Universidad de Valladolid

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER EN INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MÁSTER

TÉCNICAS AVANZADAS DE APRENDIZAJE PROFUNDO PARA LA DETECCIÓN Y ANÁLISIS DE TOS EN PACIENTES RESPIRATORIOS

Autor
Diego Asay Pérez Alonso

Tutor
Juan Pablo Casaseca de la Higuera

Valladolid, 23 de junio de 2023

TÉCNICAS AVANZADAS DE APRENDIZAJE PROFUNDO PARA LA DETECCIÓN Y ANÁLISIS DE TOS EN PACIENTES RESPIRATORIOS

Diego Asay Pérez Alonso

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

La ciencia puede divertirnos y fascinarnos a todos,
pero es la ingeniería la que cambia el mundo.
Isaac Asimov

La información es el petróleo del siglo XXI,
y la analítica es el motor de combustión.
Peter Sondergaard

La ciencia es el gran antídoto contra el
veneno del entusiasmo y la superstición.
Adam Smith

Cualquier tecnología suficientemente avanzada
es indistinguible de la magia.
Arthur C. Clarke

Del mismo modo que la electricidad transformó muchas industrias
hace aproximadamente 100 años, la inteligencia artificial cambiará
ahora también todas las grandes industrias —sanidad, transporte,
entretenimiento, fabricación— enriqueciendo las vidas de
innumerables personas.
Andrew Ng

AGRADECIMIENTOS

En primer lugar, quiero expresar mi agradecimiento al profesor Juan Pablo Casaseca de la Higuera por facilitarme la realización de este Trabajo Fin de Máster y por su orientación y colaboración durante el mismo.

Asimismo, quiero reconocer al Laboratorio de Procesado de Imagen de la Universidad de Valladolid por proveerme de los recursos necesarios para el desarrollo de este Trabajo Fin de Máster.

Mi gratitud a mi familia, compañeros y amigos, por el apoyo y el ánimo que me han ofrecido, especialmente a mis padres, mi hermano y mi pareja, que me han acompañado en los momentos más adversos.

RESUMEN

Antecedentes: La tos es un mecanismo de defensa y expulsión del aparato respiratorio que provoca una respuesta refleja y sonora. En la actualidad, el análisis de la tos como marcador sintomático del avance de una enfermedad se apoya en instrumentos poco adecuados para el seguimiento en escenarios de la vida real. Algunos solo se han evaluado en ambientes silenciosos y controlados, otros se diseñaron para resolver un problema más general que la detección de la tos o se enfocan en una población muy concreta. Asimismo, algunos enfoques no se han concebido con la eficiencia requerida para operar en un *smartphone*. Por estos motivos, los métodos de análisis de audio empleados en estos dispositivos no son capaces de manejar ambientes ruidosos, como el caso de un paciente que use su *smartphone* en el bolsillo como dispositivo de captura de datos.

Objetivo: Este Trabajo de Fin de Máster (TFM) tiene como propósito emplear técnicas de aprendizaje profundo (*Deep Learning*) para diseñar un sistema de "audición máquina" (*Machine Hearing*) que procese espectrogramas de señales acústicas y los clasifique de acuerdo a su contenido. Específicamente, se pretende reconocer los espectrogramas que contienen tos y los que no, y además clasificar cada tos a partir de sus propiedades espectrales según la enfermedad respiratorias asociada a la tos o el tipo de tos.

Métodos: Para llevar a cabo el proyecto, adquirimos 36866 señales de audio contaminadas por ruido de 20 pacientes respiratorios con distintas afecciones. La mitad de estas señales correspondieron a episodios de tos, mientras que la otra mitad no contenía ningún sonido de tos. Estas señales de audio se someten a un preprocesamiento en tres etapas. Primero, las señales de audio originales (señales de tos y no tos) se segmentan para que cada segmento dure un segundo. En segundo lugar, se transforman las señales 1D temporales en imágenes (señales 2D) mediante tres métodos. Los dos primeros métodos transforman cada clip de audio, que son señales de tiempo (1D), en señales de tiempo-frecuencia (imágenes 2D) realizando un espectrograma logarítmico o un espectrograma de mel. El tercer método aplica a los audios la técnica de ventanas deslizadas cambiando la forma del vector y transformándolo en una matriz. Posteriormente, los datos se normalizan para poder alimentar a una red neuronal recurrente convolucional (C-LSTM). La red neuronal convolucional (*Convolutional Neural Network*, CNN) extrae características de los espectrogramas de audio automáticamente para identificar "patrones" espectrales o temporales. Luego, se alimenta a una red neuronal recurrente de memoria a corto plazo (*Long Short-Term Memory*, LSTM), que predice el *frame* actual haciendo referencia a los *frames* adyacentes. De esta manera, primero detecta si el clip de audio contiene tos o no, y en caso afirmativo, procedemos a realizar un análisis posterior con el objetivo de detectar el tipo de tos o la enfermedad subyacente.

Resultados: El sistema de audiodetección de tos que obtuvo una especificidad mas alta presenta sensibilidad del 86,23 % y una especificidad del 93,90 %. Por otro lado, el método de clasificación de tos que obtuvo la mayor exactitud fue el que discriminó entre tos de pacientes con COVID-19 y tos de pacientes que tiene síntomas pero sin diagnóstico de COVID-19, que obtuvo un 58,21 %.

Conclusiones: Los resultados de este TFM abren la posibilidad de crear un dispositivo cómodo y no invasivo, con una mínima interferencia en las actividades cotidianas, capaz de detectar con carácter temprano enfermedades respiratorias, beneficiando a pacientes, profesionales sanitarios y sistemas nacionales de salud.

Palabras claves: *Deep Learning*, Tos, Detector, Análisis tiempo-frecuencia, Enfermedades respiratorias.

ABSTRACT

Background: Coughing is a defense mechanism and expulsion mechanism of the respiratory system that causes a reflexive and audible response. Currently, the analysis of cough as a symptomatic marker of disease progression relies on instruments that are poorly suited for monitoring in real-life scenarios. Some have only been evaluated in quiet and controlled environments, while others were designed to solve a problem more general than cough detection or focus on a very specific population. Additionally, some approaches have not been conceived with the required efficiency to operate on a smartphone. For these reasons, the audio analysis methods used in these devices are not capable of handling noisy environments, such as the case of a patient using their smartphone in their pocket as a data capture device.

Objective: The purpose of this Master's Thesis is to employ deep learning techniques to design a "machine hearing" system that processes spectrograms of acoustic signals and classifies them according to their content. Specifically, the aim is to recognize spectrograms that contain coughing and those that do not, as well as classify the disease associated with each cough based on their spectral properties.

Methods: To carry out the project, audio signals contaminated with noise from twenty patients with various respiratory conditions were used, along with 18,433 audio signals recorded during cough episodes and 18,433 audio signals recorded during non-cough episodes. These audio signals undergo preprocessing in three stages. First, the original audio signals (cough and non-cough signals) are segmented to have a duration of one second each. Secondly, the temporal 1D signals are transformed into images (2D signals) using three methods. The first two methods transform each audio, which are time-domain signals, into time-frequency signals by performing a logarithmic spectrogram or a mel spectrogram. The third method applies sliding windows to the audios, changing the vector shape and transforming it into a matrix. Subsequently, the data is normalized to feed into a Convolutional Long Short-Term Memory (C-LSTM) neural network. The Convolutional Neural Network (CNN) automatically extracts features from the audio spectrograms to identify spectral or temporal "patterns". Finally, the processed data is fed into a Long Short-Term Memory (LSTM) recurrent neural network, which predicts the current frame by referencing adjacent frames. In this way, it first detects if the audio contains a cough or not, and if it does, it proceeds to diagnose the respiratory disease.

Results: The audio detection system for coughs that achieved the highest specificity had a sensitivity of 86.23 % and a specificity of 93.90 %. On the other hand, the cough classification method with the highest accuracy was the one that discriminated between coughs from COVID-19 patients and coughs from patients with symptoms but without a COVID-19 diagnosis, which achieved 58.21 %.

Conclusions: The results of this Master's Thesis open up the possibility of creating a comfortable and non-invasive device with minimal interference in daily activities, capable of early detecting respiratory diseases, benefiting patients, healthcare professionals, and national health systems.

Keywords: *Deep Learning*, Cough, Detector, Time-frequency analysis, Respiratory diseases.

Capítulo 1
INTRODUCCIÓN

El propósito de este capítulo es presentar el contexto y la justificación del presente Trabajo de Fin de Master, que se basa en el empleo de técnicas de aprendizaje profundo (*Deep Learning*, DL) para la identificación y el diagnóstico de toses mediante espectrogramas de señales acústicas. Se describen los fundamentos teóricos de las señales acústicas de tos, los alcances y la metodología, así como las ventajas y los retos de este enfoque y una breve introducción al contenido de la memoria.

1.1. La tos y enfermedades respiratorias asociadas

La tos es uno de los síntomas más comunes que motiva que los pacientes acudan a consulta médica. Fisiológicamente, la tos tiene tres fases: inspiratoria, compresiva y espiratoria. La fase de inspiración implica la inhalación de un volumen variable de aire que permite el estiramiento de los músculos espiratorios, optimizando la relación longitud-tensión. La fase de compresión implica un cierre breve (200 ms) de la glotis para preservar el volumen pulmonar mientras se incrementa la presión intratorácica (hasta 300 mmHg en adultos) debido a la contracción isométrica de los músculos espiratorios contra una glotis ocluida. La fase de espiración se inicia con la apertura de la glotis, liberando un flujo espiratorio supramáximo corto (30-50 ms) (hasta 12 l/s en adultos, también llamado "pico de tos") seguido de flujos espiratorios más bajos (3-4 l/s) que se prolongan otros 200-500 ms [1].

