfermedad de Alzheimer. Para su diagnóstico se realizan varias pruebas además de evaluaciones físicas y psiquiátricas del paciente, con el objetivo de ir descartando distintos factores o condiciones posibles, como por ejemplo, otros tipos de demencia.

A la hora de determinar el diagnóstico se distinguen entre dos categorías: posible y probable. Los dos se basan en la observación y seguimiento de los síntomas, además de distintas pruebas para determinar el deterioro de dos o más funciones cognitivas, tales como memoria, habilidades lingüísticas, etc. La diferencia que existe entre estos dos diagnósticos son que en el caso posible existe una segunda enfermedad que hace la determinación más complicada, mientras que en el probable no. La única manera de confirmar el diagnóstico certero es mediante la identificación de las placas neuríticas y ovillos neurofibrilares, lo que solo es posible mediante un análisis de los tejidos que se realiza post-mortem.

En los casos con bastante evidencia que el paciente padece la enfermedad, se toman imágenes cerebrales para ayudar con el diagnóstico. Se utilizan dos tipos de neuroimagen: estructurales y funcionales. Entre las estructurales están la Resonancia Magnética (MRI) y Tomografía Axial Computarizada (TAC); para las enfermedades neurodegenerativas, estas técnicas proporcionan información sobre los cambios estructurales del cerebro a medida que evoluciona la enfermedad, como se ha mencionado anteriormente, el hipocampo se ve reducido considerablemente, siendo este un signo que aparece en etapas tempranas. Entre las funcionales están la Tomografía Computarizada por Emisión de un Solo Fotón (SPECT) y Tomografía por Emisión de Positrones (PET); que permiten la visualización de un proceso bioquímico concreto mediante el uso de radiofármacos, que para enfermedades neurodegenerativas proporcionan información sobre el flujo sanguíneo cerebral y para medir regionalmente el consumo de glucosa.

2.5. Tratamiento

En la actualidad, no existe una cura para la enfermedad de Alzheimer. Los tratamientos farmacológicos son utilizados, por lo general, para tratar los síntomas. Los fármacos utilizados principalmente son anticolinesterásicos; actúan inhibiendo la colinesterasa, que es una enzima que descompone la acetilcolina. En el cerebro de un paciente de Alzheimer, los niveles de acetilcolina son bajos, manteniendo los niveles de este neurotransmisor mediante anticolinesterásicos se consigue una mejora de los síntomas que afectan a las funciones cognitivas. La efectividad de este tipo de fármacos varía de persona a persona y además tiene una efectividad limitada. Los principales

anticolinérgicos disponibles son: donepezilo, galantamina y rivastigmina.

Existen también otros tratamientos no farmacológicos que buscan mejorar la calidad de vida del paciente, mejorar funciones cognitivas y reducir síntomas que afectan al comportamiento. Un ejemplo podría ser la escucha de música preferida para incitar recuerdos.

2.6. Imágenes estructurales por resonancia magnética

En este proyecto de diagnóstico automático, se han utilizado imágenes médicas de resonancia magnética; por lo que en esta sección se tratan de explicar los fundamentos físicos en los que se basa este tipo de tecnología, además de su implementación para la obtención de las imágenes.

2.6.1. Comportamiento de un núcleo en un campo magnético

El término de resonancia magnética viene de un fenómeno físico llamado resonancia magnética nuclear (RMN). Este fenómeno está relacionado con el espín de los núcleos de ciertos átomos por hacer que posean un pequeño momento magnético. A continuación se van a presentar estos conceptos en detalle y explicar lo que pasa cuando se somete el núcleo a un campo magnético.

El espín es una propiedad fundamental de ciertos núcleos que contienen protones o neutrones desparejados, aquellos isótopos que tengan un espín nulo no serán activados por la RMN; en cambio, aquellos que presenten un espín no nulo si se verán afectados, por ejemplo los átomos de hidrógeno ^1_1H que forman parte de la molécula del agua, elemento abundante en los tejidos del cuerpo humano. Aunque se trate de un concepto de la mecánica cuántica, podemos imaginar el espín como la rotación de un núcleo sobre su eje. Al estar cargado el núcleo positivamente por la presencia de protones, el espín produce un momento magnético μ , que es un pequeño campo magnético análogo al de un pequeño imán (figura 2.3).

Al derivar el momento magnético del espín, su orientación es la misma, definido por la ecuación:

$$\vec{\mu} = \gamma \vec{I}$$

Donde I es el espín y γ es la relación giromagnética que es una propiedad del núcleo. El espín y el momento magnético vienen dados como magnitudes vectoriales, indicando que son paralelos.

Si el núcleo se somete a un campo magnético \vec{B}_0 , en vez de alinearse su momento magnético con el campo externo, describe un movimiento de precesión, similar al de una peonza, alrededor de la dirección del campo externo. Este movimiento viene representado en la figura 2.4 y su frecuencia

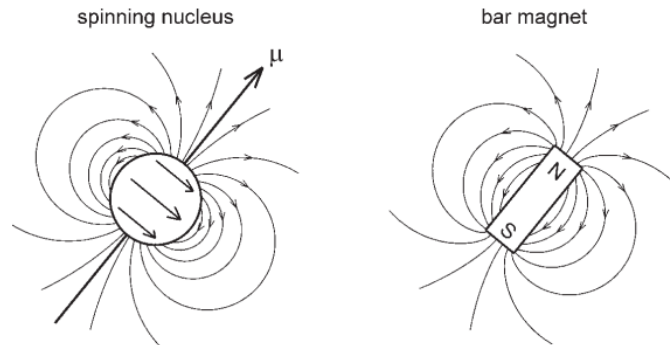


Figura 2.3: Momento magnético producido por el espín del núcleo, análogo a un imán de barra. Las flechas contenidas en el núcleo indican la dirección del espín, mientras que las de fuera indican la dirección del campo magnético [3].

angular viene dada por la frecuencia de Larmor ω_L , que viene dada por la expresión:

$$\omega_L = \gamma B_0$$

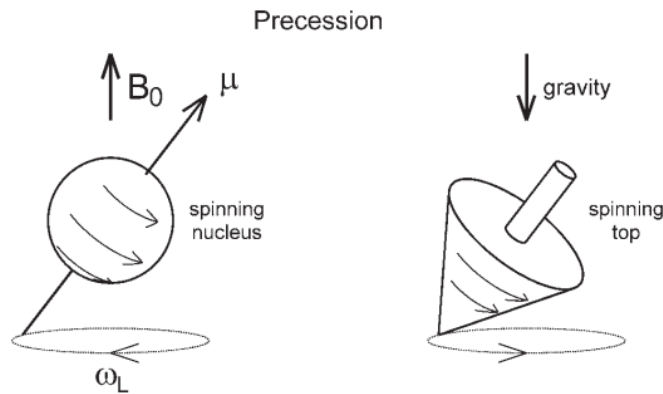


Figura 2.4: Movimiento de precesión de un núcleo en presencia de un campo magnético externo, similar al de una peonza [3].

En un determinado volumen de tejido, la suma vectorial a nivel macroscópico de los momentos magnéticos de los núcleos atómicos contenidos en ese volumen se conoce como la magnetización nuclear, y se expresa mediante \vec{M} . Al someter estos núcleos al campo magnético externo \vec{B}_0 , empiezan a experimentar el movimiento de precesión mientras algunos núcleos se alinean con el campo en su mismo sentido (paralelo) y otros se alinean en el

sentido opuesto (antiparalelo). Entonces el vector \vec{M} , que describe el campo magnético producido por todos los núcleos en ese volumen, se ve afectado por el nuevo movimiento del momento magnético de cada elemento; para entender mejor este cambio, se descompone el vector en dos componentes:

- La componente contenida en el plano transversal (perpendicular al campo \vec{B}_0), que depende de la suma de las componentes contenidas en el mismo plano de cada núcleo, por lo que vienen determinadas por el movimiento de precesión de cada elemento. Cada uno de los núcleos describe este movimiento con una fase diferente, por lo que la suma de todas las componentes se puede considerar nula, ya que se anulan entre ellas al haber otros núcleos girando con fase opuesta.
- La componente longitudinal (paralelo al campo \vec{B}_0), que dependerá de la suma de las componentes individuales en esta dirección, por lo que viene determinado número de núcleos alineados en paralelo y en antiparalelo. En este caso, habrá un número ligeramente superior de elementos alineados en el sentido del campo que en el sentido contrario, por lo que habrá una pequeña magnetización neta llamada magnetización en equilibrio \vec{M}_0 , que se encuentra en la componente longitudinal con el mismo sentido que \vec{B}_0 .

A este estado se le conoce como equilibrio, en la figura 2.5 se puede apreciar un grupo de núcleos en este estado.

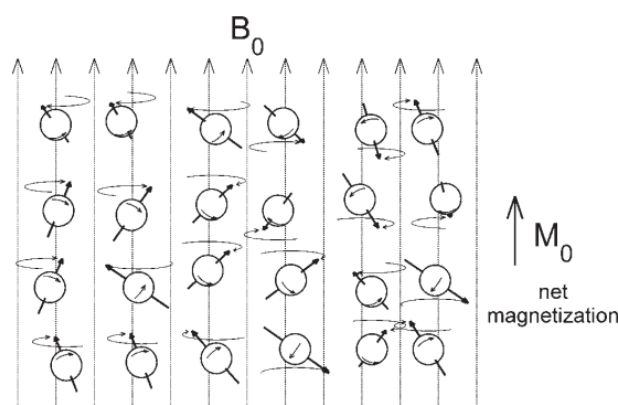


Figura 2.5: Un conjunto de núcleos sometidos a un campo magnético B_0 , con una magnetización neta resultante M_0 debido al mayor número con espín positivo [3].

En cuanto a la razón por la que hay más núcleos en paralelo que antiparalelo, es porque la energía de un momento magnético depende de su orientación con respecto al campo:

$$E = -\vec{\mu} \cdot \vec{B}_0$$

Lo que quiere decir que la energía será menor cuanto más alineado esté el momento con el campo magnético. Esta energía es minúscula en comparación a la energía térmica disponible, pero suficiente para causar una diferencia en la población de núcleos con espines opuestos. La población de en un sistema de núcleos con espines $\pm\frac{1}{2}$, como los de hidrógeno, viene determinada por:

$$\frac{N_{\downarrow}}{N_{\uparrow}} = \exp\left(-\frac{\Delta E}{kT}\right)$$

Donde N es el número de núcleos con espines en las dos direcciones (indicadas por la flecha), k es la constante de Boltzmann $1,3806488 \times 10^{-23} \frac{J}{K}$, T la temperatura corporal y ΔE la diferencia en los niveles de energía entre un núcleo de espín positivo y otro negativo en presencia de un campo magnético (efecto Zeeman, figura 2.6), que viene dada por:

$$\Delta E = \hbar\gamma B_0$$

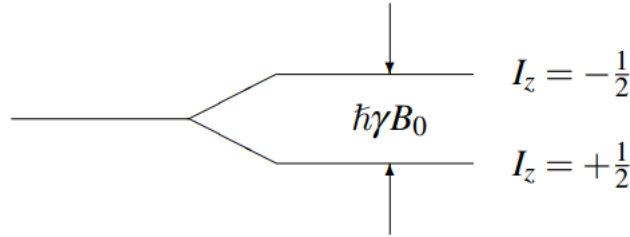


Figura 2.6: El efecto Zeeman para un espín nuclear de $I = \frac{1}{2}$, se separa en dos niveles de energía en presencia de un campo magnético [4].

Siendo \hbar la constante de Planck reducida ($\hbar = \frac{h}{2\pi}$ con $h = 6,62607015 \times 10^{-34} J \cdot s$). Con estas ecuaciones se puede concluir que la magnetización en equilibrio \vec{M}_0 es proporcional al campo externo \vec{B}_0 aplicado; en la práctica los valores de este campo suelen estar entre 1 y 3 Teslas, puesto que un valor alto aumenta el SNR.

2.6.2. Excitación del núcleo

Si al conjunto de núcleos en equilibrio se le aplica un campo magnético transversal oscilante con frecuencia angular igual a la frecuencia de Larmor ω_L , se puede alejar la magnetización nuclear del eje longitudinal, de esta forma la componente del plano transversal ya no es nula. Esto se produce porque los núcleos se sincronizan en su movimiento de precesión tanto para los que están en paralelo como los que están en antiparalelo con el eje longitudinal; a este fenómeno es al que se le conoce como resonancia magnética nuclear.

El campo magnético rotatorio en el plano transversal se denota por $\vec{B}_1(t)$ y su velocidad angular tiene que ser igual a la frecuencia de Larmor para que se cumpla la condición de resonancia $\omega_{RF} = \omega_L$, como esta frecuencia suele ser del mismo orden que las que se usan en radio, este proceso se suele describir como excitación RF. Conforme el campo rotatorio transfiere energía al sistema, la componente del plano transversal del vector de magnetización nuclear \vec{M} aumenta y la componente longitudinal disminuye (figura 2.7), esto se debe a que aumenta la población de espines antiparalelos (pasan a un nivel de energía superior).

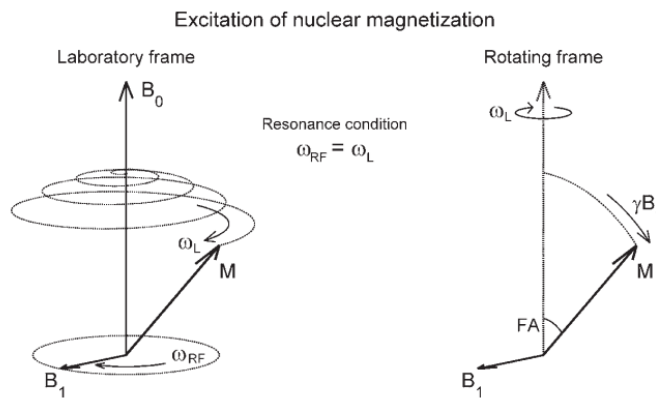


Figura 2.7: Representación de la variación del vector de magnetización nuclear \vec{M} al someterlo a un campo magnético giratorio; a la derecha se muestra la variación del ángulo con el eje longitudinal [3].

La excitación por parte del campo rotatorio $\vec{B}_1(t)$ se aplica en ráfagas cortas conocidas como pulsos RF, calculados para producir cierto grado de excitación en el sistema. Esto se puede calcular mediante el ángulo que forman la magnetización neta y el eje longitudinal, que se le conoce por FA (del inglés *flip angle*). Este ángulo es proporcional a la amplitud del campo rotatorio B_1 y la duración del pulso τ , de manera que:

$$FA = \gamma B_1 \tau$$

2.6.3. Relajación del núcleo

Una vez se acaba el pulso del campo oscilante $\vec{B}_1(t)$ los núcleos perderán esa energía suministrada por interacciones con otros núcleos y moléculas en su vecindad, haciendo que el sistema regrese al estado de equilibrio, tal y como se muestra en la figura 2.8.

Es este fenómeno llamado relajación, se puede detectar mediante unas bobinas, ya que pueden recibir la señal del vector de magnetización al variar

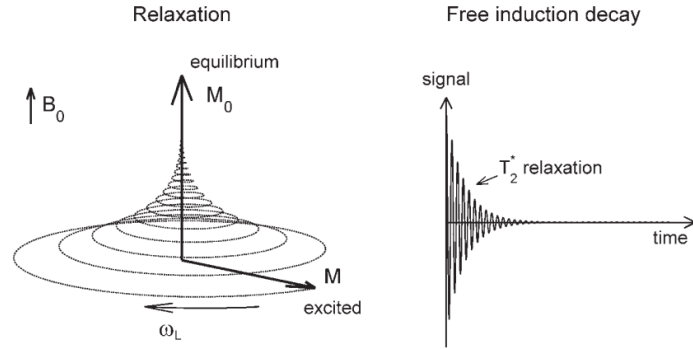


Figura 2.8: A la izquierda, el movimiento que describe el vector de magnetización nuclear \vec{M} cuando cesa el campo rotatorio y vuelve al estado de equilibrio. A la derecha, la señal que produce el vector de magnetización en la componente del plano transversal si se sitúa una bobina en uno de los ejes (x o y) [3].

y volver a la posición de equilibrio. Esta variación se puede medir en las dos componentes, cada una teniendo un tiempo de relajación característico:

- En la componente longitudinal, el vector aumenta conforme algunos de los núcleos cambian el estado de su espín de antiparalelo a paralelo, ya que el sistema sigue sometido al campo magnético \vec{B}_0 . El tiempo de relajación en la componente longitudinal se conoce como T_1 , que se considera el tiempo que tarda en llegar al 63 % del valor en equilibrio al tratarse de una función exponencial. El valor de T_1 depende del entorno en el que se encuentre el átomo, ya que varía en función de la molécula de la que forma parte el átomo y por lo tanto, del tipo de tejido. Por ejemplo, T_1 será menor en la materia blanca del cerebro por estar consistido por moléculas de lípidos, y mayor en fluidos por tener un alto porcentaje de agua. Esta diferencia de tiempo se puede traducir en una diferencia de intensidad a la hora de tomar la imagen, apareciendo la materia blanca con una intensidad clara y, por ejemplo, el fluido cefalorraquídeo en tonos oscuros.

Cuando pase el tiempo de relajación, la componente longitudinal del vector de magnetización nuclear volverá a tener el mismo valor que tenía previamente en equilibrio M_0 . Por lo que la relajación de este vector en esta componente se puede expresar mediante la función:

$$M_z(t) = M_0 + [M_z(0) - M_0] e^{-\frac{t}{T_1}}$$

Donde M_z es la magnetización en la componente longitudinal (eje z), con t el tiempo que ha pasado desde que cesó el pulso de excitación

y $M_z(0)$ el valor inicial, determinado por el valor de la magnetización antes de la excitación M_0 y el ángulo FA.

- En la componente del plano transversal, el vector disminuye conforme se desfasan los núcleos en su movimiento de precesión, ya que ya no están sometidos al campo oscilatorio $\vec{B}_1(t)$, llegando a ser nulo por regresar a la situación de equilibrio donde las fases de precesión son arbitrarias. Este proceso ocurre más rápido que para la otra componente, por lo que tiene que ser medido nada más cesar la excitación. El tiempo de relajación en la componente transversal se conoce como T_2 (figura ??), y en este caso, será menor en átomos que pertenezcan a moléculas grandes o unidas a membranas celulares, por ejemplo en hueso, y será más largo en fluidos. De la misma forma que para la otra componente, esto se traduce a una diferencia de intensidad en la imagen, pero esta vez los tiempos más largos se representan más claros.

En este caso la relajación de la componente transversal del vector de magnetización viene dada por la función:

$$M_{xy}(t) = M_{xy}(0)e^{\frac{-t}{T_2}}$$

Siendo M_{xy} la componente del vector de magnetización contenida en el plano $z = 0$ y $M_{xy}(0)$ su valor inicial.

Existe también un tiempo TR, que es el tiempo de repetición que pasa para volver a excitar el sistema y realizar otra medida de las señales. Este tiempo tiene que ser lo suficientemente largo para no interferir de manera significativa en la señal que depende de T_1 .

2.6.4. Formación de una imagen

Para la adquisición de la imagen queda resolver ciertas cuestiones, como la distinción en una señal de los distintos tejidos y su localización. Para ello se utilizan gradientes en el campo magnético \vec{B}_0 y se codifica información espacial a partir de la fase y frecuencia de la señal.

Para la selección de una sección para representar, se usa un gradiente que introduce una variación en la intensidad del campo \vec{B}_0 en la dirección longitudinal. Entonces al aplicar el campo rotatorio \vec{B}_1 a todas las secciones, sólo la sección que cumpla la condición de resonancia $\omega_{RF} = \omega_L$ absorberá la energía suministrada por el campo; esta sección se puede seleccionar gracias al gradiente, tal y como se muestra en la figura 2.9, ya que $\omega_L = \gamma B_0$. Esto hace que las demás secciones del tejido no se vean afectadas al no cumplir la condición de resonancia.

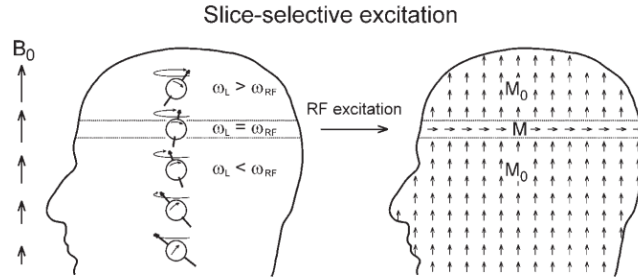


Figura 2.9: Representación del método utilizado para evaluar una sección. El campo \vec{B}_0 varía según el gradiente aplicado, haciendo que sólo la sección de interés cumpla la condición de resonancia, permitiendo así su estudio [3].

El tamaño de la sección se puede controlar mediante el ancho de banda de la ráfaga del campo oscilante y la amplitud del gradiente. La selección de la sección se hace mediante la frecuencia del campo RF.

Para identificar la contribución de cada tejido en la señal recibida, se aplica una codificación de fase y frecuencia mediante otros dos gradientes, que aumentan el campo \vec{B}_0 en la dirección de los ejes del plano transversal (figura 2.10). La codificación se realiza después de la excitación del campo magnético oscilante.

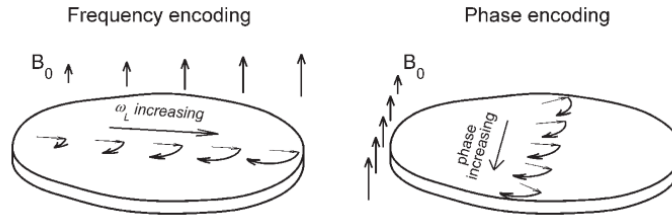


Figura 2.10: El campo \vec{B}_0 varía según los gradientes para realizar la codificación en frecuencia en uno de los ejes y la codificación en fase en el otro [3].

Al codificar en frecuencia, los tejidos encontrados en la zona con mayor intensidad de campo emitirán una señal con una frecuencia ligeramente superior a los tejidos que se encuentran en las zonas con menor intensidad, esto se consigue con un gradiente en la dirección de uno de los ejes. La señal que se detecta cuando se hace la medición, por lo tanto, contendrá componentes en distintas frecuencias que corresponderán a una zona distinta cada una. La amplitud de cada componente dará información sobre la intensidad de la señal que proviene de cada posición a lo largo del eje en el que se aplica el gradiente.

Esta información no es suficiente para formar la imagen, por lo que se realiza una codificación en fase a lo largo del otro eje, lo que permitirá obtener información en las dos dimensiones que componen la sección bajo estudio. Para conseguir esta codificación, se emite un pulso en forma de gradiente a lo largo del eje antes de la medición, lo que causa que la frecuencia sea alterada durante un breve periodo, lo suficiente para desfasar los núcleos a lo largo del eje entre ellos. La señal detectada esta vez contendrá componentes con distintas fases, proporcionando así información de los tejidos a lo largo del eje en el que se ha realizado este tipo de codificación. Para obtener la amplitud de cada componente, el proceso de excitación, codificación y medición se realiza varias veces, con intensidades cada vez mayores en el pulso aplicado.

