UNIVERSIDAD DE VALLADOLID

Escuela Técnica Superior de Ingenieros de Telecomunicación
Universidad de Valladolid

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

ESTUDIO DE ALGORITMOS DE APRENDIZAJE DINÁMICO Y ONLINE PARA APRENDIZAJE PROFUNDO APLICADO A LA DETECCIÓN DE TOS EN PACIENTES RESPIRATORIOS

———————————————————————————————

AUTOR: ALBERTO SÁNCHEZ PRIETO
TUTOR: J.P. CASASECA DE LA HIGUERA

18 de abril de 2023

TÍTULO:	ESTUDIO DE ALGORITMOS DE APRENDIZAJE DINÁMICO Y ONLINE PARA APRENDIZAJE PROFUNDO APLICADO A LA DETECCIÓN DE TOS EN PACIENTES RESPIRATORIOS
AUTOR:	ALBERTO SÁNCHEZ PRIETO
TUTOR:	J.P. CASASECA DE LA HIGUERA
DEPARTAMENTO:	TEORÍA DE LA SEÑAL Y COMUNICACIONES E INGENIERÍA TELEMÁTICA

Miembros del Tribunal

PRESIDENTE:	JAVIER AGUIAR PÉREZ
SECRETARIO:	FEDERICO SIMMROSS WATTENBERG
VOCAL:	J.P. CASASECA DE LA HIGUERA
SUPLENTE 1:	MARCOS MARTÍN FERNÁNDEZ
SUPLENTE 2:	RODRIGO DE LUIS GARCÍA
CALIFICACIÓN:	

Resumen del proyecto
En este TFG se aborda el problema de la detección de tos mediante el uso de técnicas de d*eep learning*. En particular, se propone el uso de algoritmos adaptativos para intentar solventar uno de los grandes problemas que tienen los detectores de tos; la falta de adaptación cuando se enfrentan a muestras de poblaciones con las que no han sido entrenados. Se analizarán los desafíos específicos que enfrenta la detección de tos y se expondrán diversas técnicas de aprendizaje adaptativo y *online* que pueden ser útiles para abordar estas limitaciones. Este trabajo tiene como objetivo contribuir a mejorar la detección de tos en diferentes contextos clínicos y de investigación.

Palabras clave
Aprendizaje adaptativo, Aprendizaje profundo, Detector de tos, Aprendizaje en línea, Aprendizaje activo

Keywords
Adaptive learning, Deep learning, Cough detector, Online learning, Active learning

Abstract
This Bachelor's thesis addresses the problem of cough detection through the use of *deep learning* techniques. In particular, adaptive algorithms are proposed to attempt to solve one of the major issues faced by cough detectors; the lack of adaptation when faced with samples from populations they have not been trained on. Specific challenges in cough detection will be analyzed, and various adaptive an online learning techniques that may be useful in addressing

these limitations will be presented. The aim of this work is to contribute to improving cough detection in different clinical and research contexts.

AGRADECIMIENTOS

Quiero expresar mi gratitud a todas las personas que han contribuido a la realización de este trabajo de fin de grado. En primer lugar, a mi familia, por su apoyo incondicional en todo momento, por estar ahí siempre que los he necesitado y por animarme a seguir adelante cuando las cosas se ponían difíciles.

También quiero agradecer al Laboratorio de Procesado de Imagen de la Universidad de Valladolid por su colaboración en este proyecto. Gracias por proporcionarme los medios necesarios para llevar a cabo mi investigación y por permitirme trabajar en un ambiente tan enriquecedor.

Por último, pero no menos importante, quiero agradecer al profesor Pablo Casaseca de la Higuera por su valioso apoyo y orientación durante todo el proceso. Sus consejos, sugerencias y conocimientos han sido fundamentales para el éxito de este trabajo.

Gracias de nuevo a todos los que han contribuido a hacer posible este proyecto, vuestro apoyo ha sido fundamental para su realización.

Índice general

Capítulo 1

INTRODUCCIÓN

1.1 MOTIVACIONES

La tos es un mecanismo reflejo complejo que permite mantener la función de intercambio de gases de los pulmones evitando tanto la aspiración de cuerpos extraños como liberando la vía aérea de secreciones o partículas mediante la espiración violenta. En una situación normal, la tos sirve como protector de las vías aéreas y pulmones, pero en otras circunstancias puede llegar a convertirse en un factor potencialmente dañino ya que una tos excesiva puede provocar irritaciones. Es el síntoma más común de consulta a los servicios médicos. [1]

Desde el punto de vista médico, la tos por lo general no suele ser un síntoma grave. Sin embargo, es un síntoma común en muchas enfermedades respiratorias como el asma, la bronquitis, la neuomonía o el EPOC y también en enfermedades no respiratorias como la insuficiencia cardiaca. De acuerdo a [2], las enfermedades pulmonares son responsables del 16 % de todas las muertes a nivel mundial. Aunque hay grandes avances médicos al respecto, el impacto de estas enfermedades sigue siendo significativo y se espera que continúe siéndolo durante muchos años. De hecho, se estima que en el futuro las enfermedades pulmonares representarán el 20 % de las muertes totales ocurridas en el mundo.

En los últimos años, la telemedicina ha cobrado mucha fuerza debido a la facilidad y accesibilidad que ofrece para acceder a servicios médicos y de atención de salud en línea. Esto ha llevado a una mayor investigación en la detección temprana de enfermedades, incluyendo enfermedades respiratorias.

La investigación sobre detectores de tos ha cobrado relevancia especialmente a partir de la pandemia provocada por el COVID-19 donde una detección precisa de la tos podía ayudar en la identificación temprana de posibles casos de COVID-19. La tos es un síntoma común de muchas enfermedades respiratorias que pueden ser difíciles de detectar en su etapa temprana sin la ayuda de dispositivos especializados y dichos detectores nos serían de gran ayuda para esta labor. Al identificar enfermedades de una manera prematura, las personas pueden recibir el tratamiento adecuado antes de que la enfermedad progrese y se convierta en algo más serio, esto puede ayudar a reducir el número de visitas al médico y hospitalizaciones, lo que a su vez puede reducir los costos de atención médica.

Uno de los grandes problemas que tienen este tipo de detectores es conseguir una calibración adecuada. Si el detector no está calibrado correctamente, podríamos tener diagnósticos inexactos al no adaptarse a las características de los pacientes. Dependiendo del grupo de personas a tratar, estas pueden tener patrones de tos, entornos, edades, etnias o géneros distintos, lo que puede dificultar la generalización de un detector probado con un tipo de población.

En este proyecto partiremos del sistema detector de tos que Diego Pérez implementó en [3]. En dicho trabajo se segmentaron diferentes clips de audio pertenecientes a varios pacientes consiguiendo así que tuviesen todos la misma duración (1 segundo). De este modo eliminábamos un posible sesgo a partir de la duración de los clips para posteriormente utilizar un sistema basado en *Deep learning* que hacía uso de una *Convolutional Neural Network* (CNN) cuya función era determinar si en el espectrograma de dichos segmentos aparecía un patrón de tos o no. Nuestro objetivo será la utilización de varios algoritmos adaptativos para que dicho detector no sufra cuando se enfrente a datos que no ha visto anteriormente.

1.2 OBJETIVOS

1. **Estudiar el problema de la detección de tos desde el punto de vista de la adaptación a nuevos datos**:
   Una de las principales necesidades en la detección de tos es la capacidad del modelo de adaptarse a datos que no ha visto anteriormente. La edad, el sexo, la región a la que pertenece o la presencia de afecciones respiratorias previas y otros factores afectan al modo en que una persona tose. Esto provoca la necesidad de que el modelo tenga en cuenta todas estas variaciones a la hora de enfrentarse a nuevos datos para que su desempeño no decaiga.
2. **Evaluar diferentes técnicas de aprendizaje activo en su aplicación al problema**:
   El aprendizaje activo es una técnica de aprendizaje automático que permite a un modelo seleccionar las muestras más informativas de un conjunto de datos. El objetivo será el conseguir una reducción del costo computacional a la hora de entrenar un modelo gracias a la utilización de las muestras más informativas como parte del entrenamiento.
3. **Aplicar métodos específicos de aprendizaje *online* y estudiar su desempeño**:
   El aprendizaje *online* es una técnica de aprendizaje automático que permite que el modelo se actualice continuamente a medida que se agregan nuevos datos. Esto es útil en el contexto de la detección de tos ya que pueden llegar toses nuevas (de diferentes pacientes, entornos...) a nuestro sistema y necesitamos poder adaptarnos a las mismas y detectarlas. El objetivo será estudiar el funcionamiento de este tipo de aprendizaje y si es realmente útil para nuestro problema.

1.3 FASES Y MÉTODOS

1. **Recopilación de información y lectura de artículos**: se leen y revisan diversos *papers* científicos para después definir el problema a abordar en el TFG; el estudio de algoritmos

adaptativos en *Deep Learning* para un detector de toses. Tras esto se plantea la elaboración de un proyecto para llevar a cabo los objetivos citados en la sección 1.2.

2. **Aprendizaje de Software**: se estudia y aprenden conceptos relativos al manejo de grandes volúmenes de datos y uso de redes neuronales utilizando Python. Esto incluye el manejo de librerías como `pandas`, `numPy`, `matplotlib`, `scikit-learn`, `keras`, `tensorFlow` o `theano`.
3. **Estudio sobre técnicas de Inteligencia Artificial**: se estudia y aprenden diferentes técnicas cómo el *online learning*, *active learning* y cómo implementarlas en Python.
4. **Realización de experimentos**: se realizan varias pruebas y experimentos para intentar cumplir los objetivos planteados al inicio del TFG y obtener resultados interesantes y útiles.
5. **Exposición y discusión de resultados**: se exponen y discuten los resultados obtenidos tras la realización de los experimentos.

1.4 MEDIOS NECESARIOS EMPLEADOS PARA EL DESARROLLO

Para la realización del Trabajo de Fin de Grado, utilizaremos el siguiente *hardware* y *software*:

Hardware

- PC de sobremesa con distribución Linux y las siguientes características:
  - Procesador 8xIntel®Core™i7-4790 3.60 GHz.
  - 16 GB de memoria RAM.
  - Disco duro HDD de 500 GB de capacidad.
  - GPU NVIDIA GeForce GTX 970 con 4 GB de memoria RAM.

Software

- Python [4]: Lenguaje de alto nivel con el que analizar datos y programar/desplegar algoritmos.
- LaTeX [5] : Software utilizado para la composición de este documento.

1.5 ESTRUCTURA DEL DOCUMENTO

Este documento consta de seis capítulos, incluyendo entre ellos el de introducción. El contenido de los capítulos posteriores es el siguiente:

Capítulo 2- Aprendizaje automático: en este capítulo se habla sobre cómo funciona el aprendizaje automático y cómo se encuentra dividido, profundizando sobre el *Machine Learning* y el *Deep Learning*, los métodos más utilizados en ambos casos y sus diferencias principales.

Capítulo 3- Detección de tos: en este capítulo se presenta el problema a tratar en el presente documento y un estado del arte sobre cómo se encuentra actualmente la detección de toses y el aprendizaje adaptativo.

Capítulo 4- Metodología: en este capítulo se muestra la base de datos utilizada para la realización de los experimentos, el procesado de dichos datos y una descripción sobre la red neuronal utilizada que incluye los parámetros que utilizaremos para entrenar dicha red.

Capítulo 5- Resultados y discusión: en este capítulo se explican los experimentos realizados y los resultados obtenidos

Capítulo 6- Conclusiones y líneas futuras: este capítulo recoge las principales conclusiones obtenidas en la elaboración de este documento y se proponen posibles caminos a seguir en un futuro.

Capítulo 2

APRENDIZAJE AUTOMÁTICO

En este capítulo vamos a explicar lo que es la Inteligencia Artificial (IA), el *Machine Learning* y el *Deep Learning*

2.1 INTELIGENCIA ARTIFICIAL (IA)

La Inteligencia Artificial es una disciplina que intenta replicar y desarrollar la inteligencia a través del uso de computadoras. Comprende una gran cantidad de campos como el aprendizaje, clasificación, demostración de teoremas matemáticos o diagnóstico de enfermedades. Aunque no tiene una definición clara, es una nueva forma de resolver problemas que intenta integrar el conocimiento en tales sistemas. De igual manera se puede considerar a la IA como la capacidad de las máquinas para hacer uso de algoritmos, aprender a través de datos y utilizar lo aprendido para imitar la toma de decisiones humana. [6]

Los autores Russell y Norvig diferencian varios tipos de sistemas de inteligencia artificial: [7]

- **Sistemas que piensan como humanos**: estos sistemas tratan de emular el pensamiento humano y buscan la automatización de actividades que vinculamos con procesos de pensamiento humano, toma de decisiones, resolución de problemas y aprendizaje. Un ejemplo de estos sistemas son las redes neuronales artificiales. [8]
- **Sistemas que actúan como humanos**: estos sistemas tratan de actuar como humanos imitando nuestro comportamiento. Uno de los objetivos de estos sistemas es lograr que los computadores realicen tareas que, de momento, los humanos hacen mejor. [9]
- **Sistemas que piensan racionalmente**: es decir, con lógica (idealmente), tratan de imitar el pensamiento racional del ser humano; por ejemplo, los sistemas expertos (el estudio de los cálculos que hacen posible percibir, razonar y actuar). [10] [11]
- **Sistemas que actúan racionalmente**: tratan de emular de forma racional el comportamiento humano; por ejemplo los agentes inteligentes, que están relacionados con conductas inteligentes en dispositivos tecnológicos. [12]

Podemos encontrar dos escuelas de pensamiento sobre la Inteligencia Artificial:

- **Inteligencia artificial convencional**: está basada en la realización de análisis y estadísticas del comportamiento que los seres humanos adoptan para resolver problemas, podemos dividirla de la siguiente manera:
  - **Razonamiento basado en casos (CBR)**: es una herramienta que permite resolver problemas nuevos a partir de problemas viejos documentados dentro de una base de datos. Este tipo de razonamiento se basa en analogías, buscando similitudes entre nuestro problema y problemas ya resueltos anteriormente para aplicar soluciones parecidas o idénticas a las que nos permitieron solucionar dichos problemas. [13]
  - **Sistemas Expertos**: estos sistemas se encargan de emular la forma de actuar que tendría una persona experta en un determinado campo. Son capaces de utilizar la lógica mediante procesos deductivos o inductivos para obtener resultados y siempre son capaces de mostrar los pasos lógicos que se han seguido para justificar las decisiones tomadas. [11]
  - **Redes Bayesianas**: una red bayesiana es un grafo acíclico dirigido en el que cada nodo se encarga de representar una variable aleatoria que tiene asociada una función de probabilidad condicional. La estructura de la red bayesiana provee información sobre las relaciones de dependencia e independencia condicional existentes entre las variables. Estas relaciones simplifican la representación de la función de probabilidad conjunta como el producto de las funciones de probabilidad condicional de cada variable. Las redes bayesianas proporcionan una representación gráfica para un conjunto de variables aleatorias y para las relaciones existentes entre ellas. La estructura de la red permite especificar la función de probabilidad conjunta de estas variables como el producto de funciones de probabilidad condicionadas, por lo general, más sencillas. Este enfoque representa una buena estrategia para hacer frente a problemas relacionados con la incertidumbre, donde las conclusiones no pueden ser construidas sólo a partir de un conocimiento previo sobre el problema. [14]
- **Inteligencia artificial computacional**: está centrada en el diseño de sistemas informáticos inteligentes que tratan de imitan el razonamiento humano para resolver problemas complejos, no rechaza los métodos estadísticos, pero muy a menudo aporta una vista complementaria. En este campo de investigación podemos encontrar el desarrollo de los sistemas inteligentes como los siguientes:
  - **Sistemas difusos**: estos sistemas se encargan de estudiar objetos que pertenecen a clases con fronteras indefinidas, provocando esto que las transiciones entre los conjuntos sean progresivas y no abruptas. Están basados en la utilización de la `"difusividad"`, que describe el grado de pertenencia que tiene un miembro de un conjunto a ese conjunto. [15]
  - **Redes neuronales artificiales**: consisten en un conjunto de neuronas artificiales que se encuentran conectadas entre sí mediante capas y se transmiten información mediante señales. Estas señales se transmiten desde la entrada hasta generar una salida y su objetivo es el de simular el sistema nervioso humano.

2.2 MACHINE LEARNING

Este es un campo de la inteligencia artificial que utiliza algoritmos para conseguir identificar patrones en grandes cantidades de datos y obtener predicciones.

2.2.1 TIPOS DE APRENDIZAJE

Existen 3 grupos dentro del *Machine Learning*, estos grupos se diferencian entre sí dependiendo del tipo de aprendizaje que use nuestro modelo, tenemos los siguientes:

2.2.1.1 APRENDIZAJE SUPERVISADO

Nuestro modelo se entrenará con una serie de ejemplos de la salida que esperamos encontrar para cierta entrada y, a partir de estos ejemplos obtendrá por inferencia una función o relación que nos permita obtener una salida para cada entrada. [16]
Para este tipo de aprendizaje dividiremos nuestros datos en 3 conjuntos:

- **Conjunto de entrenamiento**: en este conjunto tendremos los datos que serán utilizados para entrenar nuestro sistema. Este conjunto de datos es esencial para el proceso de aprendizaje ya que el modelo ajustará sus parámetros en función de los patrones y relaciones en los datos de entrenamiento. La calidad del entrenamiento y la precisión del modelo dependen en gran medida de conseguir una selección y representatividad adecuada del conjunto de entrenamiento de datos. Por lo tanto, es importante garantizar una distribución balanceada y suficiente cantidad de datos en el conjunto de entrenamiento para evitar sesgos y mejorar la capacidad que nuestro modelo tendrá para generalizar cuando se enfrente a nuevos datos. Llamaremos x_i a los datos de entrada utilizados para entrenar, siendo y_i su salida correspondiente
- **Conjunto de validación**: este conjunto forma parte del conjunto de entrenamiento y es utilizado para evaluar el rendimiento del modelo durante dicho entrenamiento. Gracias a esta evaluación, evitamos que nuestro sistema sufra un sobreajuste (producido cuando un modelo se ajusta demasiado a los patrones y características específicas de los datos de entrenamiento, provocando así que nuestro sistema no sea capaz de generalizar a datos no vistos previamente) ya que los hiperparámetros de nuestro sistema se irán ajustando dependiendo de los resultados obtenidos en este conjunto, consiguiendo así un modelo capaz de asimilar y comprender los patrones y características importantes del modelo pero sin llegar a causar sobreajuste.
- **Conjunto de testeo**: en este conjunto tendremos los datos que serán utilizados para evaluar el rendimiento de nuestro sistema una vez que ha sido entrenado. El objetivo de este conjunto es evaluar nuestro sistema con datos que no ha visto anteriormente para cuantificar de un modo objetivo su desempeño. Llamaremos x_i a los datos utilizados para testear, siendo $\hat{y}_i$ la salida predicha por nuestro sistema

Dentro del aprendizaje supervisado, podemos encontrarnos con 2 escenarios dependiendo del problema que querramos resolver:

2.2.1.1.1. Clasificación : en un problema de clasificación, nuestro modelo consta de 2 o más salidas donde cada salida determina una clase entre la cuál podemos clasificar una información de entrada (por ejemplo clasificar imágenes entre imágenes de perros o imágenes de gatos). Podemos dividir el proceso por el cuál funcionan los algoritmos de clasificación en 3 fases:

1. **Preprocesado**: normalmente los datos con los que partimos suelen tener errores, campos vacíos, aparición de ruido, etc... Esto puede suponer un gran problema pues cuanto mayor sea la calidad de nuestros datos, mejor desempeño tendrá nuestro sistema, las principales técnicas utilizadas para solucionar estos problemas son:
   - **Data cleaning**: realizamos una limpieza de datos eliminando los *outliers* (datos con valores que se encuentran muy distantes del resto de observaciones para esa misma variable), resolvemos las inconsistencias entre datos dejando todos con el mismo formato y tomando una decisión sobre cómo trabajar con los valores *NULL* (valores perdidos o que por un error no han sido insertados), pudiendo transformarlos en 0, ignorarlos o aplicar transformaciones para convertir dichos valores en la media que tiene esa variable obtenida a partir del resto de muestras. Esto son ejemplos pero este proceso es subjetivo y dependiendo de la naturaleza de nuestros datos utilizaremos unas técnicas u otras. [17]
   - **Data integration**: consiste en la unificación de datos provenientes de diferentes fuentes en un solo formato para poder procesar y visualizar dichos datos de manera sencilla. [18]
   - **Data transformation**: en muchas ocasiones y, dependiendo del sistema, tendremos que transformar nuestros datos normalizándolos, aplicándoles transformaciones logarítmicas o modificando su tiempo de muestreo (podemos tener un tiempo de muestreo de 15 minutos pero un sistema que funcione con muestreos de 1 hora). Este módulo se encarga de esa tarea garantizando que los datos sean precisos y confiables. [18]
   - **Data reduction**: proceso con el cuál se simplifica y comprimen grandes conjuntos de datos para conseguir que su procesado y almacenamiento sea más sencillo y eficiente. Se suelen utilizar técnicas como la eliminación de datos duplicados o la selección de un subconjunto representativo de dichos datos, disminuyendo así el número de datos a procesar. [19]
2. **Extractor de características**: es un proceso de reducción donde partimos de unas variables iniciales sin procesar y las conseguimos reducir hacia características más manejables para ser procesadas por nuestro sistema. Existen diversas técnicas para la extracción de características en imágenes y audio como:
   - **Colour Histograms**: esta técnica se basa en observar la cantidad de un determinado color que está presente en un histograma generado de la imagen a procesar. Para ello es necesario segmentar la imagen antes de procesarla. Consiste en comparar el

histograma de la imagen seleccionada con histogramas pertenecientes a una base de datos para después compararlos. Su principal ventaja es que es invariable a la rotación, traslación y escalado de una imagen y está diseñada para ignorar los píxeles de fondo de la imagen aunque si estos píxeles tienen colores similares al objeto a analizar pueden producirse errores. [20]