Dado que la fase espiratoria de la tos dura entre 230-550 ms, cualquier sonido de tos que dure menos de 200 ms podría descartarse. Además, los 200 ms antes y después de un sonido de tos pueden comprender las fases compresiva y espiratoria de baja amplitud del sonido de la tos [2].

Sin embargo, sus sonidos característicos se considera individualizado de manera similar a la voz y son generados por la vibración de las vías respiratorias grandes y las estructuras laríngeas durante el flujo turbulento en la espiración causados por las contracciones de los músculos de la pared torácica, el abdomen, el diafragma y la laringe [3]. El flujo de aire laminar, que ocurre en vías respiratorias más pequeñas, es inaudible [1]. Puede entenderse como un reflejo fisiológico natural dirigido a limpiar las vías respiratorias inferiores de residuos, especialmente de mucosidad. Es, por tanto, un mecanismo de defensa para expulsar material extraño del sistema respiratorio [4], [5].

Para que el oído humano distinga diferentes sonidos de tos, como la tos productiva y la tos improductiva, no se conoce qué vías respiratorias están involucradas en la generación ni la cantidad de secreciones requeridas. Los fisiólogos distinguen dos tipos de tos: laríngea (un reflejo verdadero, también llamado "reflejo espiratorio") y traqueobronquial. En la tos laríngea, la inspiración puede ser escasa y se desencadena en situaciones clínicas cuando los receptores laríngeos son estimulados por aspiración de material extraño. La tos traqueobronquial, por el contrario, se desencadena distal a la laringe y puede ser voluntaria [1].

La tos puede asociarse a más de un centenar de condiciones patológicas [6]. Muchas de ellas son enfermedades respiratorias, que pueden afectar a distintas partes del sistema respiratorio. Entre estas enfermedades se pueden mencionar:

- La enfermedad del sueño o apnea del sueño es un trastorno que interrumpe la respiración durante el sueño.

- La tuberculosis es una enfermedad causada por bacterias que afecta principalmente a los pulmones.
- La bronquiectasia es un enrojecimiento y engrosamiento de los bronquios.
- El neumotórax es la presencia de aire en la cavidad pleural.
- La fibrosis quística es una enfermedad genética que afecta los pulmones y otros órganos.
- La neumonía es una infección por bacterias, virus u hongos en los pulmones.
- La bronquiolitis es una infección viral de los bronquiolos.
- La sinusitis es una inflamación de los senos y conductos nasales, así como los senos paranasales.
- La rinitis es una inflamación de las membranas mucosas de la nariz.
- La amigdalitis, faringitis y laringitis es la inflamación de las amígdalas, faringe y laringe respectivamente.
- La tos ferina o pertusis o tos convulsiva es una infección bacteriana grave de los pulmones que causa tos violenta.

En cambio, otras enfermedades respiratorias son más genéricas (resfriado, influenza o gripe, alergias, etc.) y entre ellas destaca la COVID-19, causada por el virus pandémico síndrome respiratorio agudo severo coronavirus 2 (SARS-CoV-2), que apareció como un nuevo agente patógeno humano en la ciudad de Wuhan, en la provincia de Hubei, China en diciembre de 2019 [7]. Según la OMS, a las 5:43 p.m. CEST del 17 de mayo de 2023, se han confirmado 766.440.796 casos de COVID-19 en el mundo, con 6.932.591 fallecimientos. A 14 de mayo de 2023 se han administrado 13.350.702.229 dosis de vacunas [8]. Asimismo, existen dos grupos de enfermedades muy frecuentes que son:

- Enfermedad pulmonar obstructiva crónica (EPOC o COPD por sus siglas en inglés): Hace referencia al grupo de enfermedades respiratorias crónicas que provocan obstrucción de las vías respiratorias, lo que dificulta la respiración. Las enfermedades que se incluyen en la EPOC son el enfisema (enfermedad crónica en la que los alvéolos se dañan y pierden elasticidad), la bronquitis crónica (inflamación de los bronquios) y, en algunos casos, el asma (enfermedad crónica que provoca inflamación y estrechamiento de las vías respiratorias). Los síntomas de la EPOC pueden incluir tos crónica, dificultad para respirar y fatiga.
- Enfermedad Pulmonar Intersticial (EPI) [9]: es el nombre de un grupo de enfermedades que causan que el tejido alrededor de los alvéolos, que se denomina intersticio, se vuelve rígido o cicatrizado, y los sacos de aire no pueden expandirse tanto. Esto hace que sea difícil obtener suficiente oxígeno. La cicatrización se llama fibrosis pulmonar. La EPI puede ocurrir sin una causa conocida. Esto se denomina EPI idiopática. La Fibrosis Pulmonar Idiopática (FPI) es la enfermedad más común de este tipo. Asimismo, existen docenas de causas conocidas de la EPI que incluyen:
  - Enfermedades autoinmunitarias (en las cuales el sistema inmunitario ataca al cuerpo) como el lupus, la artritis reumatoidea, la sarcoidosis y la esclerodermia.
  - Inflamación de los pulmones debido a la inhalación de una sustancia extraña, como ciertos tipos de polvo, hongos o moho (neumonitis por hipersensibilidad).
  - Medicamentos (como nitrofurantoína, sulfamidas, bleomicina, amiodarona, metotrexato, oro, infliximab, etanercept y otros medicamentos quimioterapéuticos).
  - Radioterapia dirigida al tórax.
  - Las enfermedades pulmonares ocupacionales (EPOs), o relacionadas con el trabajo, son aquellas causadas o que empeoran por los materiales a los que está expuesta e inhala una persona en su lugar de trabajo. La inhalación durante un período prolongado del polvo del carbón mineral, grafito o carbón artificial causa la Neumoconiosis de los Mineros del Carbón (NMC) también conocida como enfermedad del pulmón negro, la inhalación de fibras de asbesto causa la asbestosis, la inhalación de polvo de hierro causa la siderosis, la inhalación de polvo que contiene sílice causa la silicosis y la inhalación del polvo proveniente del heno mohoso, la paja y los granos causa la neumopatía de los granjeros.

- Infección y recuperación parcial de enfermedades como el COVID-19.

Las enfermedades respiratorias pueden ser causadas por diferentes factores, como infecciones, exposición a ciertos químicos o polvo, o factores genéticos. Los síntomas de las enfermedades respiratorias pueden incluir dificultad para respirar, tos, dolor en el pecho, fatiga y falta de aire. Además, la tos puede estar asociada al estilo de vida (fumadores, personas sedentarias, etc.). Muchas de estas enfermedades no sólo son una de las mayores causas de muerte en el mundo [10], sino que constituyen además una importante carga económica para los sistemas de salud [11].

A pesar de que los sonidos de la tos transmiten información vital sobre el estado del sistema respiratorio, no existen métodos de referencia para evaluar objetivamente la tos [12]. Esto explica que, hasta hace pocos años, el estudio de la tos se haya limitado a herramientas de medición subjetiva: el profesional suele pedir al paciente que aporte su propia apreciación sobre la frecuencia y la gravedad de su tos y sobre cómo afecta a su calidad de vida. En este proceso se suelen utilizar tablas de puntuación de tos, diarios y cuestionarios sobre sintomatología como el cuestionario de tos de Leicester [13] y el índice de disnea de referencia/transicional [14], [15]. Sin embargo, este enfoque presenta algunos inconvenientes que pueden llevar a una interpretación errónea de los síntomas [16]. En primer lugar, las limitaciones reales del sistema auditivo humano y de otros instrumentos empleados (e.g., los estetoscopios), que se comportan como filtros paso bajo [17]. En segundo lugar, existe una variabilidad entre expertos [18]. Por último, los aspectos secundarios de las enfermedades subyacentes, como su comorbilidad física y psicológica [19] pueden también sesgar la interpretabilidad.

Para superar estas limitaciones, las instituciones gubernamentales han resaltado no sólo el alto potencial de la telemedicina en la gestión de enfermedades respiratorias, sino también la necesidad creciente de incrementar la investigación en esta área específica [20]. En los últimos años, y sobre todo con la irrupción de la pandemia de COVID-19, ha habido un esfuerzo creciente en la investigación de métodos que permitan medir y analizar de forma objetiva los eventos de tos. Para lograr una monitorización de la tos objetiva, precisa y fiable, son necesarios enfoques para la detección automática de la tos a partir de datos de audio [21]-[23]. Sin embargo, la mayoría de los modelos de detección actuales se basan en datos que se registran en un entorno controlado o se componen artificialmente de clips de sonido, mientras que el entorno real conlleva diferentes desafíos [24], [25].

El desafío de diseñar un sistema automatizado de detección de tos es múltiple. En comparación con el procesamiento de audio general, la detección de tos tiene sus propios desafíos. En el reconocimiento de voz, se pueden descubrir las mismas pistas de pronunciación de diferentes personas. Por otro lado, el sonido de la tos varía mucho debido a las diferentes causas, cursos de enfermedades y anatomías. Además, la aparición repentina de tos dificulta su registro. Todas estas razones impiden alcanzar una precisión similar a la detección de voz en la tarea de detección de tos [26]-[28]. Particularmente desafiante es la rara aparición de tos, que exige una alta especificidad del sistema de detección de tos para evitar falsas alarmas de otros sonidos respiratorios similares y más frecuentes, como el carraspeo, la risa o el habla. Además, se ha identificado una sensibilidad superior al 90 % para satisfacer los requisitos de los expertos [12]. Otro desafío es que un sistema de detección de tos debe ser capaz de operar continuamente durante un período de tiempo prolongado para capturar una frecuencia de tos representativa con respecto a un trastorno respiratorio, es decir, varias horas, una noche o incluso un día completo. Por lo tanto, el sistema de detección de tos debe estar conectado a la fuente de alimentación o la vida útil de la batería del sistema debe ser lo suficientemente larga. Dado que se prevé que la mayoría de los sistemas de detección de tos funcionen en el campo bajo cualquier circunstancia, funcionan con una vida útil limitada de la batería y, por lo tanto, la eficiencia energética es un criterio de diseño crítico [29].