Una vez se haya realizado la codificación en fase y en frecuencia, y obtenido las mediciones para todas las repeticiones en esa sección, los datos se adquieren como una serie de líneas contenidas en una matriz de dos dimensiones, llamada el espacio K. Aplicando una transformada de Fourier en dos dimensiones a esta matriz, se obtiene información sobre la fase, que no es de utilidad, y sobre la amplitud, con la que se obtiene la imagen reconstruida de esa sección (figura 2.11). Este proceso se realiza para distintas secciones con el fin de formar una imagen en tres dimensiones, sección a sección, del volumen bajo estudio.

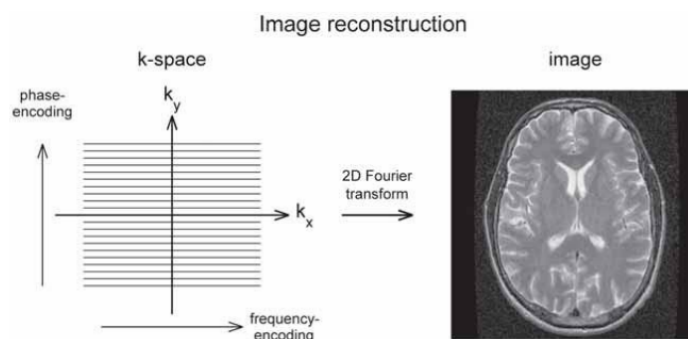


Figura 2.11: A la izquierda, la señal obtenida en el espacio K. A la derecha, la reconstrucción de la imagen de la sección bajo estudio [3].

Capítulo 3

Segmentación mediante SPM y preprocesado de imágenes

A la hora de realizar un análisis de las imágenes médicas para su correcta clasificación, es necesario efectuar su segmentación además de un preprocesado, antes de presentarlas al modelo que se encargará de realizar dicha clasificación.

La segmentación consiste en la separación de distintos tejidos de una imagen médica con el fin de extraer o aislar las regiones u órganos de interés para su posterior estudio. En este caso, para el diagnóstico automático de enfermedades neurodegenerativas, es más útil analizar imágenes del tejido mayoritariamente afectado (materia blanca o materia gris).

A la hora de realizar la segmentación se ha hecho uso de Statistical Parametric Mapping (SPM), que se refiere tanto a las técnicas estadísticas utilizadas para el análisis de imágenes cerebrales, como al conjunto de software y funciones que las implementan. Está implementado casi en su totalidad como funciones de Matlab, a excepción de algunos procesos, computacionalmente más costosos, compilados en C.

3.1. Fundamento teórico de la segmentación

La idea principal detrás de la segmentación es que SPM la realiza mediante un modelo generador estadístico. El modelo tiene en cuenta la intensidad de cada vóxel para clasificarlos en distintos tejidos. Para esto se definen distribuciones de intensidad de cada tejido, modelándolas utilizando mezclas de gaussianas y obteniendo una función objetivo. Esta función objetivo se combina con mapas de probabilidad deformables y correcciones en la falta de uniformidad de la intensidad para una mejor optimización. Permitiendo así, encontrar los parámetros óptimos para que el modelo se ajuste lo mejor posible a los datos para su correcta clasificación. A continuación se van a presentar estos conceptos en detalle para entender mejor la obtención del

modelo.

3.1.1. Intensidad del vóxel según el tipo de tejido

En las imágenes estructurales de tipo T1, la información viene dada solamente expresada en un canal (la intensidad) en cada uno de los vóxeles. Esta intensidad dependerá del tipo de tejido que se está analizando, ya que la señal será más fuerte o débil; siendo los tejidos de mayor a menor intensidad materia blanca, materia gris y líquido cefalorraquídeo.

En primera instancia la clasificación podría parecer trivial, bastaría con asociar cierto nivel de intensidad a cada tejido. Pero en la realidad existen diversos factores que podrían afectar al nivel de intensidad, que hacen que esta cambie de forma continua en vez de tomar valores más discretos. Aparte de variaciones biológicas y errores en la medición, el efecto de volumen parcial es un gran contribuyente a la hora de atribuir una intensidad concreta a un vóxel. Este efecto ocurre cuando se registra en un vóxel una región del espacio que contiene más de un tipo de tejido, la intensidad en ese caso vendrá dada por la proporción de mezcla. Un ejemplo podría ser el de la figura 3.1, donde se puede apreciar la variación de la intensidad en función de la proporción de cada tejido que se haya muestreado; en este caso, los dos tejidos en cuestión son materia blanca (mayor intensidad) y materia gris (menor intensidad).

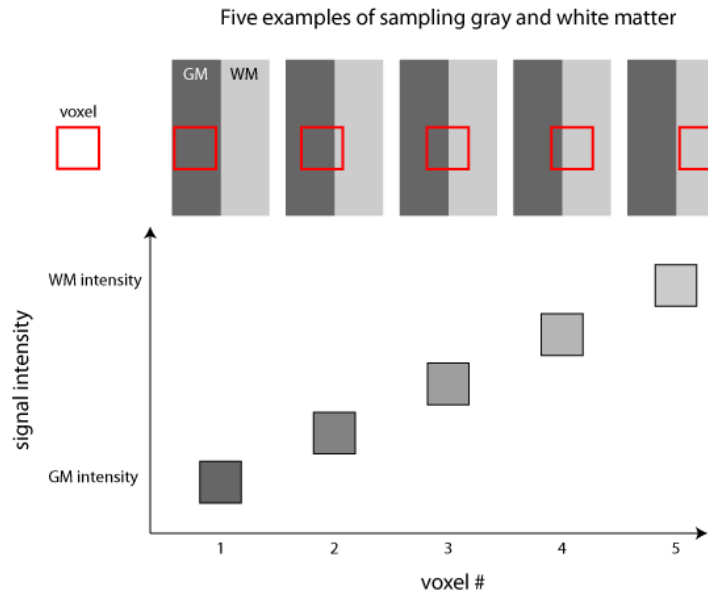


Figura 3.1: Variación de la intensidad del vóxel debido al volumen de cada tipo de tejido (materia gris o blanca) contenido en él [5].

Es debido a estos casos en los que la intensidad toma valores intermedios

que se hace interesante un enfoque probabilístico. A los vóxeles que claramente pertenecen a un tipo de tejido, se les asigna una alta probabilidad de pertenencia. Mientras que para los casos en los que el vóxel puede pertenecer a varios tejidos, se asignan valores de probabilidad intermedios proporcionales al volumen de cada clase de tejido contenido en ese vóxel. Por lo que es importante entender que el resultado obtenido tras realizar la segmentación en cada vóxel, es un conjunto de probabilidades de los distintos tejidos a los que puede pertenecer.

3.1.2. Modelado por mezcla de Gaussianas y obtención de la función objetivo

Para la obtención de las distintas probabilidades, es interesante expresar el conjunto de vóxeles que componen una imagen como una distribución en función de sus intensidades. Sabiendo que cada tipo de tejido emite señales de distinta intensidad, esta distribución se puede entender como una combinación de distintas distribuciones, una por cada tipo de tejido. Un ejemplo sería el de la figura 3.2 donde se pueden apreciar las distintas distribuciones que corresponden a cada clase de tejido.

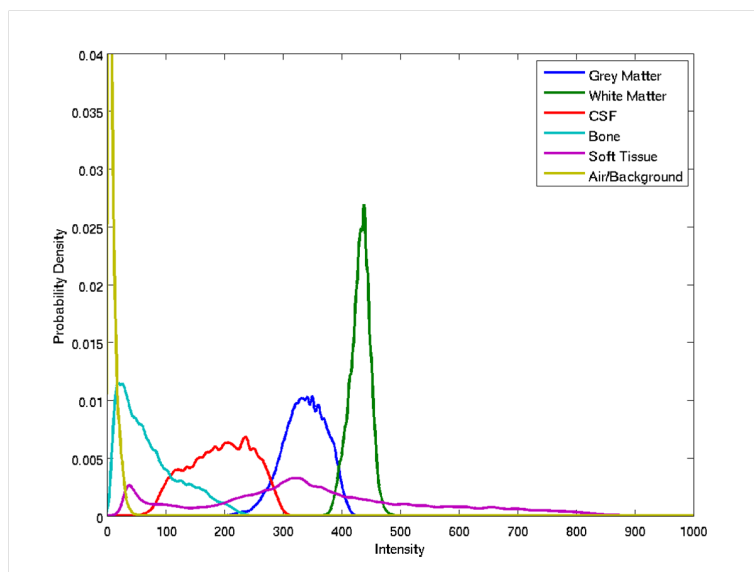


Figura 3.2: Gráfica mostrando la distribución de vóxeles según sus intensidades y la clase de tejido a la que pertenecen, en una función de densidad de probabilidad [6].

Cada una de estas distribuciones se puede modelar como una mezcla de K Gaussianas, donde la Gaussiana k -ésima está definida por su media μ_k y varianza σ_k^2 , además de su proporción de mezcla γ_k (tal que $\sum_{k=1}^K \gamma_k = 1$ y $\gamma_k \geq 0$). Para ajustar el modelo de mezcla de gaussianas es necesario maxi-

mizar la probabilidad de obtener los elementos I de datos y , por medio de los parámetros que definen las gaussianas. En una mezcla de gaussianas, la probabilidad para obtener un dato con intensidad y_i suponiendo que pertenece a la k -ésima gaussiana ($c_i=k$), que además está parametrizada por μ_k y σ_k^2 , es:

$$P(y_i|c_i = k, \mu_k, \sigma_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(y_i - \mu_k)^2}{2\sigma_k^2}\right)$$

La probabilidad a priori de que un vóxel pertenezca a la k -ésima gaussiana, independiente de su intensidad, dada la proporción de vóxeles que pertenecen a esa gaussiana es:

$$P(c_i = k|\gamma_k) = \gamma_k$$

Con estas dos expresiones, se puede calcular la probabilidad conjunta de la intensidad y_i y la pertenencia a la gaussiana k , como:

$$P(y_i, c_i = k|\mu_k, \sigma_k, \gamma_k) = P(y_i|c_i = k, \mu_k, \sigma_k)P(c_i = k|\gamma_k)$$

Integrando en todas las gaussianas:

$$P(y_i|\mu, \sigma, \gamma) = \sum_{k=1}^K P(y_i, c_i = k|\mu_k, \sigma_k, \gamma_k)$$

Ahora μ , σ y γ son vectores de parámetros que contienen la información de cada uno de los parámetros de cada gaussiana.

Asumiendo todos los elementos como independientes, la probabilidad del conjunto de datos viene dada por:

$$P(y|\mu, \sigma, \gamma) = \prod_{i=1}^I P(y_i|\mu, \sigma, \gamma) = \prod_{i=1}^I \left(\sum_{k=1}^K \frac{\gamma_k}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(y_i - \mu_k)^2}{2\sigma_k^2}\right) \right)$$

Que es la función objetivo a maximizar con respecto a los parámetros μ , σ y γ . Para ello se define una función de coste ε para minimizar, siendo el valor de la función objetivo máximo cuando el de la función de coste sea mínimo.

$$\varepsilon = -\log P(y|\mu, \sigma, \gamma) = -\sum_{i=1}^I \log \left(\sum_{k=1}^K \frac{\gamma_k}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(y_i - \mu_k)^2}{2\sigma_k^2}\right) \right)$$

La suposición de independencia de los elementos puede ser errónea, pero más adelante se revelará que las probabilidades a priori de la pertenencia de un vóxel a una gaussiana, tienen un alto grado de dependencia espacial.

3.1.3. Falta de uniformidad en la intensidad

Existe un fenómeno llamado *bias*, que corrompe las imágenes de resonancia magnética modulando la intensidad en distintas zonas de la imagen (variación espacial). Es producido por imperfecciones en el campo magnético, y aunque no supone un problema para el análisis e inspección de un profesional médico, su corrección es importante para el procesamiento automático de la imagen; en el caso de la segmentación, el hecho de variar la intensidad en ciertas zonas, afecta a la asignación de probabilidad para los distintos tipos de tejidos.

En la figura 3.3 se puede encontrar un ejemplo de una sección en el plano axial de una imagen T1 afectada por este fenómeno, el campo aplicado para su corrección y la misma imagen corregida por el campo de intensidades.

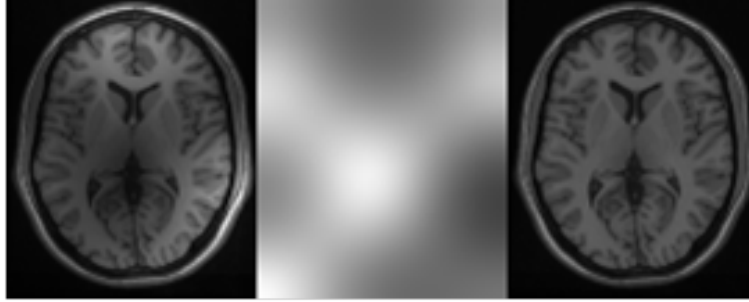


Figura 3.3: A la izquierda, la imagen original afectada por *bias*. En el centro, el campo de corrección. A la derecha, la imagen corregida [7].

La corrección del *bias* se incluye en el modelo mediante unos parámetros extra que se optimizan para contrarrestar este efecto. El campo que ajusta la variación del elemento i se denota por $\rho_i(\beta)$, donde β es el vector de parámetros desconocidos. La forma en la que se incorpora esta corrección en la función objetivo, es incluyéndola en la probabilidad de obtener la intensidad y_i dada la k -ésima gaussiana.

$$P(y_i | c_i = k, \mu_k, \sigma_k, \beta) = \frac{\rho_i(\beta)}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(\rho_i(\beta)y_i - \mu_k)^2}{2\sigma_k^2}\right)$$

Por lo que la función de coste queda definida como:

$$\varepsilon = -\sum_{i=1}^I \log \left(\rho_i(\beta) \sum_{k=1}^K \frac{\gamma_k}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(\rho_i(\beta)y_i - \mu_k)^2}{2\sigma_k^2}\right) \right)$$

3.1.4. Mapas de probabilidad deformables

Anteriormente se ha mencionado que las probabilidades a priori (*priors*) de que un vóxel pertenezca a una gaussiana venía dada por su proporción de mezcla γ . Pero esta información se puede completar para incluir dependencia espacial mediante mapas de probabilidad para distintas clases de tejidos (*tissue probability maps* o TPM para abreviar). Estos mapas de probabilidad están generados a partir de un gran número de sujetos, asignando vóxeles a las distintas clases y promediando. Permiten obtener información de la probabilidad de que un vóxel de la imagen registrada pertenezca a uno de los tipos de tejidos, independientemente de la intensidad de este. En la figura 3.4 se pueden apreciar los mapas de probabilidad normalmente utilizados para materia gris, materia blanca, líquido cefalorraquídeo y otros (el mapa de probabilidad de los otros tipos de tejidos se obtiene haciendo la diferencia de uno menos la suma de los otros tres mapas de probabilidad). En versiones más recientes de SPM, se incluyen mapas de probabilidad para el cráneo y tejidos blandos.

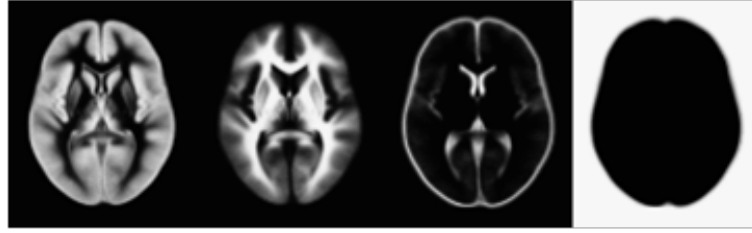


Figura 3.4: Mapas de probabilidad de distintos tejidos, provenientes del ICBM (*International Consortium for Brain Mapping*) [7].

Para incluir los mapas de probabilidad en el modelo, se permite que la probabilidad de que el vóxel i pertenezca a la gaussiana k -ésima dependa del punto del espacio y de la proporción de mezcla, según la siguiente expresión:

$$P(c_i = k|\gamma) = \frac{\gamma_k b_{ik}}{\sum_{j=1}^K \gamma_j b_{ij}}$$

La componente b_{ik} es la probabilidad, obtenida del mapa de probabilidades, de que el vóxel i pertenezca a la clase k . Esta expresión se puede mejorar utilizando mapas de probabilidad deformables, para adaptarse mejor a la forma y el tamaño de la imagen estructural que se pretende segmentar. Estas deformaciones se parametrizan mediante un vector de parámetros α , que al incluirlo en el modelo, modifica la expresión a:

$$P(c_i = k|\gamma, \alpha) = \frac{\gamma_k b_{ik}(\alpha)}{\sum_{j=1}^K \gamma_j b_{ij}(\alpha)}$$

Por lo que la función de coste queda definida como:

$$\varepsilon = - \sum_{i=1}^I \log \left(\frac{\rho_i(\beta)}{\sum_{k=1}^K \gamma_k b_{ik}(\alpha)} \sum_{k=1}^K \frac{\gamma_k b_{ik}(\alpha)}{\sqrt{2\pi\sigma_k^2}} \exp \left(- \frac{(\rho_i(\beta)y_i - \mu_k)^2}{2\sigma_k^2} \right) \right)$$

3.1.5. Optimización de la función objetivo

La optimización se realiza minimizando la función de coste ε con respecto a los vectores de parámetros μ , σ , γ , α y β . Para ello se usa un proceso iterativo, comenzando por valores estimados para los parámetros y actualizando sus valores con cada iteración para que la función de coste vaya decreciendo.

El proceso se repite hasta que haya convergencia y para actualizar el valor de los parámetros, se va alternando entre ellos para encontrar un óptimo local mientras que el resto permanece constante. De forma más detallada, se puede descomponer cada iteración de la optimización en tres pasos:

- Optimización de β mientras μ , σ , γ y α se mantienen constantes. Para ello se requiere calcular la primera y segunda derivada parcial de ε con respecto a β . La actualización del vector de parámetros vendrá dada por:

$$\beta^{(n+1)} = \beta^{(n)} - \left(\frac{\partial^2 \varepsilon}{\partial \beta^2} |_{\beta^{(n)}} + \lambda I \right)^{-1} \frac{\partial \varepsilon}{\partial \beta} |_{\beta^{(n)}}$$

Donde I es una matriz identidad y λ es un factor de escala, aumentándolo ralentiza la convergencia pero mejora la estabilidad del algoritmo.

- Optimización de α mientras μ , σ , γ y β se mantienen constantes. Este paso se efectúa de forma análoga al anterior y la actualización del parámetro α está definida por:

$$\alpha^{(n+1)} = \alpha^{(n)} - \left(\frac{\partial^2 \varepsilon}{\partial \alpha^2} |_{\alpha^{(n)}} + \lambda I \right)^{-1} \frac{\partial \varepsilon}{\partial \alpha} |_{\alpha^{(n)}}$$

- Optimización de μ , σ y γ , mientras α y β se mantienen constantes. Para la estimación de los parámetros se utiliza una estrategia de maximización de la expectación (EM), que consiste en alternar entre un paso E y un paso M. Para la iteración (n):

En el paso E, se estiman las probabilidades de pertenencia:

$$q_{ik}^{(n)} = P(c_i = k | y_i, \mu^{(n)}, \sigma^{(n)}, \gamma^{(n)}, \alpha, \beta) = \frac{P(y_i, c_i = k | \mu^{(n)}, \sigma^{(n)}, \gamma^{(n)}, \alpha, \beta)}{\sum_{j=1}^K P(y_i, c_i = j | \mu^{(n)}, \sigma^{(n)}, \gamma^{(n)}, \alpha, \beta)}$$

En el paso M, se actualizan los valores de μ , σ y γ .

$$\mu_k^{(n+1)} = \frac{\sum_{i=1}^I q_{ik}^{(n)} \rho_i(\beta) y_i}{\sum_{i=1}^I q_{ik}^{(n)}}$$

$$(\sigma_k^2)^{(n+1)} = \frac{\sum_{i=1}^I q_{ik}^{(n)} \left(\mu_k^{(n+1)} - \rho_i(\beta) y_i \right)^2}{\sum_{i=1}^I q_{ik}^{(n)}}$$

$$\gamma_k^{(n+1)} = \frac{\sum_{i=1}^I q_{ik}^{(n)}}{\sum_{i=1}^I \frac{b_{ik}(\alpha)}{\sum_{j=1}^K \gamma_k^{(n)} b_{ij}(\alpha)}}$$

Una vez se hayan optimizado correctamente los parámetros, el modelo de mezcla de gaussianas se ha ajustado correctamente a la distribución de intensidades, es capaz de corregir el *bias* y ha obtenido una mayor precisión gracias al uso de mapas de probabilidad deformables. Sabiendo las distribuciones que pertenecen a cada tejido y el grado de pertenencia de cada vóxel a una distribución, el modelo es capaz de realizar la segmentación correctamente generando una imagen estructural por tipo de tejido.

A continuación se va a presentar un ejemplo de segmentación, donde partiendo de una imagen estructural de tipo T1 (figura 3.5), se obtiene el resultado para materia blanca (figura 3.6) y materia gris (figura 3.7).

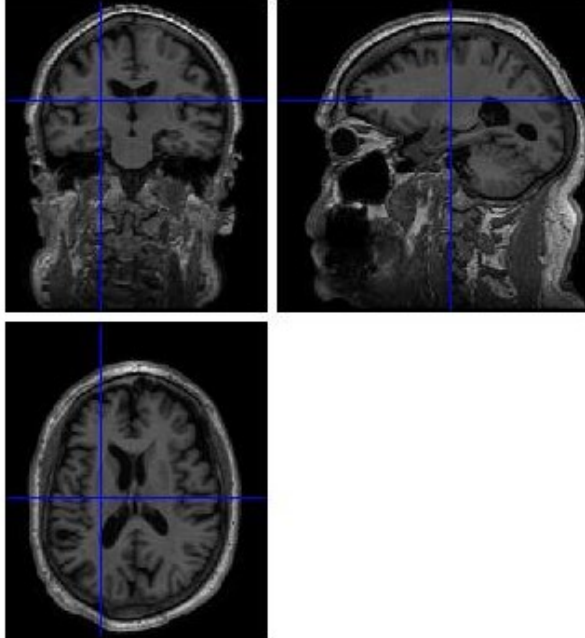


Figura 3.5: Imagen médica tipo T1 a segmentar.