- **Fourier Coefficients**: esta técnica consiste en la transformación de los bordes de la imagen al dominio complejo para posteriormente aplicar una transformada discreta de Fourier (DFT) a esos valores y compararlos con resultados almacenados en bases de datos. Es una técnica sencilla y fácil de implementar pero es muy sensible al ruido y a la posición del objeto siendo muy ineficiente al aplicar rotaciones o traslaciones en nuestra imagen. [20]

- **Principal Componenet Analysis**: esta técnica ha sido desarrollada para reducir la cantidad de datos necesarios para reconocer objetos. Consiste en representar cada objeto como un punto en un espacio n-dimensional donde cada eje representa una de las características interesantes para el sistema (estos ejes son reconocidos tras entrenar a nuestro sistema previamente). Al finalizar esta transformación podemos representar todo el sistema como una matriz que contiene todos los datos. Esta técnica es rápida y fiable pero cada vez que añadimos un nuevo conjunto de objetos debemos volver a generar la matriz de identificación. [20]

- **Short-time Fourier Transform (STFT)**: es una técnica de análisis de señales que obtiene la representación del dominio de la frecuencia de señales pertenecientes al dominio del tiempo. Se divide la señal de audio en segmentos cortos de tiempo (ventanas) y se aplica la Transformada de Fourier a cada segmento obteniendo así la representación en el dominio de la frecuencia para cada segmento. [21]
  Su expresión es la siguiente:

$$X(m,\omega) = \sum_{n=-\infty}^{\infty} x(n)w(n-m)e^{-j\omega n} \tag{2.1}$$

  donde $X(m,\omega)$ es la STFT, $x(n)$ es la señal de entrada, $w(n-m)$ es la ventana que se aplica al segmento de tiempo centrado en m, y ω es la frecuencia angular. La STFT se compone de los siguientes pasos:

  1. Se divide la señal de audio en segmentos de tiempo cortos y superpuestos. Cada segmento es multiplicado por una función de ventana.
  2. Se aplica la transformada de Fourier a cada ventana para obtener su representación en el dominio de la frecuencia.
  3. Se combinan las diferentes representaciones en el dominio de la frecuencia de cada ventana para formar una matriz de espectrograma. Con este espectrograma podemos observar cómo cambian las frecuencias a lo largo del tiempo, pudiendo extraer de aquí características relevantes

- **Mel-Frequency Cepstral Coefficients** (MFCC): Los Coeficientes Cepstrales de Frecuencia de Mel (MFCC) son un conjunto de características muy utilizadas en tareas

de reconocimiento de voz. [22]
La transformación de Mel consta de lo siguiente:

1. Dividimos la señal de audio en segmentos de tiempo cortos (podemos utilizar segmentos de milisegundos). Su expresión:

$$x(n) = \sum_{k=0}^{L-1} w(k) x_s(n - kR) \tag{2.2}$$

donde $x(n)$ es la señal de audio, $x_s(n)$ es el segmento de tiempo solapado, $w(k)$ es la ventana de tiempo, L es el tamaño de la ventana y R es el tamaño del desplazamiento.

2. Aplicamos la STFT a cada segmento de tiempo obteniendo su representación en el dominio de la frecuencia.

3. Pasamos la escala lineal de frecuencias a la escala Mel utilizando la siguiente expresión:

$$(f) = 1125 \ln\left(1 + \frac{f}{700}\right) \tag{2.3}$$

donde f es la frecuencia en Hz y $M(f)$ es la frecuencia Mel correspondiente.

4. Aplicar un banco de filtros triangulares en la escala Mel a la potencia espectral.

5. Aplicamos una transformación a los espectros de frecuencia Mel para obtener una serie de coeficientes coseno llamados los Coeficientes Cepstrales de Frecuencia de Mel (MFCC). Esta transformación viene dada por:

$$MFCC_i = \sum_{j=0}^{N-1} log_{Mel}(j) \cos\left[\frac{\pi i}{N}\left(j + \frac{1}{2}\right)\right] \tag{2.4}$$

donde $MFCC_i$ es el i-ésimo coeficiente cepstral de frecuencia de Mel, N es el número de filtros en el banco de filtros y $log_{Mel}(j)$ es el logaritmo de la potencia espectral en el j-ésimo filtro en la escala Mel.

6. Los primeros MFCC son los más informativos y relevantes y serán los introducidos al sistema.

Los MFCC proporcionan una buena representación de la información de frecuencia en la señal de audio que nos permite identificar diferentes patrones y características del audio.

3. **Clasificación**: tras la obtención de las características más relevantes del sistema, la parte final es la clasificación, siendo los algoritmos más importantes para esta tarea los siguientes:

- **Regresión logística**: algoritmo utilizado para predecir cual es el resultado de una variable categórica (la salida del modelo) en función de unas variables predictoras (la entrada del modelo). Un escenario muy común es el de calcular la probabilidad de

una salida binaria (solo puede tomar 0 o 1 como valor) a partir de una entrada x. [23] [24]
Para ello podríamos utilizar la expresión:

$$\frac{\Phi(x)}{1-\Phi(x)} = e^{\beta_0+\beta_1 x} \tag{2.5}$$

Podemos ver su gráfica en la figura 2.1

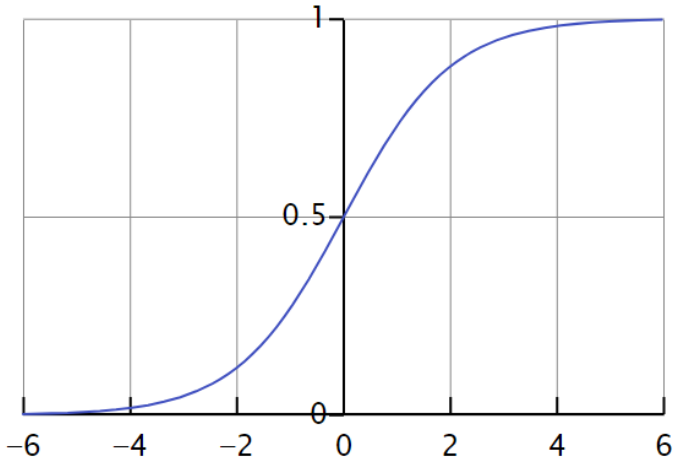


FIGURA 2.1: Función utilizada para la regresión logística con $\beta 0 + \beta 1x + e$ como eje horizontal y $\Phi(x)$ como eje vertical

- **Árbol de clasificación**: en este algortimo tenemos varias etiquetas de clase representadas por las `"hojas"` y la lógica que el árbol utiliza para determinar que etiqueta es la correcta para cada datos de entrada representadas por `"ramas"`. El algoritmo se encarga de generar las `"ramas"` oportunas a partir de los datos de entrenamiento que le mostramos. [24]
  Podemos ver una representación gráfica de como funciona en la figura 2.2:

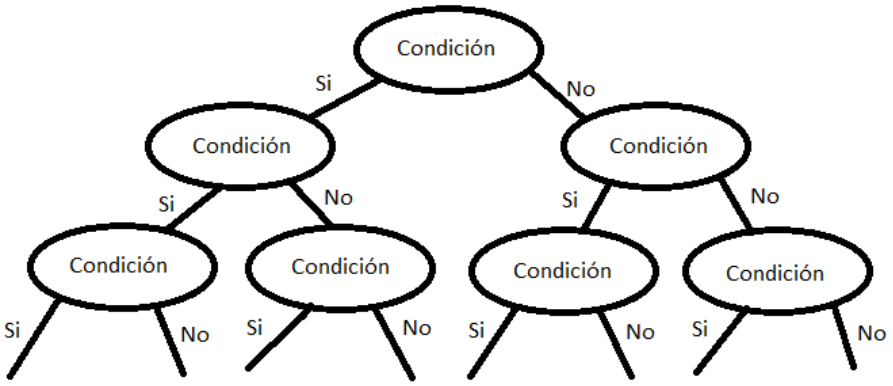


FIGURA 2.2: Representación del funcionamiento de un árbol de clasificación

- **K vecinos más próximos**: este clasificador está basado en la proximidad que existe entre sus datos para hacer predicciones individuales sobre la agrupación de dichos datos. Es muy utilizdo para problemas de clusterización o clasificación donde buscamos identificar diferentes grupos en nuestros datos. Utiliza el `"Voto mayoritario"` que consiste en asignar etiquetas a los datos en función de la etiqueta que aparezca mayoritariamente en los datos de alrededor. Un ejemplo visual de cómo funciona este algoritmo lo podemos encontrar en la figura 2.3 dónde tenemos una muestra que debe ser etiquetada y que lo será en función de las muestras que se encuentren más próximas a la misma.

FIGURA 2.3: Representación de K vecinos más próximos paso a paso. [25]

La cantidad de muestras o el radio de acción que queremos que afecten a nuestra muestra a clasificar pueden ser modificados. La medición de la distancias entre muestras (para saber si se encuentran dentro del radio de acción) puede ser ambigua ya que tenemos diferentes métricas como:

- **Distancia Euclidiana**: esta es la medida más utilizada y consiste en medir una línea recta entre 2 puntos pero tiene la limitación de solo poder usarse con

vectores reales. [26]
Su expresión es la siguiente:

$$Distancia = \sqrt{\sum_{n=1}^{\infty}(y_i - x_i)^2} \tag{2.6}$$

- **Distancia Manhattan**: esta medida se encarga de medir el valor absoluto que hay entre 2 puntos. [26]
  Su expresión es la siguiente:

$$Distancia = \sum_{n=1}^{\infty}|y_i - x_i| \tag{2.7}$$

- **Distancia Hamming**: esta medida se utiliza para comparar vectores binarios, haciendo que la distancia entre 2 vectores sea mayor cada vez que se compara un término de ambos vectores y este es diferente. Para entenderlo mejor pongamos un ejemplo: Tenemos 2 vectores binarios a los que llamaremos vector 1 y vector 2, dichos vectores tienen los siguientes valores:

  Vector 1: 0 0 1 1 1 0 1 1
  Vector 2: 0 0 1 0 1 1 1 0

  La distancia de Hamming entre ambos vectores sería de 2 pues tanto el cuarto como el último elemento de ambos vectores no coincide. [26]

Este algoritmo es fácil de implementar en cualquier sistema, tiene una gran capacidad de adaptación pues es capaz de modificarse conforme va recibiendo datos nuevos y no necesita una gran caracterización de parámetros. Por otro lado tiene el problema de no escalar bien pues conforme va creciendo nuestro sistema, este algoritmo necesita más datos (los guarda todos en memoria) generando así mayores costes y tiempo de computación

- **Support Vector Machine (SVM)**: este método de clasificación surgió como clasificador binario pero actualmente tiene un uso mucho más amplio. Esta basado en el uso de hiperplanos para conseguir separar 2 o más regiones el espacio en el que estemos trabajando, dependiendo de la separación en la cual se encuentre cada punto será etiquetado de un modo u otro. En la imagen 2.4 podemos observar como funciona este método en un plano 2d (se puede generar para planos de n dimensiones). Tenemos 2 grupos diferenciados y el algoritmo se encarga de separar dichos grupos con un plano, esto provoca que cualquier dato que introduzcamos y se encuentre por debajo del plano, será catalogado como `"Grupo Rojo"` y si el dato se encuentra por encima del plano, será catalogado como `"Grupo Azul"` [27]

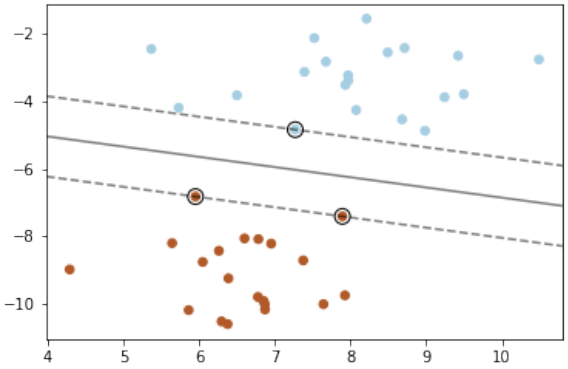


FIGURA 2.4: Ejemplo de implementacion de SVM

- **Redes Neuronales Artificiales**: este método de clasificación está basado en la replicación del sistema nervioso humano. Cómo podemos ver en la figura 2.5, las redes neuronales artificiales pueden dividirse en las siguientes capas:

  - **Capa de entrada**: está formada por las neuronas que reciben información del exterior y que posteriormente propagan a las capas ocultas. Tiene un número de neuronas igual al número de parámetros que vamos a introducir en nuestra red.

  - **Capas ocultas**: contienen valores no observables. En esta capa (pueden ser una o varias) se realiza un procesamiento de los datos obtenidos a la entrada que puede ser lineal o no.

  - **Capa de salida**: en esta capa encontramos los valores resultantes a los datos introducidos en la capa de entrada.

  El objetivo de las redes neuronales artificiales es emular el comportamiento del cerebro humano y buscan minimizar el valor de una función de pérdida que evalúa el rendimiento de la red. Normalmente este tipo de sistemas aprenden, se forman a sí mismos y tienen un gran desempeño. [28].
  Profundizaremos más sobre las mismas en el apartado 2.3.1

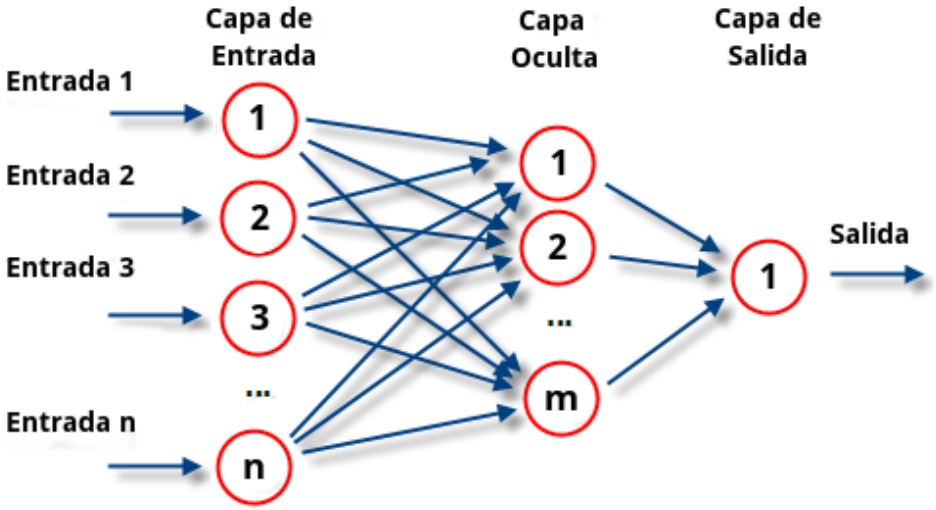
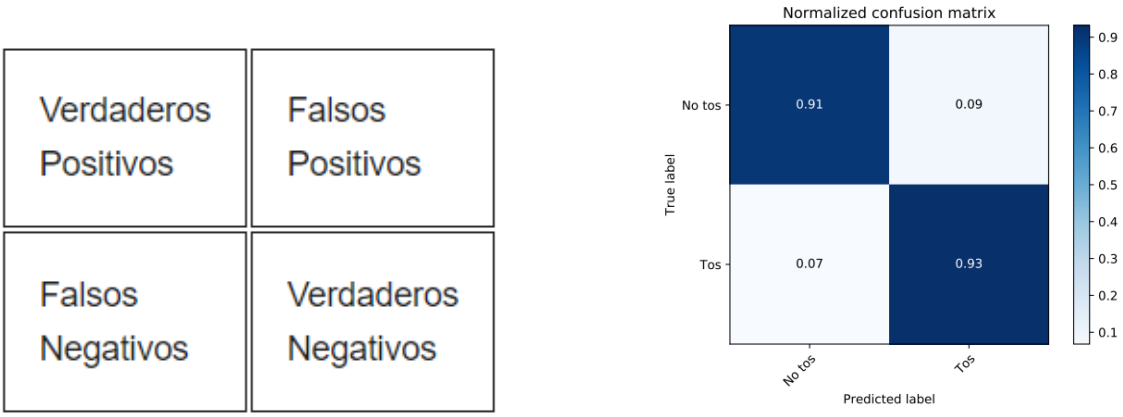


FIGURA 2.5: Estructura de una red neuronal artificial. [29]

2.2.1.1.1.1 Métricas de evaluación para un sistema de clasificación supervisado: la principal característica que tiene que tener un modelo de clasificación para ser considerado óptimo es la capacidad de generalizar las clasificaciones sobre datos no vistos anteriormente a partir de los datos de entrenamiento. Como veremos a continuación, existen diversas métricas para evaluar el rendimiento. Es importante entender el funcionamiento de todas ellas y no utilizar sólo una como guía puesto que podemos provocar que el modelo no aprenda si no que memorice y no pueda generalizar con datos que no haya visto previamente. [30]

- **Matriz de confusión normalizada** La matriz de confusión normalizada nos permite observar el porcentaje de predicciones correctas e incorrectas que hace nuestrio modelo de una manera muy visual apareciendo el porcentaje normalizado en la matriz tal como podemos observar en la figura 2.6. [31]
  Las matrices de confusión se dividen en:

  - **Verdaderos positivos (True Positive)**: el clasificador predice una clase y la predicción es correcta.
  - **Falsos negativos (False Negative)**: el clasificador predice una clase y la predicción no es correcta.
  - **Falsos positivos (False Positive)**: el clasificador predice que no era una clase y la predicción es incorrecta (también se conocen como `"falsas alarmas"`)
  - **Verdaderos negativos (True Negative)**: el clasificador predice que no era una clase y la predicción es correcta.



(a) Estructura de matriz de confusión

(b) Ejemplo de matriz de confusión

FIGURA 2.6: Matriz de confusión

De esta matriz de confusión podemos obtener diversas métricas, destacando entre ellas las siguientes:

- **Exactitud (*Accuracy*)**: este valor nos indica la cercanía que existe entre el resultado de una medición y el valor verdadero que esta tiene. Es la proporción de resultados verdaderos entre el número total de casos examinados. [31]
  Su expresión es la siguiente:

  $$Exact = \frac{VerdaderosPositivos + VerdaderosNegativos}{VerdaderosPositivos + FalsosPositivos + FalsosNegativos + VerdaderosNegativos}$$

- **Precisión (*Precision*)**: este valor nos indica la proporción que existe entre los Verdaderos Positivos y el total de positivos de nuestro sistema. [31]
  Su expresión es la siguiente:

  $$Prec = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosPositivos}$$

- **Sensibilidad (*Sensitivity*)**: este valor nos indica la capacidad que tiene nuestro sistema para discriminar casos positivos respecto a los casos negativos. Es la proporción de casos positivos que fueron bien identificados por nuestro sistema. [31]
  Su expresión es la siguiente:

  $$Sens = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosNegativos}$$

- **Especifidad (*Specifity*)**: este valor nos indica la capacidad que tiene nuestro sistema para discriminar casos negativos respecto a los casos positivos. Es la proporción de

casos negativos que fueron bien identificados por nuestro sistema. [31]
Su expresión es la siguiente:

$$Espec = \frac{VerdaderosNegativos}{VerdaderosNegativos+FalsosPositivos}$$

- ***Positive Predictive Value (PPV)***: este valor nos indica la probabilidad de que tengamos un Verdadero Positivo cuando predecimos un positivo. [31]
- ***Negative Predictive Value (NPV)***: este valor nos indica la probabilidad de que tengamos un Verdadero Negativo cuando predecimos un negativo. [31]
- **F1 Score**: este valor nos permite unir tanto la Precisión como la Sensibilidad en una expresión. Es muy utilizada cuando el número de elementos de cada clase es muy dispar y por lo tanto la distribución de dichas clases es desigual. [31]
  Su expresión es la siguiente:

  $$F1Score = \frac{2*Precision*Sensibilidad}{Precision+Sensibilidad}$$

- **Curva ROC**: la curva ROC es una representación gráfica de la sensibilidad frente a la especificidad para un sistema clasificador binario (fig 2.7), cuanto mayor sea el área por debajo de dicha curva, mejor será nuestro sistema (un sistema perfecto tendría un área de 1). [32]
  Existen varios tipos tipos de curvas ROC dependiendo de la métrica a utilizar (también se puede hacer una ponderación de todas estas métricas) las cuales son:

  - **Macro**: se calcula la métrica para cada clase y se toma el promedio no ponderado. [33]
  - **Micro**: se calcula la métrica de forma global mediante el recuento del total de verdaderos positivos, falsos negativos y falsos positivos (independientemente de las clases). [33]
  - **Ponderado**: se calcula la métrica para cada clase y se toma el promedio ponderado en función del número de muestras por clase. [33]

  aunque cada método de promedio tiene sus ventajas, una consideración común al seleccionar el método adecuado es el desequilibrio de clases. Si las clases tienen números de muestras diferentes, podría ser más informativo usar un promedio de macros, donde las clases minoritarias reciben la misma ponderación que las clases mayoritarias.
  Como norma general podemos determinar como de bueno es nuestro sistema en función del área por debajo de la curva ROC del siguiente modo:

  - **[0.5]**: Es como lanzar una moneda.