Las enfermedades respiratorias causan alrededor de 1 millón de muertes anuales en la Región Europea de la OMS. El cáncer de pulmón es la principal causa de muerte respiratoria, seguido de la EPOC, las infecciones de las vías respiratorias inferiores y la tuberculosis. Más de la mitad de todas las muertes por enfermedades respiratorias (y al menos una cuarta parte de todos los ingresos hospitalarios por enfermedades respiratorias) se deben a enfermedades causadas por el tabaquismo. Se prevé que el cáncer de pulmón y la EPOC como causas de muerte seguirán aumentando hasta 2015 y 2030 respectivamente.

El total anual estimado de los costes convencionales directos (sanitarios) más indirectos (pérdida de productividad) del asma, la EPOC, el cáncer de pulmón, la tuberculosis, las infecciones respiratorias agudas, la fibrosis quística y el síndrome de apnea obstructiva del sueño (SAOS) en los 28 países de la UE es de al menos 96.000 millones de euros. Si, además, se incluye el valor monetizado de los años de vida ajustados por discapacidad per-

didos debido a discapacidad y muerte prematura, el costo total supera la asombrosa cifra de 380 000 millones de euros. Debe tenerse en cuenta que, debido a la imposibilidad de costear muchas condiciones, incluso estos totales son subestimaciones significativas de la carga socioeconómica general de las enfermedades respiratorias en la UE [30].

Un dispositivo móvil que detecta la tos podría usarse en un teléfono inteligente para el seguimiento continuo en tiempo real de pacientes con afecciones respiratorias, mediante grabaciones de audio obtenidas de ambientes ruidosos de la vida cotidiana. Esto supondría una herramienta fiable para los profesionales, de forma que se pueda impulsar el potencial de la telemedicina en el ámbito de las enfermedades respiratorias. Además, desde la perspectiva del paciente, este sistema de seguimiento sería más confortable y provocaría una interferencia mínima en sus actividades diarias. De este modo, serían menos conscientes de su medicalización.

Desde el punto de vista del procesamiento de señales, un evento de tos es una señal no estacionaria sin una estructura formante clara y que se compone de tres fases: la fase explosiva, la fase intermedia y la fase vocal. La duración media es de aproximadamente 300 ms. Su espectro presenta generalmente un pico de alta energía alrededor de 400 Hz y un pico secundario entre 1.000 y 1.500 Hz [18].

Estas señales no tienen una estructura formante definida y se caracterizan por un contenido espectral disperso. Asimismo, los eventos de tos pueden tener: fases intermedias fuerte y débiles, fase vocal ausente y fase vocal fuerte y débiles. A pesar de que los eventos de tos presentan una forma de onda parecida, existe una variabilidad inter e intra-paciente que afecta tanto la duración como la intensidad de las tres fases [31].

1.2. Objetivos

Este TFM tiene como propósito emplear técnicas de aprendizaje profundo (*Deep Learning*) para diseñar un sistema de "audición máquina" (*Machine Hearing*) que procese espectrogramas de señales acústicas y los clasifique de acuerdo a su contenido. Específicamente, se pretende reconocer los espectrogramas que contienen tos y los que no, y además clasificar cada tos a partir de sus propiedades espectrales según la enfermedad respiratorias asociada a la tos o el tipo de tos. Para alcanzar el objetivo general, se plantean los siguientes subobjetivos específicos:

- Diseñar dos sistemas de "audición máquina" (*Machine Hearing*) que utilicen una red neuronal recurrente convolucional para clasificar espectrogramas generados a partir de clips de audio. En primer lugar, un sistema capaz de detectar la presencia de tos en los espectrogramas, usando la base de datos "Palencia" (sección 6.1.1) como conjunto de entrenamiento. En segundo lugar, un sistema capaz de clasificar la tos, usando la base de datos COUGHVID [2] (sección 6.2.1) como conjunto de entrenamiento. De esta forma, se pretende crear una herramienta que facilite la identificación temprana de patologías respiratorias a partir del análisis de la tos.
- Desarrollar un sistema que supere el rendimiento de los sistemas existentes en la literatura para la detección de tos y buscar ser capaz de clasificar los clips de audio con tos dependiendo de las enfermedades respiratorias asociadas a la tos o el tipo de tos.

Este trabajo se enmarca en el desarrollo de un monitor robusto para su futura implementación mediante un teléfono inteligente que permita supervisar en tiempo real a pacientes respiratorios. De esta forma, los pacientes podrán tener una mínima interferencia en sus actividades cotidianas, al ser supervisados por un dispositivo cómodo y no invasivo. Este dispositivo, mediante el seguimiento continuo de los parámetros respiratorios de los pacientes, podría detectar de forma precoz alteraciones pulmonares, optimizando el diagnóstico, el tratamiento y la prevención de enfermedades respiratorias, con impacto positivo en la calidad de vida de los pacientes, la eficiencia de los profesionales sanitarios y la sostenibilidad de los sistemas nacionales de salud.

1.3. Fases y métodos

Después de definir los objetivos que se quieren lograr para solucionar el problema planteado, se ha seguido una metodología de trabajo basada en el diseño, la implementación y la validación, que se compone de las siguientes fases:

a) Realizar un proceso de segmentación de las señales de audio de la base de datos propia (“Palencia”) que se empleará para la detección de tos, con el fin de obtener fragmentos de audio de un segundo que incluyan la tos a analizar y el mismo número que no incluyan tos. Aplicar un filtro y una estandarización a la base de datos pública COUGHVID [2] que se utilizará para la clasificación de los audios de tos, debido a que contiene fragmentos de audio de distintas calidades y formatos para que sean compatibles con nuestra red.

b) Adaptación de los datos para que puedan alimentar durante su aprendizaje a una red neuronal recurrente convolucional (C-LSTM), que requiere imágenes como entrada. Para ello se ha calculado el espectrograma de cada señal de audio, obteniendo las representaciones visuales de las frecuencias y amplitudes del sonido. Para obtener los mejores resultados, buscamos el tipo de espectrograma que más información relevante aporte a la red.

c) Diseño y ajuste de los parámetros e hiperparámetros de las redes neuronales recurrentes convolucionales para la detección de audios con tos y clasificación de tos según el tipo o la patología asociada.

d) La base de datos propia (“Palencia”) se utiliza como conjunto de entrenamiento para el sistema capaz de detectar la presencia de tos en los espectrogramas y la base de datos pública COUGHVID se utiliza como conjunto de entrenamiento para el sistema capaz de clasificar la tos. En ambos casos los conjuntos de datos utilizados para el entrenamiento, la validación y el test de las distintas redes neuronales no tienen solapamiento.

e) Evaluación del rendimiento de las redes entrenadas para la detección y clasificación de audios con tos según el tipo o la patología asociada, usando medidas como exactitud, sensibilidad, especificidad, área bajo la curva Característica Operativa del Receptor (Receiver Operating Characteristic, ROC), valor predictivo positivo y valor predictivo negativo, estas métricas se describen en el capítulo 3.

1.4. Medios necesarios

En este apartado se detallan el *hardware*, *software* y las fuentes bibliográficas que se han utilizado para el desarrollo de este proyecto:

Hardware

- Ordenador de sobremesa con las siguientes especificaciones:
  - Procesador Intel® Core™ i7-7700 CPU @ 3,60GHz
  - 16 GB de memoria RAM
  - Tarjeta gráfica NVIDIA GeForce® GTX 1050 con 2 GB de memoria GPU
- Conexión remota a los servidores de cálculo del Laboratorio de Procesado de Imagen (LPI) de la Universidad de Valladolid (UVa). En particular, se ha utilizado el servidor *tebas*, con las características:
  - 2 procesadores Intel® Xeon® CPU E5-2697 v4 @ 2,30GHz con 70 hilos
  - 484 GB de memoria RAM
  - Tarjetas gráficas NVIDIA® Quadro RTX™ 5000 con 16 GB de memoria GPU y NVIDIA® Quadro RTX™ 6000 con 24 GB de memoria GPU

Software

- Kubuntu 21.04 : Distribución Linux derivada de Ubuntu que utiliza KDE Plasma como entorno de escritorio.
- Anaconda : Distribución libre y abierta de los lenguajes Python y R, utilizada en ciencia de datos y aprendizaje automático.