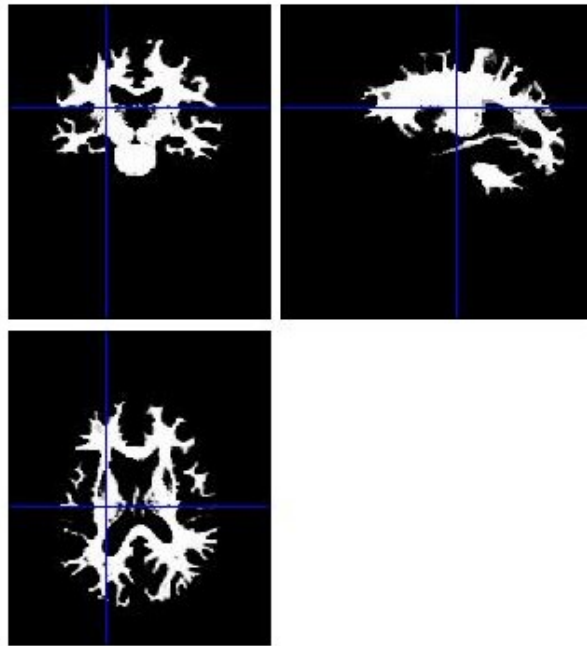


Figura 3.6: Resultado de la segmentación para materia blanca.

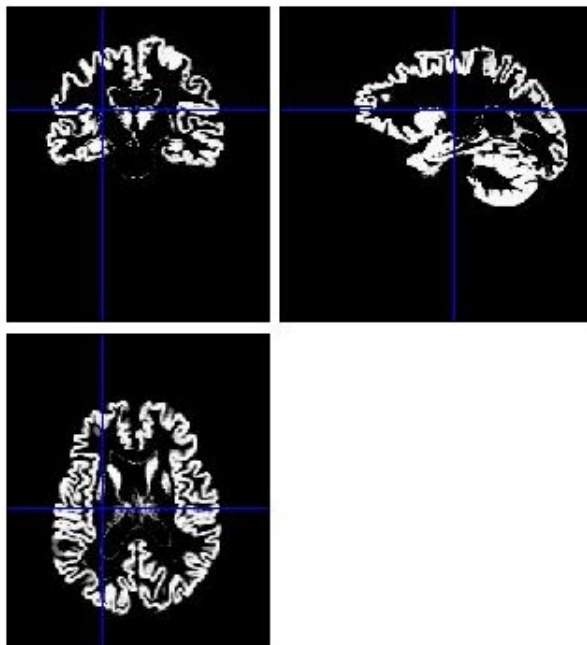


Figura 3.7: Resultado de la segmentación para materia gris.

3.2. Preprocesado de datos

Tras realizar la segmentación para aislar el tejido de interés de todas las imágenes disponibles para el análisis, los datos se cargan en el entorno de Python mediante la biblioteca de NiBabel, que permite hacer lecturas y escrituras de archivos con formatos comunes en el campo de las neuroimágenes.

El conjunto de datos se almacena en una matriz de 4 dimensiones, la primera dimensión indica el número de la imagen y las tres dimensiones restantes sirven para almacenar los datos de cada una de las imágenes (ya que se trata de imágenes estructurales en tres dimensiones). Aquí es cuando se puede apreciar el gran volumen de datos con el que se va a hacer el análisis, si se disponen de 362 imágenes de dimensiones 157, 189 y 156, con cada dato siendo de tipo *float64*. La variable que almacena los datos ocupará por lo tanto:

$$Memoria = 362 \times 157 \times 189 \times 156 \times 64 \frac{bits}{elemento} = 1,07244394 \times 10^{11} bits$$

Expresando el resultado en gigabytes:

$$Memoria = \frac{1,07244394 \times 10^{11} bits}{8 \frac{bits}{B} \times 10^9 \frac{B}{GB}} = 13,4055 GB$$

La variable puede llegar a ocupar una gran cantidad de memoria, es por ello que es importante realizar un preprocesado de los datos para intentar reducir el peso de esta variable y así aminorar el costo computacional del clasificador.

- En primer lugar cambiar el tipo de dato a *float32*, con lo que se consigue una reducción a la mitad.
- Reducir las dimensiones de las matrices a (156, 188, 156), puesto que en las últimas posiciones de la primera y segunda dimensión todos los elementos son cero, lo que no añade ningún tipo de información.
- Efectuar la media de grupos de 8 vóxeles a lo largo de todas las dimensiones de cada matriz tridimensional, con lo que se consigue una reducción a la mitad de cada dimensión; es decir, el número total de elementos se reduce entre ocho. Es cierto que esto supone una pérdida de información, por lo que se podría considerar como un compromiso entre precisión y coste computacional del proceso de clasificación. Las matrices de datos por lo tanto tendrán un tamaño de (78,94,78).

Con estas modificaciones, se puede calcular el peso de nuevo para la variable que almacena los datos:

$$Memoria = 362 \times 78 \times 94 \times 78 \times 32 \frac{bits}{elemento} = 6,62484326 \times 10^9 bits$$

De nuevo, expresando el resultado en gigabytes:

$$Memoria = \frac{6,62484326 \times 10^9 bits}{8 \frac{bits}{B} \times 10^9 \frac{B}{GB}} = 0,8281 GB$$

Lo que supone una reducción importante en el volumen de los datos, consiguiendo así un acortamiento de los tiempos de entrenamiento del modelo clasificador.

Capítulo 4

Redes neuronales artificiales

En este capítulo se busca explicar las principales ideas que hay detrás de las redes neuronales, para ellos se comenzará por definir algunos conceptos clave del Machine Learning y Deep Learning, se introducirá al perceptrón y de la estructura que puede formar además de las ideas principales detrás de su entrenamiento, se explicará el funcionamiento de los tipos principales de capas que componen a las redes convolucionales, se hará la distinción entre parámetros e hiperparámetros, y por último, se presentará la estructura utilizada en esta aplicación haciendo uso de los conceptos nombrados anteriormente.

4.1. Machine Learning y Deep Learning

Antes de proceder a explicar lo que son las redes neuronales, algunos de sus tipos y cómo funcionan, es necesario hacer una breve introducción al concepto de *Machine Learning*, así como, donde se encuentra en el campo de la computación y la inteligencia artificial.

Machine Learning (conocido en castellano como aprendizaje automático) es un enorme campo de desarrollo e investigación, que a su vez forma parte del campo de la inteligencia artificial, y se puede definir como el conjunto de técnicas que permiten a las computadoras aprender a resolver tareas sin ser explícitamente programadas.

Algunos conceptos básicos del Machine Learning son los siguientes:

- *Labels* o «etiquetas», son el objetivo que se quiere obtener o predecir con el modelo. Por ejemplo, al clasificar unas imágenes, las etiquetas son las distintas categorías a las que pertenecen esas imágenes.
- *Features* o «características», son los datos o las variables de entrada que se introducen al modelo. Siguiendo el mismo ejemplo, las características serían los píxeles de las imágenes a clasificar o datos extraídos de ellas.

- El modelo, el que permitirá establecer la relación entre las características y sus etiquetas.

El proceso de aprendizaje se realiza en dos fases distinguidas:

- Fase de entrenamiento o aprendizaje, en la que se crea el modelo y se entrena con ejemplos, para que tras un número de iteraciones, el modelo aprenda las relaciones entre características y etiquetas.
- Fase de inferencia o predicción, en la que el modelo ya entrenado hará predicciones sobre ejemplos no etiquetados, así mediante ciertas características de entrada, el modelo será capaz de predecir la salida o la etiqueta correspondiente.

En el campo de Machine Learning existen tres grandes categorías en las que se clasifican los algoritmos según su aprendizaje:

- Supervisado: cuando los datos utilizados en el aprendizaje llevan incluidos su etiqueta correspondiente (se conoce la solución para cada dato de entrenamiento). Existen varios algoritmos pertenecientes a esta categoría, algunos ejemplos son los support vector machines y las redes neuronales.
- No supervisado: en el que los datos usados para el aprendizaje no están catalogados mediante etiquetas (no se conoce la solución para ningún dato de entrenamiento), en este caso el algoritmo deberá clasificar los datos de entrada de forma autónoma. Entre los más conocidos están los algoritmos clasificación por clustering.
- Por refuerzo: donde el modelo mediante un agente aprende por prueba y error, usando un mecanismo de penalización y recompensa, que le permiten optimizar sus acciones con un conjunto de datos o en un medio, por lo general, desconocido.

En cuanto al Deep Learning, hace referencia a los algoritmos de redes neuronales artificiales por la estructura que presentan estas redes. Los modelos de este campo del Machine Learning están compuestos por varias capas de procesamiento, con diferentes niveles de abstracción donde se entrenan diferentes parámetros de forma supervisada, para que mediante una serie de transformaciones lineales y no lineales se puedan obtener salidas próximas a las etiquetas mediante unos datos de entrada. Así, estas transformaciones se van optimizando para que la salida se aproxime a la salida esperada. La figura 4.1 sirve para apreciar como se organizan estos conceptos; siendo el Deep Learning una parte del Machine Learning, que a su vez es una parte del campo de la inteligencia artificial.

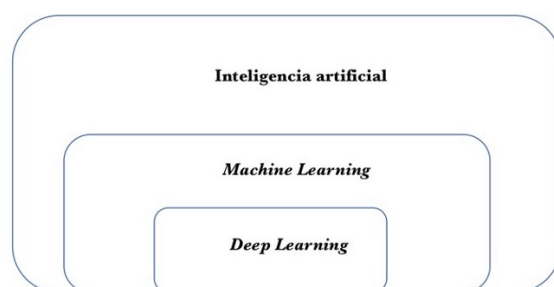


Figura 4.1: Esquema de la jerarquización del Machine Learning y Deep Learning [8].

Una representación gráfica simple de estos modelos puede ser la de la figura 4.2. Donde se pueden distinguir tres capas: la de entrada que es la primera y la encargada de recibir los datos, la de salida que se encarga de devolver las predicciones, y en medio de estas se encuentran las capas ocultas apiladas una encima de otra con diferentes estructuras y cantidad de neuronas; de ahí viene el término *deep* (profundo), que hace referencia a la profundidad de las redes que se manejan en la actualidad.

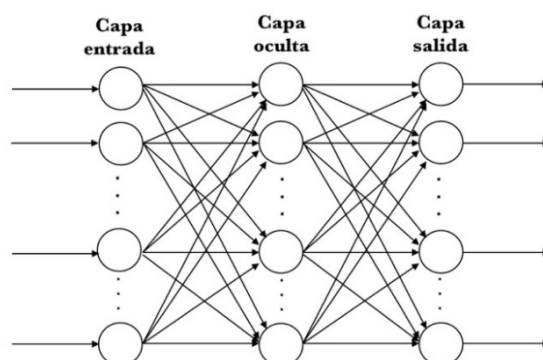


Figura 4.2: Esquema simple de modelo de Deep Learning [8].

Por último cabe destacar dos conceptos esenciales, sobretodo para el entrenamiento de un modelo. Estos dos conceptos se denominan *loss* y *overfitting* (traducidos al castellano como error y sobreajuste).

La función de *loss* es la encargada de indicar cómo de cerca se encuentra una predicción del modelo al valor deseado; cuanto más cerca se encuentre una predicción, menor será el valor del error; y viceversa (entendiéndose como predicción perfecta aquella que obtenga un valor para la función de *loss* de cero). Durante la fase de entrenamiento, al modelo se le aplican distintos algoritmos de aprendizaje automático, de manera que se vayan

modificando los parámetros de este con el objetivo de minimizar la función de *loss*.

El *overfitting* se entenderá como el fenómeno que se produce cuando un modelo se ajusta tanto a los datos de entrenamiento, que no será capaz de realizar predicciones correctamente con datos nuevos.

4.2. El perceptrón

Para entender cómo funciona una red neuronal, es necesario estudiar su elemento más básico: el perceptrón (figura 4.3).

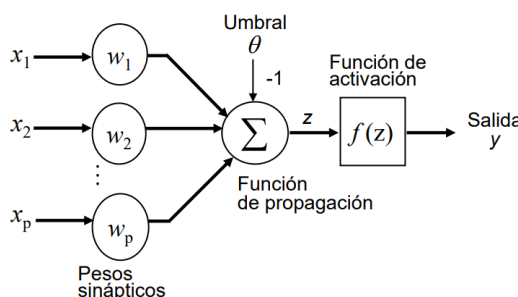


Figura 4.3: Esquema de un perceptrón.

Un perceptrón, o neurona artificial, está compuesto por varias entradas x_i a las que se les aplican unos pesos w_i , llamados pesos sinápticos, que las ponderan. Estas entradas son agregadas por una función de propagación teniendo en cuenta el umbral o sesgo θ , obteniendo el resultado z que pasa por una función de activación $f(z)$, finalmente obteniendo la salida y . Su funcionamiento se puede expresar mediante el siguiente modelo matemático:

$$y = f\left(\sum_{i=1}^p w_i x_i - \theta\right)$$

En cuanto a la función de activación, sirve para extender la salida de un nodo hasta el siguiente produciendo la activación de la neurona. Existen diferentes tipos tal y como se muestra en la figura 4.4, cada una tiene propiedades y utilidades diferentes.

Los parámetros que se buscan entrenar en la fase de aprendizaje en un perceptrón son los pesos w_i y el umbral θ . Con este modelo es imposible resolver problemas de cierta complejidad (por ejemplo problemas que no son linealmente separables), es por ello que las neuronas artificiales se combinan formando una capa, que a su vez recibe información de una capa anterior y envían información a la capa siguiente, dando forma a una estructura de red conocida como perceptrón multicapa.

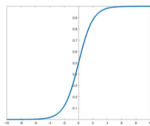
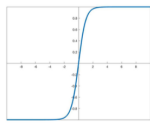
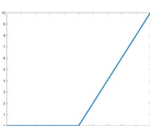
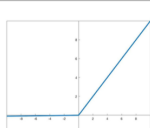
Activation function	Equation	Graph
Sigmoid	$S(x) = \frac{1}{1 + e^{-x}}$	
Tanh	$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	
ReLU	$RELU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	
Leaky ReLU	$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$	

Figura 4.4: Funciones de activación para una red neuronal.

4.3. Perceptrón Multicapa

El perceptrón multicapa o *Multi-Layer Perceptron* (MLP) es una estructura con alto grado de conectividad, compuesta por una capa de entrada, una o más capas ocultas y una capa de salida. Este tipo de redes se usan a menudo para resolver problemas de clasificación donde las categorías de clasificación son exclusivas. Los perceptrones que constituyen esta estructura cumplen con una serie de características:

- No hay enlaces entre los elementos de una misma capa, por lo que una neurona no está comunicada con las demás neuronas de su misma capa.
- Las conexiones se hacen con las neuronas de la capa anterior y la capa siguiente. Estas conexiones pueden ser de tipo global o local, dependiendo si una neurona está conectada con todas las neuronas de la siguiente capa, o sólo a un grupo determinado de ellas.
- Por lo general, el número de neuronas de la capa de entrada viene determinado por la cantidad de características (*features*) que constituyen a los datos de entrada. De igual manera, el número en la capa de salida viene determinado por la cantidad de etiquetas (*labels*).

La cantidad de parámetros a entrenar viene determinada por el núme-

ro de neuronas y la cantidad de enlaces que haya entre ellas; siendo estos parámetros los umbrales de cada neurona en el primer caso, y los pesos de los enlaces en el segundo. Estos parámetros se entrenan mediante un proceso de aprendizaje consistente de varias etapas. A continuación se va a proceder a explicar este proceso, su relación con la función de *loss* y cómo operan ciertas técnicas de optimización; con el objetivo de que el modelo sea capaz de hacer predicciones lo más exactas posibles.

4.4. Entrenamiento de una red neuronal: forward-propagation, backpropagation y función de loss

A la hora de entrenar los pesos y umbrales de cada elemento en una red neuronal se hace uso de un proceso iterativo de «ida y vuelta», que contiene diversas fases. Este proceso cumple con el objetivo de optimizar los parámetros del modelo, utilizando unos datos de entrenamiento o de ejemplo y distintas técnicas de optimización, para conseguir reducir el valor de la función de *loss* en un determinado número de iteraciones. Las fases por las que transcurre este proceso en cada iteración son:

- La primera fase, conocida como *forwardpropagation* (propagación hacia delante), en la que se presentan los datos de entrenamiento a la red neuronal mediante la capa de entrada; a partir de ahí, cada capa recibe información de la capa anterior, procesa esa información mediante las transformaciones que realizan sus neuronas y manda esa información procesada a la siguiente capa, para que esta realice su serie de transformaciones. Así se va procesando la información a través de la red, hasta que finalmente llega a la capa de salida de la cual se obtendrá el resultado, que se trata de una predicción que hace la red de la etiqueta asociada al dato de entrenamiento introducido.
- El siguiente paso es estimar el error cometido de la predicción mediante una función de *loss*, comparando esta predicción con la etiqueta correcta (al tratarse de un método de aprendizaje supervisado, se disponen de las etiquetas correctas para los datos de entrenamiento). El objetivo es conseguir minimizar este error; que a medida que se vaya entrenando el modelo, optimizando sus parámetros con cada iteración, tenderá a disminuir. La elección de la función de *loss* dependerá de qué tipo de problema se intente resolver.
- Una vez obtenido el error en la salida, mediante lo que se conoce como *backpropagation* (propagación hacia atrás), se manda esta información a través de la red en dirección opuesta (desde la capa de salida hasta la de entrada), de forma que cada neurona recibe solamente una fracción de esta información en función de su contribución relativa a la

salida; en otras palabras, las neuronas van recibiendo capa por capa una señal de error, dependiendo de su contribución relativa al error total, calculado previamente en la salida.

- Por último queda ajustar los pesos de los diferentes enlaces entre neuronas. Esto se hace por medio de un optimizador, que modifica el valor del parámetro haciendo uso del error o de la *loss* y buscando su mínimo global, para así ir optimizando el modelo.

Este proceso se repite a lo largo de varias iteraciones o *epochs*, hasta que se considere que la red puede hacer predicciones con la suficiente exactitud. Cabe destacar que para la primera iteración, los parámetros de la red se pueden asignar de forma aleatoria o escoger.

Previamente se ha mencionado el uso de un optimizador para modificar el valor de los parámetros durante el entrenamiento, ya que estos parámetros generalmente no se pueden calcular de forma analítica, un optimizador es un algoritmo de optimización iterativo. A continuación se va a explicar el optimizador de *gradient descent* (descenso de gradiente), ya que es un método muy utilizado en Machine Learning y Deep Learning; y además, es la base de muchos otros optimizadores, que con pequeñas modificaciones se adaptan para obtener una optimización más rápida en función del problema.

El método consiste en utilizar la primera derivada o gradiente de la función de *loss* para actualizar los parámetros. De forma simplificada, ya que los cálculos dependerán de la función de *loss* escogida, para aplicar este optimizador se van encadenando las derivadas del error en cada una de las capas a partir de la capa precedente, teniendo en cuenta también la función de activación de las neuronas de la capa en cuestión. Una vez se ha calculado el valor del gradiente de la función de *loss* en cada una de las neuronas, se actualizan los parámetros utilizando el gradiente pero en sentido inverso, ya que el gradiente indica el sentido de crecimiento de la función de *loss*. Usando el sentido inverso se puede avanzar en el sentido donde el error decrece, y así llegar hasta el mínimo de la función del error.

Un esquema simplificado podría ser el de la figura 4.5, simplificado en el sentido de que sólo se va a tener en cuenta la influencia del peso en la función de *loss*. Como se puede observar, el optimizador de gradiente descendiente modifica el peso en la dirección contraria al gradiente para encontrar el valor mínimo del error. La variación del parámetro para cada iteración vendrá determinado por el valor de este gradiente (ya que refleja la pendiente de la función en un punto determinado), y además, por un hiperparámetro llamado *learning rate*. El producto de estas dos magnitudes dará como resultado la variación del peso, obteniendo así su valor para la próxima iteración. Este proceso se repite hasta que el valor del peso sea tal que anule la derivada de la función de *loss*; en otras palabras, cuando se haya alcanzado el mínimo de esta función.

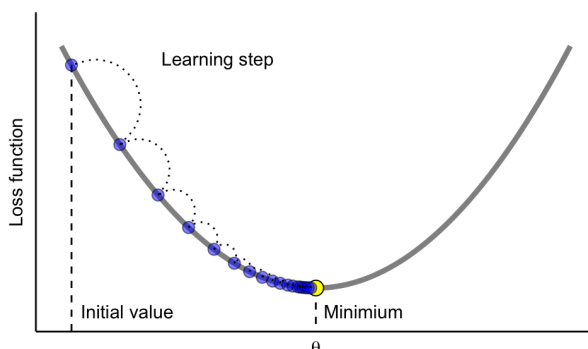


Figura 4.5: Esquema simple de optimización por gradiente descendiente [8].

4.5. Redes neuronales convolucionales

Otro tipo de redes neuronales especialmente útiles en el reconocimiento de imagen para tareas de visión artificial, son las llamadas redes neuronales convolucionales (*Convolutional Neural Networks* o CNNs para abreviar). Son parecidas a las redes neuronales ya vistas, en el sentido de que estas redes hacen uso de ciertos parámetros (pesos y sesgos) para aprender; pero con la peculiaridad de que admiten imágenes como datos de entrada, y tienen una arquitectura que les permite reconocer ciertas características y elementos en estas imágenes, con diferentes niveles de abstracción en las distintas capas.

Para entender el funcionamiento de las CNNs es fundamental comprender la operación de convolución y cómo operan las capas que la aplican. Por razones de sencillez, esta explicación se hará para el caso en el que se trate con imágenes de dos dimensiones, a continuación, se extrapolará a imágenes en tres dimensiones que son las de interés para este proyecto.

De manera general, la operación de convolución se define como:

$$h(t) = \int_{-\infty}^{\infty} f(a)g(t-a)da$$

Normalmente se expresa mediante un asterisco:

$$h(t) = f(t) * g(t)$$

En la terminología de redes convolucionales, a la primera función se le conoce como *input* o entrada, a la segunda como *kernel* o filtro, y al resultado de la operación se le suele llamar *feature map* o mapa de características.

Al trabajar con computadoras y sistemas electrónicos, las medidas van a tener valores discretos (en el caso de las imágenes, el número del pixel está representado por un entero); por lo que se define la operación de convolución

para funciones discretas como:

$$h(t) = f(t) * g(t) = \sum_{a=-\infty}^{\infty} f(a)g(t-a)$$

Aunque las redes convolucionales normalmente se van a aplicar a imágenes; es decir, los datos de entrada van a consistir en arrays de varias dimensiones llamados tensores. Para el caso en el que la entrada sea una imagen I de dos dimensiones, es interesante aplicar un *kernel* K de dos dimensiones para realizar la convolución:

$$S(i, j) = I(i, j) * K(i, j) = \sum_m \sum_n I(m, n)K(i-m, j-n)$$

Al ser conmutativa, la operación se puede expresar como:

$$S(i, j) = K(i, j) * I(i, j) = \sum_m \sum_n I(i-m, j-n)K(m, n)$$

Siendo la segunda expresión la más implementada en Machine Learning. Para entender cómo se aplica esta operación en imágenes, se va a hacer uso de un ejemplo gráfico (figura 4.6). En este ejemplo, por simplicidad se ha utilizado una imagen pequeña de 3x4 píxeles, con un solo canal; y un *kernel* cuadrado de cuatro elementos. Cada píxel viene notado por p_{ij} , siendo i el índice que indica la fila y j la columna en la que se encuentra; y de forma análoga, cada elemento del filtro por w_{mn} . El resultado de la operación es otra imagen de distinto tamaño a la original; este cambio es debido a la operación de convolución, y además depende del tamaño del filtro y de un parámetro llamado *stride* explicado en detalle en la sección de **Hiperparámetros** 4.6, pero de forma simple se puede entender como la cantidad de píxeles que pasa el *kernel* cada vez que realiza una convolución.