- **[0.5, 0.6)**: Test malo
- **[0.6, 0.75)**: Test regular
- **[0.75, 0.9)**: Test bueno
- **[0.9, 0.97)**: Test muy bueno
- **[0.97, 1)**: Test excelente

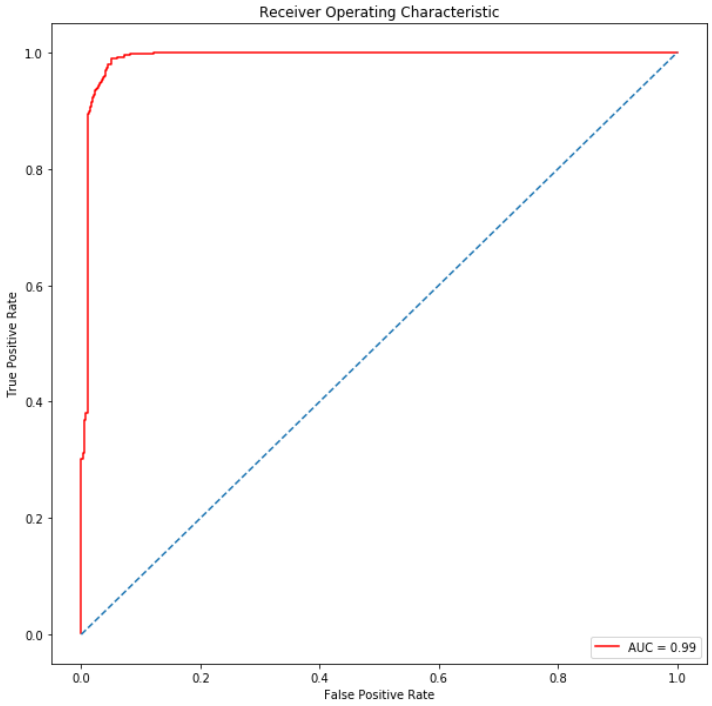


FIGURA 2.7: Ejemplo de curva ROC casi perfecta (AUC = 0.99)

- **Validación cruzada**: este método nos garantiza que los resultados obtenidos son independientes de la partición que hayamos creado para separar nuestros datos en datos de entrenamiento y datos de test. Su funcionamiento consiste en calcular la media entre las evaluaciones (podemos evaluar con una matriz de confusión, por ejemplo) realizadas de las diferentes particiones. [34]
  Existen 3 tipos:

  - **Validación cruzada de K iteraciones**: Para este método dividimos nuestros datos en K subconjuntos. Utilizaremos uno de estos subconjuntos como nuestros datos de pruebas y los K-1 restantes serán nuestros datos de entrenamiento. Ahora repetiremos este proceso K veces utilizando cada uno de los posibles datos de prueba y finalmente calcularemos la media aritmética entre todas las evaluaciones (realizamos una evaluación por cada subconjunto). Con este método podemos ser muy precisos pues no está influenciado por los datos que elegimos como prueba y test ya que

utilizamos todas las combinaciones posibles y calculamos la media de los resultados pero a cambio es costoso computacionalmente. Podemos ver un ejemplo del mismo en la figura 2.8. [35]

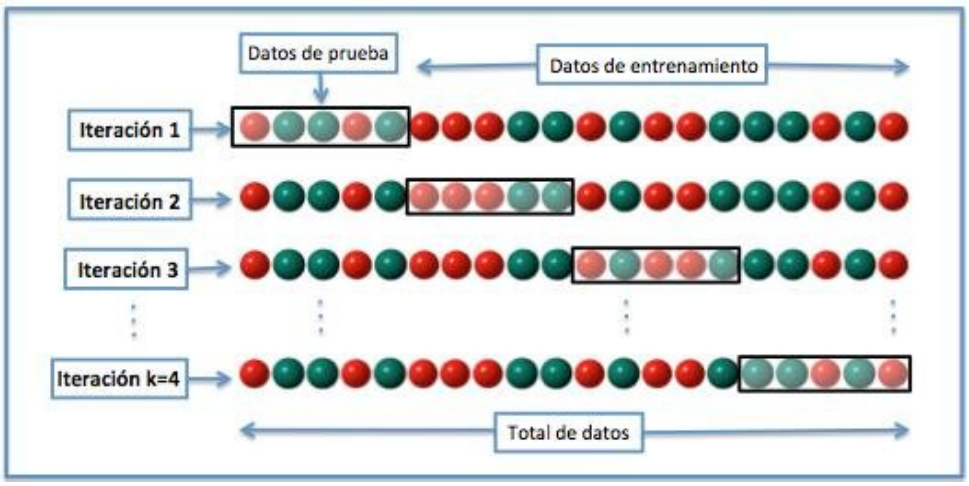


FIGURA 2.8: Ejemplo de Validación cruzada de K iteraciones (K=4). [36]

- **Validación cruzada aleatoria**: para este método dividimos nuestros datos de manera aleatoria en subconjuntos de prueba y de test. Repetiremos este proceso k veces realizando la evaluación correspondiente a cada subconjunto y calcularemos la media aritmética del total de evaluaciones. Este método es útil pues no tenemos un máximo de iteraciones como sí ocurre en la Validación cruzada de K iteraciones pero tiene la desventaja de que hay datos que pueden evaluarse en varias ocasiones y otros que pueden ser no evaluados al ser escogidos todos ellos de manera aleatoria. Podemos ver un ejemplo del mismo en la figura 2.9. [35]

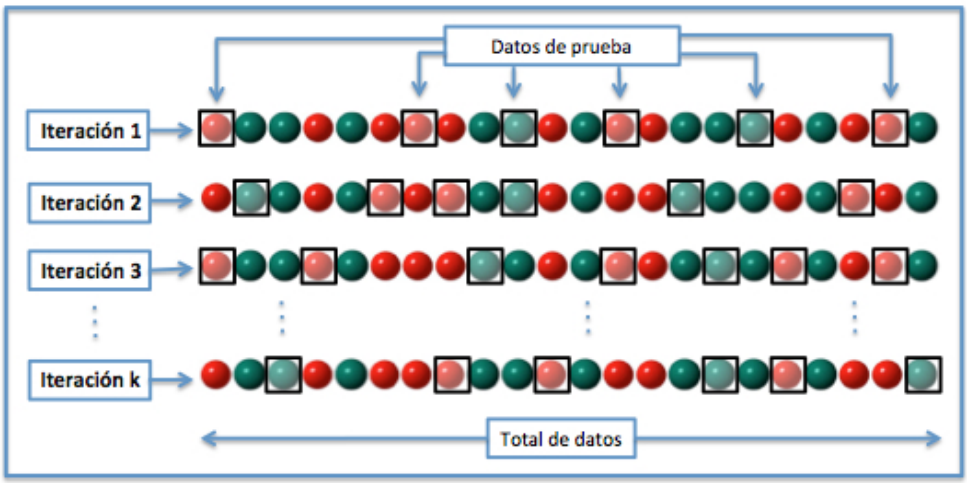


FIGURA 2.9: Ejemplo de Validación cruzada aleatoria. [36]

- **Validación cruzada dejando uno fuera**: para este método dividimos nuestros datos creando un subconjunto de test con solo un dato y subconjunto de entrenamiento con todos los datos restantes. Este proceso será repetido N veces (N es igual al número de datos totales que tengamos), se evaluará cada iteración y posteriormente realizaremos la media aritmética de estas evaluaciones. Suele utilizarse cuando se dispone de pocos datos ya que permite maximizar el uso de datos disponibles para realizar la evaluación pero a su vez se asume un coste computacional muy grande. Podemos ver un ejemplo del mismo en la figura 2.10. [37]

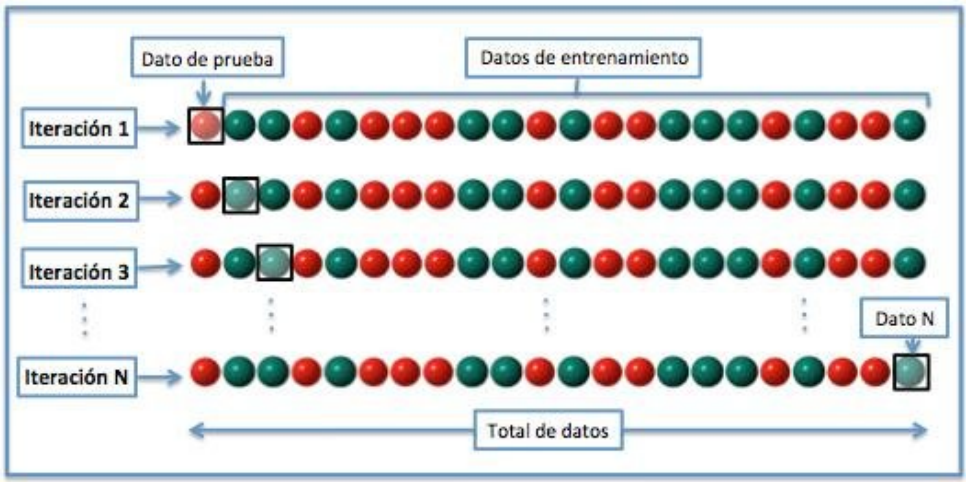


FIGURA 2.10: Ejemplo de Validación cruzada dejando uno fuera. [36]

2.2.1.1.2. Regresión: en un problema de regresión, entrenamos nuestro modelo para obtener una salida con un valor continuo a partir de unos datos de entrada (por ejemplo predecir el valor de mercado de una casa teniendo en cuenta su ubicación, metros cuadrados, año de construcción...). Los algoritmos más importantes son los siguientes:

- **Regresión lineal**: algoritmo utilizado para obtener una relación lineal entre una entrada dada y la salida que esta produce (la entrada sería una variable independiente y la salida dependiente). Es un modelo muy simple pero muy útil para determinados escenarios dónde nuestro problema no es demasiado complejo puesto que este algoritmo es ligero computacionalmente hablando. [23]
  Podemos ver un ejemplo en la figura 2.11

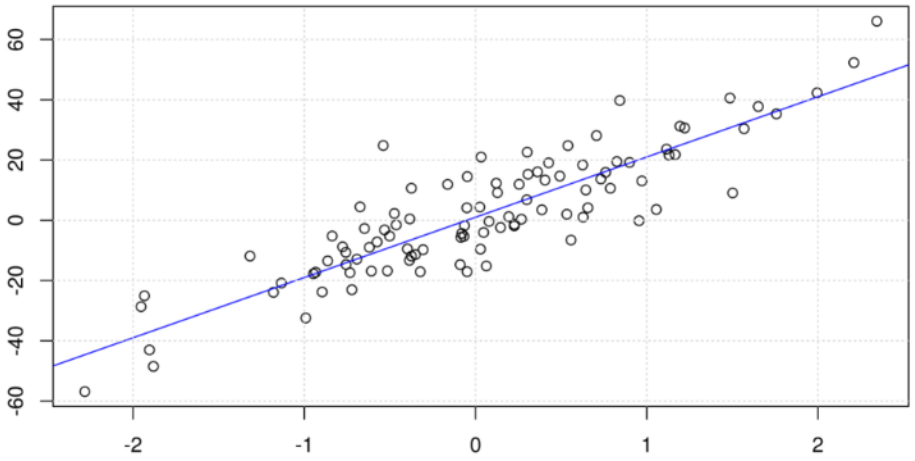


FIGURA 2.11: Representación de una regresión lineal

- **Regresión polinomial**: algoritmo utilizado para obtener una relación entre unos datos de entrada y su salida pero sin la necesidad de ser lineal, podemos utilizar ecuaciones de distintos grados para realizar nuestra regresión obteniendo así diferentes curvaturas y aproximarnos más a la solución. Podemos ver un ejemplo en la figura 2.12

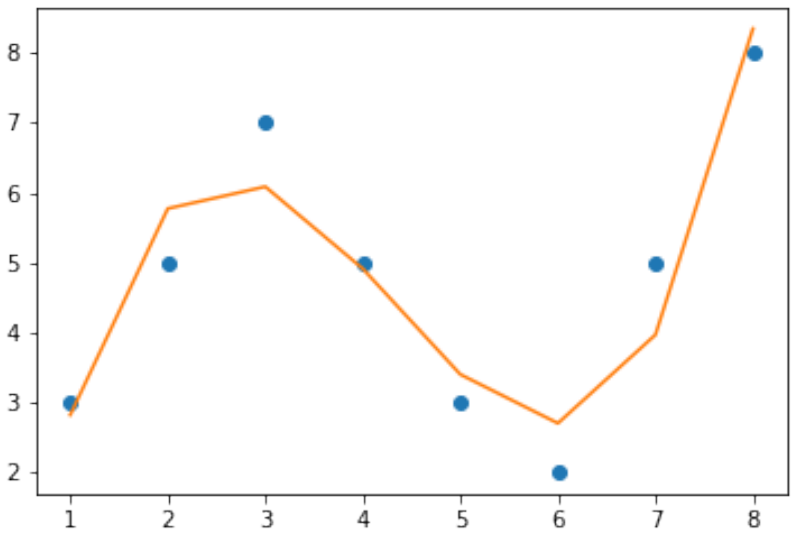


FIGURA 2.12: Representación de una regresión polinomial

2.2.1.2 APRENDIZAJE NO SUPERVISADO

Método de aprendizaje donde basándose en unas observaciones nuestro sistema es capaz de ajustarse a las mismas y obtener una salida a partir de inferencia sin necesidad de conocimiento previo. No puede predecir o clasificar datos pero sí que nos permite entender la estructura de los mismos. Su principal uso es el de agrupar los datos de entrada según los patrones, características similares que encuentre, etc... Este tipo de aprendizaje es muy usado en el mundo real pues como se desconoce cuales deberían de ser los resultados podemos utilizarlo para muchos problemas donde no dispongamos de los mismos. [38] [39]
Se puede dividir en los siguientes:

- **Agrupamiento**: consiste en la búsqueda de estructuras o tipos de patrones que se van repitiendo en una estructura de datos formando agrupaciones a partir de ellos.
- **Reducción de datos**: consiste en la reducción del conjunto de datos original a otro conjunto de menor dimensión identificando para ello las variables más irrelevantes (y eliminándolas). Esto nos permite mejorar la carga computacional del modelo y su complejidad

2.2.1.3 APRENDIZAJE POR REFUERZO

Método de aprendizaje dónde no tenemos una etiqueta de salida y tampoco buscamos agrupar nuestros datos en función de sus características, el objetivo es que la máquina aprenda por si misma mediante un mecanismo donde se intenta maximizar la recompensa en vez de minimizar el error o función coste como ocurre en el aprendizaje supervisado. [40] [41]
Podemos definir los siguientes componentes:

- **Agente**: nuestro modelo a entrenar
- **Ambiente**: entorno con el que nuestro agente se relaciona y que limita y establece normas dependiendo de su estado
- **Acción**: todas las posibles acciones que nuestro agente puede realizar en cierto instante
- **Estado**: indicadores que pertenecen al ambiente y determinan cómo se encuentra en un determinado momento
- **Recompensas**: premio o castigo a partir de una acción realizada por el agente y que determinaran si lo está haciendo bien o mal.

Al inicio, el agente recibe un "`estado inicial`" y realiza una "`acción`", esta "`acción`" modifica el ambiente y en la siguiente iteración, el agente recibe el "`estado`" y la "`recompensa`" por parte del "`ambiente`". Si la recompensa obtenida es positiva, ese comportamiento se verá reforzado mientras que si es negativa, se verá penalizado. Esto esta basado en el proceso de decisión de Markov. [42] Un uso muy claro de este tipo de aprendizaje se encuentra en los videojuegos, en los cuales tenemos un escenario y miles de posibilidades distintas donde cada movimiento que realizamos tiene un impacto en dicho escenario. Sería imposible hacer un modelo con aprendizaje supervisado puesto que hay muchas variables que no podemos tener en cuenta y

no podemos entrenar con todas las situaciones posibles recurriendo así al uso del aprendizaje por refuerzo

2.2.2 ACTIVE LEARNING

El *Active Learning* es una práctica englobada dentro del aprendizaje automático en la cual un algoritmo de aprendizaje va realizando consultas a una fuente de información (normalmente una persona) que se encarga de etiquetar datos con su salida esperada que anteriormente no se encontraban disponibles. Es utilizado cuando tenemos una gran cantidad de datos sin etiquetar y tendría un gran costo humano hacerlo manualmente, cuando esto ocurre podemos recurrir a la utilización de algoritmos de aprendizaje que consulten de manera activa a una fuente de datos o a un usuario que etiquete manualmente dichos datos para obtener sus etiquetas. Como el usuario es el que elige la cantidad de datos a etiquetar que pasará al modelo, en muchas ocasiones nuestro sistema puede aprender conceptos con un menor número de muestras (si estas muestras son muy relevantes para el sistema) comparado a las que necesitaría sin nuestro etiquetado aunque también tenemos el riesgo de que nuestro algoritmo desborde ya que las muestras que hemos etiquetado sean muy poco informativas y provoquen errores en nuestro sistema. Lo normal es guiarnos por una `"Puntuación de confianza"` que determine que muestras serán más interesantes de consultar. [43]

2.2.2.1 ESCENARIOS

Se puede dividir en los siguientes escenarios:

- **Membership Query Synthesis**: en este escenario, nuestro algoritmo genera su propia instancia a partir de una distribución normal que determine a partir de ciertas consultas que realice. Baum [44] realizó pruebas utilizando la interpolación para optimizar las consultas pero más tarde se demostró que este tipo de consultas no tienen un buen funcionamiento para tareas de visión puesto que el sentido humano no es capaz de reconocer el tipo de consulta óptima encontrada por el algoritmo y por lo tanto, no podemos etiquetar los datos de manera correcta. [45]

- **Stream-Based Selective Sampling**: en este escenario la obtención de etiquetas de estancias no etiquetadas es muy poco costosa y por lo tanto podemos extraer muestras de nuestra distribución actual y decidir si solicitamos la etiqueta de la misma o no. Si la distrbución de los datos de entrada es uniforme, este escenario se comportará como *Membership Query Synthesis* pero si es no uniforme o desconocida tenemos la garantía de que las consultas son razonables ya que proceden de una distribución subyacente real. La decisión de solicitar el etiquetado o no puede plantearse de varios modos. [46]

- **Pool-Based Sampling**: en este escenario, se extraen las instancias de nuestro conjunto de datos y a cada una de estas instancias se le asigna un *Score* o `"Puntuación de confianza"`

en base a lo bien o mal que el sistema entiende dichas instancias. Tras esto el sistema consulta las etiquetas de las instancias que tienen menor confianza para obtenerlas y vuelve a realizar un entrenamiento, repitiendo este proceso de manera iterativa. El principal problema de este escenario es la puntuación de confianza ya que cómo veremos a continuación hay varios método para calcularla. [47]

2.2.2.2 PUNTUACIÓN DE CONFIANZA

Tenemos varios tipos de `"Puntuación de confianza"`, esta puntuación nos permite reconocer las clases que mejor o peor desempeño tienen y, en función de esto, decidir que hacer con las mismas, se encuentra dividido en:

- **Least Confidence**: este método toma la probabilidad que tiene cada dato de pertenecer a una clase y las ordena de menor a mayor, dando así prioridad a las clases que tienen menos probabilidad de contener a dicho dato. [48]
  Su expresión sería la siguiente:

$$S_{LC} = argmax_x(1 - P(\hat{y}/x)) \tag{2.8}$$

$$\hat{y} = argmax_y(P(y/x)) \tag{2.9}$$

- **Margin Sampling**: este método toma la diferencia entre la mayor y la segunda mayor probabilidad que tiene un dato de pertenecer a cierta clase y prioriza las diferencias más pequeñas haciendo así que los puntos donde el modelo duda más entre la primera y segunda clase más probable, sean priorizados. [49]
  Su expresión es la siguiente:

$$S_{MS} = argmin_x(P(\hat{y}/x) - P(\hat{y}_{max-1}/x)) \tag{2.10}$$

- **Entropy**: este método viene desde el campo de la termodinámica y puede entenderse como una medida del desorden de un sistema. Si la entropía es alta, el sistema estará más desordenado mientras que si es baja, el sistema tendrá mayor orden. Este concepto puede usarse utilizando la certeza que tiene un modelo. Si nuestro modelo tiene mucha certeza sobre a que clase pertenece el dato de entrada, el valor de *Entropy* será bajo mientras que si nuestro modelo no tiene certeza sobre la clase a la que pertenece el dato, *Entropy* tendrá un valor mayor pues hay un mayor desorden. [50]
  Se priorizarán los puntos con mayor entropía y su expresión viene dada por:

$$S_E = argmax_x(-\sum i_P(\hat{y}_i/x) log P(\hat{y}_i/x)) \tag{2.11}$$

2.2.3 ONLINE LEARNING

El *Online Machine Learning* es un método de *Machine Learning* en el cual los datos se encuentran disponibles secuencialmente (no disponemos de todos ellos al inicio) y es usado para conseguir mejorar nuestras predicciones futuras entrenando con cada nuevo dato que recibimos al contrario que las técnicas basadas en *Batch Learning*, las cuales generan el mejor predictor a base de entrenar con `"lotes"` de datos. El *Online Learning* es utilizado en situaciones donde es necesario una actualización constante de los datos y, por lo tanto, nuestro algoritmo de predicción debe estar modificándose en busca de nuevos patrones continuamente.