- Python : Lenguaje de programación interpretado y multiparadigma, que soporta orientación a objetos, programación imperativa y programación funcional.
- Spyder : Entorno de desarrollo integrado (*Integrated Development Environment*, IDE) multiplataforma y de código abierto para programación científica en Python. Ofrece un editor de código con resaltado de sintaxis, depurador, consola interactiva, explorador de variables y otras herramientas útiles para el desarrollo.
- Jupyter notebook : Entorno computacional interactivo basado en la web.
- TensorFlow [32] : Biblioteca de código abierto para aprendizaje automático, desarrollada por Google. Permite crear y entrenar modelos de aprendizaje profundo con una gran escalabilidad y rendimiento, aprovechando el uso de unidades de procesamiento gráfico (GPU).
- Keras [33] : Biblioteca de Redes Neuronales de código abierto escrita en Python que se ejecuta sobre TensorFlow, Microsoft Cognitive Toolkit o Theano.
- NumPy [34] : Extensión de Python que le añade mayor soporte para vectores y matrices, y una biblioteca de funciones matemáticas para operar con ellos.
- SciPy [35] : Biblioteca libre y de código abierto para Python que contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, transformada rápida de Fourier (*Fast Fourier Transform*, FFT), procesamiento de señales y de imagen, resolución de Ecuaciones Diferenciales Ordinarias (EDOs) y otras tareas para la ciencia e ingeniería.
- Librosa [36] : Biblioteca para análisis de música y audio. Proporciona herramientas para extraer características de la música y el audio.
- TensorFlow I/O [37] : Colección de sistemas de archivos y formatos de archivo que no están incluidos entre las opciones de compatibilidad integradas de TensorFlow. Proporciona extensiones adicionales útiles de Dataset, transmisión y sistemas de archivos.
- FFmpeg [38] : Software libre que puede grabar, convertir y reproducir audio y video en diferentes formatos.
- Matplotlib [39] : Biblioteca para la generación de gráficos a partir de datos en listas o arrays en Python y su extensión matemática NumPy.
- Pandas [40] : Biblioteca de software escrita como extensión de NumPy para manipulación y análisis de datos en Python. Ofrece estructuras de datos flexibles y eficientes para trabajar con datos tabulares o multidimensionales.
- Scikit-learn [41] : Biblioteca para aprendizaje automático de software libre para Python que se integra con las bibliotecas numéricas y científicas NumPy y SciPy.
- tgt [42] : Paquete de Python para procesar, consultar y manipular los archivos TextGrid de Praat. Facilita la lectura, escritura y modificación de estos archivos que contienen anotaciones temporales del habla.

- Praat [43] : Software gratuito para el análisis científico del habla usado en lingüística. Permite realizar diferentes tareas como grabación, edición, segmentación, etiquetado, análisis acústico y síntesis del habla.
- LaTeX[44] : Sistema de composición de textos orientado a la creación de documentos escritos con alta calidad tipográfica.
- TeXstudio [45]: Editor de texto gratuito que se utiliza para escribir documentos en LaTeX.

Bibliografía

- Hemeroteca de la Escuela Técnica Superior de Ingenieros de Telecomunicación (ETSIT).
- Repositorios electrónicos suscritos por la UVa.

Respecto a los datos sobre los que se aplicará el análisis y evaluación de los métodos, se utilizará la base de datos ("Palencia") disponible dentro del grupo en el que se desarrolla el proyecto, el LPI y la base de datos COUGHVID [2]. Estas bases de datos se explican con detalle en las secciones 6.1.1 y 6.2.1.

1.5. Estructura del documento

El documento se compone de 8 capítulos, entre los que se encuentra el presente capítulo. A continuación se presenta un breve resumen de su contenido:

Capítulo 1 - Introducción. Se presenta la temática del documento, así como la motivación, los objetivos propuestos, las fases y medios necesarios para lograrlos y la estructura del presente documento.

Capítulo 2 - Reconocimiento de patrones. Se define el concepto de reconocimiento de patrones, Inteligencia Artificial y *Machine Learning*. Se describen las diferentes técnicas de *machine hearing* y los principales clasificadores que se pueden emplear.

Capítulo 3 - Técnicas para evaluar el rendimiento de un clasificador. Se describe las principales métricas para la evaluación del rendimiento del sistema y se explica la curva ROC, la validación cruzada y las formas de promediar los resultados.

Capítulo 4 - Redes neuronales. Se introducen las redes neuronales y una visión general del *Deep Learning*, describiendo los diferentes tipos de redes que se pueden diseñar con esta filosofía.

Capítulo 5 - Estado del arte. En este capítulo se revisan los monitores de tos anteriores para la detección de eventos de audio con tos. Además, se muestran trabajos anteriores sobre la detección de enfermedades respiratorias, en concreto COVID-19.

Capítulo 6 - Metodología. Este capítulo describe las bases de datos y la composición de los *folds* empleadas para el entrenamiento y la evaluación de la red. En este capitulo se procede a la descripción tanto de las bases de datos utilizadas como de la composición de los *folds* en los entrenamientos y test de la red. También se detallan los pasos realizados para el preprocesamiento de las señales, que incluye la adaptación y la preparación de los datos para su uso en la red, así como la definición, el diseño y el entrenamiento de la red neuronal recurrente convolucional.

Capítulo 7 - Resultados y discusión. Este capítulo presenta el análisis y la discusión de los resultados obtenidos en la detección de segmentos de audio con tos y en la clasificación de audios con tos según la patología y el tipo de tos subyacentes. Además, se realizan comparaciones con los trabajos previos existentes en este ámbito.

Capítulo 8 - Conclusiones y líneas futuras. En este capítulo final se resumen las principales conclusiones derivadas de este Trabajo de Fin de Master y se plantean posibles líneas de trabajo futuro surgidas a partir de su desarrollo.

Además de un capítulo en el que se recogen las fuentes bibliográficas y referencias consultadas para la redacción del presente documento, se añade un capítulo de los acrónimos utilizados y un apéndice con el código necesario para construir las redes neuronales.

Capítulo 2

RECONOCIMIENTO DE PATRONES

El reconocimiento de patrones es una subárea de la inteligencia artificial y de la ciencia de la computación que se centra en la capacidad de una computadora para detectar patrones y tendencias en grandes conjuntos de datos. Esto se logra mediante el uso de algoritmos de aprendizaje automático, como clasificadores, que entrenan a un modelo con ejemplos previamente etiquetados para que pueda realizar predicciones precisas sobre nuevos datos.

El reconocimiento de patrones se basa en el aprendizaje automático, que es el proceso a través del cual una computadora aprende a realizar tareas sin ser explicitamente programada para hacerlas. En el aprendizaje automático, se proporciona a la computadora un gran conjunto de datos y se le permite .[en]trenarse.[a] sí misma para detectar patrones y tendencias en esos datos. A medida que procesa más y más datos, la computadora puede mejorar su capacidad para detectar patrones y tomar decisiones basadas en esos patrones.

El reconocimiento de patrones se utiliza en una amplia variedad de aplicaciones, como la detección de fraude, el análisis de mercado, la detección de intrusiones en la seguridad de la información, la identificación de patrones en grandes conjuntos de datos médicos, la clasificación de objetos, el reconocimiento de voz y de lenguaje natural (también conocido como "machine hearing" o "machine listening"), el análisis de imágenes y vídeos, la predicción de resultados futuros, la detección de anomalías y el análisis de sentimientos en el texto.

Un sistema automático de reconocimiento de patrones es un sistema diseñado para detectar patrones y tendencias en grandes conjuntos de datos sin la intervención humana. Estos sistemas se pueden entrenar utilizando técnicas de aprendizaje automático para mejorar su capacidad para detectar patrones a medida que procesan más y más datos.

Un modelo general clásico de un sistema automático se compone de tres partes principales: una sección de entrada, una sección de procesamiento y una sección de salida. La sección de entrada recibe información del entorno externo y la convierte en una forma que puede ser utilizada por el sistema. La sección de procesamiento lleva a cabo las operaciones necesarias para transformar la información de entrada en una salida deseada. Por último, la sección de salida devuelve la información procesada al entorno externo en una forma que pueda ser interpretada por el usuario o por otro sistema.

Además de las tres secciones principales, un sistema automático también puede incluir una sección de retroalimentación, que permite al sistema tomar decisiones y ajustar su comportamiento en función de la información recibida. La retroalimentación puede ser utilizada para mejorar el rendimiento del sistema y permitir que se adapte a cambios en el entorno.

Un clasificador de Deep Learning es un tipo de sistema automático que se utiliza para clasificar diferentes objetos o categorías. Por ejemplo, un clasificador de imágenes puede ser entrenado para identificar diferentes objetos en una imagen, como gatos, perros, árboles, etc. La sección de entrada del clasificador recibe la imagen y la convierte en una forma que pueda ser utilizada por el sistema. La sección de procesamiento utiliza una red neuronal profunda (Deep Learning) para analizar la imagen y asignarle una etiqueta o clase. Por último, la sección de salida devuelve la etiqueta asignada a la imagen. La sección de retroalimentación recibe información sobre la precisión del clasificador y la utiliza para ajustar y mejorar su desempeño en el futuro.

2.1. Inteligencia Artificial

La inteligencia artificial (IA) es un campo de la informática que se ocupa del desarrollo de sistemas informáticos capaces de simular el comportamiento y pensamiento humano. También se puede definir como campo interdisciplinario que se centra en la creación de sistemas que puedan realizar tareas que requieren inteligencia humana, como el razonamiento, el aprendizaje y la resolución de problemas. La historia de la IA se remonta a muchos siglos atrás, cuando se empezaron a desarrollar mecanismos y dispositivos capaces de imitar la inteligencia humana. La idea de crear máquinas inteligentes que puedan resolver problemas y tomar decisiones por sí mismas ha fascinado a la humanidad desde hace mucho tiempo. Sin embargo, el desarrollo de la IA como un campo de estudio científico y tecnológico se remonta a la década de 1950. A continuación se presentan algunos hitos importantes en la historia de la IA, que dividiremos en las siete etapas que vemos en la Figura 2.1 y una etapa precursora.