Supongamos que el *kernel* actúa como una ventana realizando operaciones a lo largo y ancho de la imagen de entrada. Esta operación se efectúa haciendo la suma de los productos de cada elemento del filtro con el píxel que cubre ese elemento. El resultado es un valor que corresponderá al píxel de la imagen de salida. En el ejemplo se ha remarcado con colores esta operación para el primer grupo de cuatro píxeles, que dan como resultado el primer elemento de la matriz resultante. A continuación, se desplaza la ventana un píxel hacia la derecha y se realiza el mismo proceso de nuevo. Una vez recorrida la imagen de izquierda a derecha y de arriba hacia abajo, se obtiene la imagen o matriz resultante, cuya dimensión dependerá de cuantas veces se pueda mover la ventana en la imagen de entrada. En el ejemplo cada elemento de la imagen de salida contiene las operaciones realizadas para su obtención, con el fin de facilitar su entendimiento.

En este caso, los parámetros a entrenar son un sesgo y cada uno de los elementos que compone la matriz del *kernel*; pero usar solamente un filtro

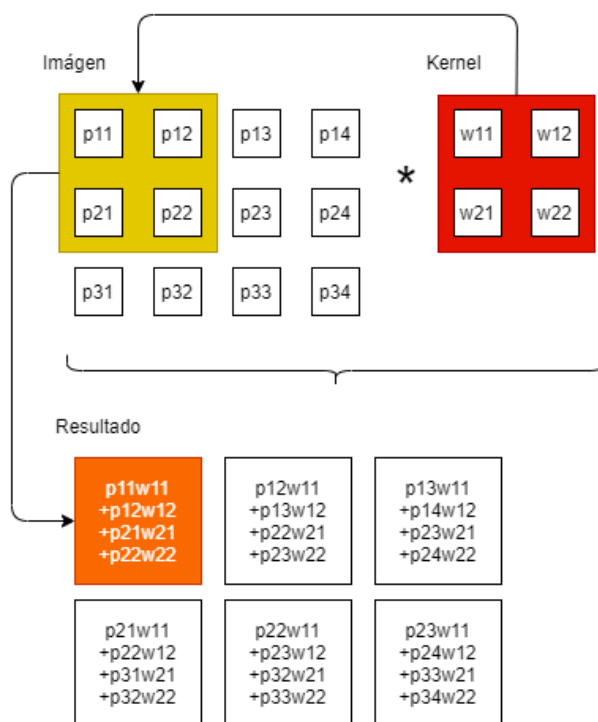


Figura 4.6: Diagrama de operación de convolución en una imagen.

limita la cantidad de características que se pueden obtener de una imagen, es por eso que en una capa convolucional se usan varios filtros, lo que permite una mejor extracción de diferentes características. Cada convolución con cada uno de estos filtros dará como resultado su propia salida como se puede observar en la figura 4.7. Por lo tanto, la cantidad de parámetros que utilizará la red será proporcional a la cantidad de filtros que se quieran aplicar, a su tamaño y a la cantidad de capas definidas para componer la estructura.

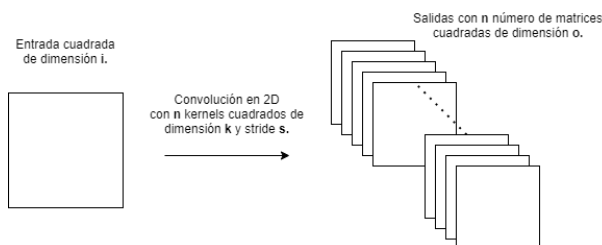


Figura 4.7: Esquema representando el cambio de dimensión y el número de salidas en una capa convolucional.

La dimensión de las matrices de salida en 4.7 vendrá dada por esta

expresión:

$$o = \frac{i - k}{s} + 1$$

Donde i es la dimensión de la imagen cuadrada, k la dimensión del *kernel* cuadrado, s el stride y o la dimensión de la imagen resultante.

Las capas de convolución se suelen utilizar en combinación con capas de *pooling*. Utilizada normalmente inmediatamente después de la convolución para condensar y simplificar la información que se ha obtenido. El *pooling* consiste en reducir un grupo determinado de valores en uno (generalmente un cuadrado de píxeles), si se hace a lo largo de toda la matriz se reducirá su tamaño para así disminuir la complejidad y el número de características. Las dimensiones de la matriz resultante dependerá del tamaño de la ventana usada para agrupar los elementos y, como se verá más adelante, del *stride* definido. Un ejemplo podría ser el de la figura 4.8.

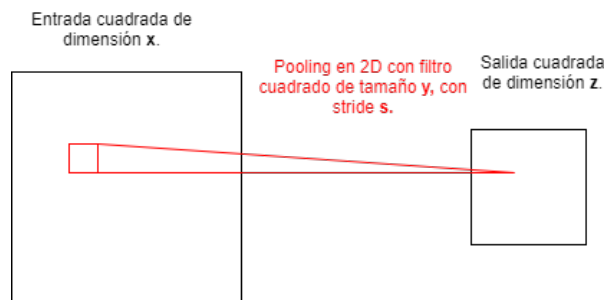


Figura 4.8: Esquema representando el cambio de tamaño tras una capa de *pooling*.

La dimensión del resultado en 4.8 vendrá dada por:

$$z = \frac{x - y}{s} + 1$$

De la misma forma, x es la dimensión de la imagen cuadrada, y la dimensión de la ventana que realiza la operación, s el stride y z la dimensión de la imagen resultante. Hay distintos tipos de *pooling* dependiendo de la operación que utilizan para efectuar la reducción; por ejemplo, en el *max-pooling* el valor resultante será el máximo de los valores de los elementos que se están comparando en esa zona, mientras que el *average-pooling* realiza el promedio de los valores de estos elementos.

Al aplicar este tipo de capas después de una capa convolucional, el *pooling* se efectúa en cada una de las matrices que resultan de realizar la convolución con cada uno de los filtros. Por lo tanto, la salida de la capa de *pooling* tendrá tantas matrices como la salida de la capa de convolución que está inmediatamente antes. Tal y como se muestra en la figura 4.9.

Entendiendo el funcionamiento de las capas de convolución y de *pooling*, se pueden construir redes convolucionales. Estas capas se pueden combinar

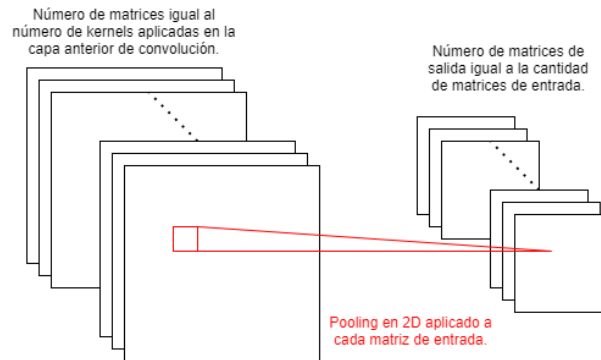


Figura 4.9: Esquema indicando el número de salidas en función del número de entradas, tras una operación de *pooling*.

además las capas de perceptrones ya vistas, por medio de una capa de *reshape* que aplana los datos a un tensor de una dimensión.

En cuanto a las imágenes en tres dimensiones, las operaciones de convolución y *pooling* son muy parecidas a las de las imágenes en dos dimensiones. En este caso, las ventanas para realizar las operaciones y los *kernels* de convolución serán cubos o prismas cuadrangulares en vez de cuadrados o rectángulos. Una representación de una convolución en tres dimensiones se encuentra en la figura 4.10, donde a una imagen tridimensional se le aplica una convolución con un filtro cúbico, dando como resultado otra con dimensiones menores. Cada vóxel del resultado viene de convolucionar veintisiete vóxeles de la entrada (por aplicar un filtro cúbico de $3 \times 3 \times 3$).

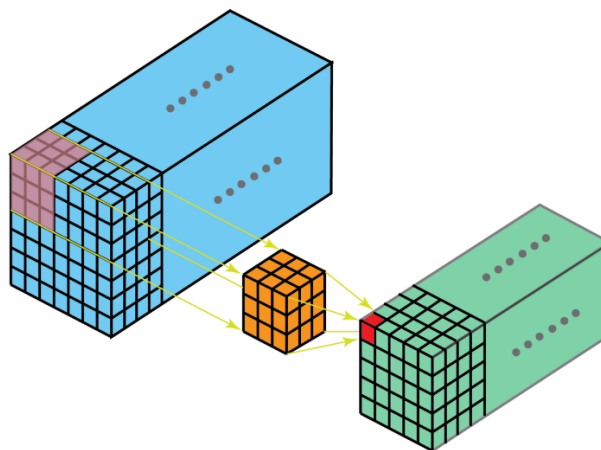


Figura 4.10: Representación de convolución en tres dimensiones con un *kernel* cúbico.

4.6. Hiperparámetros

Una vez llegado hasta este punto en el entendimiento del Deep Learning, es necesario hacer una aclaración y distinguir entre parámetros e hiperparámetros. Los parámetros se pueden considerar como las variables de configuración internas del modelo cuyos valores se calculan con los datos de entrada; por ejemplo, como se ha mencionado anteriormente, los pesos y los sesgos de las distintas neuronas y sus enlaces, o las matrices de pesos y el sesgo para definir los *kernels* en las redes convolucionales. En cambio, los hiperparámetros se consideran variables de configuración externas al modelo y se especifican por el propio programador.

Existen hiperparámetros a nivel de estructura de la red, como el número de capas, la cantidad de neuronas, el tipo de función de activación para una capa, etc. Y hay hiperparámetros a nivel del entrenamiento de la red, como el número de iteraciones, el *learning rate*, etc. A continuación se va a proceder a explicar algunos de los hiperparámetros de mayor interés para este proyecto, tanto generales como específicos para CNNs.

- Número de *epochs*: es el número de iteraciones para la fase de entrenamiento, la cantidad de veces que se hace pasar los datos de entrenamiento a través del modelo. El programador deberá encontrar un compromiso entre aumentar las *epochs* para encontrar una mayor precisión e intentar evitar el *overfitting*.
- *Batch size*: es el tamaño del lote de datos que se usan para entrenar. Los datos se pueden agrupar en lotes que pueden ser procesados por la red. Para esta aplicación, debido al tamaño de los datos de entrada, cada imagen se ha procesado de forma individual, por lo que el tamaño del lote es igual a uno.
- *Learning rate* (ratio de aprendizaje): es el factor que multiplica al valor del gradiente a la hora de aplicar el optimizador para actualizar los pesos durante la fase de entrenamiento. Determina como de grande será el paso para escoger los nuevos parámetros para la siguiente iteración. Su valor óptimo es muy dependiente del problema; un *learning rate* alto acelera el proceso de aprendizaje; pero puede llevar a no alcanzar el punto mínimo puesto que al ser tan grande el paso, el valor del parámetro podría oscilar entre dos puntos cercanos a este y nunca llegar a converger (en casos muy extremos podría causar divergencia). En cambio, un valor bajo garantiza llegar a la solución óptima con el problema de que el proceso podría ser demasiado lento. En la figura 4.11 se pueden apreciar las consecuencias de escoger un *learning rate* demasiado alto o bajo para una misma función de *loss*.

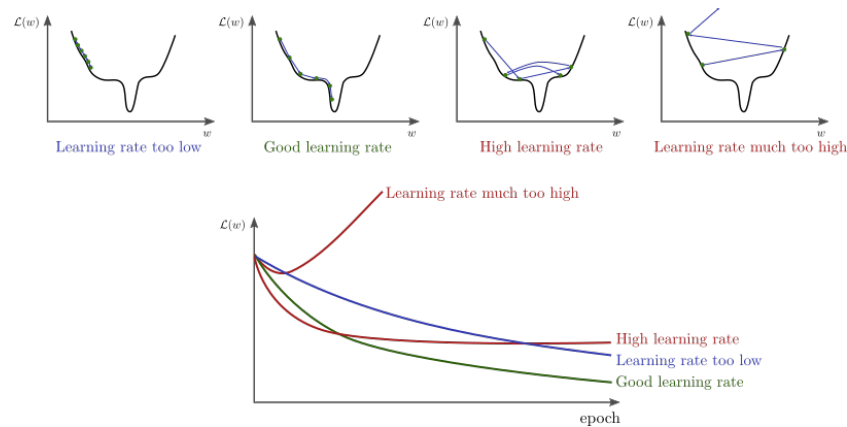


Figura 4.11: Gráficas que representan distintos *learning rates* para el mismo problema [9].

Existe un hiperparámetro llamado *learning rate decay* que permite disminuir el valor del ratio de aprendizaje conforme van pasando las iteraciones. Esto permite tener valores altos al comienzo del entrenamiento para acelerar el proceso de aprendizaje; pero a su vez, tener valores pequeños, conforme los parámetros se acerquen a la solución, para garantizar una mayor precisión.

- *Momentum*: a veces al buscar la solución óptima, el optimizador se puede quedar atascado en un mínimo local o en un punto de silla. Para solventar este problema y permitir llegar al mínimo global, se utiliza el *momentum*, que se puede considerar como un factor que va entre cero y uno, y se calcula tomando la media ponderada del gradiente de pasos anteriores para poder superar el obstáculo. Una función de *loss* en la que este hiperparámetro sería muy útil es la de la figura 4.12, ya que tiene ciertos mínimos locales a los que el optimizador puede converger.

Existen ciertos hiperparámetros que son exclusivos para las capas de tipo convolucional. A continuación se van a desarrollar los principales y más interesantes para este proyecto.

- Número de filtros y sus tamaños: tanto para capas convolucionales como para capas de *pooling*, estos hiperparámetros, que actúan sobre todo a nivel de estructura, van a determinar la dimensión de los datos de salida. También determinan la cantidad de parámetros a entrenar, debido a que los filtros se componen por matrices de pesos; es decir, cuanto mayor sea el número de matrices de pesos y cuanto mayor sea el tamaño de estas matrices, el coste de entrenamiento de la red será mayor. El programador deberá encontrar un compromiso entre

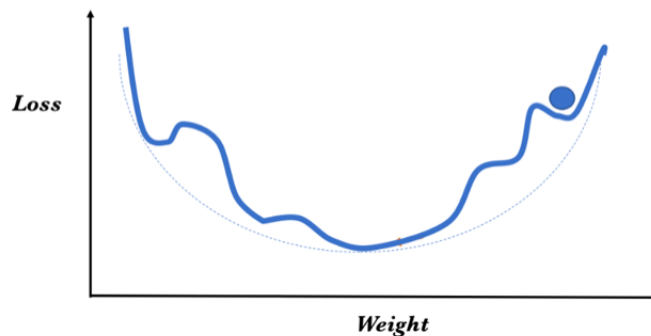


Figura 4.12: Gráfica de una función de *loss* con mínimos locales [8].

una mayor precisión con una mayor cantidad de parámetros y el coste computacional que esto supondría.

- **Stride:** ya nombrado en otros apartados anteriores, el stride es el hiperparámetro que determina el paso de la ventana al realizar una operación de convolución o de *pooling*; por ejemplo si la entrada es una imagen y se usa un stride de uno, la ventana dará pasos de solamente un pixel. Este hiperparámetro es muy importante ya que permite controlar la cantidad de grupos de elementos que se vana procesar, y además permite al programador controlar la reducción de las dimensiones de las matrices al realizar la operación. El stride se puede definir para cada una de las dimensiones de los datos de entrada; lo que quiere decir que si se desea reducir una de las dimensiones más que la otra u otras, es posible hacerlo con este hiperparámetro.
- ***padding*:** como se ha visto anteriormente, debido a la ventana que va realizando las operaciones, el tamaño de las matrices de salida tiende a reducir. El *padding* permite aumentar la dimensión de los datos de entrada justo antes de realizar la operación, para que la dimensión de los datos de salida sea la misma que la de los datos de entrada antes de aplicar el *padding*. Esto se consigue agregando ceros al contorno de las matrices de entrada, para que al realizar la operación, se mantenga el tamaño de las matrices sin introducir datos que podrían afectar al resultado final. Para entender este proceso se va a hacer uso de un ejemplo de una matriz en dos dimensiones (figura 4.13).

La imagen 4.13 era originalmente de 5x5 píxeles antes de aplicarle *padding*; si a esta imagen se le aplica una operación de convolución con *kernels* de dimensión 3x3 y un *stride* de uno en ambas dimensiones, el resultado sería una imagen de 3x3 debido al paso de la ventana para agrupar los elementos. Ahora bien, si a esta imagen se le emplea *padding* añadiendole ceros alrededor, se obtendría la imagen 4.13. Rea-

0	0	0	0	0	0	0
0	1	2	3	4	5	0
0	6	7	8	9	10	0
0	11	12	13	14	15	0
0	16	17	18	19	20	0
0	21	22	23	24	25	0
0	0	0	0	0	0	0

Figura 4.13: Representación de una matriz de números a la que se le ha aplicado *padding* [8].

lizando la misma operación con el mismo tamaño de *kernel* y *stride*, el resultado será esta vez una imagen de 5x5; la misma dimensión que la imagen original.

Por lo que ahora la dimensión de la salida de 4.7, suponiendo el uso de *padding*, será:

$$o = \frac{i + 2p - k}{s} + 1$$

Y la de *pooling* en 4.8 será:

$$z = \frac{x - y + 2p}{s} + 1$$

Donde *p* se refiere al *padding* en ambas ecuaciones.

4.7. Autoencoders

Este apartado se centrará en un tipo de estructura de red neuronal llamada *autoencoder*. Su peculiaridad reside en que su entrenamiento se efectúa utilizando los mismos datos de entrada como etiquetas, por lo que esta estructura intentará replicar a su salida la entrada que reciba. Funcionan comprimiendo la entrada en una representación de espacio latente, normalmente llamada *code*; y después, reconstruyendo la salida a partir de esta representación para que coincida con la entrada. Su estructura se puede descomponer en dos partes:

- El *encoder*, que se encarga de comprimir los datos. Puede ser descrito por la función $h=f(x)$, donde x es la entrada y h la representación comprimida de esta.

- El *decoder*, que reconstruye los datos introducidos al *encoder* a partir de su compresión. La función que lo describe es $r=g(h)$, donde r es la reconstrucción.

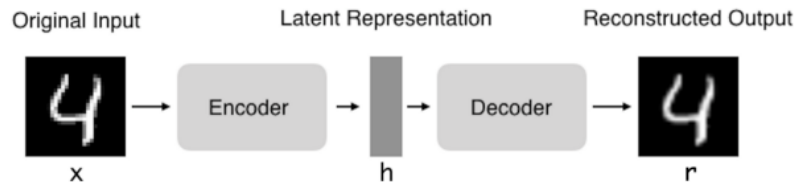


Figura 4.14: Arquitectura de un *autoencoder*[10].

Juntando el *encoder* y el *decoder* se obtiene un *autoencoder*, tal y como se muestra en 4.14; cuya función puede ser descrita por $r=g(f(x))$ y cuyo objetivo es aproximar la salida r a la entrada x .

Las principales aplicaciones de esta estructura son la de eliminación de ruido en datos y la reducción de dimensionalidad (la compresión de la información que se ha mencionado anteriormente). Existe también un tipo interesante de *autoencoders* llamado *variational autoencoders* (VAEs), que permiten generar nueva información a partir de los datos que se han usado para entrenarlos; por ejemplo, generar imágenes de caras humanas nuevas a partir de un conjunto de imágenes de distintos rostros reales.

En este estudio se va a hacer uso solamente de su capacidad compresiva, puesto que es especialmente útil para la extracción de características importantes en las imágenes médicas tridimensionales. Se simplificará un gran volumen de datos en un sólo vector de características para cada una de las imágenes; que a su vez servirá como información de entrada a un clasificador estadístico, que se encargará de clasificar las imágenes e intentará predecir los distintos grupos a los que pueden pertenecer los pacientes.

Para la construcción de un *autoencoder* se pueden usar los dos tipos de redes ya descritas anteriormente, individualmente o de forma combinada, siempre y cuando se tengan en cuenta las dimensiones que toma la información, y esta se remodele correctamente a lo largo de la red.

En el caso de utilizar capas de neuronas densamente conectadas, basta con ir reduciendo el número de neuronas en las capas del *encoder* hasta llegar a la capa del *code* o cuello de botella, que será la que menos neuronas contenga de toda la red. En el *decoder* en cambio, hay que ir aumentando el número de neuronas por capa, hasta que la última capa contenga la misma cantidad de neuronas que la primera capa del *encoder*; necesario si se quiere conseguir la misma dimensión a la salida que a la entrada. Una representación de un *autoencoder* con esta estructura se puede encontrar en la figura 4.15.

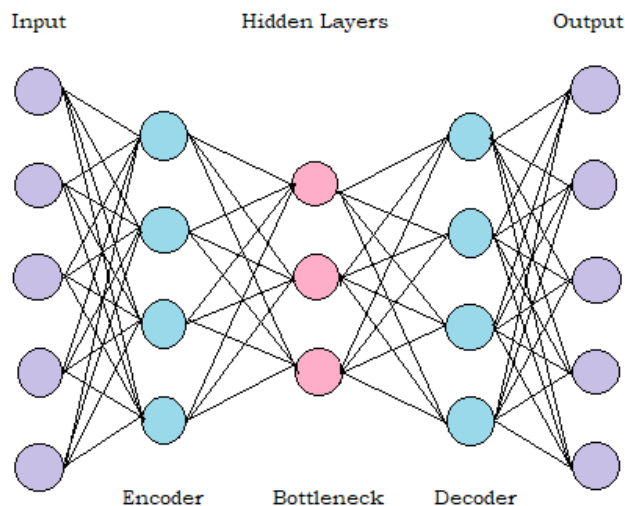


Figura 4.15: Ejemplo de un *autoencoder* constituido por varias capas de neuronas densamente conectadas.

A la hora de construir un *autoencoder* con CNNs, como se ha explicado anteriormente, tanto las capas de convolución como las de *pooling* son útiles para reducir la dimensionalidad en el *encoder*. Pero para aumentar de nuevo el tamaño de los datos ya comprimidos, es necesario utilizar capas de convolución transpuesta y de *upsampling* en el *decoder*. Así se podrá recuperar la dimensión original de las matrices, además de permitir calcular el error entre la predicción y la entrada original.