Los algoritmos basados en *Online Machine Learning* son propensos a sufrir *Catastrophic forgetting*, se denomina así a la tendencia que sufren las redes neuronales artificiales de olvidar completa y abruptamente información almacenada previamente sustituyéndola por información nueva [51][52], este problema suele ser mitigado gracias a la utilización de *Incremental Learning*, un método de aprendizaje automático en el que los datos de entrada se utilizan continuamente para ampliar los conocimientos del modelo existente a través del entrenamiento sin tener que reentrenar nuestro modelo completamente cada vez que recibimos dichos datos. Es una técnica de aprendizaje supervisado y no supervisado que se aplica cuando los datos de entrenamiento están disponibles gradualmente en el tiempo. [53]
Los principales campos de aplicación del *Online Learning* son los siguientes:

- **Personalización de recomendaciones**: sistemas de recomendación en línea utilizan *Online Learning* para personalizar las recomendaciones de los usuarios en tiempo real, a medida que van interactuando con el sistema.
- **Análisis de datos en tiempo real**: sistemas como la detección de fraudes o la detección de intrusiones utilizan *Online Learning* para identificar patrones en datos en tiempo real y tomar acciones en consecuencia.
- **Procesamiento de lenguaje natural**: aplicaciones como asistentes virtuales y bots para aplicaciones de mensajería como *Whatsapp* o *Telegram*, utilizan *Online Learning* para mejorar su capacidad de comprensión y respuesta a los usuarios en tiempo real.
- **Control de dispositivos**: dispositivos como termostatos inteligentes y sistemas de seguridad domésticos utilizan *Online Learning* para mejorar su capacidad de tomar decisiones y realizar acciones en tiempo real.
- **Detectores de tos que aprenden en función de nuevos datos**[1]: sistemas de detección que tratan de detectar la presencia de tos en un audio. Estos sistemas utilizan el *Online Learning* para ir actualizándose y ajustándose a diferentes tipos de voces, entornos más o menos ruidosos, etc. Son muy útiles pues pueden tener varias implementaciones, algunos ejemplos son los siguientes:
    1. Durante la pandemia de COVID-19, podríamos utilizar estos detectores para intentar identificar personas infectadas y así reducir la propagación del virus.
    2. En hospitales podríamos monitorear pacientes de un modo más eficiente, identificando toses persistentes o inusuales.

[1] Tema tratado en el presente TFG

3. En las plantas alimentarias podríamos determinar cómo es la calidad del aire pues la tos es un factor importante de contaminación. Podríamos dictaminar en tiempo real cómo se encuentra el aire de nuestra planta y actuar en consecuencia.

2.3 Deep Learning

El aprendizaje automático (*Machine Learning*) y el aprendizaje profundo (*Deep Learning*) son dos subcampos del aprendizaje automático que se diferencian en cómo se representan y aprenden las características. Ambas disciplinas comparten un objetivo común: aprender una representación a partir de datos de entrada para realizar tareas específicas, como la clasificación o la regresión. Sin embargo, existen algunas diferencias importantes en cuanto a la forma en que se representan y aprenden las características.

En el *Machine Learning* tradicional, se utiliza un enfoque conocido como extractor de características manual para identificar y seleccionar las características más relevantes. Este proceso requiere de la selección cuidadosa y la ingeniería de las características a partir de los datos de entrada, y se basa en la idea de que una representación adecuada de los datos conseguirá una mejora en el rendimiento de la tarea objetivo. En este enfoque, el modelo de aprendizaje automático es alimentado con las características extraídas manualmente en lugar de los datos brutos.

El *Deep Learning*, por otro lado, utiliza un enfoque automático para extraer características a través de una estructura de red neuronal. Las características son aprendidas automáticamente a partir de los datos de entrada en un proceso conocido como aprendizaje de características. La estructura de la red neuronal permite que el modelo aprenda una representación de los datos gracias a sus varias capas de procesamiento, cada una de las cuales puede aprender una representación más abstracta y compleja. Este proceso permite al modelo aprender características relevantes para la tarea objetivo, lo que puede resultar en un rendimiento mejor que con un enfoque de extractor de características manual.

En resumen, la principal diferencia entre *Machine Learning* y *Deep Learning* es la forma en que se representan y aprenden las características. Mientras que el aprendizaje automático tradicional utiliza un enfoque manual para extraer características, el aprendizaje profundo utiliza un enfoque automático a través de redes neuronales para aprender características relevantes para la tarea objetivo. [54]

2.3.1 Redes neuronales

Las redes neuronales son el método más utilizado en *Deep Learning* y buscan simular el funcionamiento del sistema nervioso humano en computadoras para realizar procesado de datos. Está basado en la utilización de un gran número de neuronas organizadas en diferentes capas que se intercomunican entre sí mediante enlaces, cada uno de los cuales lleva asociado un determinado peso. Dicho peso representa la información utilizada por nuestra red para la resolución de problemas. Las redes neuronales aprenden mediante la experiencia, son entrenadas con ejemplos ilustrativos y ellas mismas generan un modelo que les permite realizar tareas en base a lo que han aprendido. Además de esto, las redes neuronales pueden generalizar dicho problema y conseguir resolver nuevos casos a los que anteriormente no habían sido expuestas. La información es procesada paralelamente entre nuestras neuronas ya que muchas de estas pueden estar funcionando a la vez, esto provoca que tengamos una gran capacidad de cálculo pues aunque de manera individual cada neurona solo pueda realizar procesamientos simples, si

interconectamos una gran cantidad de ellas conseguiremos mucha capacidad de procesamiento. Las redes neuronales artificiales utilizan una representación numérica de la información y permiten desde un valor real continuo hasta un valor discreto o binario. [28]

2.3.1.1 NEURONAS

Son el elemento más pequeño en el que se divide nuestra red neuronal, en la figura 2.13 podemos observar como consta de unas entradas que recibe como señal (generalmente son señales enviadas desde otras neuronas a las que está conectada y ponderada por el peso de su respectiva conexión) que se sumarán dando lugar a la entrada neta. También disponen de una salida, lugar desde donde enviamos información a otras neuronas. Dicha salida es calculada al aplicar a nuestra entrada neta cierta función matemática (llamada función de activación).

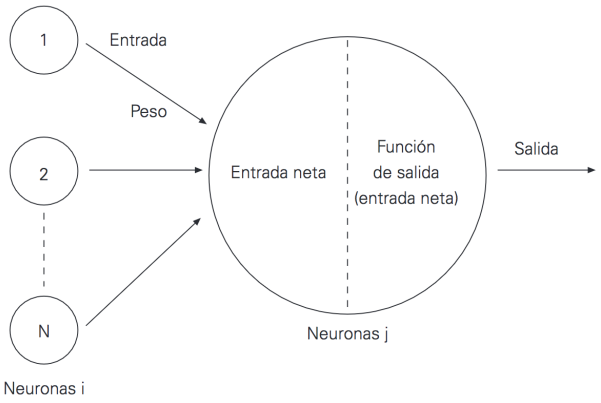


FIGURA 2.13: Estructura de una neurona. [55]

2.3.1.2 FUNCIONES DE ACTIVACIÓN

Dentro de las funciones de activación podemos encontrar varias, entre las que destacan:

- **Sigmoide**: esta función transforma los datos de entrada a una escala comprendida entre 0 y 1, los valores altos tienden asintóticamente a 1 y los valores bajos tienden asintóticamente a 0. Tiene una lenta convergencia y nos permite buenos rendimientos en la última capa de la red. [56]
  Su representación podemos observarla en la figura 2.14 y viene dada por la ecuación:

$$f(x) = \frac{1}{1 - e^{-x}} \tag{2.12}$$

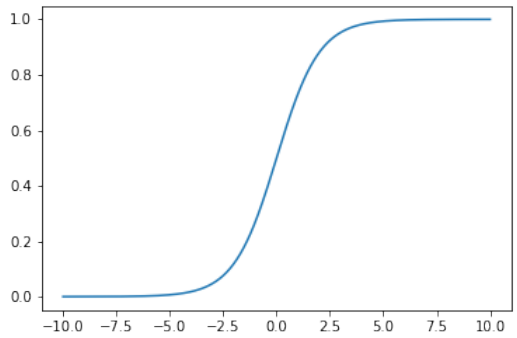


FIGURA 2.14: Representación de la función Sigmoide

- **Tangente hiperbólica**: esta función transforma los datos de entrada a una escala comprendida entre -1 y 1, los valores más grande tienden asintóticamente a 1 y los valores más bajos tienden asintóticamente a -1. Es utilizada en problemas donde nuestra red tiene que decidir entre 2 opciones. [56]
  Podemos observar su representación gráfica en la figura 2.15 y su ecuación es:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2.13}$$

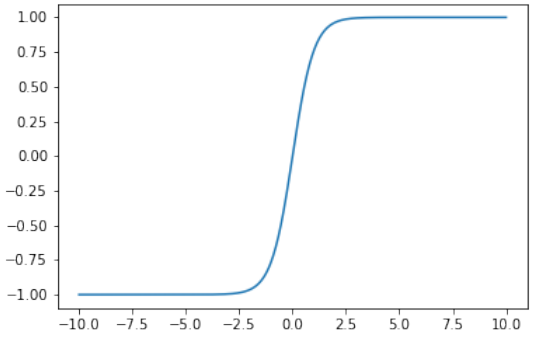


FIGURA 2.15: Representación de la función Tangente hiperbólica

- **SoftMax**: esta función transforma las salidas a un formato probabilístico de modo que la suma de todas estas probabilidades sean igual a 1. [56]
  Es utilizada en problemas donde buscamos obtener una representación en forma de probabilidades, viene dada por la ecuación:

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{-2x_k}} \tag{2.14}$$

- **ReLU**: Esta función consiste en anular los valores negativos (convirtiéndolos en 0 a la salida) y no modificar el valor de los valores positivos a la entrada (pasándolos a la salida sin modificar). Es la función más utilizada en redes convolucionales y tiene un buen comportamiento al tratar con imágenes. [56]
  Podemos observarla gráficamente en la figura 2.16 y viene dada por la ecuación:

$$f(x) = \begin{cases} 0 & si \quad x \leq 0 \\ \\ x & si \quad x \geq 0 \end{cases} \tag{2.15}$$

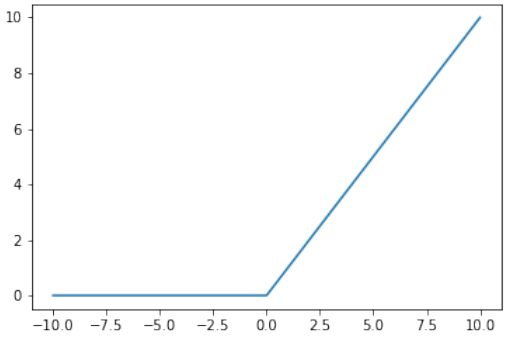


FIGURA 2.16: Representación de la función ReLu

2.3.1.3 TIPOS DE REDES

Podemos encontrar varios tipos de redes neuronales dependiendo de como las clasifiquemos, podemos hacerlo del siguiente modo:

- **Clasificación por el número de capas**:
  - **Redes monocapa**: son las redes más simples ya que no tienen capas intermedias y por lo tanto su capa de entrada está directamente conectada a su capa de salida. [57]
  - **Redes multicapa**: estas redes contienen varias capas ocultas entre la capa de entrada y la capa de salida con neuronas que en algunos casos estarán conectadas entre sí y en otros no. [57]
- **Clasificación por el tipo de conexión**
  - **Redes no recurrentes**: la información de la red es transportada en un solo sentido (desde la capa de entrada a la de salida) y no existe la realimentación. Todo esto provoca que estas redes no tengan memoria y que actualmente no sean muy utilizadas. [57]
  - **Redes multicapa**: la información de la red puede ser compartida entre neuronas de una misma o distintas capas gracias a la realimentación provocando que nuestra red

tenga memoria. Estas redes son más versátiles y tienen mayor capacidad de cálculo que las no recurrentes y por ello su uso está más extendido actualmente. [57]

- **Clasificación por grado de conexiones**

  - **Redes totalmente conectadas**: cada capa de nuestra red está totalmente conectadas con la capa anterior y posterior gracias a que todas las neuronas se encuentran conectadas entre sí. [57]

  - **Redes parcialmente conectadas**: las neuronas de nuestras capas no se encuentran conectadas completamente entre sí. [57]

- **Redes convolucionales**: las redes convolucionales tratan de emular el funcionamiento de un cerebro biológico imitando para ello las neuronas que se encuentran en la corteza visual. En dicha corteza existen dos tipos de células encargadas de la detección de bordes y orientación. Las primeras células son conocidas como células simples y poseen varias regiones que se excitan o inhiben formando diferentes patrones en una dirección, tamaño y posición para cada célula en particular. Si recibimos una señal visual que tiene la misma dirección y posición que uno de los patrones de nuestras células, dicha célula se activará y emitirá una señal. El segundo tipo de células son las células complejas y funcionan igual que las simples pero siendo sensibles a la orientación recibida por un estímulo visual, activándose y emitiendo señal cuando la orientación de dicho estímulo es igual al patrón de orientación de la célula. La estructura que forman todas estas células en la corteza visual es muy importante pues a medida que nos vamos alejando de las primeras capas que reciben estos estímulos visuales directamente, vamos perdiendo sensibilidad a la posición, dirección y orientación pues las células empiezan a depender de los estímulos y propagaciones de las células en capas anteriores. [58] [59]
  Podemos dividir una red convolucional en las siguientes capas:

  - **Capa de entrada**: esta capa se encarga de almacenar los valores de entrada a nuestra red (normalmente cada entrada es el equivalente al valor de un pixel de la imagen a procesar).

  - **Capa convolucional**: esta capa se encarga de utilizar máscaras o kernels para procesar la imagen obtenida en la capa de entrada. Para ello partimos de una máscara de un tamaño elegido (suele ser 3x3) y procedemos a desplazar dicho kernel a través de nuestra imagen inicial aplicando el producto escalar obteniendo así un nuevo mapa de valores tal como podemos observar en la imagen 2.17

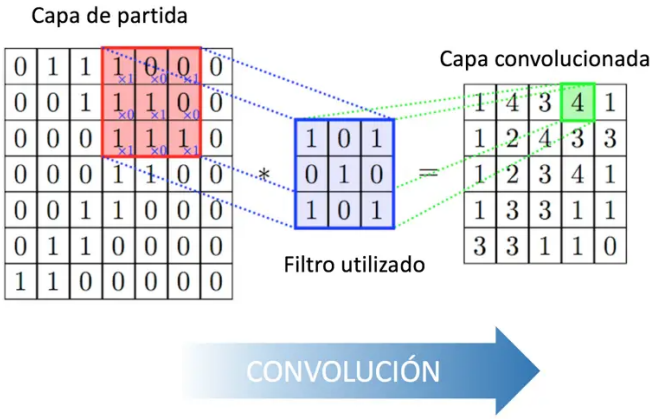


FIGURA 2.17: Representación del funcionamiento de una capa convolucional 3x3. [60]

- **Capa de pooling**: esta capa se encarga de simplificar nuestro sistema reduciendo para ello la dimensionalidad de la matriz de entrada tal y cómo podemos ver en la figura 2.18 (El método Average Pooling selecciona la media de los valores seleccionados y el método Max Pooling se queda con el valor más alto, en esta figura se utiliza el método Max Pooling). Nuestro sistema perderá precisión pues estamos eliminando información pero a su vez favorecemos la invariabilidad a pequeñas variaciones a la entrada pues hemos disminuido el número de características a analizar. Suele ser de 2x2.

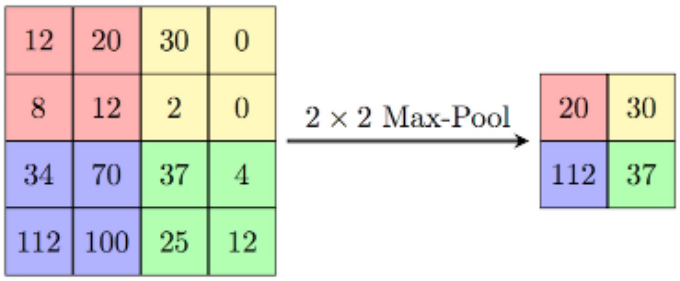


FIGURA 2.18: Representación del funcionamiento de una capa de Pooling 2x2 (método Max Pooling). [61]

- **Capa fully conected**: esta capa se encuentra a la salida de nuestro sistema y se aplica tras el uso de una capa *Flatten* que se encarga de eliminar la dimensionalidad de la información. Normalmente tienen una función de activación *SoftMax* y devuelven la probabilidad de cada clase posible (las redes convolucionales son mayormente utilizadas para clasificación)

2.3.1.4 FUNCIÓN DE COSTE

La función de coste determina la diferencia que existe entre un valor y ese mismo valor estimado por nuestra red neuronal. Estas funciones de coste son utilizadas para ajustar parámetros de la red y existen los siguientes:

- **RMSE (Error cuadrático medio)**: esta función de coste penaliza los errores grandes y suele utilizarse para optimizar regresiones. [62]
  Su expresión es la siguiente:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{2.16}$$

- **MAE (Error absoluto medio)**: esta función de coste no penaliza tanto los errores grandes y es más fácil de interpretar que el RMSE pero su convergencia es más complicada. [62]
  Su expresión es la siguiente:

$$MAE = \sum_{i=1}^{n} \frac{|\hat{y}_i - y_i|^2}{n} \tag{2.17}$$

- **MASE (Error absoluto medio escalado)**: esta función es similar al MAE pero está escalado por sus propios valores. Es muy fácil de interpretar pero su convergencia se vuelve más complicada. [62]
  Su expresión viene dada por:

$$MASE = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{\frac{n}{n-1} \sum_{i=2}^{n} |\hat{y}_n - y_{n-1}|} \tag{2.18}$$

- ***Categorical Crossentropy***: es una medida de precisión que se utiliza para variables categóricas y es fácil de interpretar pues tiene una escala univariante y es simétrica. [63]
  Su expresión viene dada por:

$$L(\Phi) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} \hat{y}_i log(p_{ij}) \tag{2.19}$$

2.3.1.5 Entrenamiento de una red neuronal

Durante el entrenamiento de una red neuronal, los datos de entrenamiento se dividen en pequeños conjuntos llamados *batches* que son procesados y utilizados para actualizar los pesos de la red neuronal. Cada iteración se denomina como *epoch* y consiste en procesar todos los *batches* de los datos de entrenamiento.
El tamaño del *batch* es un hiperparámetro muy importante para el entrenamiento de la red y tiene un gran impacto en su desempeño. Si el tamaño del *batch* es demasiado grande, necesitaremos una gran cantidad de recursos computacionales para entrenar el sistema. Por otro lado, un tamaño de *batch* demasiado pequeño provocaría que el modelo fuese propenso a la variabilidad y la pérdida de precisión. Por lo tanto, debemos buscar un equilibrio entre el tamaño del *batch* y la eficiencia del entrenamiento.
El modo de aprendizaje de una red neuronal consiste en mejorar la solución a través de iteraciones, hasta que se alcance un rendimiento aceptable. Esto se logra encontrando un conjunto de pesos para nuestras neuronas adecuados que permitan a la red llevar a cabo una tarea específica, partiendo de pesos iniciales aleatorios. Se evalúa el rendimiento actual de la red mediante una función de error basada en los pesos (esta función de error suele ser el error cuadrático medio) y el objetivo es encontrar cuál es el conjunto de pesos que consigue minimizar esa función. El método de entrenamiento utiliza una regla de actualización que, en función de los patrones de entrada, modifica iterativamente los pesos hasta llegar al punto óptimo de la red neuronal. [64]
Existen varios métodos de optimización como los siguientes:

- **Gradiente descendente**: este método iterativo busca minimizar la función de coste utilizando las derivadas parciales de dicha función. Para ello definiremos la función de coste como $f(w)$ siendo w el conjunto de todos los pesos de la red. Los pasos a seguir son:

  1. Comenzamos con un conjunto de pesos (podemos escogerlos nosotros, escogerlos aleatoriamente o utilizar algún método) con el que calculamos la dirección en la que se encuentra el máximo error.