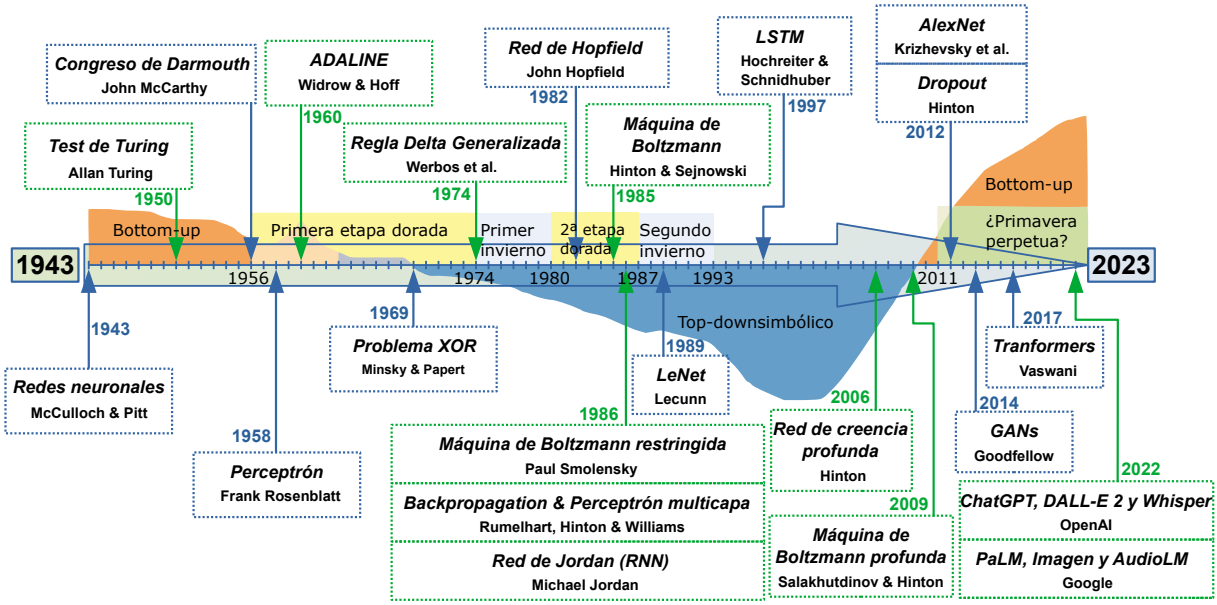


Figura 2.1: Historia de la Inteligencia Artificial [46], [47].

2.1.1. Evolución histórica

Precursores (Hasta 1952): Uno de los primeros en explorar el concepto de la inteligencia artificial (IA) fue el matemático y filósofo griego Aristóteles, quien propuso la idea de un autómata capaz de moverse por sí mismo, este concepto de IA ha estado en evolución desde el siglo XVII, cuando el matemático y filósofo René Descartes [48], [49] propuso que una máquina podría simular el uso de lenguaje y, por tanto, que piensa. El filósofo francés considera la posibilidad de que puedan construirse sofisticadas máquinas, mas estas no podrían igualar al ser humano en cuanto a la flexibilidad que va de la mano de la inteligencia genuina, la cual también da lugar a acciones inteligentes en tanto sensibles al contexto. Luego, aunque las máquinas simularan el uso de lenguaje, contarían solo como ingeniosos y sofisticados artefactos, cuya conducta además, en lo global, no sería adecuada como la humana [50].

En las décadas de 1940 y 1950, un puñado de científicos de diversos campos (matemáticas, psicología, ingeniería, economía y ciencias políticas) comenzaron a discutir la posibilidad de crear un cerebro artificial. En 1942 la matemática Ada Byron (Lovelace) desarrolló el primer algoritmo con capacidad para ser procesado por la máquina analítica creada por Charles Babbage [51], aunque nunca pudo probarlo, por lo que es considerada la primera programadora de la historia. Los primeros en describir lo que los investigadores posteriores llamarían una red neuronal fueron Walter Pitts y Warren McCulloch que analizaron redes de neuronas artificiales idealizadas y demostraron cómo podrían realizar funciones lógicas simples en 1943 [52]. Donald Hebb propuso en 1949 la "regla de Hebb"

[53], esta ley, explica a grandes rasgos el aprendizaje neuronal y se convirtió en la precursora de las técnicas de entrenamiento de redes neuronales artificiales de hoy en día.

Sin embargo, la IA como lo conocemos hoy en día comenzó a desarrollarse en la década de 1950, cuando se llevaron a cabo varios trabajos importantes en el campo. En 1950, el matemático y científico de la computación británico Alan Turing publicó un artículo titulado "Computing Machinery and Intelligence" [54], en el que propuso un test de inteligencia para determinar si una máquina era inteligente. Este test, conocido como el "Test de Turing", consiste en hacer preguntas a una máquina y a un humano y determinar si el examinador puede distinguir quién es la máquina y quién es el humano basándose únicamente en las respuestas [55].

El nacimiento de la inteligencia artificial (1952-1956): Etapa de creación de las técnicas básicas para representar el comportamiento inteligente tanto en el ámbito de los métodos de representación como de los lenguajes de programación. En esta etapa se llevaron a cabo los primeros experimentos y demostraciones de sistemas de IA, incluyendo *Audrey* un sistema de reconocimiento de voz que permitía reconocer dígitos en lenguaje natural hablado que presentó Bell Labs en 1952.

El término "inteligencia artificial" se acuñó por primera vez en 1956 durante el congreso organizado por el matemático y científico de la computación John McCarthy sobre IA en Dartmouth College de New Hampshire (USA), en la cual participaron científicos de diversas ramas del saber. Desde entonces, la IA ha sido un campo en constante evolución, con muchos avances y logros notables en el aprendizaje automático, el procesamiento del lenguaje natural y el reconocimiento de patrones. En la actualidad, la IA es una tecnología importante en una amplia variedad de campos y se espera que continúe desarrollándose a un ritmo acelerado en el futuro cercano.

IA simbólica (1956-1974): En esta etapa se estudiaron las posibilidades de aplicación industrial de la Inteligencia Artificial. Rosenblatt desarrolla en 1958 una teoría para un sistema nervioso hipotético llamado perceptrón [56]. Bernard Widrow y Ted Hoff desarrollaron en 1960, la denominada "Ley de Widrow-Hoff" [57], una importante variación del algoritmo de aprendizaje del perceptrón, que dio lugar al modelo ADALINE, que constituyó la primera red neuronal artificial aplicada a un problema real: la eliminación de los ecos de las líneas telefónicas por medio de filtros adaptativos.

IBM demuestra Shoebox en la Feria Mundial de Seattle de 1962 una máquina que puede entender hasta 16 palabras habladas en inglés. El MIT desarrolla ELIZA entre 1964 y 1966 uno de los primeros programas en procesar lenguaje natural e intentaba mantener una conversación de texto coherente con el usuario y entre los años 1968 y 1970 también en el MIT se desarrolla SHRDLU que es un programa de computadora para la comprensión del lenguaje natural.. Edward Feigenbaum desarrolló Dendral entre 1965 Y 1975 para interpretar la estructura molecular que fué el primer sistema experto en ser utilizado para propósitos reales y en 1967 Richard Greenblatt desarrolló el primer programa de ajedrez capaz de ejecutar una partida en condiciones llamado MacHack VI. Además Minsky y Seymour Papert publican en 1969 "Perceptrons"donde demostraron que el perceptrón no era capaz de aprender una función XOR [58] y en el año 1970 Peter Toma crea un sistema de traducción automática llamado SYSTRAN.

El primer invierno de IA (1974-1980): El campo de las redes neuronales fue cerrado casi por completo durante 10 años debido a la devastadora crítica de Marvin Minsky en "Perceptrons". Además el tremendo optimismo de los investigadores de IA había elevado las expectativas hasta niveles imposibles ya que no supieron apreciar la dificultad de los problemas a los que se enfrentaban y cuando los resultados prometidos no se materializaron, la financiación de la IA desapareció. En esta etapa finalizan cinco años de investigación de reconocimiento de voz financiada por DARPA entre 1971 y 1976 que condujo a la creación de Harpy por Carnegie Mellon, una máquina capaz de comprender 1011 palabras. James Anderson en 1977 desarrolló el Asociador Lineal y su extensión conocida como red *Brain-State-in-a-Box*, que permitieron modelizar funciones arbitrariamente complejas [59].

Auge (1980-1987): En la década de 1980, corporaciones de todo el mundo adoptaron una forma de programa de IA llamada "sistemas expertos" y el conocimiento se convirtió en el foco de la investigación principal de IA, la característica principal ha venido siendo la aceptación de la IA como una actividad informática profesional, de tal forma que desde 1981 las aplicaciones técnicas no son sólo desarrolladas por los laboratorios especializados, sino también por empresas que realizan aplicaciones para la industria y el mundo empresarial en general.

En 1982 el informático británico David Marr publica su famoso libro "Vision: A Computational Investigation into the Human Representation and Processing of Visual Information" [60], en el que describe cómo se puede desarrollar un sistema informático capaz de reconocer objetos a partir de imágenes y John Hopfield presentó su trabajo sobre redes neuronales describiendo con claridad y precisión una variante del Asociador Lineal inspirada en

la minimización de la energía presente en los sistemas físicos, conocida como "red de Hopfield" [61]. También, en ese mismo año Fujitsu comenzó el desarrollo de "computadoras pensantes" para diversas aplicaciones en robótica. En 1985 Judea Pearl acuñó el término red bayesiana [62], y Geoffrey Hinton y Terry Sejnowski inventaron un tipo de red neuronal recurrente estocástica llamada la Máquina de Boltzmann [63] en honor al físico austriaco Ludwig Boltzmann.