En una capa de convolución transpuesta la idea principal es aumentar la resolución de la imagen introducida por medio de matrices de pesos (*kernels*) y *padding*. Es importante saber que la convolución transpuesta no es una deconvolución, ya que no revierte la operación de convolución; pero para el *decoder* es muy útil, ya que utiliza parámetros entrenables y aumenta la dimensión de los datos introducidos, permitiendo así hacer una reconstrucción de la imagen original a partir de los datos comprimidos.

Para entender como operan este tipo de capas, se va a descomponer el proceso en una serie de pasos, tal y como se muestra en la figura 4.16. Se supone una matriz a la que se le aplica una convolución transpuesta con *kernels* cuadrados de dimensión \mathbf{k} , con *stride* \mathbf{s} y *padding* \mathbf{p} . El primer paso es calcular los parámetros \mathbf{z} y \mathbf{p}' a partir de los parámetros que se han utilizado para definir la operación, utilizando las ecuaciones que se muestran en la figura. El siguiente paso es insertar entre los elementos de la entrada una cantidad \mathbf{z} de filas y columnas de ceros; si \mathbf{z} fuese igual a cero, no se insertaría ceros, por lo que la matriz de entrada permanecería inalterada. Después se añade un padding de \mathbf{p}' ceros alrededor de la matriz. Por último,

se realiza una convolución a la imagen resultante del paso anterior con un *stride* siempre unitario. El resultado a la salida es una matriz de dimensión mayor que la original.

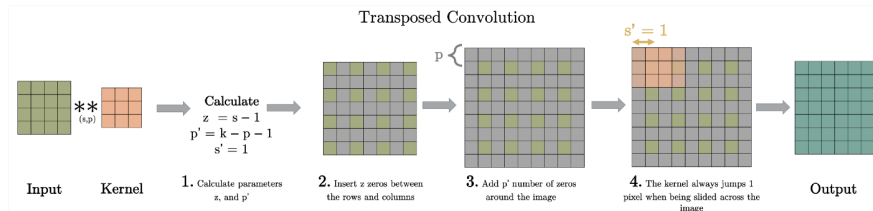


Figura 4.16: Representación por pasos de una operación de convolución transpuesta.

El *upsampling* sirve para aumentar el tamaño de la información mediante la repetición de los elementos de la matriz de entrada en grupos de elementos en la matriz de salida, el tamaño de estos grupos depende de la ventana escogida. Tal y como se muestra en la figura 4.17, por medio de la operación de *upsampling*, se ha aumentado cada dimensión por un factor de dos al utilizar una ventana cuadrada de 2x2.

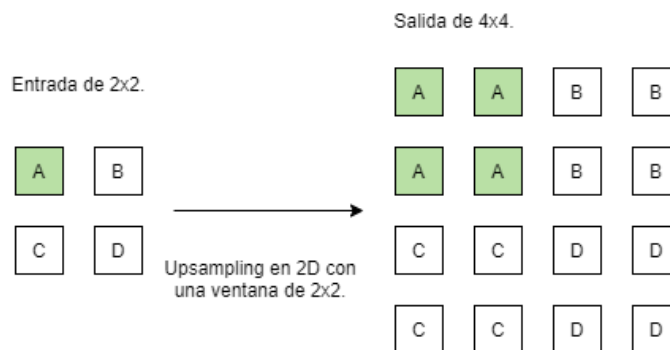


Figura 4.17: Ejemplo sencillo de un *upsampling*.

Con estas operaciones, ya es posible la construcción de un *autoencoder* convolucional (CAE). Para su entrenamiento se define una función de *loss* (por lo general el error cuadrático medio suele funcionar bien con esta estructura), y escoger un optimizador. Pero a diferencia de una red neuronal, las etiquetas con las que se entrena el modelo son los propios datos de entrada; es por eso que se suelen considerar como una técnica de aprendizaje auto-supervisado.

Capítulo 5

Clasificadores estadísticos

Después de haber comprimido la gran cantidad de información de las matrices de datos en vectores de características, queda la tarea de utilizar estos vectores para la clasificación en los tres grupos distintos de pacientes. Para ello, se utilizan una serie de algoritmos pertenecientes al campo del Machine Learning llamados clasificadores estadísticos.

Los clasificadores estadísticos son algoritmos de aprendizaje supervisado, que se encargan de agrupar o discriminar distintos elementos, definidos por un vector de características, en diferentes clases. Utilizan datos ya etiquetados como ejemplos de entrenamiento para componer una función de clasificación, que establece la relación entre los datos y sus etiquetas. De esta forma, el clasificador es capaz de hacer predicciones sobre datos con etiqueta desconocida y clasificarlos en una de las posibles clases.

El objetivo principal de este capítulo es definir a los clasificadores estadísticos (haciendo especial énfasis en la clasificación binaria), explicar el funcionamiento de los clasificadores utilizados en este proyecto y los métodos utilizados para su evaluación.

5.1. Definición

Considerando que los datos que se disponen para la clasificación están formados por un conjunto de vectores de características $x_i \in \mathbb{R}^m$, $i=1, \dots, n$, con m como dimensión del espacio de características \mathcal{H} . Se pueden dividir estos datos en un subconjunto de entrenamiento \mathcal{X} y otro de prueba o test \mathcal{Y} , tal que $\mathcal{X} \cup \mathcal{Y} = \mathcal{H}$.

Se define el vector de características x_i como un vector de entrenamiento si $x_i \in \mathcal{X}$, siendo $\mathcal{X} \subset \mathbb{R}^m$ el subconjunto de entrenamiento. De forma análoga, se define el vector de características \bar{x}_j como un vector de test si $\bar{x}_j \in \mathcal{Y}$, siendo $\mathcal{Y} \subset \mathbb{R}^m$ el subconjunto de test. Una condición necesaria para la partición de los dos subconjuntos es que estos sean independientes; es decir, que la intersección de los dos subconjuntos sea igual al conjunto

vacío ($\mathcal{X} \cap \mathcal{Y} = \emptyset$).

Por lo tanto, un clasificador se puede expresar mediante una función $f = f(x_i, \omega)$, formada a partir de los datos de entrenamiento $x_i \in \mathcal{X}$ de n dimensiones, con sus correspondientes etiquetas y_i , y que depende de un vector de parámetros ω . Se encarga de establecer un valor $z_j \in \{\pm 1\}$, que corresponde a las dos clases en una clasificación binaria, mediante un vector de test $\bar{x}_j \in \mathcal{Y}$. Expresado matematicamente como:

$$\begin{aligned} f : \mathbb{R}^n &\longrightarrow \{\pm 1\} \\ \bar{x}_j \in \mathcal{Y} &\longmapsto f(x_j, \omega) = z_j \end{aligned}$$

Tal y como se explicó en la sección 4.1, el proceso de aprendizaje del modelo (en este caso el clasificador) se realiza en dos fases: la fase de entrenamiento, donde el clasificador utiliza los vectores de entrenamiento para aprender, mediante los parámetros ω , la relación entre estos vectores y sus etiquetas asociadas; y la fase de evaluación, en la que el clasificador realiza predicciones sobre los vectores de test, asociándolos a una de las posibles clases \mathcal{C}_i , con $i \in 1, 2$ para una clasificación binaria.

El tipo de clasificador óptimo dependerá de la distribución de los elementos de cada clase en el espacio de características. Por ejemplo, un clasificador lineal no proporcionará buenos resultados para una distribución que no sea linealmente separable, por lo que habrá que utilizar otro tipo de clasificador no lineal, que a su vez requerirá un mayor número de parámetros, aumentando la complejidad del clasificador; para un proceso de clasificación en el que se dispongan pocos datos de ejemplo, aumentar la complejidad puede ser un efecto contraproducente. En la figura 5.1 se pueden apreciar dos distribuciones ejemplo para un espacio de características bidimensional; en la primera distribución, los datos se pueden clasificar mediante un clasificador lineal (representado por la recta); en cambio, para la segunda distribución será necesario utilizar un clasificador con una función polinomial.

A continuación se van a describir en detalle los clasificadores utilizados para este proyecto.

5.2. Árboles de decisión

Los árboles de decisión (Breiman et al., 1984) son métodos de aprendizaje supervisado que no se basan en el entrenamiento de parámetros, si no en la formulación de una serie de reglas que permiten la categorización de ciertas condiciones que se dan repetidamente en los datos de entrenamiento; es decir, encuentra patrones en las características de los vectores de datos.

Del inglés *Decision Trees*, reciben este nombre por el tipo de estructura ramificada con la que operan, formada por:

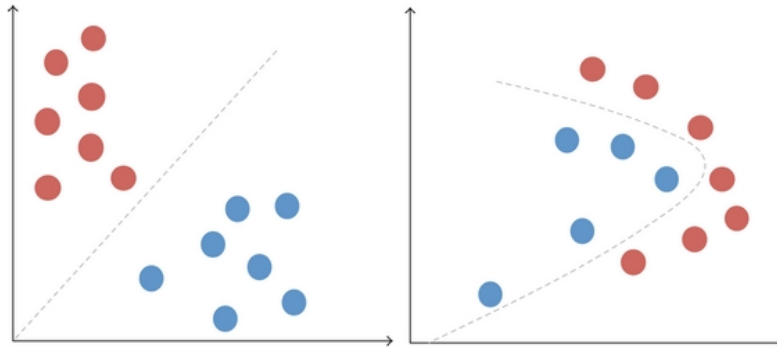


Figura 5.1: Dos distribuciones binarias en un espacio de características bidimensional. A la izquierda se han separado las dos clases mediante una recta. A la derecha se han separado mediante un polinomio [11].

- **Nodo raíz:** es el primer nodo del que se parte haciendo la primera evaluación dado un vector de características.
- **Nodos de decisión:** en los que se evalúan si el vector de características cumple o no ciertas condiciones. De ellos pueden partir varias ramas que llevan a otros nodos de decisión o a los nodos hojas.
- **Nodos hojas:** son los últimos nodos de la estructura y determinan la clase del vector de características introducido, en función del nodo hoja al que se haya llegado.

Los árboles de decisión se pueden utilizar tanto para clasificaciones binarias, como para clasificaciones de varias clases (multiclase). Su construcción se efectúa mediante la partición en los nodos de decisión en dos subconjuntos, mediante expresiones booleanas que clasifican los datos según si la expresión es verdadera o falsa. Algunas de sus ventajas son:

- Son fáciles de entender y de interpretar, ya que se permite la visualización de la estructura.
- Su coste computacional asciende de manera logarítmica en función de la cantidad de datos utilizados para su entrenamiento.

Entre sus desventajas principales están:

- Son propensos al sobreajuste debido a que se crean árboles con demasiada complejidad, por lo que no generalizan bien los datos. Existen mecanismos para contrarrestar este efecto, por ejemplo establecer un mínimo de muestras requeridas para formar un nodo hoja.
- Existen algunos casos en los que los árboles de decisión tienen dificultades, por ejemplo la función XOR.

5.3. Maquinas de vectores de soporte

Ideados a finales de los años setenta por Vladimir Vapnik (Vapnik, 1982) [22] en AT&T y recogidos en su libro *The nature of Statistical Learning Theory*[23], las máquinas de vectores de soporte (SVM de *Support Vector Machines*) son una serie de algoritmos pertenecientes al aprendizaje supervisado con interesantes aplicaciones en el reconocimiento de patrones y objetos. Para esta aplicación, se aprovecharán las grandes capacidades que tienen estos algoritmos en espacios de alta dimensionalidad para realizar clasificaciones binarias, por lo que al tener tres clases distintas de pacientes (Control, MCI y AD) se harán clasificaciones entre elementos de dos grupos (p.ej. Control vs AD). A continuación se van a presentar las ideas principales que envuelven a los SVM lineales.

5.3.1. SVM para clases linealmente separables

Las SVM se basan en la construcción del hiperplano óptimo de decisión capaz de separar los vectores x_i , $i=1,2,...,l$, del conjunto de test

$$(y_1, x_1), \dots, (y_l, x_l)$$

en función de la clase a la que pertenezcan \mathcal{C}_1 o \mathcal{C}_2 , según sus etiquetas $y \in \{-1, 1\}$. Para ello, se define el plano g mediante su ecuación general:

$$g(x) = w^T(x) + b = 0$$

donde w es el vector de peso ortogonal al plano y b el sesgo. El proceso de optimización consiste en encontrar los parámetros w_i , $i=1,...,N$, óptimos, pertenecientes al vector w , que determina al hiperplano óptimo.

Utilizando los vectores x_i , $i=1,2,...,N$, pertenecientes al conjunto de entrenamiento, se pretende diseñar el hiperplano que los clasifique debidamente y que tenga la misma distancia de los puntos más cercanos de las dos clases \mathcal{C}_1 y \mathcal{C}_2 . A esta distancia se le conoce como el margen y se exigen estas condiciones para maximizar la posibilidad de clasificar correctamente un vector de características diferente a los ya presentados en el conjunto de entrenamiento; es decir, para que el clasificador tenga una mejor capacidad de generalización. Una representación de estos conceptos se puede encontrar en la figura 5.2, donde el hiperplano corresponde a una recta que separa los elementos de dos clases en un espacio de características bidimensional.

Como la distancia de un punto x al hiperplano es:

$$dist(x, g) = \frac{|g(x)|}{||w||}$$

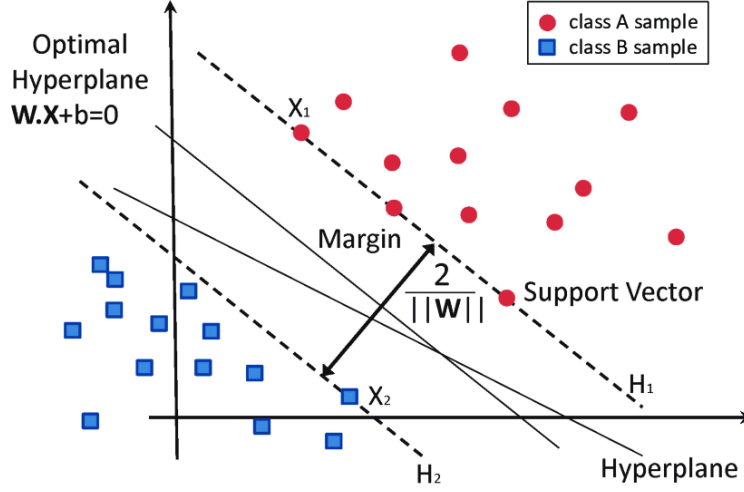


Figura 5.2: Máquina de vectores de soporte separando elementos de dos clases linealmente separables [12].

Siendo $\|w\|$ la norma o módulo del vector, que vale:

$$\|w\| = \sqrt{\sum_i^N w_i^2}$$

Escogiendo unos valores para w y b tales que para los puntos más cercanos de \mathcal{C}_1 y \mathcal{C}_2 , $g(x)$ valga 1 y -1 respectivamente, se consigue simplificar el problema de optimización a maximizar el margen:

$$\frac{1}{\|w\|} + \frac{1}{\|w\|} = \frac{2}{\|w\|}$$

con las condiciones:

$$w^T x + b \geq 1, \forall x \in \mathcal{C}_1,$$

$$w^T x + b \leq -1, \forall x \in \mathcal{C}_2$$

Si consideramos la etiqueta y_i igual a 1 para un vector de características x_i perteneciente a la clase \mathcal{C}_1 e igual a -1 para un vector de características x_i perteneciente a la clase \mathcal{C}_2 ; el hiperplano óptimo dependiente de w y b , se obtiene minimizando la función de coste:

$$\Phi(w) = \frac{1}{2} \|w\|^2$$

sujeto a la condición de la inecuación:

$$y_i(w^T x_i + b) \geq 1 \text{ para } i = 1, 2, \dots, N$$

La solución a este problema de optimización no lineal con restricciones (problema de programación convexa), conforme a Karush-Kuhn-Tucker, viene dada en los puntos de silla de la función Lagrangiana:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}w^T w - \sum_{i=1}^N \alpha_i [y_i(w^T x_i + b) - 1]$$

donde α es el vector que recoge los distintos α_i , llamados multiplicadores de Lagrange.

Y se deben cumplir las siguientes condiciones conforme a Karush-Kuhn-Tucker:

$$\frac{\partial \mathcal{L}(w, b, \alpha)}{\partial b} = 0$$

$$\frac{\partial \mathcal{L}(w, b, \alpha)}{\partial w} = 0$$

$$\alpha_i \geq 0 \text{ para } i = 1, 2, \dots, N$$

$$\alpha_i [y_i(w^T x_i + b) - 1] = 0 \text{ para } i = 1, 2, \dots, N$$

Con las dos primeras condiciones y la función lagrangiana, se obtiene que:

$$\sum_{i=1}^N \alpha_i y_i = 0$$

$$w = \sum_{i=1}^N \alpha_i y_i x_i$$

El vector w óptimo será combinación lineal de los $N_{sv} \leq N$ vectores de características para los multiplicadores α_i no nulos:

$$w = \sum_{i=1}^{N_{sv}} \alpha_i y_i x_i$$

A estos vectores se les conoce como vectores de soporte, y son los que en la inecuación $y_i(w^T x_i + b) \geq 1$ cumplen la igualdad; por lo que los vectores de soporte se encuentran en uno de los dos hiperplanos:

$$w^T x + b = \pm 1$$

recordando, que estos vectores de características son los más cercanos al clasificador de todo el conjunto de entrenamiento.

Al hiperplano óptimo que se encarga de hacer la clasificación se le conoce como maquina de vectores de soporte y es único. Para hallarlo hay que obtener los parámetros óptimos, que no siempre es fácil. Es por ello que el problema se puede considerar en su representación dual, sustituyendo $w = \sum_{i=1}^N \alpha_i y_i x_i$ en la función lagrangiana $\mathcal{L}(w, b, \alpha)$ y operando, se obtiene:

$$W(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j}^N \alpha_i \alpha_j y_i y_j x_i^T x_j$$

con las condiciones:

$$\sum_{i=1}^N \alpha_i y_i = 0$$

$$\alpha \geq 0$$

El problema se puede expresar por lo tanto como maximizar $W(\alpha)$, de esta forma se calculan los multiplicadores de Lagrange α_i óptimos. El vector w de parámetros que define el hiperplano óptimo se puede calcular explícitamente mediante la expresión:

$$w = \sum_{i=1}^N \alpha_i y_i x_i$$

y finalmente, queda el umbral b que se obtiene en la práctica a partir de las condiciones:

$$\alpha_i [y_i (w^T x_i + b) - 1] = 0 \text{ para } i = 1, 2, \dots, N$$

y realizando un promedio de todos los valores obtenidos.

Puesto que la matriz Hessiana asociada a la función de coste es definida positiva, se sabe que la función tiene un mínimo local para estos parámetros; y además, debido a las restricciones impuestas mediante funciones lineales, se sabe que ese mínimo es global y único.

5.3.2. SVM para clases no linealmente separables

Un problema que existe, y que se encuentra a menudo para distribuciones reales, es que los vectores de características no se encuentran distribuidos perfectamente en dos grupos linealmente separables que se puedan separar mediante un hiperplano. En la realidad es común encontrarse datos atípicos que, por las delimitaciones del hiperplano, puedan estar contenidos en la región equivocada. Con este tipo de distribuciones cualquier intento de construir una maquina de vectores de soporte dejará vectores de características correctamente clasificados, incorrectamente clasificados y algunos vectores contenidos dentro del margen.

Es por ello que al vector de restricciones se le incluye un nuevo vector de variables ξ , para que tenga en cuenta los tres tipos de vectores de características que hay. De manera que:

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

Los vectores de características clasificados correctamente cumplen la inecuación para $\xi = 0$, aquellos que se encuentran dentro de la banda que hay entre los vectores de soporte corresponden con $0 < \xi \leq 1$ y los clasificados de forma errónea con $\xi_i > 1$. En la figura 5.3 se puede apreciar esta distinción de forma gráfica.

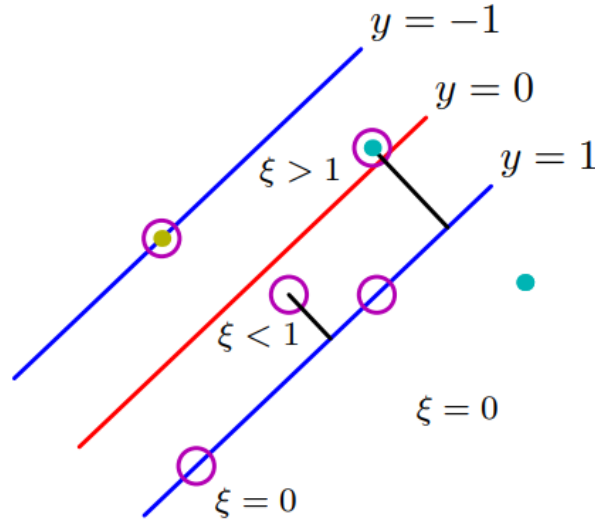


Figura 5.3: Valores de ξ según la posición de un dato con respecto a la recta clasificadora para una distribución bidimensional de clases no linealmente separables [13].

Con esta nueva variable la tarea de optimización reside en maximizar el margen, pero a diferencia del apartado anterior, esta vez teniendo en cuenta la cantidad de datos con $\xi_i > 1$ e intentando que esta sea lo menor posible; es decir, que se encuentre el hiperplano óptimo que sea capaz de dividir el mayor número de elementos en su clase correcta. La función de coste, por lo tanto, queda definida:

$$\Phi(w, b, \xi) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i$$

Sujeto a las condiciones

$$y_i(w^T x_i + b) \geq 1 - \xi_i \quad i = 1, 2, \dots, N$$

$$\xi_i > 0 \quad i = 1, 2, \dots, N$$

Donde C es una constante positiva que se encarga de moderar la influencia de ξ .

De forma análoga al caso anterior, ya que se trata de un problema de programación convexa, se define la función Lagrangiana:

$$\mathcal{L}(w, b, \xi, \alpha, \mu) = \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \mu_i \xi_i - \sum_{i=1}^N \alpha_i [y_i (w^T x_i + b) - 1]$$

Cabe destacar que las variables ξ_i y μ_i que son los multiplicadores de Lagrange asociados, no forman parte del problema expresamente, más bien están asociados a C .

De manera similar al anterior caso, el problema se puede expresar mediante su problema de optimización dual

$$W(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

Que se maximiza con respecto a α , bajo las condiciones:

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N$$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

Para este caso el multiplicador de Lagrange α_i tiene que estar comprendido entre cero y C . El vector w que define al hiperplano óptimo se calcula de la misma forma $w = \sum_{i=1}^N \alpha_i y_i x_i$.

5.3.3. SVM no lineales

Las máquinas de vectores de soporte no lineales consisten en elevar la dimensión del espacio de características a una dimensión, donde los vectores de características se pueden clasificar mediante un hiperplano lineal.