  2. Calculamos el gradiente $\nabla f(w)$ que nos indica cual es la dirección en la que se encuentra el máximo crecimiento de nuestra función para w_0.

  3. Actualizamos el valor de nuestros pesos w utilizando para ello el sentido opuesto al dado por el gradiente $\nabla f(w)$ ya que así nos encontraremos en la dirección de máximo decrecimiento. Estas actualizaciones de los pesos serán más bruscas o suaves dependiendo de la tasa de aprendizaje que utilicemos (viene dado por la letra α)

  Este valor de α es crucial para el buen funcionamiento del método pues si utilizamos un α demasiado pequeño, el método necesitará más tiempo para acercarse al mínimo y si α es demasiado grande, el método no será lo suficientemente preciso para encontrar el mínimo. Podemos ver un ejemplo de este método en la siguiente figura (2.19) para 2d ya que es fácil de visualizar, aunque se generaliza para n dimensiones. [65]

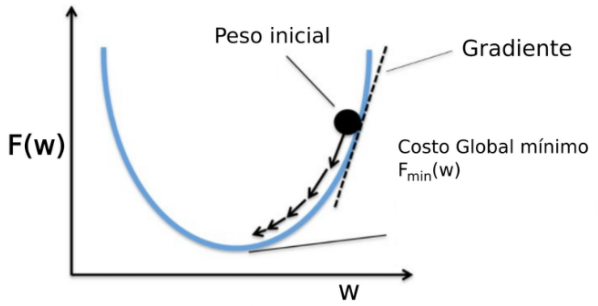


FIGURA 2.19: Representación de el Gradiente descendente. [66]

- **AdaMax**: primero, definiremos las normas del vector gradiente del siguiente modo:

  $L_1 = \nabla$
  $L_2 = \sqrt{\nabla^2}$
  $L_n = \sqrt[n]{\nabla^n}$
  Siendo ∇ el vector de gradientes

  Muchos algoritmos utilizan la varianza (L_2) para funcionar ya que a partir de la norma 2 empezamos a perder estabilidad en los cálculos. Este método busca minimizar la función de coste utilizando para ello la norma infinita ($L_p \to \infty$) del gradiente ya que, sorprendentemente podemos simplificar la expresión de la misma de un modo muy sencillo para obtener la ecuación: [67]

  $$u_t = \text{máx}(\beta_2 \cdot u_{t-1}, |\nabla_t|) \tag{2.20}$$

  Donde: β_2 es una tasa de decrecimiento exponencial y u_t es la norma infinita ponderada exponencialmente.

- **Backpropagation**: el *Backpropagation* es un algoritmo clave para conseguir que las redes neuronales sean lo más parecidas posibles a nuestro cerebro humano, está basado en la capacidad de organización que tienen las neuronas que se encuentran en las capas intermedias de las redes, las cuales son capaces de extraer y reconocer características o patrones en los datos de entrada. El algoritmo de *Backpropagation* nos permite identificar falta de datos para completar patrones ya aprendidos durante la fase de entrenamiento gracias a su funcionamiento, el cual es el siguiente:

  1. **Inicialización de los pesos de la red**: Se inicializan aleatoriamente todos los pesos de la red.
  2. **Propagación hacia adelante**: Se propaga la entrada de una muestra de entrenamiento a través de la red, multiplicando los valores de entrada por los pesos de las diferentes conexiones entre capas obteniendo la salida de la red.
  3. **Cálculo del error**: Se calcula la diferencia entre la salida esperada de la red y la salida calculada en los pasos anteriores, es decir, el error de la red.

4. **Retropropagación del error**: El error se propaga hacia atrás a través de la red, calculando el gradiente de la función de error con respecto a cada peso en la red.
5. **Actualización de los pesos**: Se actualizan los pesos de la red utilizando el gradiente calculado en el paso anterior, con el objetivo de minimizar el error en la salida de la red. El *Learning Rate* es un hiperparámetro que determina cuánto se deben actualizar los pesos en función del gradiente en cada actualización. Si nuestra tasa de aprendizaje o *Learning Rate* es alta puede que la red se salte mínimos locales y no converja, mientras que un valor bajo de *Learning Rate* puede hacer que la red tarde demasiado en converger.
6. **Repetición de los pasos 2-5**: Se repiten los pasos 2 a 5 para todas las muestras de entrenamiento en el conjunto de datos. Este proceso se denomina una época de entrenamiento.
7. **Evaluación de la red**: Se evalúa la red en un conjunto de datos de prueba para comprobar si su desempeño y si es capaz de generalizar de un modo correcto.

Este algoritmo fue creado como solución a la concatenación de errores que ocurre habitualmente en una red neuronal pues cuando una neurona/nodo no tiene un peso adecuado, ese error se va transmitiendo al resto de capas y así sucesivamente, aumentando el error de nuestra red. [68]

2.3.2 TRANSFER LEARNING

Se conoce como *Transfer Learning* al conjunto de métodos que nos permiten transferir información que nos ha resultado útil a la hora de resolver ciertos problemas para resolver otros. En el campo del *Deep Learning* es muy útil pues nos permite reutilizar modelos que necesitan grandes tiempos de cálculo y recursos como base para la resolución de nuevos problemas. Está basado en la utilización de los conocimientos adquiridos por una configuración (*source*) para resolver un problema determinado (*objetive*). La implementación de *Transfer Learning* dentro de *Deep Learning* está basada en la utilización de redes neuronales entrenadas previamente para trasladarlas a mi nuevo problema. Normalmente utilizamos como redes ya entrenadas modelos de un alto rendimiento que han sido testados sobre una gran cantidad de datos y que sería computacionalmente muy costoso entrenar por nuestra cuenta. [69]
Podemos distinguir entre los siguientes tipos de *Transfer Learning* dependiendo de cómo se realice la transferencia de información:

- **Aprendizaje por transferencia inductiva**: para este tipo de *Transfer Learning*, tenemos los mismos datos de base tanto para la *source* como para el *objetive* pero las tareas finales de ambos son diferentes (por ejemplo podemos utilizar un modelo entrenado que detecte diferente objetos para crear un modelo que detecte la presencia de coches)
- **Aprendizaje por transferencia no supervisada (*Unsupervised Transfer Learning*)**: este tipo de *Transfer Learning* parte de la idea de que es más común obtener grandes cantidades de datos sin etiquetar que etiquetados. En este caso tanto los datos de la *source* y del *objetive* son parecidos pero sus tareas son diferentes y los datos se encuentran sin etiquetar

- **Aprendizaje por transferencia transductiva (*Transductive Transfer Learning*)**: este tipo de *Transfer Learning* utiliza tareas similares para *source* y *objetive* pero sus datos o distribución de probabilidad son diferentes

Cuando utilizamos *Transfer Learning* podemos proceder de 2 modos:

2.3.2.1 Utilizar modelos entrenados previamente para extraer características

Esta estrategia está basada en la utilización de todas las capas de un modelo pre-entrenado (*source*) excepto la última (la de salida), cambiando esta por una capa que nos permita obtener los resultados deseados en *objetive*. Un claro ejemplo de esto es AlexNet, una arquitectura de Red Neuronal Convolucional creada por Alex Krizhevsky [70] que en 2012 ganó el *ImageNet Large Scale Visual Recognition Challenge* al ser la red con el menor error a la hora de evaluar una base de datos (ImageNet) que contiene más de 14 millones de imágenes pertenecientes a más de 20000 categorías diferentes etiquetadas a mano. AlexNet es una red muy utilizada en *Transfer Learning* pues partiendo de ella podemos conseguir modelos para clasificar imágenes con un gran rendimiento.

2.3.2.2 Ajustar modelos pre-entrenados

Esta es una técnica compleja puesto que además de reemplazar la capa de salida de la red pre-entrenada, también reemplazaremos otras capas dónde los hiperparámetros no se ajustan exactamente a nuestro nuevo problema ya que generalmente las últimas capas de los modelos suelen encargarse de tareas más específicas que las primeras (estas se encargan de identificar características más generales). Si congelamos el peso de algunas capas a la hora de entrenar, podemos ajustar los hiperparámetros de otras capas para afinar la extracción de características determinadas y potenciar la red.

2.3.3 Online Deep Learning

Existe un puente entre el *Online Learning* y el *Deep Learning* si abordamos el problema de cómo aprenden las Redes Neuronales Profundas (DNNs) a partir de un flujo de datos utilizando una configuración *Online*. Un enfoque sería el de utilizar el proceso de entrenar nuestra DNN con la *Standard Back-propagation*, utilizando una sola instancia cada ronda del entrenamiento *Online*. Esta aproximación es simple pero no es efectiva ya que tiene muchas limitaciones en la práctica.

Uno de los principales problemas es el elegir una profundidad adecuada para nuestro modelo antes de empezar a aprender de la *DNN Online*. Si nuestro modelo es demasiado complejo, el proceso de aprendizaje convergerá muy lento y no cumpliremos el objetivo de aprendizaje *Online*, por otro lado, si nuestro modelo es demasiado simple la capacidad de aprender estará muy restringida y, sin esta capacidad de profundidad no podremos comprender patrones complejos. Si utilizamos *Batch learning*, la forma para solventar este problema es la utilización de datos de validación (que ajustan los hiperparámetros) pero esta implementación no es posible si

utilizamos *Online Learning* puesto que no podemos generar un conjunto de datos de validación al variar nuestro modelo cada vez que recibimos nuevos datos.

2.3.3.1 ONLINE HEDGE BACKPROPAGATION (OHBP)

Para intentar paliar estos problemas y crear una implementación de *Online Deep Learning* funcional, Doyen Sahoo et. al en [71] crearon un modelo basado en el *Backpropagation* pero utilizando una estrategia de cobertura (*Hedge*) a la que llamaremos *Online Hedge Backpropagation* (OHBP). En el *Backpropagation* tradicional se utiliza un conjunto de datos fijo para entrenar una red neuronal, el algoritmo se ejecuta en varias iteraciones en cada una de las cuales se presentan todos los datos de entrenamiento y se actualizan los pesos de la red neuronal. Sin embargo, OHBP se basa en un proceso de entrenamiento incremental, en el que se presentan los datos de entrenamiento uno por uno y se actualizan los pesos de la red neuronal en consecuencia (con un *Learning Rate* que podemos decidir).

La idea detrás de OHBP es que al entrenar una red neuronal de forma incremental, esta se puede adaptar a los cambios en los datos de entrenamiento conforme se van presentando. Utiliza el mecanismo de *Hedge*, el cual consiste en que nuestra predicción será una combinación ponderada de los diferentes clasificadores que tiene la red en vez de funcionar como el método original, donde se obtiene a través de un clasificador que se nutre solo de h^L (ver figura 2.20). Gracias al *Hedge* podemos observar como la capacidad efectiva de la red se va adaptando en cada momento pues los pesos de cada clasificador variarán con cada nuevo dato provocando que la red aumente o disminuya su profundidad conforme lo vaya necesitando. Los pasos para el entrenamiento son los siguientes:

1. Inicializar los pesos de la red neuronal de forma aleatoria.
2. Recibir un nuevo dato de entrenamiento.
3. Propagar este dato de entrenamiento a través de la red neuronal para obtener la salida deseada.
4. Calcular la diferencia que existe entre la salida deseada y la salida real (cáclulo de error).
5. Actualizar los pesos de la red neuronal utilizando el algoritmo de *Hedge Backpropagation*.
6. Repetir los pasos 2-5 con cada nuevo dato de entrenamiento que se reciba

OHBP tiene varias ventajas sobre el *Backpropagation* tradicional. Por ejemplo, permite a la red neuronal adaptarse a los cambios en los datos de entrenamiento de manera dinámica pues sus pesos van cambiando a medida que se los datos van llegando. También nos otorga una mayor flexibilidad en la cantidad de memoria que se encuentra disponible, ya que solo necesitamos almacenar los datos presentados recientemente y no todos. [72]
En la figura 2.20 podemos observar la estructura del OHBP, la cuál consta de:

- **Línea azul**: representan el flujo de avance para el cálculo de las características de la capa oculta.

- **Línea naranja**: indican la salida tras aplicar la función softmax y el uso de los distintos decisores a continuación.

- **Línea verde**: indican el flujo de actualización *online* utilizando el enfoque de Hedge Backpropagation.

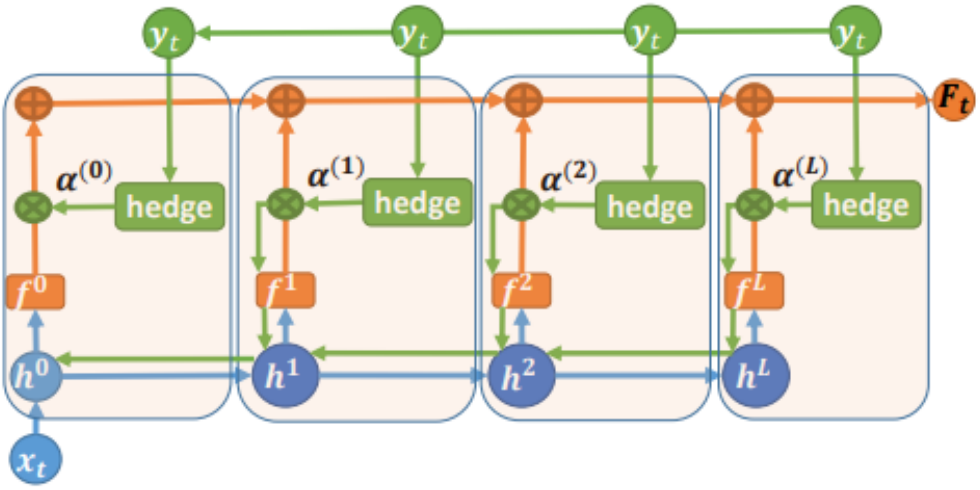


FIGURA 2.20: Estructura de OHBP. [72]

2.3.3.2 MODELO MATEMÁTICO

Estamos considerando una *Deep Neural Network* con L capas ocultas (la mayor capacidad de la red es una con L capas) y la función de predicción viene dada por la siguiente ecuación:

$$\mathbf{F}(x) = \sum_{l=0}^{L} \alpha^{(l)} \mathbf{f}^{(l)} \tag{2.21}$$

donde:

$\mathbf{f}^{(l)} = softmax(\mathbf{h}^{(l)}\Theta^{(l)}), \forall l = 0, ..., L$
$\mathbf{h}^{(l)} = \sigma(W^{(l)}\mathbf{h}^{(l-1)}), \forall l = 1, ..., L$
$\mathbf{h}^{(0)} = \mathbf{x}$

En esta nueva arquitectura hemos introducido dos nuevos parámetros: $\Theta^{(l)}$ y α que deben ser aprendidos.
A diferencia de la red original de *Backpropagation* donde la predicción final viene dada por el valor de $\mathbf{h}^{(t)}$, aquí nuestra predicción viene ponderada por los diferentes clasificadores aprendidos

utilizando $\mathbf{h}^{(0)}, ..., \mathbf{h}^{(L)}$.
La predicción total del modelo es una ponderación de las predicciones de todos los clasificadores dónde el peso de casa clasificador viene denotado por $\alpha^{(l)}$ y la función de pérdida del modelo viene dada por: $\zeta(\mathbf{F}(x), y) = \sum_{l=0}^{L} \alpha^{(l)} \zeta(\mathbf{f}^{(l)}(x), y) \forall l = 0, ..., L$
Durante el proceso de aprendizaje *online*, necesitamos aprender $\alpha^{(l)}, \theta^{(l)} y W^{(l)}$.
Para aprender $\alpha^{(l)}$ utilizaremos el *Hedge Algorithm* [73] haciendo que en la primera iteración todos los pesos de α sean uniformemente distribuidos. En cada iteración, el clasificador $\mathbf{f}^{(l)}$ hace una predicción $\hat{y}_t^{(l)}$.
Al final de cada ronda, los pesos de α se normalizan del siguiente modo: $\sum_{l=0}^{L} {\alpha_t}^{(l)} = 1$.
Para aprender los parámetros $\theta^{(l)}$ en todos los clasificadores, utilizaremos el *Online Gradient Descent* [74].
Para actualizar $W^{(l)}$ debemos usar una función adaptativa y aplicarle el *Online Gradient Descent* pues esta métrica viene dada por las derivadas del error que se retropropaga desde cada clasificador y no desde la salida como ocurre en el *Backpropagation* tradicional.

Capítulo 3

DETECCIÓN DE TOS

La detección de toses por audio utilizando *Deep Learning* es un problema de clasificación donde haciendo uso de una DNN buscamos clasificar audios como toses.

Cuando nos enfrentamos a un problema de este tipo, normalmente utilizamos las siguientes técnicas para obtener resultados satisfactorios:

1. **Preprocesado de datos**: debemos normalizar la amplitud del audio y convertirlo en una representación numérica que pueda ser procesada por la red neuronal. Las técnicas mas relevantes están basadas en la utilización de *Mel-Frequency Cepstral Coefficients* (MFCC) y *Short-time Fourier Transform* (STFT).
2. **Diseño de la red neuronal**: se suelen utilizar redes neuronales convolucionales o redes neuronales recurrentes Estas redes utilizan filtros para extraer características de la señal de audio y a través de sus capas obtenemos una salida (que será la clasificación).
3. **Entrenamiento de la red neuronal**: se usan datos de entrenamiento para realizar el entrenamiento de la red.
4. **Evaluación de la red neuronal**: tras haber entrenado la red neuronal, pasamos a testearla. Para ello, utilizaremos datos de test para los que generaremos predicciones de clasificación y posteriormente compararemos dichas predicciones con la clasificación real.
5. **Optimización de la red neuronal**: podemos mejorar el desempeño de nuestra red mediante el ajuste de hiperparámetros, profundidad de la red...

3.1 ESTADO DEL ARTE

Existen varios estudios científicos sobre el problema de detección de tos. A continuación presentaremos algunos de los más relevantes junto a estudios relacionados con algoritmos adaptativos y el trabajo de Diego Pérez Alonso [3], trabajo en el cual profundizaremos e intentaremos mejorar con diferentes métodos:

Un sistema de audición de máquina para la detección robusta de tos basado en una representación de alto nivel de características de audio específicas de la banda

En agosto del año 2019, Jesús Monge Álvarez et al. [75] presentaron un sistema de detección automática de tos basado en el análisis de señales de audio. Este sistema no utilizaba una DNN y se puede dividir en los siguientes pasos:
En primer lugar, se calcularon características espectrales de corta duración en las siguientes cinco bandas de frecuencia: [0; 0,5), [0,5; 1), [1; 1,5), [1,5; 2), [2; 5,5125] kHz. Para obtener un resultado robusto en escenarios donde hubiese ruido, se seleccionaron y combinaron las características más relevantes.
En segundo lugar, se obtuvo una representación de los datos a un alto nivel gracias al cálculo de la media y la desviación estándar de los descriptores de corta duración en *frames* de 300ms a largo plazo. Por último, se utilizó una SVM entrenada con datos de diferentes entornos ruidosos para llevar a cabo la detección de la tos. El sistema se evaluó utilizando una base de datos que contiene diferentes señales de audio que emulan tres escenarios de la vida real (con diferentes contenidos de ruido). El sistema obtuvo una sensibilidad del 92.71 % y una especificidad del 88.58 %.