Werbos [64], Parker [65] y LeCun [66] formularon una nueva regla, la denominada "Regla Delta Generalizada", que dió lugar a un avance muy significativo en el aprendizaje supervisado. En 1986 Rumelhart desarrollo back-propagation [67] que es un algoritmo de aprendizaje supervisado para redes neuronales artificiales que ofreció una solución muy potente para la construcción de redes neuronales más complejas al evitar los problemas observados en el aprendizaje del perceptrón simple. Este algoritmo constituye desde entonces una de las reglas de aprendizaje de mayor utilización para el entrenamiento de la red conocida como perceptrón multicapa [68]. Además los RBM fueron inventados en 1986 por Paul Smolensky, aunque inicialmente bajo el nombre de Harmonium y en el mismo año Michael Jordan creo la primera SRN llamada red de Jordan [69].

El segundo invierno de IA (1987-1993): La fascinación de la comunidad empresarial por la IA aumentó y disminuyó en la década de 1980 en el patrón clásico de una burbuja económica. Yann LeCun en 1989 presentó el prototipo de una red neuronal convolucional diseñada para el reconocimiento de caracteres manuscritos e impresos que luego se llamó LeNet [70]. En la década de 1990 como un experimento de computación afectiva el MIT crea Kismet que es una cabeza de robot que puede reconocer y simular emociones. En 1990 el investigador de robótica Rodney Brooks apuntó directamente a la hipótesis del sistema de símbolos físicos, argumentando que los símbolos no siempre son necesarios ya que "el mundo es su propio mejor modelo" [71]. En 1991, Turk y Pentland desarrollan la tecnología denominada Eigenfaces [72], capaz de reconocer un rostro humano dentro de una fotografía.

IA (1993–2011): El campo de la IA, que ahora tiene más de medio siglo, finalmente logró algunos de sus objetivos más antiguos. Parte del éxito se debió al aumento de la potencia de las computadoras y parte se logró centrándose en problemas aislados específicos y persiguiéndolos con los más altos estándares de responsabilidad científica. En 1997 el equipo de Dragon Systems lanzó el sistema de reconocimiento de voz Dragon NaturallySpeaking, el primer sistema mediante el cual no era necesario realizar pausas entre cada palabra, además, era capaz de entender 100 palabras por minuto. En noviembre de 1997 se publica en la revista Neural Computation el artículo principal de LSTM [73] escrito por Sepp Hochreiter y Jürgen Schmidhuber. El 11 de mayo de 1997 el programa de ordenador Deep Blue [74] de IBM derrota al campeón mundial de ajedrez Garry Kasparov y también el 11 de mayo pero de 1999 se introdujo el primer modelo de consumo de una serie de mascotas robóticas diseñadas y fabricadas por Sony llamada AIBO. Posteriormente la década de los años 2000, comienza con la presentación de la compañía japonesa Honda del primer robot autónomo, el robot ASIMO, capaz de moverse de manera autónoma y realizar tareas simples. Geoffrey Hinton en 2006 propuso la red de creencia profunda (DBN) [75] que es un modelo gráfico generativo, y más tarde en 2009, Salakhutdinov y Hinton propusieron una arquitectura de red neuronal profunda llamada "máquina de Boltzmann profunda" (DBM) [76]

Aprendizaje profundo, big data e inteligencia artificial general (2011-presente): En las primeras décadas del siglo XXI, el acceso a grandes cantidades de datos (conocidos como "big data"), computadoras más baratas y rápidas, y técnicas avanzadas de aprendizaje automático se aplicaron con éxito a muchos problemas en toda la economía. De hecho, McKinsey Global Institute estimó en su famoso artículo "Big data: La próxima frontera para la innovación, la competencia y la productividad" [77] que "para 2009, casi todos los sectores de la economía estadounidense tenían al menos un promedio de 200 terabytes de datos almacenados".

2.1.2. Grandes hitos en los últimos años

IBM desarrolló Watson, un sistema de IA capaz de responder a preguntas formuladas en lenguaje natural, que en febrero de 2011 compitió y ganó en el popular programa de televisión "Jeopardy!", derrotando a sus dos oponentes humanos. El 4 de octubre de 2011, Apple anunció el sistema de reconocimiento de voz comercial basado en IA, llamado "Siri", que se incluiría con el iPhone 4S. El 14 de junio de 2011, Google anunció en su evento Inside Google Search que comenzaría a implementar "Voice Search" en Google.com durante los próximos días. En Android 4.1+ (Jelly Bean), se fusionó con "Google Now", presentado por Google el 27 de junio de 2012 como parte de la demostración principal de Android 4.1 Jelly Bean en Google I/O 2012. En la Conferencia de desarrolladores de Microsoft BUILD en abril de 2014 se demostró por primera vez "Cortana". En noviembre

de 2014, Amazon anunció el asistente virtual “Alexa”. En Google I/O 2016, Google presentó su nuevo asistente personal inteligente “Google Assistant”, que sería una especie de evolución de “Google Now”. Sin embargo, a diferencia de “Google Now”, el Asistente puede establecer un diálogo bidireccional con el usuario.

La primera red neuronal profunda que ganó el desafío de clasificación de imágenes ILSVRC [78], [79] fue AlexNet-8 (SuperVision) [80], que en 2012 redujo el error (top-5) al 15,3 %, en comparación con el 26,2 % de la segunda mejor entrada. Posteriormente ZFNet (Clarifai) ganó ILSVRC-2012, Inception (GoogLeNet) [81] ganó ILSVRC-2014 y ResNet (MSRA) [82] ganó ILSVRC-2015, donde los investigadores de Microsoft informaron que sus CNN excedían la capacidad humana en las tareas de ILSVRC. Las últimas ediciones fueron ganadas por ResNeXt [83] en ILSVRC-2016 y SENet [84] en ILSVRC-2017, que redujo el error (top-5) al 2,251 %, superando la entrada ganadora de 2016 que obtuvo un 3,57 %. Finalmente cabe destacar tres redes neuronales que obtuvieron resultados de vanguardia en muchos puntos de referencia de visión por computadora como son ViT [85] y CLIP [86], [87] en 2021, y CoCa [88] en 2022.

En 2016 cabe destacar YOLO [89]-[99], que hace uso de una única red neuronal convolucional para detectar objetos en imágenes en tiempo real. En enero de 2018 se abrió la primera tienda sin cajeros del mundo llamada Amazon Go [100]. Posteriormente se desarrollaron sistemas de IA capaces de superar a los médicos humanos en la detección de ciertos tipos de cáncer como el cáncer de pulmón [101] en 2019 y el cáncer de mama [102] en 2020. El equipo de Meta AI en abril de 2023 publicó SAM [103], un modelo de Inteligencia Artificial (IA) capaz de “recortar” cualquier objeto, en cualquier imagen, con un solo clic, ya que ha aprendido una noción general de lo que son los objetos y esta comprensión permite al sistema la segmentación rápida con generalización de disparo cero (*zero-shot*) a objetos e imágenes desconocidos sin requerir capacitación adicional.

En 2022 se han desarrollado varias IA capaces de crear imágenes a partir de una breve descripción, como DALL-E 2 [104], [105], cuyas imágenes son seleccionadas por CLIP; GLIDE [106]; Midjourney [107]; Imagen [108]; y Stable Diffusion [109]. Llegando a ganar el primer lugar en la competencia de arte digital en la Feria Estatal de Colorado de 2022 una imagen de Midjourney llamada *Théâtre d'Opéra Spatial*. Además, basándose en Stable Diffusion se ha creado Riffusion [110] para generar música; basándose en DALL-E 2 se ha creado Bing Image Creator [111]; y basándose en Imagen se han creado Imagen Editor [112] para editar imágenes e Imagen Video [113] para generar vídeos. Por otra parte se ha desarrollado Instant NeRF [114] para generar escenas 3D en segundos a partir de imágenes 2D, que ha dado lugar a DreamFusion [115], que crea escenas 3D a partir de texto usando imágenes 2D, y ClimateNeRF [116], que simula entornos climáticos.

En marzo de 2016 un sistema de aprendizaje automático llamado “AlphaGo” [117] desarrollado por la empresa “DeepMind” derrota al campeón mundial de Go, Lee Sedol. En octubre de 2017, presentaron una versión mejorada llamada “AlphaGo Zero” [118] que no utiliza datos humanos. En diciembre de 2017 presentan “AlphaZero” [119], que utiliza un enfoque generalizado para derrotar a “Stockfish” en ajedrez y “Elmo” en shogi y “AlphaGo Zero” en Go. En noviembre de 2019, se presenta “MuZero” [120] igualando el desempeño de “AlphaZero” en ajedrez y shogi, mejorandolo en Go y superando el rendimiento medio y mediano de R2D2 [121] en toda la suite de juegos de Atari. A finales de 2021, se propuso una variante más eficiente de “MuZero”, denominada “EfficientZero” [122].

En diciembre de 2018, “AlphaStar” [123]-[125] de DeepMind venció a “MaNa”, uno de los jugadores profesionales de StarCraft más fuertes del mundo, por 5-0. En julio de 2019, la nueva versión restringida de “AlphaStar” [126] compitió de forma anónima y en agosto de 2019, “AlphaStar” [127] había alcanzado el nivel de gran maestro, ubicándose entre el 0,2 por ciento superior de los jugadores humanos [128]. DeepMind también ha desarrollado IAs competentes en otros juegos como Quake III Arena Capture the Flag [129], OpenAI en Dota 2 [130] y Meta AI en póquer [131]. En Minecraft OpenAI ha llegado a fabricar herramientas de diamante observando vídeos [132], [133] mientras que DeepMind lo logró a través del aprendizaje por refuerzo [134].

DeepMind presentó “AlphaTensor” [135], [136], basándose en “AlphaZero”, que descubre nuevos algoritmos para multiplicar matrices de forma más eficiente, una IA para el control magnético de plasmas tokamak a través del aprendizaje por refuerzo profundo [137] y “AlphaFold” [138]-[140] para la predicción de la estructura de proteínas.