Esto se consigue mediante una función de mapeo Φ definida mediante un *kernel*. Una función *kernel* permite definir una noción de similitud devolviendo el producto interno entre dos datos en un espacio de características apropiado. Un ejemplo de *kernel* utilizado para SVM es el *kernel* polinomial, definido por:

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$$

Siendo d el grado del polinomio.

Por lo tanto el problema de clasificación queda nuevamente definido por:

$$W(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i,j}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

A maximizar $W(\alpha)$ con respecto a α , sujeto a las condiciones:

$$0 \leq \alpha_i \leq C, \quad i = 1, 2, \dots, N$$

$$\sum_{i=1}^N \alpha_i y_i = 0$$

5.4. Medidas de rendimiento

Una manera de evaluar el comportamiento de un clasificador ante un conjunto de datos es mediante las medidas de rendimiento. Mediante estas medidas se puede efectuar un análisis comparativo de distintos modelos, tanto para la parte de extracción de características como para la de determinar un clasificador, con el fin de optimizar la clasificación.

Antes de presentar estas medidas, es necesario entender los tipos de resultados posibles, ya que estas medidas dependerán del tipo de resultado obtenido en la fase de evaluación. En el caso de una clasificación binaria las etiquetas pueden ser positivas o negativas, es por ello que existen dos errores posibles: que el clasificador asocie un valor positivo a un vector de características cuya etiqueta es en realidad negativa o que asocie un valor negativo a una etiqueta positiva. Los posibles resultados vienen resumidos en la tabla 5.1, dependen de la clase que se le haya atribuido a un dato por el clasificador y la etiqueta original de ese dato; y pueden ser: Positivo Verdadero (PV) en caso de que ambas sean positivas, Negativo Falso (NF) cuando se ha hecho una predicción negativa siendo la etiqueta en realidad positiva, Positivo Falso (PF) cuando la predicción es positiva pero la etiqueta original negativa y Negativo Verdadero (NV) en el caso de que ambas sean negativas. Se define la sensibilidad de un clasificador como la habilidad que

		Etiqueta	
		Positiva	Negativa
Predicción	Positiva	PV	PF
	Negativa	NF	NV

Cuadro 5.1: Tabla resumen de resultados posibles de la predicción de un clasificador en función de la etiqueta original.

tiene de distinguir positivos verdaderos.

$$\text{Sensibilidad} = \frac{\text{cantidad de PV}}{\text{cantidad de PV} + \text{NF}}$$

Por lo que un clasificador con una muy alta sensibilidad será capaz de asociar correctamente los datos cuyas etiquetas originales sean positivas. Se podría decir por lo tanto, que si de tal clasificador se obtiene un resultado negativo, este resultado es muy fiable.

De la misma forma, se define la especificidad de un clasificador como la habilidad que tiene de distinguir negativos verdaderos.

$$\text{Especificidad} = \frac{\text{cantidad de } NV}{\text{cantidad de } NV + PF}$$

En este caso, un clasificador con muy alta especificidad será capaz de asociar correctamente los datos cuyas etiquetas originales sean negativas. Si tal clasificador proporciona un resultado positivo, se podría decir entonces que este resultado es muy fiable.

Además de la sensibilidad y especificidad, es necesario evaluar la precisión de un clasificador, que se define mediante:

$$\text{Precisión} = \frac{\text{cantidad de } PV + NV}{\text{cantidad de } PV + NF + PF + NV}$$

El clasificador ideal será aquel que tenga valores altos de sensibilidad, especificidad y precisión de manera simultánea, y no solamente un valor alto en una de las medidas.

Existen también otras medidas que pueden ser interesantes para esta aplicación llamadas valores predictivos. Los valores predictivos tanto positivo (vpp) como negativo (vpn), reflejan la validez de un resultado positivo o negativo. En un clasificador con vpp alto, si se obtiene un resultado positivo, se podrá considerar como un resultado fiable. Pero este tipo de medidas tienen una limitación, y es que tienen una estrecha relación con la prevalencia (en la epidemiología es la proporción de individuos que pertenece a una clase). Si se da el caso que en el conjunto de prueba hay un número mayor de elementos de una clase que de la otra, tiene más sentido utilizar las fracciones de probabilidad; siendo la positiva (fpp):

$$fpp = \frac{\text{sensibilidad}}{1 - \text{especificidad}}$$

y la negativa:

$$fpn = \frac{\text{especificidad}}{1 - \text{sensibilidad}}$$

puesto que no dependen de la proporción de elementos que pertenecen a cada clase.

5.5. Validación cruzada

Una vez llegado hasta este punto, surge la cuestión de como hacer la repartición de las imágenes disponibles entre un conjunto de entrenamiento y un conjunto de validación o de prueba. Hecho importante para evitar el *overfitting* de un modelo, y que este sea capaz de hacer predicciones en un conjunto de datos no visto anteriormente. Para resolver esta cuestión se hace uso de la validación cruzada.

La validación cruzada (*cross-validation* en inglés) es una técnica utilizada para evaluar la capacidad de generalización de un modelo sobre un conjunto independiente de datos. Generalmente, permite estimar la precisión de un modelo predictivo en la práctica.

A la hora de aplicar validación cruzada, se divide el conjunto de datos disponible en subconjuntos menores y complementarios. A continuación, se realiza el entrenamiento del modelo con uno de estos subconjuntos, y la validación con otro subconjunto. El hecho de utilizar ciertos subconjuntos, normalmente aleatorios, puede dar lugar a variabilidad en la evaluación (ya que un subconjunto puede tener datos favorables o desfavorables); por lo que la evaluación se lleva a cabo en múltiples rondas utilizando particiones distintas de los datos. Los resultados obtenidos tras aplicar esta técnica, serán el promedio de los resultados obtenidos en la fase de validación del modelo en cada una de las distintas rondas, ya que se conocen las etiquetas del subconjunto de validación correspondiente.

Cabe destacar que en cada una de las rondas de evaluación, se deberán reiniciar los parámetros del modelo para que la fase de entrenamiento se realice con el subconjunto que corresponde; así se asegura que las evaluaciones de las distintas iteraciones sean independientes, y no haya influencia de otro subconjunto de datos utilizado en el entrenamiento de otra ronda anterior.

5.5.1. Validación cruzada k-fold

La validación cruzada k-fold consiste en dividir el conjunto de datos en k número de subconjuntos, donde el entrenamiento se efectúa mediante k-1 subconjuntos y el subconjunto restante es utilizado para la validación. Este proceso se reitera k veces, utilizando en cada iteración un subconjunto de validación diferente. De tal forma que tras las k iteraciones, todos los elementos del conjunto muestral hayan sido utilizados para la fase de validación. Este método se suele utilizar cuando los algoritmos de aprendizaje son computacionalmente costosos o se dispone de un gran volumen de muestras.

El valor de k escogido dependerá de varios factores y es importante para que los resultados representen la capacidad del modelo de hacer buenas predicciones. Estos factores son la varianza, que puede cambiar en función de los datos utilizados para entrenar el modelo; el sesgo, que puede llevar a

una sobrestimación de las competencias del modelo y disminuye conforme aumenta k ; y el coste computacional, que aumenta conforme aumenta k debido a que determina el número de iteraciones que se realizarán. El objetivo por lo tanto, es encontrar un compromiso entre estos factores y escoger el valor que proporcionará los resultados más fiables para el modelo.

En la práctica, unos valores típicos serían de $k=5$ o $k=10$, ya que empíricamente han mostrado que sus resultados no sufren de un sesgo excesivo ni una varianza demasiado alta. Para esta aplicación, el primer valor de $k=5$ sería más interesante, ya que el entrenamiento del modelo es bastante costoso de por sí, y un valor de $k=10$ supondría el doble de tiempo para la evaluación completa del modelo. Un esquema para entender mejor la evaluación por validación cruzada se puede encontrar en 5.4.

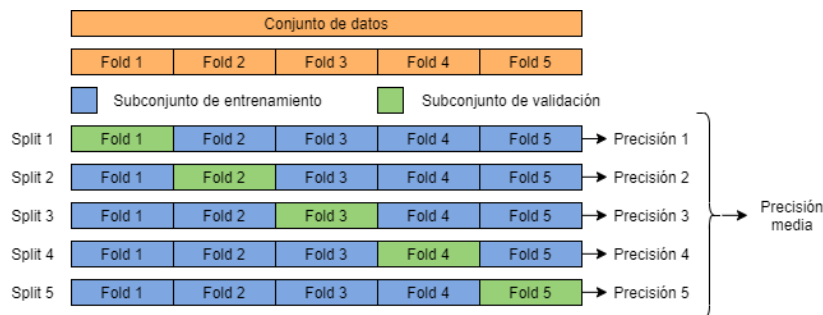


Figura 5.4: Validación cruzada para $k=5$.

Capítulo 6

Implementación

Este capítulo tiene la función de describir la parte práctica de este trabajo, ya sea la implementación de los conceptos descritos, la construcción del modelo o la evaluación de este.

La mayor parte de la implementación se ha realizado en Python, el único proceso que se ha realizado fuera del entorno ha sido la segmentación, para el que se ha hecho uso de su software específico SPM en el entorno de Matlab. A lo largo de este capítulo se mencionarán ciertas librerías o programas complementarios que han sido de utilidad.

6.1. Bases de datos

La base de datos utilizada para el aprendizaje del modelo contiene 362 imágenes de resonancia magnética tipo T1 de distintos pacientes. Las imágenes se pueden clasificar en tres grupos según el diagnóstico de los profesionales médicos encargados del seguimiento de los pacientes, estos tres grupos serán las tres clases que se distinguirán para la clasificación automática.

- Normal o control, para los pacientes diagnosticados sanas.
- MCI, para los pacientes diagnosticados con deterioro cognitivo leve (MCI del inglés *mild cognitive impairment*) y para aquellos pacientes que llevan un largo periodo en este estado (LMCI del inglés *long mild cognitive impairment*).
- AD, para aquellos pacientes que han sido diagnosticados con enfermedad de Alzheimer.

El número de imágenes pertenecientes a cada categoría queda reflejada en la tabla 6.1.

	Control	MCI	AD
Número de pacientes	101	182	79

Cuadro 6.1: Tabla que muestra el número de imágenes pertenecientes a cada clase.

6.2. Carga de datos y preprocesado

Una vez realizada la segmentación de las imágenes, tal y como se describe en la sección 3.1, obteniéndose así las imágenes estructurales de la materia gris, llega el momento de cargar los datos en el entorno de Python y procesarlos antes de presentarlos al modelo clasificador. El módulo de **carga_y_procesado_datos** incluye tres funciones que se encargan de esta labor.

- La primera función **cargar_datos** se encarga de cargar las imágenes a una matriz de cuatro dimensiones (la primera dimensión indica el número de la imagen y las demás son para guardar los datos de cada imagen) mediante un bucle, utilizando la librería de Nibabel. También crea un vector en el que se van registrando las etiquetas de cada imagen con valores de 0 para el grupo de control, 1 para MCI y 2 para AD; esto se consigue accediendo al documento de Excel que contiene la información de cada paciente, y mediante su número de identificación, se extrae el diagnóstico de forma automática. Cada vez que se registra una etiqueta, se suma uno en una de las posiciones del vector **cantidad_datos** dependiendo de la etiqueta registrada; de esta forma, al final del bucle se sabrán la cantidad de imágenes que pertenecen a cada clase.
- La segunda función **procesado_datos** es la encargada de realizar el procedimiento descrito en la sección 3.2, para ello se utilizan tres bucles que realizan la media de los elementos que ocupan las posiciones pares con los que ocupan las posiciones impares. El uso de tres bucles es debido a que los datos están contenidos en matrices de tres dimensiones, por lo que cada bucle se encarga de reducir una dimensión diferente. Puesto que el algoritmo es costoso (tardó más de un día en realizar todas las operaciones), la variable resultante, que recoge todas las matrices de datos reducidas, se guarda en un archivo externo para así poder cargar los datos directamente en el experimento en vez de tener que repetir todo el proceso.
- La última función **clasificacion_binaria** se utiliza para extraer los datos de la variable en función de sus etiquetas, y guardarlos en otra variable donde sólo se recogen los datos que pertenecen a las dos ca-

tegorías de interés para la clasificación binaria. Las etiquetas también se extraen de su variable correspondiente y se almacenan en otra de la misma forma que los datos. Tanto la nueva variable de datos como la de etiquetas se almacenan en un archivo externo, con el fin de poder cargar los datos directamente en el experimento de clasificación binaria correspondiente.

6.3. Construcción e implementación del modelo

6.3.1. Contrucción del Autoencoder

Las diferentes estructuras probadas para el *autoencoder* se recogen en el módulo **Autoencoders**, donde las funciones se encargan de construir la estructura capa por capa, dejando la capa central variable para definir desde el módulo en el que se realiza el experimento, ya que el número de neuronas en la capa central determinará la dimensión de los vectores de características, que serán introducidos al clasificador.

A continuación se presentan las tres estructuras ideadas, siendo la primera un prototipo con el que se hicieron varias pruebas y la base con la que se idearon las otras dos. La segunda es una versión mejorada de la primera por el hecho de eliminar las capas de *pooling* y *upsampling*, utilizando en su lugar capas convolucionares de *strides* variables para ir reduciendo la dimensión de los datos, y de la misma forma capas de convolución transpuesta para hacer la reconstrucción en el *decoder*, mejorando así el rendimiento. La tercera estructura implementa capas de convolución aparte de las que se utilizan para la reducción del tamaño de los datos, con el fin de una mejor extracción de características. Las estructuras recibirán el nombre de **AE-1** (figura 6.1), **AE-2** (figura 6.2) y **AE-3** (figura 6.3), según el orden en el que han sido diseñadas.

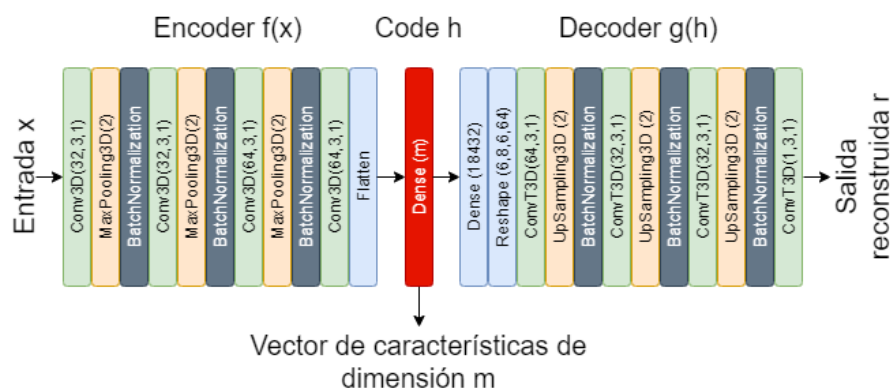


Figura 6.1: Diagrama que muestra la estructura capa por capa de **AE-1**.

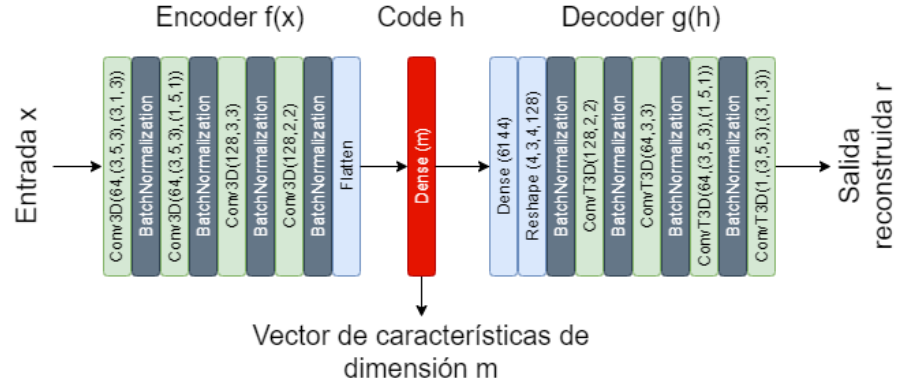


Figura 6.2: Diagrama que muestra la estructura capa por capa de **AE-2**.

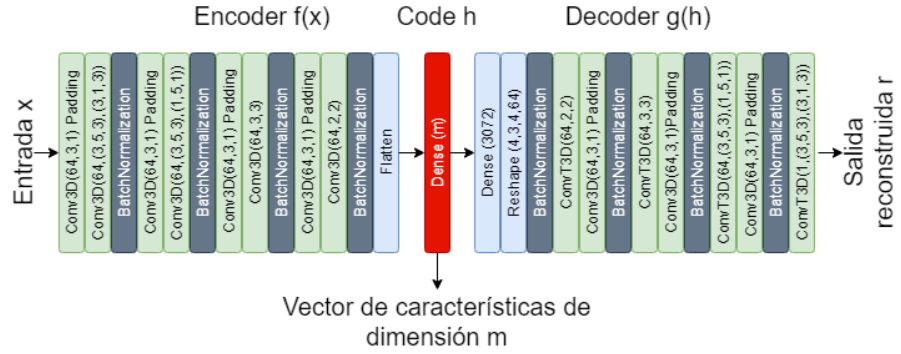


Figura 6.3: Diagrama que muestra la estructura capa por capa de **AE-3**.

La construcción de las estructuras se hizo con la librería de **Keras**, que es una API para Python basada en la plataforma de **Tensorflow**, que permite definir redes neuronales y construirlas capa por capa, además de incluir los algoritmos necesarios para el aprendizaje de estas.

La nomenclatura seguida para las distintas capas es:

- Para las capas de convolución 3D y convolución transpuesta 3D, (a, b, c) : donde a es el número de *kernels* utilizados en esa capa, b es el vector que define el tamaño del *kernel*, si viene dado por un número se trata de un *kernel* cuadrado; y c es el *stride*, que puede ser un vector para indicar como se desplaza el filtro por cada una de las tres dimensiones, o un entero que indica que se aplicará ese *stride* en todas las dimensiones. El *padding* viene indicado para aquellas capas donde se haya utilizado para mantener la dimensión.
- Para las capas de neuronas densamente conectadas (perceptrón multicapa), el número entre paréntesis corresponde con el número de neuronas que tiene esa capa.

- En las capas de *MaxPooling* y *UpSampling*, el número indica el tamaño de la ventana cuadrada usada para realizar la operación.
- Las capas de *flatten* sirven para aplanar los datos a un vector, y las de *Reshape* para redimensionar los datos desde un vector a una matriz de dimensiones especificadas entre el paréntesis.

La capa del centro, que se compone de m número de neuronas, se define desde la función encargada de construir la estructura. Esto se ha hecho para poder probar con distintos valores de m desde el módulo donde se realiza el experimento, y así no tener que cambiar las funciones que construyen las estructuras. Este valor es importante, ya que determina la dimensión de los vectores de características utilizados para el aprendizaje y evaluación del clasificador estadístico. Para la obtención de los resultados se utilizó un valor de $m = 15$.

La función de *loss* utilizada es la de error cuadrático medio, ya que en un *autoencoder* la salida es la reconstrucción de la entrada, por lo que interesa que sea lo más parecida posible. La función de *loss* queda definida por:

$$\text{Loss} = \frac{1}{N} \sum_i (x_i - r_i)^2 = \frac{1}{N} \sum_i (x_i - g(f(x_i)))^2$$

Siendo x_i los datos de entrada y r_i las salidas (reconstrucción de la entrada).

Las capas de *BatchNormalization* se introdujeron en una etapa temprana del diseño debido a que en algunas de las pruebas de las estructuras, los *autoencoders* proporcionaban el mismo vector de características para distintos datos de entrada, impidiendo así que el clasificador aprendiese las relaciones entre las etiquetas y la información contenida en los vectores de características (ya que todos eran iguales). Esta capa se encarga de normalizar las entradas, y en este caso permitió que la capa del centro del *autoencoder* aprendiese a extraer las características.

El algoritmo optimizador que se ha utilizado es el de **Adam** (*adaptive moment estimation*), que es el que mostraba un mejor rendimiento y una convergencia más rápida de los diferentes algoritmos probados (RMSProp, SGD).

La función de activación de cada capa es **ReLU** (linear uniforme rectificadora); a excepción de la capa central, que utiliza una función de activación lineal para poder obtener más valores en el vector de características, de manera que los valores negativos no se rectificasen a cero. Se probó utilizar una función de activación de **Softmax** en la última capa, pero este planteamiento producía un error mayor y una convergencia más lenta.

Por último cabe destacar que se instaló y utilizó el **Toolkit CUDA** de NVIDIA, compatible con la pataforma de **Tensorflow**, que permite la utilización de una GPU para el aprendizaje de redes neuronales. Este tipo de

tecnología, por su arquitectura, es más eficiente que un procesador convencional a la hora de realizar operaciones con tensores, por lo que disminuyen drásticamente los tiempos de entrenamiento. En este caso, tras su implementación, el tiempo que se necesitaba para entrenar una red neuronal se redujo en un factor de 20.

6.3.2. Evaluación del modelo

Las funciones utilizadas para la evaluación de los distintos modelos y para la realización de las diversas pruebas, se encuentran en el módulo **expression_resultados**. Estas funciones utilizan la librería de **matplotlib** para la representación de gráficas, que serán de utilidad para realizar la evaluación del modelo. Las funciones de este módulo son:

- La primera función **generar_plot**, se encarga de representar los valores de la función de *loss*, tanto para el entrenamiento como la evaluación del *autoencoder*, con respecto a las *epochs*; es decir, genera una gráfica con el error de entrenamiento y de validación, para cada iteración.
- La función **representar_matrizconfusion** genera una figura de la matriz de confusión y otra de la matriz de confusión normalizada. Las matrices de confusión permiten visualizar el rendimiento de un clasificador, comparando las etiquetas en las que se han clasificado ciertos datos con las etiquetas verdaderas de esos datos. Para generar estas matrices se utiliza la librería de **scikit-learn**, que permite generar la matriz a partir del vector que contiene las predicciones y del vector que contiene las etiquetas originales. Se llama también a la función **generar_matrizconfusion** que permite definir ciertos parámetros para su mejor visualización, por ejemplo asocia los valores de las etiquetas utilizadas para el clasificador (0, 1 y 2) a sus clases originales (Control, MCI y AD) para su correcta visualización en la matriz de confusión.

6.3.3. Funcionamiento del programa principal

El módulo principal es donde se realiza el entrenamiento y la evaluación del modelo en las distintas rondas de validación cruzada, haciendo uso de las funciones descritas en los dos apartados anteriores. Para entender mejor la estructura y funcionamiento del programa, se va a hacer uso del diagrama 6.4 y explicar cada uno de los bloques.

- Cargar datos en función de la clasificación: se cargan tanto las matrices de datos como las etiquetas, en función de si se va a realizar una clasificación binaria o multiclase. La carga se hace desde archivos externos que han sido creados con las funciones del módulo **carga_y_procesado_datos**, ya explicado en la sección 6.2.