Clasificación de eventos de tos por red neuronal profunda pre-entrenada

En noviembre del año 2015, Jia-Ming Liu Jia-Ming Liu et al. [76] desarrollaron un modelo de *Deep Neural Network* a través de un proceso de entrenamiento que primero realizaba un pre-entrenamiento sin supervisión y ajuste fino. Para capturar información temporal se utilizó un *Hidden Markov Model* (HMM) donde la red neuronal profunda predecía la probabilidad de observación asociada con cada estado del HMM. Para la clasificación final, se empleó el algoritmo de decodificación Viterbi, y se diseñaron tres HMM para la clase de tos y uno para la clase de no tos. La representación de características se basó en *Mel Frequency Cepstral Coefficients* (MFCC). Las grabaciones fueron realizadas por una grabadora digital portátil con un micrófono colocado en el cuello del paciente, y las señales de audio se adquirieron en un entorno hospitalario corriente. Antes del análisis, se segmentaron las todas las secuencias de audio en clips de 10 segundos y se realizó una compensación de forma manual para eliminar el desequilibrio de clases, dejando la proporción de no tos/tos a 2. En un experimento donde se utilizaron dieciséis pacientes para el entrenamiento y seis pacientes para la prueba, el modelo alcanzó una sensibilidad del 83.6 % y una especificidad del 90.9 %.

Detección de eventos de tos en audio mediante el uso de Momentos Hu Locales

En 2019, Jesús Monge et al. [77] publicaron un artículo donde se ponía de manifiesto la importancia de la telemedicina y de cómo esta rama es muy útil para mejorar la atención médica en enfermedades respiratorias. Uno de los síntomas más comunes de enfermedades respiratorias es la tos y con un detector que no modificase la vida normal del usuario que lo lleva, podríamos conseguir detecciones muy tempranas de diversas enfermedades respiratorias. El artículo

discute la importancia de monitorear en tiempo real la tos para la salud y cómo las tecnologías modernas pueden permitir la detección automática de la tos gracias al uso de dispositivos móviles como *smartphones* y *smartwatches*. El artículo propone el uso de momentos Hu locales como un conjunto de características robustas para el sistema de detección automática de tos y muestra su rendimiento en varios ambientes ruidosos. Se analiza el equilibrio entre el rendimiento y la eficiencia del sistema y se compara con otros conjuntos de características y clasificadores de audio buscando un detector eficiente y que sea cómodo de utilizar en el día a día. Se evaluó una base de datos que incluía ruidos ambientales y 16 horas de grabaciones de pacientes reales con problemas respiratorios y los mejores resultados obtenidos fueron utilizando los momentos locales Hu como características y un clasificador k-NN, obteniendo una sensibilidad y especificidad de hasta el 88.51 % y el 99.77 % respectivamente.

Clasificación de eventos de tos por red neuronal profunda

En 2019, Diego Pérez Alonso [3] desarrolló el modelo de DNN que podemos encontrar en la sección 4.2. Esta red se utilizó para la detección de toses en audios pertenecientes a 20 pacientes (véase la base de datos descrita en 4.1) usando el método de *k-fold cross validation* con folds balanceados. Los resultados obtenidos podemos observarlos en la tabla 3.1. Lo que se puede inferir a partir de los resultados obtenidos es que el modelo tiene un buen desempeño para la detección de toses en la base de datos utilizada en el estudio. Sin embargo, la generalización de este modelo a otros contextos o bases de datos puede no ser tan efectiva. Esto se debe a que las características de la base de datos utilizada para entrenar el modelo pueden ser diferentes de las de otros conjuntos de datos, lo que podría afectar su capacidad para detectar toses en otros pacientes o en diferentes entornos. Por lo tanto, es necesario el reconocimiento de las características y pacientes que más influyan en el rendimiento de la red junto con el uso de algoritmos adaptativos como los que presentaremos a continuación:

- **Active Learning**: el uso de *Active Learning* puede proporcionar varias mejoras respecto al aprendizaje pasivo.
  En primer lugar, puede reducir significativamente la cantidad de datos presentes en el set de entrenamiento. Al elegir los datos más informativos podemos conseguir modelos más eficientes con menos datos.
  En segundo lugar, el *Active Learning* puede ser útil en situaciones en las que los datos son costosos o difíciles de obtener ya que cómo hemos dicho antes, podemos obtener la información más relevante del sistema de un modo más eficiente que un aprendizaje pasivo sin la necesidad de utilizar grandes cantidades de datos.
  En tercer lugar, el *Active Learning* puede aumentar la eficiencia del modelo al mejorar su capacidad para aprender de manera autónoma a medida que recibe nuevos datos.

- **Online Learning**: el uso de *Online Learning* puede proporcionar varias mejoras respecto al aprendizaje pasivo.
  En primer lugar puede ser muy útil en aplicaciones en las que el conjunto de datos es grande y es actualizado frecuentemente, ya que consigue que el modelo se adapte a los cambios sin la necesidad de volverlo a entrenar entero. Esto podría ocurrir en un escenario donde recibiésemos audios de pacientes constantemente y tuviésemos que detectar cuando

un audio ha sido tos.
En segundo lugar, puede ser útil en situaciones en las que el costo de etiquetar datos es alto o el etiquetado es imposible, ya que el modelo puede aprender de forma no supervisada a partir de los datos no etiquetados.

	Sensibilidad	**Especificidad**	**Área bajo la curva ROC**	**PPV**	**NPV**	**Precisión**
3-Fold	0.86	0.90	0.95	0.90	0.87	0.88
5-Fold	0.85	0.93	0.95	0.92	0.86	0.89
10-Fold	0.86	0.93	0.96	0.92	0.87	0.89

TABLA 3.1: Media de las métricas para la detección de tos de [3]

Evaluación de un detector de tos en un entorno realista

En 2020, Carlos Hoyos et al. [78] presentaron un detector de tos para smartphone. Su funcionamiento era el siguiente:

1. Se va dividiendo el audio proveniente del micrófono en tramos de 50ms con una superposición de 25ms. Tras esto se aplica un filtro para eliminar segmentos de datos no importantes
2. Por cada espectrograma se extraen las características más importantes.
3. Se usa un clasificador basado en k-NN para detectar los eventos de tos.

El sistema creado fue validado en condiciones de laboratorio pero como estas condiciones prácticamente no existen en la vida real, la aplicación incorpora un pequeño módulo de entrenamiento que permite la adaptación a nuevos entornos. Primero se evaluó la aplicación con diferentes pacientes en entornos exteriores tras haberla entrenado en un entorno de laboratorio, provocando esto una bajada de la sensibilidad promedio a un valor de 60 módulo de entrenamiento consiguiendo aumentar la sensibilidad hasta un 85.85 Esto verifica el hecho de que el cambio de población a la hora de evaluar un detector de tos es crítico y necesitamos utilizar técnicas para conseguir reducir su impacto

Cómo podemos observar todos estos trabajos tienen que ver con la detección de tos y la utilización de diversas técnicas para conseguir un mejor rendimiento a la hora de crear un detector pero hasta la fecha no hay trabajos enfocados en el uso del aprendizaje activo con *deep learning* para clasificadores de tos

Capítulo 4

METODOLOGÍA

Como se ha comentado anteriormente, el objetivo de este trabajo es el estudio de diferentes algoritmos adaptativos en redes neuronales para resolver el problema planteado en el capítulo 3. Para ello utilizaremos los datos y estructura de red neuronal presentes en [3] y realizaremos diferentes pruebas para determinar cuáles son las muestras más relevantes de dichos datos y qué tipo de algoritmos funcionan mejor sobre ellos.

4.1 BASE DE DATOS UTILIZADA

Para nuestro estudio hemos utilizado una base de datos que contiene muestras de audio de veinte pacientes del Complejo Asistencial Universitario de Palencia con diferentes enfermedades respiratorias. Se utilizó un Sony Xperia Z2 para grabar los audios almacenando los archivos en un formato WAV de 16 bits a 44,1 kHz. A cada paciente se le indicó que guardasen dicho dispositivo como lo harían normalmente para captar las muestras en un entorno real donde podemos encontrar ruido. [3]
Un resumen de los datos que podemos encontrar en la base de datos lo podemos encontrar en la tabla 4.1

ID	Enfermedad	Edad	Sexo	Número de toses	Tipo de tos
0	Respiratoria Aguda	36	Masculino	265	Aguda
1	Bronquiectasias & Asma	23	Femenino	0	Crónica que no es EPOC
2	Sarcoidosis	37	Masculino	163	Crónica que no es EPOC
3	Respiratoria Aguda	53	Masculino	352	Aguda
4	Respiratoria Aguda	18	Femenino	602	Aguda
5	EPOC	79	Masculino	767	EPOC
6	Cáncer de pulmón	84	Masculino	921	Cáncer de pulmón
7	Neumonía bilateral	44	Masculino	2401	Aguda
8	Cáncer de pulmón	62	Masculino	809	Cáncer de pulmón
9	Apnea Obstructiva del sueño	70	Femenino	1268	
10	Bronquiectasias	63	Femenino	508	Crónica que no es EPOC
11	EPOC y Apnea Obstructiva del sueño	48	Femenino	586	
12	EPOC	69	Femenino	901	
13	Neumonía unilobular	31	Femenino	1579	Aguda
14	Neuomonía ATÍPICA	83	Masculino	504	Aguda
15	Neumonía y EPOC	87	Masculino	1166	
16	Tromboembolismo pulmonar	70	Femenino	917	
17	EPOC	50	Femenino	2033	EPOC
18	Asma	70	Femenino	697	Crónica que no es EPOC
19	EPOC y Hamartoma pulmonar	60	Masculino	615	
20	EPOC y Fibrosis pulmonar	67	Masculino	1379	

TABLA 4.1: Base de datos utilizada para realizar los experimentos. [3]

4.1.1 PROCESADO DE DATOS

Los audios pertenecientes a esta base de datos tienen un problema ya que los clips de audio donde aparece una tos habitualmente tienen una duración menor a un segundo mientras que los clips donde no hay toses suelen tener una duración mayor a un segundo. Esto podría provocar que nuestra red interpretase como una característica muy relevante la duración y basase sus predicciones en la misma consiguiendo de este modo tener un sistema sesgado. En la figura 4.1 podemos observar como este sesgo existe también en el espectrograma de los clips.

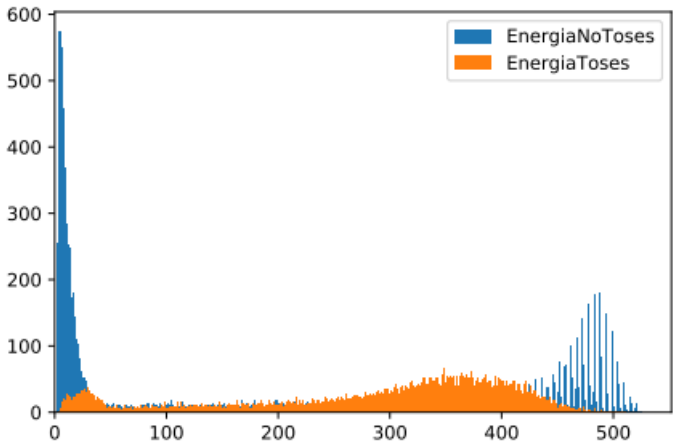


FIGURA 4.1: Energía de los espectrogramas de los clips de audio sin recortar a 1 segundo. [3]

Para evitarlo, se han configurado todos los clips de audio para que su duración sea de 1 segundo del siguiente modo:

- Primero se utiliza el *software* Praat que nos permite etiquetar manualmente en que momento comienza una tos y gracias a él generaremos los nuevos clips empezando cada nuevo clip en el momento en el que Praat indique que comienza una tos y finalizando 1 segundo después
- Para los clips de audio sin tos concatenaremos todos los clips de audio sin tos utilizando Praat y después lo dividiremos en segmentos de audio de 1 segundo.

Con esto habremos eliminado el sesgo por la duración de los audios y observando la figura 4.2 vemos cómo también hemos eliminado el sesgo de los espectrogramas ya que aunque haya diferencias entre los niveles de energías para los clips con toses y sin ellas, esto se debe a una característica del audio en sí.

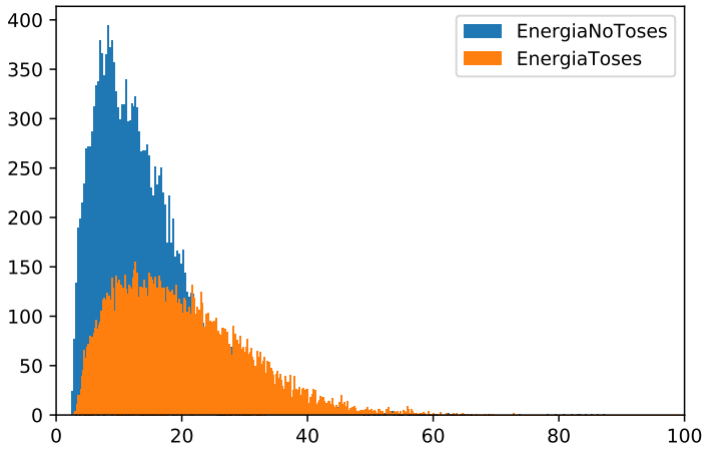


FIGURA 4.2: Energía de los espectrogramas de los clips de audio recortados a 1 segundo. [3]

Tras esto, se diezma la señal desde 44.1 kHz hasta 8.82 kHz para reducir la carga computacional y el espectrograma final será introducido en forma de imagen a nuestra red neuronal[1]

[1]Proceso extraido de [3]

4.2 RED NEURONAL UTILIZADA

Hemos utilizado la red de la figura 4.3. Esta red tiene una primera capa convolucional de 32 filtros con un kernel (2,2), su función de activación es una *reLu* y admite una entrada de la forma (45,100,1) puesto que los espectrogramas a analizar tienen un tamaño de (45,100) y debemos añadir una tercera dimensión al ser una capa convolucional. Después podemos observar una capa de *MaxPooling2D*, en esta capa se reducen las dimensiones de nuestros datos para no tener una gran carga computacional. Tras esta, tenemos una capa de *dropout* utilizada para reducir el sobreajuste de la red ya que provoca que durante un entrenamiento se ignoren un cierto número de salidas de la capa aleatoriamente. El resto de capas *MaxPooling2D* y *dropout* tienen las mismas características pero vamos aumentando el número de filtros en un factor 2. La capa *flatten* se encarga de aplanar nuestros datos para pasarlos a la siguiente capa donde tenemos 512 neuronas y una función de activación *relu* y por úlitmo la capa de salida, una capa densa de dos neuronas con función de activación *softMax* que nos permitirá diferenciar entre los clips con tos y sin ella. Después de realizar varias simulaciones con esta estructura de red neuronal, decidimos eliminar las capas de *dropout* consiguiendo mayor balanceo de sensibilidad y especificidad de la red[2] .

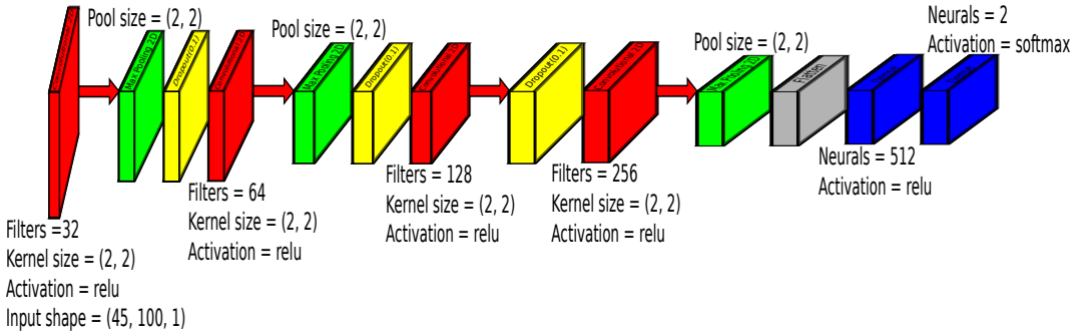


FIGURA 4.3: Estructura de la red neuronal. [3]

[2]Red extraida de [3]

4.3 ENTRENAMIENTO DE LA RED NEURONAL

Para que nuestra red neuronal sea capaz de clasificar los datos de entrada entre las clases `"Tos"` y `"No Tos"` debemos entrenarla. Durante este entrenamiento podemos modificar ciertos parámetros de la red para intentar mejorar el desempeño de la misma, se han seleccionado los siguientes:

- **Función de pérdida**: *Categorical crossentropy*
- **Función de activación**: *adaMax*
  - $\alpha = 0{,}002$
  - $\beta_1 = 0{,}9$
  - $\beta_2 = 0{,}999$
- **Métricas de evaluación**: utilizaremos las matrices de confusión y el área bajo las curvas ROC
- **Número de *epoch* utilizado en la red**: 50
- **Tamaño de *batch***: 128
- **Porcentaje de datos de entrenamiento utilizados para validación**: 20

Para utilizar el OHBP debemos determinar una tasa de aprendizaje, hemos utilizado la propuesta en [71]:

- **Learning Rate**: 0.01

Capítulo 5

RESULTADOS Y DISCUSIÓN

A continuación detallaremos los experimentos realizados durante el TFG. Nuestro primer objetivo fue determinar cuales eran los pacientes más conflictivos para nuestro modelo ya que después los utilizaremos para probar el funcionamiento de algoritmos adaptativos. Denominamos como pacientes conflictivos aquellos que cuando se encuentran en el grupo de testeo provocan una bajada del rendimiento del sistema.

5.1 VALIDACIÓN CRUZADA K-FOLD

En primer lugar se realizaron simulaciones con el método `"Validación cruzada K-Fold"`. Aunque este método divide los datos de forma aleatoria, para nuestras primeras simulaciones realizamos esa división de manera manual replicando los grupos utilizados en [3] (estos grupos son balanceados y proporcionan buenos resultados). Utilizamos un 3-Fold dividiendo nuestros pacientes del siguiente modo:

Fold 1: pacientes 3, 7, 11, 12, 15, 18
Fold 2: pacientes 4, 5, 9, 14, 16, 17
Fold 3: pacientes 0, 2, 6, 8, 10, 13, 19

Aunque cuando entrenamos con los Folds 1 y 2 y testeamos con el 3 los resultados son muy buenos, los resultados de testear tanto con el Fold 1 como con el Fold 2 no son los más óptimos pues tal y como podemos observar en su sensibilidad y NPV, ambos tienen problemas a la hora de detectar `"No Tos"` (tabla 5.1).

Testeo	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Fold 1	0.66	0.90	0.84	0.94	0.52	0.73
Fold 2	0.73	0.86	0.85	0.89	0.68	0.78
Fold 3	0.90	0.86	0.94	0.86	0.91	0.89

TABLA 5.1: Métricas para 3-Fold

Tras esto realizamos el experimento inverso, dividiendo nuestros pacientes en 5 grupos balanceados (extraídos de [3]), entrenando nuestra red con uno de los grupos y testeándola con los cuatro restantes:

Fold 1: pacientes 0, 3, 5, 7
Fold 2: pacientes 2, 4, 12, 17
Fold 3: pacientes 10, 13, 16, 19
Fold 4: pacientes 6, 8, 11, 20
Fold 5: pacientes 9, 14, 15, 18

obteniendo los resultados de las tabla :

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Fold 1	0.75	0.70	0.80	0.67	0.78	0.72
Fold 2	0.84	0.70	0.85	0.63	0.88	0.75
Fold 3	0.82	0.81	0.90	0.81	0.83	0.82
Fold 4	0.82	0.86	0.91	0.87	0.81	0.84
Fold 5	0.81	0.82	0.89	0.82	0.81	0.81

TABLA 5.2: Métricas para 5-Fold

Para los Folds 3, 4 y 5 las curvas ROC y métricas son buenas pero si observamos los resultados con el Fold 1 y Fold 2 nuestra red no obtiene un gran rendimiento, siendo su principal problema la detección de `"Tos"` Después de estas pruebas y de hacer varias simulaciones más introduciendo y eliminando pacientes a la hora de entrenar podemos observar como el paciente 12 y el paciente 7 son conflictivos para nuestra red puesto que siempre que están presentes en el grupo de testeo y no en el de entrenamiento, el rendimiento de la misma disminuye. Para probar esto, hemos procedido a entrenar la red con el resto de pacientes y testearla solo con el paciente 12 y 7 pero los resultados obtenidos han sido buenos (Áreas bajo la ROC de 0.98).

5.2 DIFERENCICACIÓN POR TOSES

Ahora diferenciaremos a los pacientes por los tipos de tos que padecen, testeando nuestra red con los pacientes de una enfermedad concreta y entrenándola con el resto, los dividiremos del siguiente modo:

Tos aguda: pacientes 0, 3, 4, 7, 13, 14
Tos crónica que no es EPOC: pacientes 2, 10, 18
EPOC: pacientes 5, 12, 17
Cáncer de pulmón: pacientes 6, 8
No clasificada: pacientes 9, 11, 15, 16, 19, 20

Los resultados obtenidos los podemos observar en la tabla 5.3:

Testeo	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Aguda	0.93	0.91	0.97	0.91	0.93	0.92
Crónica que no es EPOC	0.92	0.97	0.96	0.89	0.92	0.90
EPOC	0.85	0.85	0.91	0.85	0.85	0.85
Cáncer de pulmón	0.95	0.85	0.95	0.85	0.95	0.90
No clasificada	0.85	0.89	0.95	0.89	0.85	0.87

TABLA 5.3: Curvas ROC testeando con cada tipo de Tos

Tras realizar esto vemos que de todas las enfermedades con las que se testea, la que tiene unas peores métricas es el EPOC (tabla 5.3).