Google lanzó un modelo de lenguaje llamado “BERT” [141] en 2018, que se utilizó para mejorar significativamente la precisión de las tareas de procesamiento del lenguaje natural, como la comprensión del lenguaje y la clasificación de texto, basándose en Transformer [142], una arquitectura de red neuronal que Google Research inventó y abrió en 2017. Posteriormente lanzaron “LaMDA” [143] en 2021, una IA que es capaz de mantener conversaciones naturales con humanos, hasta tal punto que el ingeniero de Google Blake Lemoine llegó a asegurar que tiene consciencia después de entrevistar a “LaMDA” [144], [145]. Más tarde, buscando cuál es el tamaño

óptimo del modelo y la cantidad de tokens de entrenamiento para un presupuesto de cómputo dado, desarrollaron Chinchilla [146], que supera a Gopher [147] con el costo de cómputo de entrenamiento a pesar de tener 70 billones de parámetros en comparación con los 280 billones de Gopher.

En 2022 Google anunció el LLM PaLM [148], que superó en rendimiento a todos los modelos conocidos hasta el momento y en el que se basan varias IAs en campos específicos como PaLM-SayCan [149], para que un robot realice tareas físicas siguiendo instrucciones textuales; Minerva [150], capaz de resolver cuestiones matemáticas y científicas usando razonamiento paso a paso; y Med-PaLM [151], para brindar respuestas seguras y útiles en el dominio médico. Finalmente, en 2023 anunciaron PaLM 2 [152], que impulsa el chatbot Bard [153] y el copiloto de Workspace [154], además de ser la base de Med-PaLM 2 [155], ahora con capacidades multimodales para sintetizar información como radiografías y mamografías; Codey, un modelo básico de texto a código; y Sec-PaLM, una versión especializada de PaLM 2 entrenada en casos de uso de seguridad y un salto potencial para el análisis de ciberseguridad [156].

Por su parte, OpenAI creó en 2018 GPT [157], en 2019 GPT-2 [158], gracias al cual se desarrolló el primer sistema de inteligencia artificial experto en comprender y escribir en lengua española, llamado MarIA [159], y en 2020 GPT-3 [160], capaz de realizar una amplia variedad de tareas de lenguaje natural, incluyendo la traducción, la escritura de código y la respuesta a preguntas, generando texto coherente y realista. Este último ha dado lugar a Codex [161], que traduce el lenguaje natural a código e impulsa a GitHub Copilot [162], y cuyo competidor es AlphaCode [163] de DeepMind. Más tarde, a partir de un modelo de la serie GPT-3.5, se desarrolló ChatGPT [164] usando aprendizaje por refuerzo a partir de retroalimentación humana (RLHF), mediante los mismos métodos que InstructGPT [165], dando lugar a una IA que interactúa de forma conversacional, capacitada para seguir una instrucción en un "prompt" y proporcionar una respuesta detallada. Posteriormente, lanzaron GPT-4 [166], que se introdujo en Microsoft 365 Copilot [167], en Bing [168] para potenciar las búsquedas, lo que ha hecho que ocurran sucesos similares a lo que pasó con LaMDA de Google [169], y en Windows Copilot [170], sustituyendo a Cortana.

También debemos mencionar otros grandes proyectos en este campo, como Jurassic-1 [171] de AI21 Labs, MT-NLG [172], resultado de un esfuerzo conjunto entre Microsoft y NVIDIA, BLOOM [173], desarrollado en el proyecto BigScience, y OPT-175B [174] de Meta AI. Estos dos últimos modelos democratizan el acceso a modelos lingüísticos a gran escala. Meta AI también ha entrenado un modelo de lenguaje en las secuencias de millones de proteínas naturales, llamado ESM [175], [176], para completar los espacios en blanco en una secuencia de proteínas, presentando evidencia de que los modelos de lenguaje aprenden las propiedades de las proteínas, como su estructura y función. Otro modelo destacado de Meta AI es NLLB [177], que es capaz de ofrecer traducciones de alta calidad directamente entre más de 200 idiomas, incluidos idiomas de bajos recursos como asturiano, luganda, urdu y más.

Con respecto a modelos de voz a texto, tenemos a Whisper [178], que es un sistema de reconocimiento automático de voz (ASR) de OpenAI entrenado con 680 mil horas de datos supervisados multilingües y multitarea recopilados de la web. Además, permite la transcripción en varios idiomas, así como la traducción de esos idiomas al inglés, y su principal competidor es Chirp [179] de Google, basado en el modelo USM [180]. También podemos hablar de una nueva generación de códecs de audio impulsados por IA, como Satin [181] de Microsoft, Lyra V2 [182] de Google y EnCodec [183] de Meta AI.

Por otro lado, respecto a los modelos de texto a voz, tenemos a VALL-E [184], un enfoque de modelado de lenguaje para la síntesis de texto a voz (TTS) de Microsoft, entrenado con 60 mil horas de habla en inglés, donde emergen capacidades de aprendizaje en contexto y se puede utilizar para sintetizar voz personalizada de alta calidad con solo una grabación registrada de 3 segundos de un hablante desconocido. También existe Voicebox [185] de Meta AI, siendo uno de los primeros en este campo WaveNet [186] de DeepMind, que entrenaba solo con 44 horas. Un enfoque relacionado es el de MusicLM [187] de Google Research, un modelo para generar música de alta fidelidad a partir de descripciones de texto que se basa en AudioLM [188] como componente generativo y confía en MuLan [189] para la generación de texto necesaria para el entrenamiento. También existe MusicGen [190] de Meta AI y una versión comercial llamada Mubert [191].

Observando estos modelos de IA, podemos pensar en combinarlos. Por ejemplo, utilizando el modelo Whisper para transcribir de voz a texto, el modelo ChatGPT como controlador [192], que, interactuando de forma conversacional, nos proporcione una respuesta detallada y, finalmente, el modelo VALL-E para sintetizar voz personalizada de alta calidad, como vemos en la Figura 2.2.

Otro enfoque es el que ha dado DeepMind con Gato [193], una red neuronal que con los mismos pesos puede ejecutar Atari, subtitular imágenes, chatear, apilar bloques con un brazo robótico real y mucho más, decidiendo en función de su contexto si generar texto, pares de unión, pulsaciones de botones u otros tokens. Además de modelos de IA multimodales como ImageBind [194] de Meta AI, el primer modelo capaz de vincular información de seis modalidades: texto, imagen/video, audio, sensores de profundidad (3D), térmicos (radiación infrarroja) y unidades de medición inercial (IMU), que calculan el movimiento y la posición.

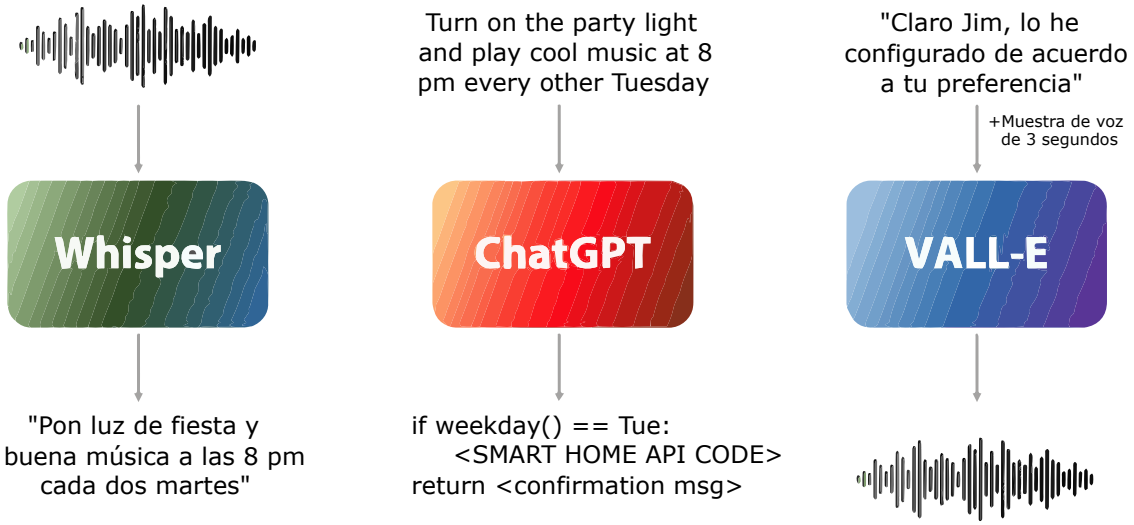


Figura 2.2: Ejemplo de concatenación de Whisper, ChatGPT y VALL-E.

2.1.3. Clasificación

Como hemos visto, la IA es un campo en constante evolución que ha dado lugar a una gran cantidad de investigación y desarrollo en los últimos años. A medida que esta tecnología avanza, es importante comprender y clasificar las distintas formas en las que puede manifestarse. En esta sección, exploraremos las distintas clasificaciones de la IA según su capacidad de pensamiento y acción, comparándolas con la mente humana, considerando las funciones que pueden realizar y el áreas de aprendizaje. Esta clasificación nos permitirá comprender mejor la complejidad y el potencial de la IA, así como sus aplicaciones y desafíos. Si hacemos la clasificación de la IA comparándola con la mente humana tenemos tres tipos, siendo estos:

- Inteligencia artificial estrecha (ANI): También llamada IA limitada o IA débil se refiere a sistemas que tienen habilidades limitadas y se enfocan en realizar una tarea específica. Estos sistemas tienen la capacidad de aprender y adaptarse a nuevas situaciones, pero su alcance es limitado y su conocimiento se basa en experiencias previas. Ejemplos de sistemas de IA limitada incluyen el asistente virtual Siri y el sistema de recomendación de Netflix.