- Construcción y compilación autoencoder: se define y compila el autoencoder con los hiperparámetros correspondientes, tal y como se describió en la sección 6.3.1.
- Guardar parámetros autoencoder: se guardan los pesos y los sesgos del autoencoder antes de realizar las rondas de validación cruzada para poder restablecerlos tras cada ronda.
- Validación cruzada: se dividen los datos en cinco subconjuntos y se realizan el entrenamiento y evaluación, tanto del *autoencoder* como del clasificador, en cinco rondas (*folds*). En cada una de estas rondas se usa un subconjunto de evaluación diferente, tal y como se mencionó en la sección 5.5.1. Para la división de los datos y etiquetas, y la declaración del bucle, se ha utilizado la librería de **scikit-learn**
- Entrenamiento autoencoder: se definen los parámetros para realizar el entrenamiento del *autoencoder*, para esta aplicación, para la mayoría de los experimentos se han utilizado veinte iteraciones (*epochs*). Se recuerda que la estructura de *autoencoder* utiliza los propios datos de entrada como etiqueta u objetivo, por lo que en la función **fit** de la librería de **Keras** hay que especificar la variable de datos dos veces. La evaluación del *autoencoder* se realiza después de cada iteración con el subconjunto de test. Los valores de la función de *loss*, tanto para el entrenamiento como la evaluación, se almacenan en la variable **historial**.
- Definición clasificador: la librería de **scikit-learn** incluye una gran variedad de algoritmos para el aprendizaje automático. Es en esta parte donde se define al clasificador que se vaya a utilizar para el experimento (**SVM** para las clasificaciones binarias, y árboles de decisión o **SVM** con estrategia *one-vs-all*). Al estar contenido dentro del bucle de validación cruzada, el clasificador se define en cada ronda, para reiniciar sus parámetros y que tenga que ser entrenado desde cero con un conjunto distinto.
- Obtención vector de características del conjunto de entrenamiento: usando la función **predict** con el *encoder* ya entrenado, se obtienen los vectores de características del conjunto de entrenamiento; es decir, las imágenes tridimensionales se comprimen en un vector de información.
- Entrenamiento clasificador: con el vector de características del conjunto de entrenamiento se procede a realizar el aprendizaje del clasificador, en este caso las etiquetas serán las distintas clases a las que pertenecen los datos (los tres grupos mencionados en la sección 6.1).
- Obtención vector de características del conjunto de evaluación: para evaluar el clasificador es necesario obtener también los vectores de

características de los datos del conjunto de evaluación. Este paso se efectúa de forma análoga al de obtención vector de características del conjunto de entrenamiento.

- Predicción del clasificador sobre los vectores de características de evaluación: con el clasificador ya entrenado y los vectores de características de evaluación, se realizan las predicciones de las clases a las que pueden pertenecer los datos mediante la función **predict**. Las predicciones se van almacenando en la variable **predicted_labels** para que al final del experimento, se puedan comparar las etiquetas verdaderas con las etiquetas estimadas del clasificador para todos los conjuntos de evaluación.
- Obtención de precisión del clasificador y del error del autoencoder: se calcula la precisión del clasificador para la ronda de validación cruzada y se almacena en el vector **score** para obtener la precisión media de todas las rondas al final. También se llama a la función **generar_plot** (descrita en 6.3.2) utilizando la variable **historial**, con el fin de evaluar el proceso de aprendizaje del *autoencoder*.
- Cargar parámetros autoencoder: una vez hechas las predicciones y calculada la precisión, se reinician los parámetros del *autoencoder* para poder entrenarlo desde los valores iniciales en la siguiente ronda de validación cruzada.
- Obtención medidas de rendimiento/generación matriz de confusión: una vez finalizadas todas las rondas de validación cruzada, se calcula la precisión media, se obtienen las medidas de rendimiento para la clasificación binaria y se generan las matrices de confusión para la clasificación multiclase.

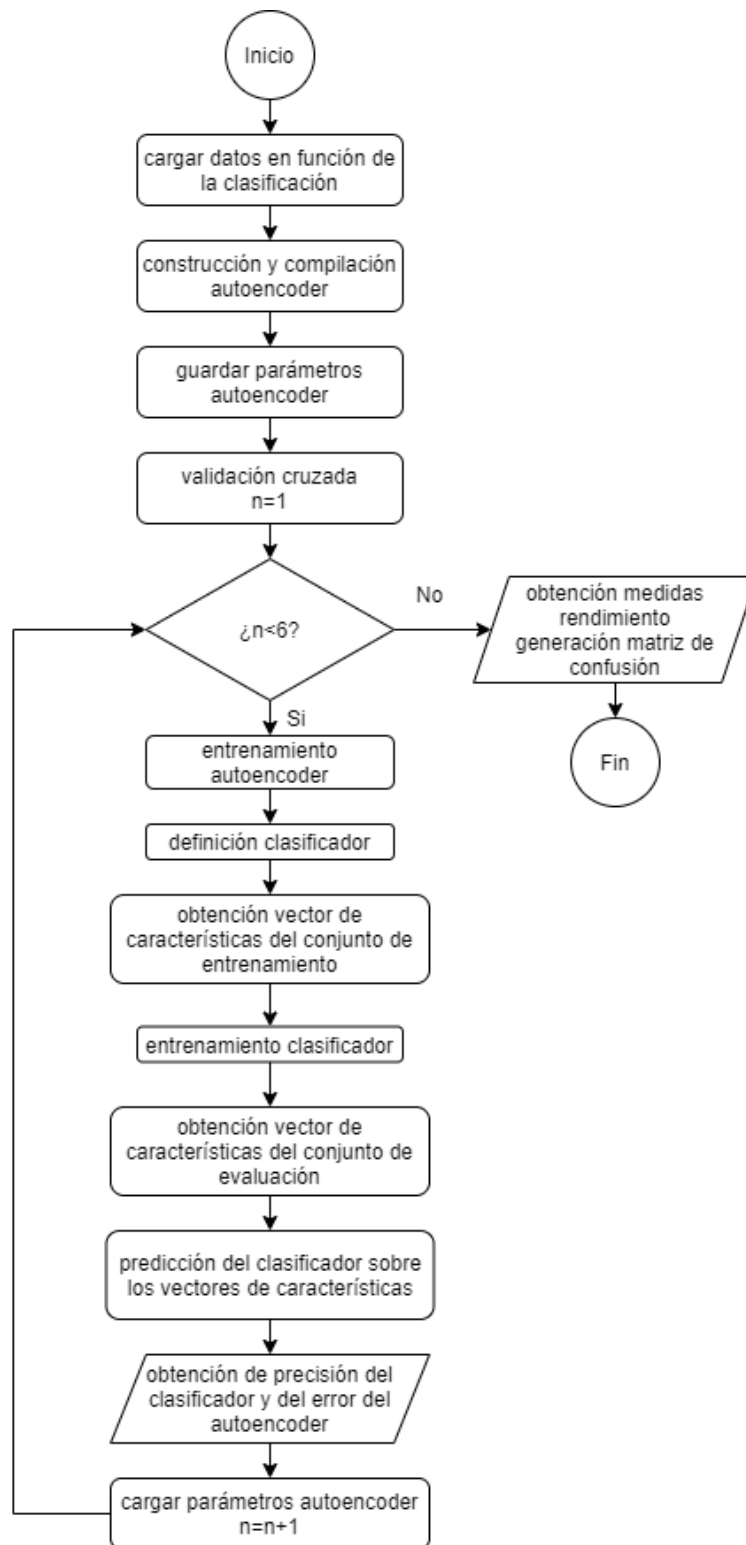


Figura 6.4: Diagrama de flujo del módulo principal.

Capítulo 7

Análisis de los resultados

A continuación se van a presentar los principales resultados de las clasificación multiclase y la clasificación binaria.

7.1. Clasificación multiclase

Para la clasificación multiclase se utilizó el *autoencoder-3*, haciendo un análisis de la relación entre la precisión media y la dimensión de los vectores de características, para los dos tipos de clasificadores. Los resultados obtenidos se pueden apreciar en la gráfica 7.1.

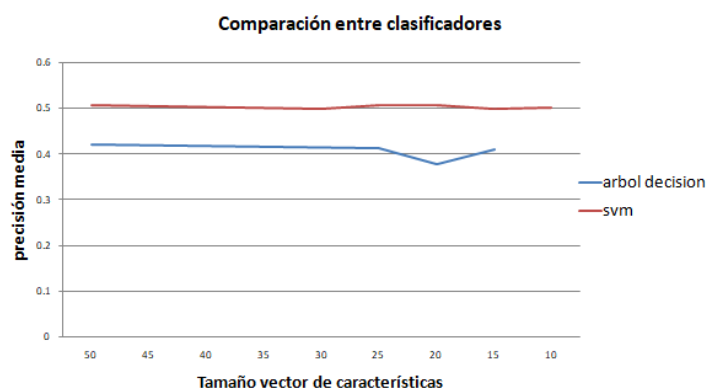


Figura 7.1: Gráfica mostrando precisión media según el número de neuronas de la capa central del *autoencoder-3*.

Como se puede apreciar, la precisión media no varía mucho con respecto a la dimensión de los vectores de características, ya que no se han estudiado puntos lo suficientemente bajos como para que suponga una disminución en la precisión. Por esta razón, para la obtención de los resultados principales se han utilizado 15 neuronas en la capa central del *autoencoder*, ya que no

supone una pérdida de precisión y mejora el tiempo de entrenamiento del modelo al disminuir el número de parámetros para las conexiones de la capa central.

También cabe destacar que el clasificador basado en máquinas de vectores de soporte tiene un mejor rendimiento que el clasificador basado en árboles de decisión.

Para obtener los resultados del modelo, se evaluó también la función de *loss* del autoencoder tanto para los datos de entrenamiento como los de test, obteniendo para el primer fold, la gráfica 7.2 para el clasificador basado en árboles de decisión y la gráfica 7.3 para el que está basado en SVM.

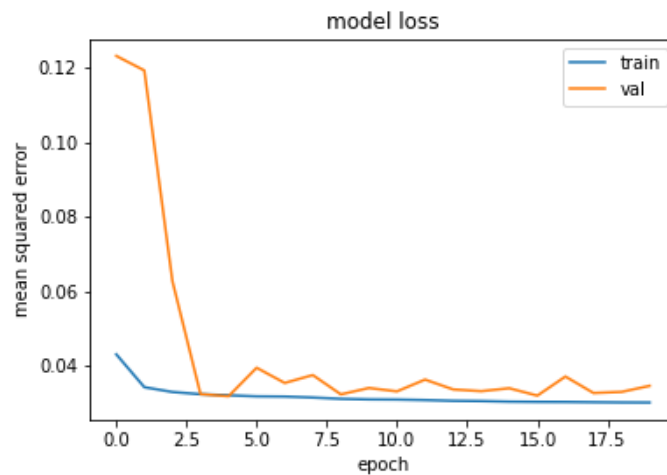


Figura 7.2: Gráfica que muestra el error cuadrático medio del *autoencoder-3* para la clasificación con árboles de decisión.

El error disminuye tanto para las sucesivas iteraciones del entrenamiento como para las de validación, esto es un signo que el *autoencoder* no se está sobreajustando a los datos de entrenamiento; ya que si ese fuera el caso, el error de los datos de test no tendería a disminuir como el error de los datos de entrenamiento.

Los resultados obtenidos por los clasificadores vienen recogidos en la tabla 7.1.

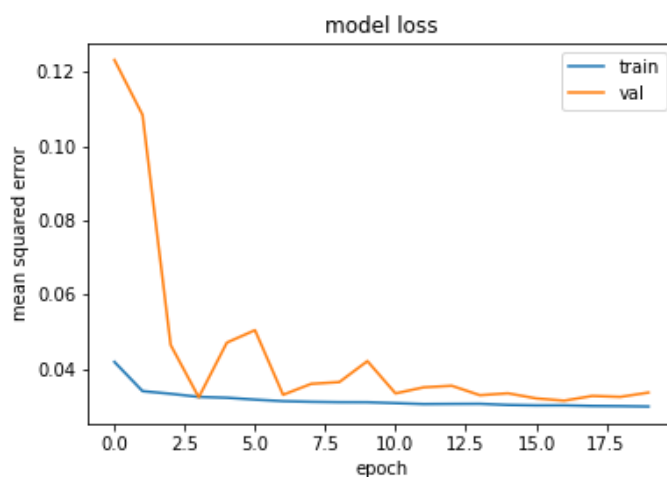


Figura 7.3: Gráfica que muestra el error cuadrático medio del *autoencoder-3* para la clasificación con máquinas de vectores de soporte.

Clasificación Multiclase	DT	SVM
Precisión fold 1	0,4384	0,5753
Precisión fold 2	0,4384	0,4521
Precisión fold 3	0,4306	0,4444
Precisión fold 4	0,375	0,5833
Precisión fold 5	0,3611	0,4444
Precisión media	0,4087	0,4999
Desviación típica	0,0375	0,0726

Cuadro 7.1: Resultados para la clasificación multiclase.

Como era de esperar, el clasificador basado en máquinas de vectores de soporte ha obtenido una precisión mayor aunque con una desviación típica mayor. Para visualizar los resultados a nivel de etiquetas, se utilizan las matrices de confusión (figuras 7.4 y 7.5).

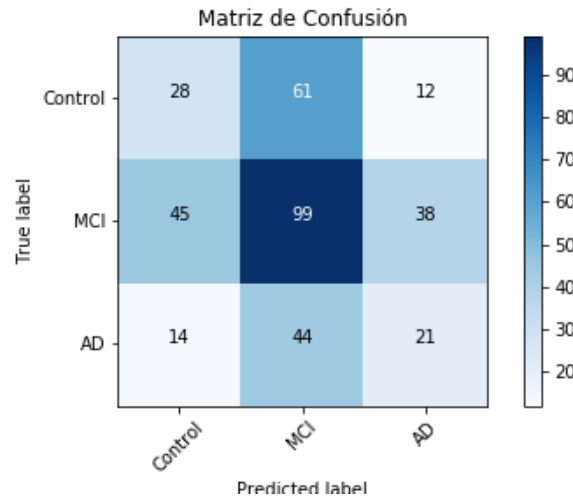


Figura 7.4: Matriz de confusión del clasificador basado en árboles de decisión.

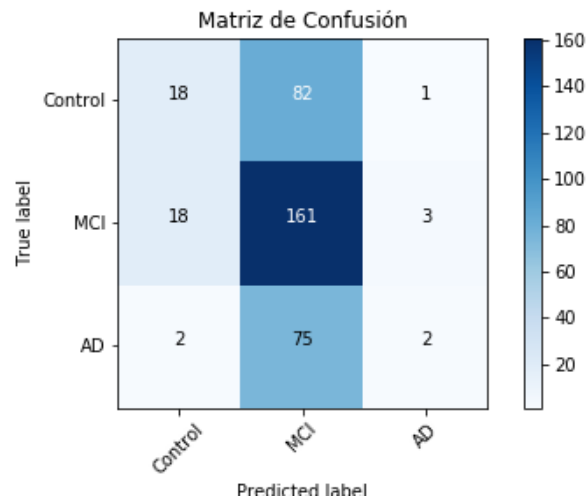


Figura 7.5: Matriz de confusión del clasificador basado en máquinas de vectores de soporte.

Tras analizar las matrices de confusión para ambos casos, se puede concluir que aunque el clasificador basado en SVM tenga una mayor precisión, el de árboles de decisión es capaz de detectar mejor los pacientes sanos (control) y con enfermedad de Alzheimer (AD). La mayor precisión se debe por lo tanto, a que el clasificador de SVM identifica bien a los pacientes con deterioro cognitivo leve y esta es la clase con mayor cantidad de datos, mientras que el clasificador de árboles de decisión categoriza ciertos elementos de este grupo en Control y AD. Sería interesante ver los resultados que se obtendrían

con una muestra mayor de imágenes de pacientes sanos y enfermos.

Cabe destacar que las clasificaciones entre varias clases pueden ser a veces complicadas, sobretodo en este tipo de casos en los que los elementos de las distintas clases pueden llegar a solaparse entre ellos.

7.2. Clasificación binaria

El problema de clasificación multiclase se puede plantear en varios problemas de clasificación binaria; de manera que si antes se tenía un problema de clasificación entre Control-MCI-AD, se puede plantear como tres problemas binarios: AD-Control, MCI-Control y AD-MCI.

Para los resultados principales se ha utilizado de nuevo el *autoencoder-3* con una máquina de vectores de soporte como clasificador; pero primero se mostrará la precisión obtenida para varias pruebas de clasificación AD vs Control, con las tres estructuras planteadas para el *autoencoder*, con distintos valores para la cantidad de neuronas de la capa central. Los resultados se encuentran recogidos en la tabla 7.2

Nº neuronas	Precisión AE-1	Precisión AE-2	Precisión AE-3
50	0,7222	0,7278	0,7611
25	0,7333	0,7389	0,7333
20	0,7278	0,75	0,7611

Cuadro 7.2: Resultados de la precisión obtenida para las distintas estructuras.

Como se puede comprobar, el *autoencoder-3* ha obtenido ligeramente una mejor precisión de media que los otros dos, justificando así su uso para la obtención de los resultados finales.

De la misma forma que en el apartado anterior, se han obtenido los errores de entrenamiento y test del *autoencoder* para las tres clasificaciones binarias (gráficas 7.6, 7.7 y 7.8)

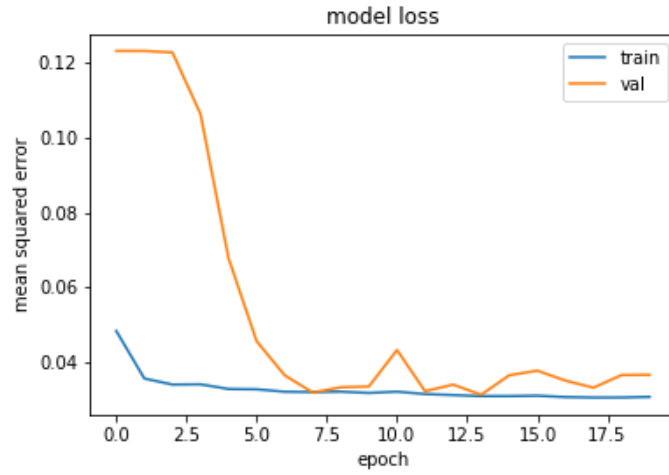


Figura 7.6: Gráfica que muestra el error cuadrático medio del *autoencoder-3* en uno de los folds, para la clasificación AD vs Control.

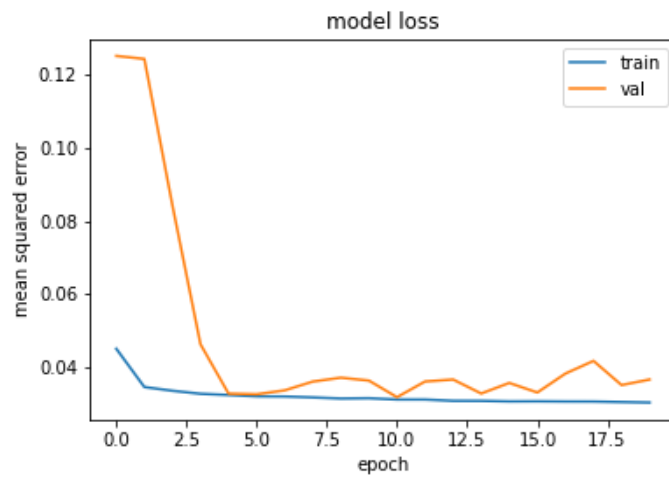


Figura 7.7: Gráfica que muestra el error cuadrático medio del *autoencoder-3* en uno de los folds, para la clasificación MCI vs Control.

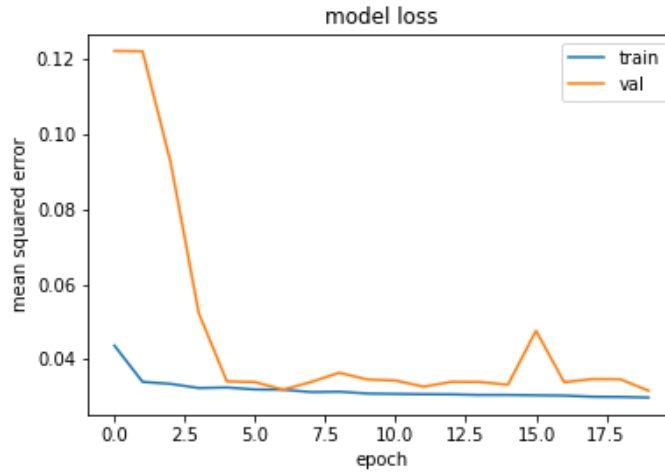


Figura 7.8: Gráfica que muestra el error cuadrático medio del *autoencoder-3* en uno de los folds, para la clasificación AD vs MCI.

De nuevo, aunque existan ciertos "relieves" en la función, el error de los datos de test tiende a bajar.

Los resultados obtenidos por el modelo para las clasificaciones binarias vienen recogidos en la tabla 7.3

Clasificación binaria	AD vs Control	MCI vs Control	AD vs MCI
Precisión fold 1	0,8889	0,7719	0,9057
Precisión fold 2	0,6944	0,6316	0,6346
Precisión fold 3	0,7778	0,5789	0,6154
Precisión fold 4	0,6944	0,75	0,7308
Precisión fold 5	0,6944	0,5714	0,6154
Precisión media	0,75	0,6608	0,7004
Desviación típica	0,0856	0,0947	0,1243
Sensibilidad	0,6582	0,989	0,0127
Especificidad	0,8218	0,0594	1
fpp	3,6934	0,9303	-
fpn	0,4159	0,185	0,9873

Cuadro 7.3: Resultados de las clasificaciones binarias utilizando un clasificador SVM.

En la clasificación entre pacientes sanos y pacientes con Alzheimer se han obtenido buenos resultados, así como buenas medidas de rendimiento; al tener un valor de fpp alto, quiere decir que si el clasificador da un valor positivo, este es muy fiable. En cambio en las otras dos clasificaciones aunque la precisión media no sea baja, una de las medidas de rendimiento si; esto se debe a que el clasificador ha considerado la mayoría de los datos

como pertenecientes a la clase de MCI, que es la clase con mayor número de pacientes. Cabe destacar que la fracción de probabilidad positiva (fpp) del experimento MCI vs Control no se ha asignado, puesto que como la especificidad es de 1 (el clasificador no ha registrado ningún positivo falso) al calcular la fpp se estaría dividiendo entre cero.

Capítulo 8

Conclusiones y trabajo futuro

8.1. Conclusiones

EL modelo es capaz de clasificar entre pacientes sanos y pacientes con Alzheimer; en cambio, no es capaz de diferenciar correctamente entre pacientes con deterioro cognitivo leve y las otras categorías. El problema resulta, por lo tanto, demasiado complejo cuando interviene la clase de MCI, para el modelo propuesto. Parte de la causa de esto puede ser el hecho de haber tenido que reducir los datos a la mitad con la media por el peso de las matrices de datos, con lo que se pierde cierto nivel de detalle en la imagen original.