5.2.1 PACIENTES CON EPOC

Vamos a centrarnos en dichos pacientes y procedemos a dividirlos entre hombres y mujeres para entender si hay diferencias significativas por género:

Hombres con EPOC: pacientes 5, 19 y 20
Mujeres con EPOC: pacientes 11, 12 y 17

Si entrenamos nuestra red con el grupo de mujeres que padezcan EPOC y la testeamos con los hombres que padecen dicha enfermedad y viceversa obtendremos la tabla 5.4

Testeo	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Hombres	0.73	0.75	0.81	0.75	0.73	0.74
Mujeres	0.86	0.67	0.84	0.72	0.83	0.75

TABLA 5.4: Curvas ROC entre hombres y mujeres con EPOC

Como podemos observar, las métricas obtenidas no son tan buenas como cabría esperar entre pacientes que padecen la misma enfermedad. Esto nos indica que puede haber diferencias significativas entre los espectrogramas de hombres y mujeres con EPOC.

5.2.2 HOMBRES Y MUJERES CON EPOC

Tras observar que los pacientes con EPOC son los pacientes que nuestra red clasifica con mayor dificultad y que hay cierta diferencia entre hombres y mujeres con EPOC, el objetivo del siguiente experimento es determinar si alguno de los pacientes en concreto es muy problemático para el testeo de la red. Para ello, utilizaremos todos los hombres con EPOC para entrenar excepto uno, que se utilizará para testear y repetiremos esto mismo con mujeres (tablas 5.5 y 5.6)

Testeo	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Paciente 5	0.87	0.54	0.80	0.65	0.81	0.70
Paciente 19	0.73	0.71	0.79	0.71	0.72	0.72
Paciente 20	0.80	0.68	0.79	0.72	0.77	0.74

TABLA 5.5: Matrices de confusión para hombres con EPOC

Testeo	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Paciente 11	0.93	0.41	0.84	0.61	0.85	0.67
Paciente 12	0.86	0.72	0.88	0.75	0.84	0.79
Paciente 17	0.59	0.79	0.79	0.74	0.66	0.69

TABLA 5.6: Curva ROC para mujeres con EPOC

Tras ver los resultados podemos dictaminar que para nuestra red es difícil identificar `"No Tos"` tanto de mujeres como de hombres con EPOC (es posible que los entornos donde se encontrasen estos pacientes fuesen ruidosos). Especialmente las de la paciente 11 puesto que nuestra red se equivoca un 59 % de las ocasiones cuando en el audio no aparece una tos y en las del paciente 5, donde obtenemos un error del 46 % cuando en su audio no aparece tos. A la hora de detectar `"Tos"` nuestra red funciona correctamente excepto para la paciente 17 dónde tenemos un 41 % de errores.
En experimentos posteriores se utilizarán los pacientes identificados hasta este punto como conflictivos para ayudarnos a determinar cómo se comporta un modelo.

5.3 Active Learning

Ahora nos ayudaremos de algoritmos adaptativos para intentar mejorar el desempeño de nuestra red. Empezaremos utilizando *Active Learning*, para ello, entrenaremos nuestra red con unos archivos de audio base (100 audios, 500 audios y 1000 audios) escogidos aleatoriamente de nuestra base de datos. Después iremos aumentando de 10 en 10 los audios utilizados para el entrenamiento siendo estos 10 audios añadidos los 10 peores audios en el set de testeo obtenidos en la iteración anterior (se añade al entrenamiento el audio y su etiqueta correcta) en términos de *Least Confidence* y *Margin Sampling* (en este caso son iguales pues solo hay 2 clases para etiquetar) y volveremos a entrenar la red entera. Los resultados se presentan en las tablas 5.7, 5.8 y 5.9

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
100	0.80	0.76	0.84	0.77	0.79	0.78
110	0.73	0.80	0.85	0.79	0.75	0.77
120	0.77	0.80	0.86	0.79	0.77	0.79
130	0.83	0.77	0.87	0.78	0.82	0.80
140	0.83	0.78	0.87	0.79	0.82	0.81
150	0.82	0.81	0.88	0.81	0.82	0.81
160	0.80	0.83	0.89	0.82	0.80	0.81

TABLA 5.7: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento. Comienzo en 100 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
500	0.82	0.75	0.85	0.77	0.80	0.79
510	0.81	0.84	0.89	0.83	0.81	0.82
520	0.82	0.83	0.90	0.83	0.83	0.83
530	0.84	0.82	0.91	0.83	0.84	0.83
540	0.83	0.85	0.92	0.85	0.83	0.84
550	0.82	0.87	0.92	0.86	0.83	0.85
560	0.84	0.85	0.92	0.85	0.85	0.85

TABLA 5.8: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento. Comienzo en 500 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
1000	0.82	0.80	0.88	0.80	0.81	0.81
1010	0.82	0.87	0.91	0.83	0.83	0.84
1020	0.85	0.85	0.92	0.85	0.85	0.85
1030	0.86	0.87	0.93	0.86	0.86	0.86
1040	0.84	0.89	0.94	0.88	0.85	0.86
1050	0.84	0.87	0.94	0.89	0.85	0.87
1060	0.85	0.89	0.94	0.88	0.86	0.87

TABLA 5.9: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento. Comienzo en 1000 audios

Como podemos observar, utilizando esta técnica podemos mejorar en gran medida el rendimiento de nuestro sistema con la incorporación al entrenamiento de muestras que son muy relevantes para nuestra red. El valor de todas las métricas aumenta a medida que añadimos nuevos datos llegando a mejorar el área bajo la curva ROC hasta 7 puntos, la especificidad 3 puntos y la precisión 6 puntos. Podemos conseguir así una red con gran rendimiento pero necesitamos cumplir la condición de poder etiquetar manualmente las muestras añadidas en cada iteración al entrenamiento, algo que no siempre es posible.

5.4 ACTIVE LEARNING Y TRANSFER LEARNING

Ahora añadiremos un método de *Transfer Learning*. Para ello repetiremos el experimento de la sección 5.3, utilizando como punto de partida los 100, 500 y 1000 audios utilizados en dicha sección pero en este caso congelaremos todas las capas excepto las 2 últimas cada vez que entrenemos la red llegando a los resultados de las tablas (5.10, 5.11 y 5.12):

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
100	0.80	0.76	0.84	0.77	0.79	0.78
110	0.93	0.52	0.85	0.66	0.89	0.73
120	0.87	0.69	0.85	0.74	0.84	0.78
130	0.36	0.94	0.83	0.86	0.60	0.65
140	0.85	0.72	0.85	0.75	0.83	0.79
150	0.76	0.80	0.85	0.79	0.77	0.78
160	0.79	0.80	0.86	0.80	0.77	0.79

TABLA 5.10: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento congelando todas las capas de la red excepto las 2 últimas. Comienzo en 100 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
500	0.82	0.75	0.85	0.77	0.80	0.79
510	0.88	0.70	0.87	0.75	0.85	0.79
520	0.77	0.82	0.87	0.81	0.78	0.79
530	0.83	0.78	0.88	0.78	0.82	0.80
540	0.78	0.82	0.87	0.81	0.79	0.80
550	0.76	0.83	0.88	0.82	0.78	0.80
560	0.79	0.82	0.88	0.81	0.79	0.80

TABLA 5.11: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento congelando todas las capas de la red excepto las 2 últimas. Comienzo en 500 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
1000	0.82	0.80	0.88	0.80	0.81	0.81
1010	0.83	0.85	0.90	0.85	0.83	0.84
1020	0.77	0.89	0.91	0.88	0.79	0.83
1030	0.79	0.88	0.91	0.87	0.80	0.83
1040	0.76	0.89	0.92	0.88	0.79	0.83
1050	0.85	0.84	0.92	0.84	0.85	0.84
1060	0.79	0.88	0.92	0.87	0.81	0.84

TABLA 5.12: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento congelando todas las capas de la red excepto las 2 últimas. Comienzo en 1000 audios

Como podemos observar, el desempeño de nuestro sistema es menor cuando congelamos las primeras capas. Esto se debe a que utilizando esta técnica, tenemos una menor capacidad de adaptación y cambio pues los pesos de todas las capas excepto las 2 últimas no pueden modificarse. Por contraparte ganamos velocidad de entrenamiento ya que al tener que calcular menos pesos, nuestra red se entrena más rápido.

5.5 ACTIVE LEARNING SIN ETIQUETAS HUMANAS

Ahora repetiremos el experimento de la sección 5.3, utilizando como punto de partida los 100, 500 y 1000 audios utilizados en dicha sección pero en este caso solo dispondremos de etiquetas para los conjuntos iniciales de datos. Esto provocará que los 10 audios que añadamos a nuestro conjunto de entrenamiento provenientes del conjunto de test, serán añadidos con la etiqueta que nuestra red neuronal le haya adjudicado (que puede ser o no su etiqueta correcta) obteniendo las tablas (5.13, 5.14 y 5.15):

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
100	0.80	0.76	0.84	0.77	0.79	0.78
110	0.51	0.90	0.84	0.84	0.65	0.70
120	0.57	0.89	0.84	0.83	0.67	0.73
130	0.58	0.89	0.85	0.84	0.68	0.73
140	0.73	0.84	0.86	0.82	0.75	0.78
150	0.77	0.83	0.87	0.82	0.78	0.80
160	0.81	0.83	0.89	0.82	0.81	0.82

TABLA 5.13: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento (su etiqueta es la dictaminada por la red neuronal y puede que no sea correcta). Comienzo en 100 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
500	0.82	0.75	0.85	0.77	0.80	0.79
510	0.83	0.80	0.88	0.82	0.83	0.84
520	0.82	0.86	0.91	0.86	0.83	0.84
530	0.87	0.82	0.92	0.83	0.86	0.85
540	0.82	0.87	0.92	0.86	0.83	0.84
550	0.83	0.86	0.92	0.85	0.83	0.84
560	0.85	0.85	0.92	0.85	0.85	0.85

TABLA 5.14: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento (su etiqueta es la dictaminada por la red neuronal y puede que no sea correcta). Comienzo en 500 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
1000	0.82	0.80	0.88	0.80	0.81	0.81
1010	0.82	0.86	0.92	0.86	0.83	0.84
1020	0.81	0.88	0.92	0.87	0.82	0.84
1030	0.82	0.88	0.92	0.87	0.83	0.85
1040	0.81	0.89	0.92	0.88	0.82	0.85
1050	0.80	0.90	0.93	0.89	0.82	0.85
1060	0.83	0.89	0.93	0.88	0.84	0.86

TABLA 5.15: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 peores muestras) para añadir nuevos audios al entrenamiento (su etiqueta es la dictaminada por la red neuronal y puede que no sea correcta). Comienzo en 1000 audios

Podemos observar como en la tabla 5.13 tenemos una gran bajada de sensibilidad y NPV a partir de la segunda iteración y nos cuesta tres iteraciones recuperar niveles anteriores, esto puede deberse a que nuestro modelo haya etiquetado mal varias muestras de `"Tos"` muy relevantes para el sistema y las haya utilizado para entrenar la red, obteniendo resultados no deseados ya que aunque el área bajo la curva ROC mantiene buenos niveles, nuestro detector de toses tendría una clara tendencia al Falso Positivo.

Ahora repetiremos el experimento anterior pero utilizando los 10 audios con menor *Least Confidence* y *Margin Sampling* (los audios que el sistema está más seguro de haber clasificado bien) para añadir al entrenamiento de la siguiente iteración. Utilizaremos de nuevo la etiqueta elegida por nuestro sistema en vez de la etiqueta correcta.

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
100	0.80	0.76	0.84	0.77	0.79	0.78
110	0.73	0.80	0.83	0.79	0.75	0.77
120	0.76	0.82	0.86	0.80	0.77	0.79
130	0.75	0.85	0.87	0.83	0.77	0.80
140	0.71	0.89	0.89	0.87	0.75	0.80
150	0.67	0.92	0.90	0.89	0.73	0.79
160	0.68	0.91	0.90	0.89	0.74	0.80

TABLA 5.16: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 mejores muestras) para añadir nuevos audios al entrenamiento (su etiqueta es la dictaminada por la red neuronal y puede que no sea correcta). Comienzo en 100 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
500	0.82	0.75	0.85	0.77	0.80	0.79
510	0.82	0.79	0.88	0.80	0.82	0.81
520	0.83	0.85	0.91	0.85	0.83	0.84
530	0.73	0.91	0.91	0.89	0.77	0.82
540	0.76	0.90	0.92	0.88	0.79	0.83
550	0.80	0.87	0.92	0.87	0.82	0.84
560	0.81	0.87	0.92	0.86	0.82	0.84

TABLA 5.17: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 mejores muestras) para añadir nuevos audios al entrenamiento (su etiqueta es la dictaminada por la red neuronal y puede que no sea correcta). Comienzo en 500 audios

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
1000	0.82	0.80	0.88	0.80	0.81	0.81
1010	0.51	0.97	0.91	0.95	0.66	0.74
1020	0.55	0.97	0.91	0.95	0.68	0.76
1030	0.63	0.96	0.91	0.94	0.72	0.79
1040	0.70	0.93	0.91	0.91	0.76	0.82
1050	0.75	0.90	0.91	0.88	0.78	0.82
1060	0.76	0.90	0.92	0.88	0.79	0.83

TABLA 5.18: Resultado de las métricas utilizando *Least Confidence* y *Margin Sampling* (10 mejores muestras) para añadir nuevos audios al entrenamiento (su etiqueta es la dictaminada por la red neuronal y puede que no sea correcta). Comienzo en 1000 audios

Como podemos ver, los resultados son buenos en términos de área bajo la curva ROC pero nuestro sistema pierde rendimiento en términos de sensibilidad y NPV, teniendo una menor capacidad de detectar `"Tos"`. Esto es provocado porque si añadimos al entrenamiento muestras que nuestro sistema está seguro de detectar bien, apenas estaremos proporcionando información relevante para mejorar la red y este no será capaz de adaptarse a nuevas muestras.

5.6 ONLINE DEEP LEARNING

Por último utilizaremos el algoritmo explicado en la sección 2.3.3.1 para comprobar el funcionamiento del *online learning*, hemos utilizado el repositorio de github [79] para implementar el código.

Este algoritmo parte de cero y se va entrenando y actualizando de forma constante con los datos que le vamos proporcionando. Hemos utilizado todos los datos pertenecientes a la base de datos de la sección 4.1 para realizar este entrenamiento además de fijar el tamaño de capas de la red a 5 (hemos comprobado que subiendo el número de capas no aumenta el rendimiento pero sí aumenta el tiempo de cómputo). Los resultados se expondrán en tablas que contendrán las métricas más relevantes en varios segmentos de los datos para ver cómo el algoritmo va mejorando o empeorando su rendimiento mientras entrena a través de toda la base de datos y el

"`Acumulado total`" que contiene las métricas obtenidas midiendo la totalidad de los datos y no las métricas separadas por segmentos.

Para el primer experimento, hemos introducido todos los datos en orden aleatorio al algoritmo obteniendo los resultados de la tabla 5.19.

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Segmento 0-0.5 %	0.65	0.50	0.57	0.58	0.57	0.58
Segmento 10-15 %	0.71	0.71	0.71	0.70	0.71	0.71
Segmento 30-40 %	0.81	0.79	0.80	0.80	0.80	0.80
Segmento 40-50 %	0.81	0.80	0.80	0.79	0.81	0.80
Segmento 60-80 %	0.82	0.82	0.82	0.82	0.82	0.82
Acumulado total	0.79	0.77	0.78	0.78	0.79	0.78

TABLA 5.19: Métricas para el OHBP con 5 capas de profundidad

Como podemos observar, las métricas son menores a las del resto de experimentos. Que este algoritmo no sea tan preciso como los anteriores se debe a que el objetivo del mismo es ser capaz de mantener un aprendizaje *online* actualizándose los pesos de la red con cada dato nuevo, consiguiendo ser muy versátil y adaptativo a nuevos datos pero perdiendo rendimiento (siendo aún así un sistema muy balanceado pues todas las métricas son similares y no está desbalanceado a la hora de detectar casos positivos o negativos en su acumulado total). Vemos como introduciendo los datos al entrenamiento de modo aleatorio vamos mejorando en todas las métricas a medida que vamos avanzando. Esto es un caso ideal pues el entrenar de este modo provoca que el modelo sea alimentado con información muy relevante ya que utiliza muestras de todos los pacientes de una manera constante. Para ver cómo responde el modelo a entrenar primero con muestras de pacientes conflictivos, hemos entrenado el algoritmo introduciéndole primero las muestras de los pacientes 17, 19, 11, 5, 12 y 7 (los cuales hemos identificado como más conflictivos en secciones anteriores) y después el resto de muestras obteniendo los resultados de la tabla 5.20 (las muestras de estos pacientes conflictivos se introducen aleatoriamente, no están ordenadas por paciente).

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Segmento 0-0.5 %	0.53	0.66	0.59	0.60	0.60	0.60
Segmento 10-15 %	0.74	0.63	0.67	0.67	0.71	0.69
Segmento 30-40 %	0.87	0.85	0.86	0.85	0.87	0.86
Segmento 40-50 %	0.82	0.79	0.79	0.79	0.81	0.80
Segmento 60-80 %	0.83	0.80	0.81	0.81	0.83	0.82
Acumulado total	0.80	0.77	0.78	0.79	0.79	0.79

TABLA 5.20: Métricas para el OHBP con 5 capas de profundidad introduciendo al principio los datos de pacientes conflictivos

Cómo vemos el modelo mejora ligeramente en el acumulado total pero es en el segmento 30-40 % donde observamos una gran mejoría, esto ocurre porque en este segmento se encuentra el final de los datos de pacientes conflictivos y nuestro sistema (tras entrenar previamente con datos de esos pacientes) es capaz de adaptarse mucho mejor a ellos (aunque en el segmento inicial se haya perdido rendimiento tal y cómo podemos ver). Cuando empezamos a introducir el resto de datos, tenemos una bajada de rendimiento volviendo a valores cercanos a los manejados en

la tabla 5.19.
Ahora probaremos cómo se comporta el sistema dejando los datos de pacientes más conflictivos para el final llegando a los resultados de la tabla 5.21

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Segmento 0-0.5 %	0.66	0.49	0.58	0.57	0.59	0.58
Segmento 10-15 %	0.75	0.71	0.73	0.73	0.73	0.73
Segmento 30-40 %	0.78	0.78	0.77	0.77	0.78	0.78
Segmento 40-50 %	0.80	0.80	0.80	0.81	0.79	0.80
Segmento 60-80 %	0.86	0.87	0.86	0.87	0.86	0.86
Acumulado total	0.80	0.77	0.78	0.78	0.79	0.78

TABLA 5.21: Métricas para el OHBP con 5 capas de profundidad introduciendo al final los datos de pacientes conflictivos

Los resultados acumulados totales son muy similares al experimento anterior pero vemos como en este caso llegamos al mejor rendimiento de la red cuando nos encontramos en el segmento 60-80 %, esto es debido a que en este segmento se encuentran los datos de los pacientes más conflictivos. Podríamos pensar que el rendimiento decaería pero el modelo es capaz de adaptarse a los nuevos datos gracias al entrenamiento previo.

Los resultados obtenidos hasta aquí indican que el OHBP es capaz de adaptarse en diferentes situaciones cuando los datos van llegando sin estar ordenados por paciente (aunque si que hayamos hecho un subgrupo de pacientes conflictivos, este subgrupo no seguía un orden si no que se encontraban todos los datos de dichos pacientes colocados de forma aleatoria), este sería un caso bastante ideal donde recibimos información relevante de muchos pacientes constantemente, ahora probaremos como se adapta agrupando todos los datos por pacientes.

Para el siguiente experimento iremos entrenando nuestro modelo de paciente en paciente (el orden en el que se introducen al algoritmo es aleatorio) obteniendo la tabla 5.22

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Segmento 0-0.5 %	0.64	0.52	0.58	0.57	0.59	0.58
Segmento 10-15 %	0.68	0.63	0.65	0.64	0.67	0.65
Segmento 30-40 %	0.78	0.75	0.76	0.76	0.77	0.77
Segmento 40-50 %	0.79	0.75	0.77	0.75	0.79	0.77
Segmento 60-80 %	0.85	0.84	0.84	0.84	0.84	0.84
Acumulado total	0.79	0.75	0.77	0.76	0.78	0.77

TABLA 5.22: Métricas para el OHBP con 5 capas de profundidad entrenando por pacientes

Comparando las dos tablas (5.19 y 5.22), se puede ver que hay algunas diferencias en las métricas para cada segmento y para el acumulado total. En general, el modelo entrenado por pacientes (tabla 5.22) parece tener una sensibilidad ligeramente mejor y una especificidad ligeramente peor que el modelo entrenado de forma aleatoria (tabla 5.19). Además, el modelo entrenado por pacientes parece tener un área bajo la curva ROC y una precisión ligeramente mejores pero ambos modelos son muy parecidos y no encontramos diferencias significativas. Procedemos ahora a utilizar los pacientes conflictivos que hemos utilizado antes para entrenar en primer

lugar (en este caso los datos se introducen por orden de paciente más conflictivo) obteniendo la tabla 5.23.