Los profesionales médicos a la hora de determinar si un paciente sufre de deterioro cognitivo leve, realizan distintas pruebas y evaluaciones teniendo en cuenta los síntomas del paciente para efectuar el diagnóstico. Por lo que disponen de información que puede no estar reflejada en las imágenes de resonancia magnética.

Las máquinas de vectores de soporte han mostrado una mayor precisión en la clasificación multiclase, pero esto puede ser a que han clasificado correctamente más elementos del grupo MCI, que el clasificador de árbol de decisión, al ser el grupo MCI el de mayor número de pacientes. Sería interesante estudiar que pasaría si usase la misma cantidad de pacientes por clase, puesto que el clasificador de árbol de decisión ha clasificado bien más elementos de las otras dos clases.

Otra manera de aumentar la precisión sería utilizando, además de las imágenes estructurales, imágenes funcionales (PET o SPECT), analizándolas de forma similar e incluyendo la información adicional que proporcionan en el vector de características.

8.2. Proyección de futuro

Este proyecto ha servido como introducción al mundo del aprendizaje automático, por lo que es evidente que se puede ampliar. A continuación se describen ciertos aspectos que se pueden tener en cuenta como trabajo futuro.

- Intentar optimizar el modelo utilizando imágenes funcionales, tal y como se ha descrito en las conclusiones.
- Optimizar el *autoencoder* aún más haciendo más pruebas con distintos hiperparámetros aumentando su complejidad si fuese necesario.
- Realizar el entrenamiento en un equipo con mejores especificaciones, para así poder entrenar el *autoencoder* con más iteraciones y sin tener que reducir las matrices de datos de dimensión para no perder nivel de detalle.
- Probar con distintos *kernels* en las máquinas de vectores de soporte, sobretodo para las clasificaciones con MCI, por si se encuentra un espacio de mayor dimensionalidad donde las clases son separables mediante un hiperplano lineal.
- Como ejercicio didáctico, implementar algoritmos de aprendizaje no supervisado para la clasificación con el objetivo compararlos con los algoritmos de aprendizaje supervisado. Implementar también otros tipos de clasificadores.

Anexo

Aquí se recoge el código utilizado.

Módulo: carga_y_procesado_datos

```
import numpy as np
import pandas as pd
from pathlib import Path
import nibabel as nib

def cargar_datos(dataPath,CLF):

    dataPath='C:\\Users\\User\\Documents\\6\\TFG\\Imágenes\\muestras\\'
    print(dataPath)
    dataset=pd.read_excel(dataPath +'dataset.xls')
    df=pd.DataFrame(dataset)
    img_data=np.ndarray(shape=(362,157,189,156), dtype='float32')
    cantidad_datos=[0,0,0]
    if CLF==1:
        labels=np.ndarray(shape=(362,1), dtype='float32')
        x=0
        for file in Path(dataPath).glob('**/*.nii'):
            id=(str(file)[56:66])
            lista_estado=df[df['Subject ID']==id]['DX Group']
            estado=lista_estado.iloc[0]
            if estado=='Normal':
                labels[x]=[0]
                cantidad_datos[0]+=1
            elif estado=='AD':
                labels[x]=[2]
                cantidad_datos[2]+=1
            elif estado=='LMCI':
                labels[x]=[1]
                cantidad_datos[1]+=1
            elif estado=='MCI':
                labels[x]=[1]
                cantidad_datos[1]+=1
            img= nib.load(file)
            img_data[x,:,:]=img.get_fdata()
            print(str(x) + ' de 361')
            x +=1
    else:
        labels=np.ndarray(shape=(362,3), dtype='float32')
        x=0
        for file in Path(dataPath).glob('**/*.nii'):
            id=(str(file)[56:66])
            lista_estado=df[df['Subject ID']==id]['DX Group']
            estado=lista_estado.iloc[0]
```

```

        if estado=='Normal':
            labels[x,:]=[1,0,0]
        elif estado=='AD':
            labels[x,:]=[0,0,1]
        elif estado=='LMCI':
            labels[x,:]=[0,1,0]
        elif estado=='MCI':
            labels[x,:]=[0,1,0]
        img= nib.load(file)
        img_data[x,:,:]=img.get_fdata()
        print(str(x) + ' de 361')
        x +=1
    return labels,cantidad_datos

def procesado_datos(img_data, savePath):
    X=img_data[:,0:156,0:188,:]
    Y=np.ndarray(shape=(362,156,188,78), dtype='float32')
    Z=np.ndarray(shape=(362,156,94,78), dtype='float32')
    X_mean=np.ndarray(shape=(362,78,94,78), dtype='float32')
    for k in range(78):
        Y[:, :, :, k]=(X[:, :, :, 2*k]+X[:, :, :, 2*k+1])/2
        print('k='+str(k))
    for j in range(94):
        Z[:, :, :]=(Y[:, :, 2*j,:]+Y[:, :, 2*j+1,:])/2
        print('j='+str(j))
    for i in range(78):
        X_mean[:, i, :, :]=(Z[:, 2*i, :, :]+Z[:, 2*i+1, :, :])/2
        print('i='+str(i))
    np.save(Path(savePath), X_mean)

def clasificacion_binaria(input_data, labels, cantidad_datos, tipo_clasificacion):
    j=0
    if tipo_clasificacion == 'ADvsNORMAL':
        datos_extraidos=
np.ndarray(shape=(cantidad_datos[0]+cantidad_datos[2],78,94,78),dtype='float32')
        etiquetas_extraidas=np.empty(shape=(cantidad_datos[0]+cantidad_datos[2],1),dtype='float32')
        for i in range(len(labels)):
            if labels[i]==0:
                datos_extraidos[j]=input_data[i]
                etiquetas_extraidas[j]=labels[i]
                j+=1
            elif labels[i]==2:
                datos_extraidos[j]=input_data[i]
                etiquetas_extraidas[j]=labels[i]
                j+=1
        np.save(Path('C:\\Users\\User\\Documents\\6\\TFG\\datos procesados\\ADvsNORMAL'),
datos_extraidos)
        np.save(Path('C:\\Users\\User\\Documents\\6\\TFG\\datos procesados\\ADvsNORMAL_labels'),
etiquetas_extraidas)
    elif tipo_clasificacion== 'MCIvsNORMAL':

```

```

    datos_extraidos=
np.ndarray(shape=(cantidad_datos[0]+cantidad_datos[1],78,94,78),dtype='float32')
    etiquetas_extraidas=np.empty(shape=(cantidad_datos[0]+cantidad_datos[1],1),dtype='float32')
    for i in range(len(labels)):
        if labels[i]==0:
            datos_extraidos[j]=input_data[i]
            etiquetas_extraidas[j]=labels[i]
            j+=1
        elif labels[i]==1:
            datos_extraidos[j]=input_data[i]
            etiquetas_extraidas[j]=labels[i]
            j+=1
    np.save(Path('C:\\Users\\User\\Documents\\6\\TFG\\datos
procesados\\MCIvsNORMAL'),datos_extraidos)
    np.save(Path('C:\\Users\\User\\Documents\\6\\TFG\\datos procesados\\MCIvsNORMAL_labels'),
etiquetas_extraidas)
    elif tipo_clasificacion == 'ADvsMCI':
        datos_extraidos=
np.ndarray(shape=(cantidad_datos[2]+cantidad_datos[1],78,94,78),dtype='float32')
        etiquetas_extraidas=np.empty(shape=(cantidad_datos[2]+cantidad_datos[1],1),dtype='float32')
        for i in range(len(labels)):
            if labels[i]==2:
                datos_extraidos[j]=input_data[i]
                etiquetas_extraidas[j]=labels[i]
                j+=1
            elif labels[i]==1:
                datos_extraidos[j]=input_data[i]
                etiquetas_extraidas[j]=labels[i]
                j+=1
        np.save(Path('C:\\Users\\User\\Documents\\6\\TFG\\datos
procesados\\ADvsMCI'),datos_extraidos)
        np.save(Path('C:\\Users\\User\\Documents\\6\\TFG\\datos procesados\\ADvsMCI_labels'),
etiquetas_extraidas)

```

Módulo: Autoencoders

```
import keras
```

```
from keras import layers
from keras import models
```

```

def build_autoencoder_1(capa_intermedia):
    #encoder

    encoder=models.Sequential()
    encoder.add(layers.Conv3D(32,3,1,activation='relu',input_shape=(78,94,78,1)))
    encoder.add(layers.MaxPooling3D(2))
    encoder.add(layers.BatchNormalization())
    encoder.add(layers.Conv3D(32,3,1,activation='relu'))
    encoder.add(layers.MaxPooling3D(2))

```

```

encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(64,3,1,activation='relu'))
encoder.add(layers.MaxPooling3D(2))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(64,3,1,activation='relu'))
encoder.add(layers.Flatten())
encoder.add(layers.Dense(capa_intermedia,activation='relu'))

```

```

#decoder

```

```

decoder=models.Sequential()
decoder.add(layers.Dense(18432,activation='relu'))
decoder.add(layers.Reshape((6,8,6,64), input_shape=(18432,)))
decoder.add(layers.Conv3DTranspose(64,3,1,activation='relu'))
decoder.add(layers.UpSampling3D(2))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(32,3,1,activation='relu'))
decoder.add(layers.UpSampling3D(2))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(32,3,1,activation='relu'))
decoder.add(layers.UpSampling3D(2))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(1,3,1,activation='relu'))

```

```

return encoder, decoder

```

```

def build_autoencoder_2(capa_intermedia):

```

```

#encoder

```

```

encoder =models.Sequential()
encoder.add(layers.Conv3D(64,(3,5,3),(3,1,3),activation='relu',input_shape=(78,94,78,1)))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(64,(3,5,3),(1,5,1),activation='relu'))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(128,3,3,activation='relu'))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(128,2,2,activation='relu'))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Flatten())
encoder.add(layers.Dense(capa_intermedia))

```

```

#decoder

```

```

decoder=models.Sequential()
decoder.add(layers.Dense(6144,activation='relu'))
decoder.add(layers.Reshape((4,3,4,128), input_shape=(6144,)))
decoder.add(layers.BatchNormalization())

```

```

decoder.add(layers.Conv3DTranspose(128,2,2,activation='relu'))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(64,3,3,activation='relu'))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(64,(3,5,3),(1,5,1),activation='relu'))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(1,(3,5,3),(3,1,3),activation='relu'))

return encoder, decoder

```

```
def build_autoencoder_3(capa_intermedia):
```

```
    #encoder
```

```

encoder=models.Sequential()
encoder.add(layers.Conv3D(64,3,1,activation='relu',padding='same',input_shape=(78,94,78,1)))
encoder.add(layers.Conv3D(64,(3,5,3),(3,1,3),activation='relu'))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(64,3,1,activation='relu',padding='same'))
encoder.add(layers.Conv3D(64,(3,5,3),(1,5,1),activation='relu'))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(64,3,1,activation='relu',padding='same'))
encoder.add(layers.Conv3D(64,3,3,activation='relu'))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Conv3D(64,3,1,activation='relu',padding='same'))
encoder.add(layers.Conv3D(64,2,2,activation='relu'))
encoder.add(layers.BatchNormalization())
encoder.add(layers.Flatten())
encoder.add(layers.Dense(capa_intermedia))

```

```
    #decoder
```

```

decoder=models.Sequential()
decoder.add(layers.Dense(3072,activation='relu'))
decoder.add(layers.Reshape((4,3,4,64), input_shape=(3072,)))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(64,2,2,activation='relu'))
decoder.add(layers.Conv3D(64,3,1,activation='relu',padding='same'))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(64,3,3,activation='relu'))
decoder.add(layers.Conv3D(64,3,1,activation='relu',padding='same'))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(64,(3,5,3),(1,5,1),activation='relu'))
decoder.add(layers.Conv3D(64,3,1,activation='relu',padding='same'))
decoder.add(layers.BatchNormalization())
decoder.add(layers.Conv3DTranspose(1,(3,5,3),(3,1,3),activation='relu'))

```

```
    return encoder, decoder
```

Módulo: expresión_resultados

```
from matplotlib import pyplot as plt
import numpy as np
from sklearn import metrics
import itertools

def generar_plot(historial,numero_fold,nombre_archivo):
    plt.plot(historial.history['loss'])
    plt.plot(historial.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('mean squared error')
    plt.xlabel('epoch')
    plt.legend(['train', 'val'], loc='upper right')
    plt.savefig(str(numero_fold)+nombre_archivo+'.png')
    plt.savefig(str(numero_fold)+nombre_archivo+'.pdf')

def representar_matrizconfusion(true_labels,predicted_labels,class_list,nombre_archivo):
    cm = metrics.confusion_matrix(true_labels, predicted_labels, labels=class_list)
    np.set_printoptions(precision=2)

    # Plot non-normalized confusion matrix
    plt.figure()
    generar_matrizconfusion(cm, classes=class_list, normalize=False, title='Matriz de Confusión')
    plt.savefig(nombre_archivo+'matriz-confusion.png')

    # Plot normalized confusion matrix
    plt.figure()
    generar_matrizconfusion(cm, classes=class_list, normalize=True, title='Matriz de Confusión
Normalizada')
    plt.savefig(nombre_archivo+'matriz-confusion-normalizada.png')

def generar_matrizconfusion(matriz_confusion, classes, normalize=False, title='Confusion matrix'):
    if normalize:
        matriz_confusion = matriz_confusion.astype('float') / matriz_confusion.sum(axis=1)[:, np.newaxis]
        print("Matriz de confusión normalizada")
    else:
        print('Matriz de confusión sin normalizar')

    plt.imshow(matriz_confusion, interpolation='nearest', cmap=plt.get_cmap('Blues'))
    plt.title(title)
    plt.colorbar()

    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, ['Normal','MCI','AD'], rotation=45)
    plt.yticks(tick_marks, ['Normal','MCI','AD'])

    fmt = '.3f' if normalize else 'd'
```

```
umbral = matriz_confusion.max() / 2.
```

```
for i, j in itertools.product(range(matriz_confusion.shape[0]), range(matriz_confusion.shape[1])):  
    plt.text(j, i, format(matriz_confusion[i, j], fmt), horizontalalignment="center",  
            color="white" if matriz_confusion[i, j] > umbral else "black")
```

```
plt.tight_layout()  
plt.ylabel('True label')  
plt.xlabel('Predicted label')
```

Módulo principal: clasificación-SVM-Multiclase (se modifica en función del tipo de clasificación)

```
import numpy as np  
import pandas as pd  
from pathlib import Path  
import nibabel as nib  
from cargayprocesado_datos import cargar_datos  
from cargayprocesado_datos import procesado_datos  
from cargayprocesado_datos import clasificacion_binaria  
from matplotlib import pyplot as plt  
import expresion_resultados  
#####  
  
dataPath='C:\\Users\\User\\Documents\\6\\TFG\\Imágenes\\muestras\\'  
loadPath='C:\\Users\\User\\Documents\\6\\TFG\\datos procesados\\img_data.npy'  
  
labels, cantidad_datos=cargar_datos(dataPath, 1)  
#procesado_datos(img_data, savePath)  
  
input_data= np.load(Path(loadPath))  
  
#####  
  
compresion=15  
  
input_data= input_data.reshape((362,78,94,78,1))  
  
  
from keras import layers  
from keras import models  
  
import Autoencoders  
  
encoder, decoder =Autoencoders.build_autoencoder_3(compresion)  
inp=layers.Input(input_data.shape[1:])  
code = encoder(inp)  
reconstruction = decoder(code)
```

```

autoencoder = models.Model(inp,reconstruction)
autoencoder.summary()
autoencoder.compile(loss='MeanSquaredError', optimizer='Adam')

autoencoder.save_weights('autoencoder.h5')

from sklearn import svm, metrics, multiclass
from sklearn.model_selection import KFold

kfold=KFold(n_splits=5, shuffle=False)
score=[]
predicted_labels=np.array([])
true_labels=np.array([])
numero_fold=1
for train, test in kfold.split(input_data):
    print(f'entrenamiento fold: {numero_fold}')
    historial=autoencoder.fit(x=input_data[train], y=input_data[train], batch_size=2, epochs=20,
verbose=1,validation_data=(input_data[test],input_data[test]))
    clf = multiclass.OneVsRestClassifier(svm.SVC())
    j=0
    k=0
    codigo_entrenamiento=np.empty(shape=(int(len(train)),compresion),dtype='float32')
    codigo_prueba=np.empty(shape=(int(len(test)),compresion),dtype='float32')
    labels_train=np.empty(shape=(int(len(train)),1),dtype='float32')
    labels_test=np.empty(shape=(int(len(test)),1),dtype='float32')
    for i in train:
        img_train=input_data[i]
        codigo_entrenamiento[j]=encoder.predict(img_train[None])[0]
        labels_train[j]=labels[i]
        j+=1

    print(codigo_entrenamiento.shape)
    clf.fit(codigo_entrenamiento, labels_train)

    for i in test:
        img_test=input_data[i]
        codigo_prueba[k]=encoder.predict(img_test[None])[0]
        labels_test[k]=labels[i]
        k+=1
    prediction=clf.predict(codigo_prueba)

    predicted_labels=np.append(predicted_labels,prediction)
    true_labels=np.append(true_labels,labels_test)
    print('prediccion: ' + str(prediction))
    print('etiqueta: ' + str(labels_test))
    score.append(clf.score(codigo_prueba,labels_test))
    average=np.sum(score)/len(score)

```



```
expresion_resultados.generar_plot(historial,numero_fold,'SVM-Multiclass')  
autoencoder.load_weights('autoencoder.h5')
```

```
numero_fold +=1
```

```
expresion_resultados.representar_matrizconfusion(true_labels,predicted_labels,clf.classes_, 'SVM-  
Multiclass-')
```

Bibliografía

- [1] Hippus, H. and Neundörfer, G., “The discovery of Alzheimer’s disease,” *Dialogues in clinical neuroscience*, Vol. 5, 03 2003, pp. 101–8.
- [2] Loof, A. and Schoofs, L., “Alzheimer’s Disease: Is a Dysfunctional Mevalonate Biosynthetic Pathway the Master-Inducer of Deleterious Changes in Cell Physiology?” *OBM Neurobiology*, Vol. 3, 10 2019, pp. 1–1.
- [3] Prasad, P. V., *Magnetic Resonance Imaging, Methods and Biologic Applications*, Humana Press, 1st ed., 2006.
- [4] Polzehl, J. and Tabelow, K., *Magnetic Resonance Brain Imaging, Modeling and Data Analysis Using R*, Springer, 1st ed., 2019.
- [5] Peelle, J., “Tissue class segmentation,” .
- [6] Ashburner, J., “fMRI Preprocessing,” .
- [7] Ashburner, J., “Image registration,” .
- [8] jordi Torres, *Deep Learning — Aprendizaje profundo Introducción práctica con Keras (primera parte)*, 2018.
- [9] Hammelr, B. D., “What learning rate should I use?” 2019.
- [10] Hubens, N., “Deep inside: Autoencoders,” 2018.
- [11] Hira, Z. and Gillies, D., “A Review of Feature Selection and Feature Extraction Methods Applied on Microarray Data,” *Advances in Bioinformatics*, Vol. 2015, 07 2015, pp. 1–13.
- [12] García-Gonzalo, E., Fernández-Muñiz, Z., Garcia Nieto, P. J., Sánchez, A., and Menéndez, M., “Hard-Rock Stability Analysis for Span Design in Entry-Type Excavations with Learning Classifiers,” *Materials*, Vol. 9, 06 2016, pp. 531.
- [13] Bishop, C. M., *Pattern Recognition and Machine Learning*, Springer, 2006.

- [14] Stelzmann, R. A., Norman Schnitzlein, H., and Reed Murtagh, F., "An english translation of alzheimer's 1907 paper, "über eine eigenartige erkankung der hirnrinde"," *Clinical Anatomy*, Vol. 8, No. 6, 1995, pp. 429–431.
- [15] Flint Beal M, Richardson E, and Martin J, *Enfermedad de Alzheimer y demencias afines*, Editorial Interamericana Mc Graw-Hill, México, 14th ed., 1998.
- [16] "2020 Alzheimer's disease facts and figures," *Alzheimer's & Dementia*, Vol. 16, No. 3, 2020, pp. 391–460.
- [17] Tom, S. E., Hubbard, R. A., Crane, P. K., Haneuse, S. J., Bowen, J., McCormick, W. C., McCurry, S., and Larson, E. B., "Characterization of Dementia and Alzheimer's Disease in an Older Population: Updated Incidence and Life Expectancy With and Without Dementia," *American Journal of Public Health*, Vol. 105, No. 2, 2015, pp. 408–413.
- [18] Blennow, K., Hampel, H., and Zetterberg, H., "Biomarkers in Amyloid-beta Immunotherapy Trials in Alzheimer's Disease," *Neuropsychopharmacology*, Vol. 39, No. 1, 2013, pp. 189–201.
- [19] Blennow, K., Hampel, H., and Zetterberg, H., "Biomarkers in Amyloid-beta Immunotherapy Trials in Alzheimer's Disease," *Neuropsychopharmacology*, Vol. 39, No. 1, 2013, pp. 189–201.
- [20] Blennow, K., Hampel, H., and Zetterberg, H., "Biomarkers in Amyloid-beta Immunotherapy Trials in Alzheimer's Disease," *Neuropsychopharmacology*, Vol. 39, No. 1, 2013, pp. 189–201.
- [21] B, F. L.-B. S., "An active and socially integrated lifestyle in late life might protect against dementia," .
- [22] VAPNIK, V. and CORTES, C., "Support-Vector Networks," *Machine Learning*, 1995.
- [23] Bishop, C. M., *The Nature of Statistical Learning Theory*, Springer-Verlag New York, 2nd ed., 2000.
- [24] Stoeckel, J., *Outils de classification pour l'aide au diagnostic: application à la maladie d'Alzheimer et à d'autres pathologies cérébrales*, Ph.D. thesis, École Nationale Supérieure des Mines de Paris, 2003.
- [25] Segovia Román, F., *Análisis de imágenes funcionales cerebrales mediante modelos de mezcla de gaussianas y mínimos cuadrados parciales para el diagnóstico de alteraciones neurológicas*, Ph.D. thesis, Universidad de Granada, 2011.

- [26] Ashburner and Friston, “Chapter 6 - Segmentation,” 2007.
- [27] Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, MIT Press, 2016.
- [28] BURGESS, C. J., “A Tutorial on Support Vector Machines for Pattern Recognition,” *Data Mining and Knowledge Discovery*, 1998.
- [29] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J., *Classification And Regression Trees*, Chapman & Hall/CRC, 2nd ed., 1984.
- [30] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research*, Vol. 12, 2011, pp. 2825–2830.