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Segmento 0-0.5 %	0.54	0.66	0.60	0.60	0.60	0.60
Segmento 10-15 %	0.74	0.62	0.68	0.67	0.70	0.68
Segmento 30-40 %	0.86	0.85	0.85	0.86	0.86	0.86
Segmento 40-50 %	0.81	0.77	0.79	0.77	0.81	0.79
Segmento 60-80 %	0.81	0.85	0.83	0.84	0.82	0.83
Acumulado total	0.79	0.78	0.79	0.78	0.79	0.79

TABLA 5.23: Métricas para el OHBP con 5 capas de profundidad entrenando primero con los datos de pacientes más conflictivos (se introducen los datos ordenados por paciente)

Cómo podemos ver, los resultados son muy similares a los obtenidos en la tabla 5.21 así que ahora probaremos si hay diferencias al introducir los pacientes más significativos al final del entrenamiento obteniendo la tabla 5.24

Entrenamiento	Sensibilidad	Especificidad	Área bajo la curva ROC	PPV	NPV	Precisión
Segmento 0-0.5 %	0.65	0.50	0.57	0.58	0.57	0.58
Segmento 10-15 %	0.71	0.71	0.71	0.70	0.71	0.71
Segmento 30-40 %	0.81	0.79	0.80	0.80	0.80	0.80
Segmento 40-50 %	0.81	0.80	0.80	0.79	0.81	0.80
Segmento 60-80 %	0.82	0.82	0.82	0.82	0.82	0.82
Acumulado total	0.79	0.77	0.78	0.78	0.79	0.78

TABLA 5.24: Métricas para el OHBP con 5 capas de profundidad entrenando al final con los datos de pacientes más conflictivos (se introducen los datos ordenados por paciente)

Al comparar los valores de cada métrica en las dos tablas (5.21 y 5.24), se puede observar que el modelo de la tabla 5.21 parece tener un mejor rendimiento en general respecto al de la tabla 5.24. Tiene una mayor sensibilidad en la mayoría de los segmentos, lo que significa que es más probable que identifique una `"Tos"` de forma correcta. También tiene una mayor especificidad en varios segmentos, lo que significa que es más probable que identifique una `"No tos"`. Además, tiene un área bajo la curva ROC ligeramente mayor en la mayoría de los segmentos de prueba y en el acumulado total, lo que indica un mejor rendimiento general del modelo en la discriminación entre toses y no toses. El resto de métricas son muy similares tanto para los segmentos como para el total acumulado pero si observamos el segmento 60-80 % podemos ver cómo las métricas de la tabla 5.24 no son tan buenas como las de 5.21 lo cual denota que este algoritmo tiene mejor comportamiento cuando recibe datos de los pacientes más conflictivos y estos datos están colocados de forma aleatoria respecto a recibirlos de un modo ordenado por paciente.

Capítulo 6

CONCLUSIONES Y LÍNEAS FUTURAS

6.1 CONCLUSIONES

Los algoritmos de aprendizaje adaptativo han demostrado ser una herramienta muy útil en el campo de la Inteligencia Artificial y el *Machine Learning*. Estos algoritmos son capaces de aprender y mejorar con el tiempo, ajustando su comportamiento y su capacidad de respuesta para adaptarse a las nuevas situaciones y a los cambios en los datos que procesan.

Una de las grandes ventajas de los algoritmos de aprendizaje adaptativo es que pueden alcanzar grandes resultados con una cantidad relativamente pequeña de datos de entrenamiento. Esto es particularmente relevante en aplicaciones donde la cantidad de datos de entrenamiento es limitada o la obtención de los mismos es una tarea complicada, como en el caso del reconocimiento de voz o de la clasificación de imágenes.

En el presente trabajo, hemos simulado una situación de *Active Learning* observando como al añadir un número reducido de muestras a los datos de entrenamiento (utilizando *Least Confidence* y *Margin Sampling* como puntuación de confianza) cada vez que volvemos a entrenar un modelo podemos conseguir mejorar el desempeño del mismo de una manera muy significativa. Por ejemplo, hemos conseguido aumentar el área de la curva ROC, la especificidad y la precisión hasta en 9, 10 y 7 puntos respectivamente añadiendo solo 60 muestras a nuestro set de entrenamiento. También hemos usado la técnica de *Transfer Learning* que consiste en congelar las capas más profundas de la red cada vez que volvemos a entrenar un modelo viendo cómo esto aporta una mayor velocidad a la hora de realizar los cálculos del modelo pero no consigue mejorar el rendimiento de una manera tan significativa cómo lo hace el *Active Learning* tradicional, consiguiendo una mejora del área de la curva ROC, la especificidad y la precisión de hasta 4, 8 y 3 puntos respectivamente.

Otro de los métodos utilizados ha sido el *Active Learning* pero utilizando para volver a entrenar nuestro modelo las etiquetas que el propio sistema asignaba a las muestras elegidas. Los resultados obtenidos denotan que aunque también conseguimos una mejora del modelo, podemos sufrir pérdida de rendimiento en ciertos momentos ya que es posible que muchos de los datos con los que re-entrenemos el modelo sean erróneos. También podemos ver que si utilizamos para re-entrenar las muestras cuyo nuestro sistema ha estado más seguro de predecir bien (las peores

en términos de *Least Confidence* y *Margin Sampling*), el sistema pierde rendimiento en términos de sensibilidad y NPV, teniendo una menor capacidad de detectar "Tos". Esto es provocado porque si añadimos al entrenamiento muestras las cuales nuestro sistema está seguro de detectar bien, apenas estaremos proporcionando información relevante para mejorar el modelo y este no será capaz de adaptarse a nuevas muestras

Otro de los métodos que hemos implementado en este TFG es el *Online Learning*, que consiste en ir actualizando un modelo de clasificación a medida que se van presentando nuevos datos, en lugar de entrenar el modelo con un conjunto fijo de datos y luego utilizarlo para clasificar nuevas imágenes. De este modo, conseguimos un modelo capaz de adaptarse a nuevas situaciones y aprender de forma continua.

Sin embargo, es importante tener en cuenta que el rendimiento del aprendizaje *online* puede ser difícil de mejorar una vez que se ha alcanzado un cierto nivel. Esto se debe a que nuestro sistema debe poder adaptarse a nuevas imágenes pero a su vez debe seguir pudiendo interpretar y entender las imágenes vistas anteriormente, esto provoca que nuestro sistema no tenga un rendimiento tan alto como el de otras técnicas basadas en *batch learning* pero que sea capaz de generalizar mejor para nuevos datos (escenario en el cual las técnicas de *batch learning* no son tan eficientes)

En concreto hemos implementado el algoritmo de la sección 2.3.3 con una profundidad máxima de 5 capas en el cual hemos observado como en las etapas iniciales se adapta mejor a ser entrenado con datos aleatorios frente a ser entrenado por pacientes (podemos ver una mejoría en el segmento 10-15 % de 3, 6 y 6 puntos en sensibilidad, área bajo curva ROC y precisión respectivamente) mientras que en las etapas finales consigue un rendimiento ligeramente mejor entrenando paciente por paciente (podemos ver una mejoría en el segmento 60-80 % de 3, 2 y 2 puntos en sensibilidad, área bajo curva ROC y precisión respectivamente). Este algoritmo es capaz de adaptarse a cualquier tipo de situación (datos ordenados de forma aleatoria, ordenados por paciente, ordenados por pacientes más conflictivos...) llegando a tener unas métricas finales muy similares para todos los experimentos (sensibilidad sobre 0.79, especificidad sobre 0.78, área bajo curva roc sobre 0.78 y precisión sobre 0.78) siendo muy balanceado y sin tendencia a dar falsos positivos (uno de los grandes problemas de un clasificador).

6.2 Limitaciones y Líneas Futuras

En el ámbito de la Inteligencia Artificial y el *Machine Learning*, los algoritmos de aprendizaje adaptativo han demostrado ser una herramienta útil para mejorar la eficiencia y precisión de los sistemas de reconocimiento de voz. Sin embargo, la validación de estos modelos se ha limitado principalmente a pruebas con unos conjuntos de datos determinados.

Es necesario destacar que, para que los modelos de reconocimiento de voz sean aplicables en la práctica clínica, debemos garantizar la precisión en una variedad de condiciones de captura de audio, incluyendo entornos con gran cantidad de ruido y variaciones en la calidad del audio.

Por lo tanto, se recomienda la implementación de estos modelos en varios entornos clínicos reales (además del que ya tenemos) para poder evaluar de un modo mas exacto el rendimiento

de nuestros métodos adaptativos.

Además de esto, se podría plantear la realización de un detector de enfermedades a partir de la tos apoyado en el uso de algoritmos adaptativos

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

BIBLIOGRAFÍA

[1] Adelaida Lamas, Marta Ruiz de Valbuena, and Luis Máiz. Tos en el niño. *Archivos de Bronconeumología*, 50(7):294–300, 2014.

[2] N Ambrosino, W Aniwidyaningsih, I Annesi-Maesano, P Bakke, F Blasi, S Borg, K Bracke, G Brussele, G Burge, A Bush, N Cano, K-H Carlsen, NH Chavannes, L Clancy, J-F Cordier, U Costabel, V Cottin, P Cullinan, M Decramer, M Demedts, S Drysdale, JS Elborn, KF Chung, M Fletcher, J Gerritsen, GJ Gibson, A Greenough, A Gulsvik, RP Gupta, G Hardavella, D Heederik, M Hecker, M Humbert, N Kunzli, T Lerut, ES Limb, RK Mayer, GB Migliori, N Navani, B Nemery, L Nicod, D Nowak, S Olofsson, P Palange, P Pelosi, L Perez, U Persson, C Pison, R Rapp, RL Riha, W Seeger, Y Sibille, AK Simonds, M Simoni, G Sotgiu, S Spiro, I Steenbruggen, R Stevenson, DP Strachan, J Svenson, P Tonnesen, J Townsend, T Troosters, S Turner, P Van Schil, R Varraso, G Viegi, JA Wedzicha, T Welte, and S Williams. *La salud pulmonar en Europa-Hechos y cifras*. European Lung Foundation, 2014.

[3] Diego Pérez Alonso. Análisis de señales de tos para detección temprana de enfermedades respiratorias. Master's thesis, Universidad de Valladolid, Valladolid, España, 2019.

[4] `https://docs.python.org/es/3/tutorial/`.

[5] `https://es.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes`.

[6] Michael Haenlein and Andreas Kaplan. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, 61(4):5–14, 2019.

[7] Peter Norvig Russell, Stuart J.; Norvig. *Artificial intelligence: a modern approach (3rd edition)*. Prentice Hall, 2009.

[8] Richard Bellman. *An introduction to artificial intelligence: can computers think?* San Francisco: Boyd Fraser Pub, 1978.

[9] Kevin Rich, Elaine; Knight. *Artificial intelligence (2nd edition)*. New York: McGraw-Hill, 1991.

[10] Patrick Henry Winston. *Artificial intelligence (3rd edition)*. Reading, Mass.: Addison-Wesley Pub, 1992.

[11] Sebastian Badaro, Leonardo Ibañez, and Martin Agüero. Sistemas expertos: Fundamentos, metodologías y aplicaciones. *Ciencia y Tecnología*, 1, 12 2013.

[12] Nilsson and Nils J. *Artificial Intelligence: A New Synthesis*. San Francisco: Kaufmann, 1998.

[13] J. M. Cruz Ac, J. N. Zaragoza Grifé, and J. A. González Fajardo. Un sistema de razonamiento basado en casos para apoyar la toma de decisiones en la industria de la construcción. *Ingeniería*, 2013.

[14] Carlos Enrique Hernández Reyes José Carlos Santiesteban Rojas, Dianet Utria Pérez. Definición de redes bayesianas y sus aplicaciones. *Vinculando*, 2012.

[15] Panayotis Brea, EbertTremante. Una visión de la teoría difusa y los sistemas difusos enfocados al control difuso. *Ingeniería Industrial. Actualidad y Nuevas Tendencias*, 2014.

[16] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[17] Ihab F Ilyas and Xu Chu. *Data cleaning*. Morgan & Claypool, 2019.

[18] Salvador García, Sergio Ramírez-Gallego, Julián Luengo, José Manuel Benítez, and Francisco Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1):1–22, 2016.

[19] Hui Yang. Data preprocessing. *Pennsylvania State University: Citeseer*, 2018.

[20] Matevž Kunaver and Jurij Tasic. Image feature extraction - an overview. pages 183 – 186, 02 2005.

[21] Lutfiye Durak and Orhan Arikan. Short-time fourier transform: two fundamental properties and an optimal implementation. *IEEE Transactions on Signal Processing*, 51(5):1231–1242, 2003.

[22] Guillermo Arturo Martínez Mascorro and Gualberto Aguilar Torres. Reconocimiento de voz basado en mfcc, sbc y espectrogramas. *Ingenius*, (10):12–20, 2013.

[23] Irene Moral Peláez. Modelos de regresión: lineal simple y regresión logística. *Revista Seden*, 14:195–214, 2016.

[24] Mónica Lizares Castillo. Comparación de modelos de clasificación: regresión logística y árboles de clasificación para evaluar el rendimiento académico. 2017.

[25] `https://www.ibm.com/es-es/topics/knn`.

[26] M Arriagada Rodríguez and IDEL Proyecto. Comparación de métricas de distancia en el algoritmo k-vecinos más cercanos para el problema de reconocimiento automático de dígitos manuscritos. *Pontificia Universidad Católica de Valparaíso, Facultad de Ingeniería, Escuela de Ingeniería Informática*, 2015.

[27] Alexandre Kowalczyk. *Support Vector Machines Succinctly*. 2017.

[28] Raquel Flórez López and José Miguel Fernández Fernández. *Las redes neuronales artificiales*. Netbiblo, 2008.

[29] `https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/`.

[30] J. A. Swets. Signal detection theory and roc analysis in psychology and diagnostics: Collected papers. *Lawrence Erlbaum Associates, In*, 1996.

[31] Sofia Visa, Brian Ramsay, Anca Ralescu, and Esther Knaap. Confusion matrix-based feature selection. volume 710, pages 120–127, 01 2011.

[32] James Fogarty, Ryan S. Baker, and Scott E. Hudson. Case studies in the use of roc curve analysis for sensor-based estimates in human computer interaction. GI '05, page 129–136, Waterloo, CAN, 2005. Canadian Human-Computer Communications Society.

[33] Zhe Hui Hoo, Jane Candlish, and Dawn Teare. What is an roc curve? *Emergency Medicine Journal*, 34(6):357–359, 2017.

[34] Devijver P. A. Kittler J. *Pattern recognition : a statistical approach*. 1982.

[35] Payam Refaeilzadeh, Lei Tang, and Huan Liu. Cross-validation. *Encyclopedia of Database Systems*, 532–538:532–538, 01 2009.

[36] Oscar M. Cumbicus-Pineda. *Categorización automática de tweets sobre el tema político electoral aplicando algoritmos de clasificación supervisada*. PhD thesis, 07 2017.

[37] Geoffrey Webb, Claude Sammut, Claudia Perlich, Tamás Horváth, Stefan Wrobel, Kevin Korb, William Noble, Christina Leslie, Michail Lagoudakis, Novi Quadrianto, Wray Buntine, Lise Getoor, Galileo Namata, Jiawei Jin, Jo-Anne Ting, Sethu Vijayakumar, Stefan Schaal, and Luc De Raedt. *Leave-One-Out Cross-Validation*. 01 2010.

[38] Isabel Cristina Pérez Verona and Leticia Arco García. Una revisión sobre aprendizaje no supervisado de métricas de distancia. *Revista Cubana de Ciencias Informáticas*, 10(4):43–67, 2016.

[39] Horace B Barlow. Unsupervised learning. *Neural computation*, 1(3):295–311, 1989.

[40] Juan Ignacio Bagnato. *Aprende Machine Learning en español : teoría + práctica Python*. Amazon Italia Logistica, Torrazza Piemonte, 2020.

[41] Marco A Wiering and Martijn Van Otterlo. Reinforcement learning. *Adaptation, learning, and optimization*, 12(3):729, 2012.

[42] Santiago Babío Fernández and Enrique Zuazua Iriondo. Procesos de decisión de markov y q-learning. 2021.

[43] Burr Settles. Active learning. *Synthesis lectures on artificial intelligence and machine learning*, 6(1):1–114, 2012.

[44] E.B. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks*, 2(1):5–19, 1991.

[45] Liantao Wang, Xuelei Hu, Bo Yuan, and Jianfeng Lu. Active learning via query synthesis and nearest neighbour search. *Neurocomputing*, 147:426–434, 2015. Advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012).

[46] Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[47] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers, 1994.

[48] Mingkun Li and Ishwar K Sethi. Confidence-based active learning. *IEEE transactions on pattern analysis and machine intelligence*, 28(8):1251–1261, 2006.

[49] Jin Zhou and Shiliang Sun. Improved margin sampling for active learning. In *Chinese Conference on Pattern Recognition*, pages 120–129. Springer, 2014.

[50] Yawar Siddiqui, Julien Valentin, and Matthias Nießner. Viewal: Active learning with viewpoint entropy for semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9433–9443, 2020.

[51] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109–165. Academic Press, 1989.

[52] Roger Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97(2):285–308, 1990.

[53] Jeffrey Schlimmer and Doug Fisher. A case study of incremental concept induction. pages 496–501, 01 1986.

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[55] Idae Jin. Introducción a las redes neuronales artificiales, 2019. Acceso el 2021-09.

[56] Fernando Izaurieta and Carlos Saavedra. Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*, 2000.

[57] Raquel Flórez López and José Miguel Fernández Fernández. *Las redes neuronales artificiales*. Netbiblo, 2008.

[58] D H Hubel and T N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *J. Physiol.*, 148(3):574–591, October 1959.

[59] Jordi Torres. *Python deep learning*. Marcombo, 2020.

[60] `https://www.diegocalvo.es/red-neuronal-convolucional/`.

[61] `https://anderfernandez.com/blog/`.

[62] Vahid Mirjalili and Sebastian Raschka. *Python machine learning*. Marcombo, 2020.

[63] Yaoshiang Ho and Samuel Wookey. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access*, 8:4806–4813, 2019.

[64] C Aggarwal Charu. Neural networks and deep learning: a textbook, 2018.

[65] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[66] Nbv-net: A 3d convolutional neural network for predicting the next-best-view. `https://www.researchgate.net/figure/Figura-211-Gradiente-descendiente-Busca-la-direccion-hacia-donde-la-funcion-F-w_fig10_329453137`. [accessed 14 Jan, 2023].

[67] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[68] Timothy P Lillicrap, Adam Santoro, Luke Marris, Colin J Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, 2020.

[69] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[70] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[71] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. Online deep learning: Learning deep neural networks on the fly, 2017.

[72] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. Online deep learning: Learning deep neural networks on the fly, 2017.

[73] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[74] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. ICML'03, page 928–935. AAAI Press, 2003.

[75] Jesús Monge-Álvarez, Carlos Hoyos-Barceló, Luis Miguel San-José-Revuelta, and Pablo Casaseca-de-la Higuera. A machine hearing system for robust cough detection based on a high-level representation of band-specific audio features. *IEEE Transactions on Biomedical Engineering*, 66(8):2319–2330, 2019.

[76] J.-M. Liu, M. You, Z. Wang, G.-Z. Li, X. Xu, and Z. Qiu. Cough event classification by pretrained deep neural network. *BMC medical informatics and decision making*, 15(4):S2, Nov 2015.

[77] Jesús Monge-Álvarez, Carlos Hoyos-Barceló, Paul Lesso, and Pablo Casaseca-de-la Higuera. Robust detection of audio-cough events using local hu moments. *IEEE Journal of Biomedical and Health Informatics*, 23(1):184–196, 2019.

[78] Carlos Hoyos Barcelo, Jose Garmendia-Leiza, Maria Dolores Aguilar Garcia, Jesús Monge, Diego Pérez-Alonso, Carlos Alberola-López, and Pablo Casaseca-de-la Higuera. *Evaluation in a Real Environment of a Trainable Cough Monitoring App for Smartphones*, pages 1175–1180. 01 2020.

[79] Vu Luong. Online neural network (onn-tensorflow), 2021.