- Inteligencia artificial general (AGI): También llamada IA fuerte se refiere a sistemas que tienen habilidades más avanzadas y pueden realizar una amplia gama de tareas. Estos sistemas que busca emular la inteligencia humana en su conjunto son capaces de aprender y adaptarse a situaciones complejas y variadas, lo que les permite resolver problemas de forma autónoma, y pueden realizar una amplia variedad de tareas sin necesidad de ser programados específicamente para cada una de ellas. Esta es la etapa más avanzada de la IA y, aunque se ha logrado cierto progreso en esta dirección, aún no se ha alcanzado completamente.

- Superinteligencia artificial (ASI): Se refiere a una máquina que supera la inteligencia humana en todos los dominios cognitivos y actualmente no existe.

Ahora nos enfocaremos en la segunda clasificación, es decir, según las funciones que puede realizar. En esta categoría, podemos identificar cuatro tipos de IA, que se diferencian en su capacidad para tomar decisiones, aprender, comprender emociones y tener una conciencia propia. Estos cuatro tipos son:

- Máquinas reactivas: Se trata de la etapa más básica de la IA, en la que un sistema puede tomar decisiones y realizar acciones basándose en su entorno inmediato, pero sin tener memoria ni capacidad de aprendizaje. Ejemplos de sistemas de IA reactiva incluyen el robot explorador de Marte Sojourner y el sistema de juego de ajedrez Deep Blue.
- Máquinas de memoria limitada: Capaces de realizar tareas específicas y aprender para mejorar, pero tienen memoria limitada. Ejemplo: Vehículos autónomos, asistentes virtuales y tecnología de reconocimiento facial.
- Teoría de la mente: Busca desarrollar máquinas que puedan comprender emociones, deseos, necesidades y creencias humanas, para tener una interacción social.
- Autoconsciente: Además de comprender emociones, deseos, necesidades y creencias humanas, tendrán emociones, deseos, necesidades y creencias propias.

Continuamos con la tercera clasificación de Stuart Russell y Peter Norvig [195], donde vemos las cuatro categorías de inteligencia artificial según su capacidad de pensamiento y acción [196]:

- Sistemas que piensan como humanos: Intentan emular la forma en que los humanos perciben, razonan y aprenden. Como por ejemplo una red neuronal artificial que se utiliza para el reconocimiento de imágenes.
- Sistemas que actúan como humanos: Imitan el comportamiento humano en tareas como la percepción, el movimiento y la manipulación de objetos. Como por ejemplo un robot humanoide.
- Sistemas que piensan racionalmente: Se enfocan en el razonamiento lógico y la toma de decisiones basadas en reglas y algoritmos. Como puede ser un sistema experto que utiliza una base de conocimiento y reglas para tomar decisiones y resolver problemas en un campo específico.
- Sistemas que actúan racionalmente: Imitan la acción racional de los humanos, como la planificación, la ejecución de tareas y la resolución de problemas. Como puede ser un agente inteligente que se utiliza en el comercio electrónico para ayudar a los usuarios a realizar compras en línea de manera eficiente y racional.

Por último, hablaremos de las áreas en las que se puede especializar la IA, siendo algunas de las más importantes:

- Aprendizaje automático (*Machine Learning*, ML): Se enfoca en desarrollar algoritmos y técnicas que permitan a las máquinas aprender de los datos que se les proporcionan.
- Redes neuronales: Son un tipo de modelo inspirado en el cerebro humano que es utilizado en el aprendizaje automático, para procesar y analizar datos.
- Aprendizaje profundo (*Deep Learning*, DL): Subárea de aprendizaje automático que utiliza redes neuronales profundas (con muchas capas) que pueden aprender a partir de grandes cantidades de datos para realizar tareas complejas y aprender representaciones abstractas de los datos.
- Procesamiento del lenguaje natural (*Natural Language Processing*, NLP): Campo de la inteligencia artificial que se centra en desarrollar sistemas que puedan comprender (NLU), generar (NLG) y manipular lenguaje humano natural.
- Visión computacional: Rama de la inteligencia artificial que se enfoca en cómo las computadoras pueden interpretar y comprender imágenes y vídeos. La generación de imágenes es un campos complementario y estrechamente relacionado que utiliza ese conocimiento para crear imágenes sintéticas de alta calidad.
- Audición máquina o *Machine hearing*: Subárea de IA que se enfoca en procesamiento y interpretación de señales sonoras. Incluye tareas como reconocer el lenguaje hablado (ASR), síntesis de voz (TTS) y el análisis de escena auditiva computacional (CASA).

En las siguientes secciones hablaremos más sobre alguno de estos conceptos.

2.2. Audición máquina

Como se ha indicado anteriormente *machine hearing* es un campo de la inteligencia artificial que se centra en desarrollar sistemas informáticos que imiten la forma en que los humanos escuchan y comprenden el sonido mediante la combinación de técnicas de procesamiento de señales, modelos de la psicología del oído humano y algoritmos de aprendizaje automático. Estos últimos son capaces de mejorar la precisión de estos sistemas y desarrollar nuevas aplicaciones analizando sonidos e identificando patrones, como los sonidos de diferentes palabras o instrumentos.

Existe una gran variedad de elementos sonoros como la voz humana, la música, el silencio, los ruidos y efectos sonoros, que serán analizados por separado aunque en la práctica se combinen de manera natural. En general, la audición máquina tiene el potencial de mejorar una amplia gama de tecnologías y aplicaciones diferentes como la síntesis de voz que convierte el texto en habla sintética (TTS), la identificación de una persona en base a su voz (SR), la mejora de la calidad de la señal de audio (SE), el reconocimiento de las emociones en el habla humana (SER), el análisis de escena auditiva computacional (CASA) que busca la comprensión de la escena acústica compleja e identificación de los distintos sonidos presentes en una señal de audio, el análisis musical que puede identificar instrumentos individuales en una grabación musical, la localización de sonidos capaz de determinar la ubicación de una fuente de sonido en un entorno ruidoso y el reconocimiento automático del habla (ASR), siendo su principal aplicación la transcripción del lenguaje hablado en tiempo real (STT) y una de sus partes más importantes la representación matemática de la relación entre la señal de audio y el texto que se está diciendo, que llamamos modelado acústico. Estos términos están estrechamente relacionados y su combinación crea sistemas capaces de comprender y responder al habla humana de manera más natural y efectiva.

El objetivo de este trabajo es la detección de tos en señales de audio ruidosas de varios pacientes con distintas patologías respiratorias. Esto implica el uso del CASA para el análisis de sonidos ambientales y el ASR para el reconocimiento del habla y otros sonidos en las grabaciones de audio que contienen toses. De esta forma, se pretende contribuir a la detección precoz de enfermedades respiratorias mediante el procesamiento de señales de audio ruidosas. También se podría emplear el SR para identificar al paciente o el SE para mejorar la calidad de la señal de audio de la tos.

Para analizar sonidos, los sistemas de audición máquina extraen las propiedades físicas del audio mediante el modelado matemático de la señal acústica como ondas sonoras. Estas propiedades físicas del audio son el objeto de estudio de la acústica, una rama de la física que se ocupa de cómo se genera, se propaga y se percibe el sonido. El sonido es una onda mecánica que necesita un medio elástico e inercia para transmitirse y su energía acústica puede transformarse en energía potencial y cinética.[197].

Hay varias características físicas del audio que se pueden utilizar para describir la señal de audio en diferentes dominios. En el dominio del tiempo, estas características incluyen la tasa de cruce por cero, la amplitud, la energía y el ritmo. En el dominio de la transformada de Fourier, las características incluyen la autocorrelación, la forma del espectro, la envolvente del espectro, la tonalidad y el croma. Además, también hay técnicas de búsqueda coincidente basadas en Gabor y representaciones del tensor de codificación dispersa. Otros dominios que se utilizan para describir la señal de audio incluyen el dominio de la transformación de wavelet, el dominio de la imagen, el dominio cepstral y el dominio del espectro de alto orden, así como el espacio de fase y el espacio propio [198].

Los sistemas de audición máquina también utilizan modelos basados en la psicología del oído humano para tener en cuenta las características perceptivas del audio en su análisis. Esto les permite interpretar los sonidos de manera más precisa y natural, y mejorar su capacidad para realizar tareas como el reconocimiento de habla o la identificación de instrumentos musicales. Otra forma de obtener características perceptivas es a través del análisis de las reacciones fisiológicas del oyente, como la actividad eléctrica del cerebro o la actividad muscular en la cara.

Las características perceptivas o prosódicas son aquellas que se refieren a cómo el sonido es percibido por el oído humano y tratan de modelar cómo la información acústica se procesa y se percibe en el cerebro humano, considerando aspectos que pueden ser diferentes a sus características físicas. Por ejemplo, la claridad, presencia o nitidez son utilizados para describir la calidad de una señal de audio. La intensidad percibida de un sonido no siempre coincide con su intensidad física al igual que ocurre con la duración. Además, el oído humano es más sensible a algunas frecuencias que a otras, lo que puede afectar la forma en que un sonido es percibido. También es importante tener en cuenta el contexto en el que se produce un sonido, ya que puede afectar su percepción, por lo que se debe tener en cuenta la relación entre los diferentes sonidos y la historia previa de los sonidos para

ayudar a interpretar su significado. Por otro lado, pueden proporcionar información con significado semántico en el contexto de los oyentes humanos como la entonación y el ritmo de la voz que pueden indicar el estado emocional del hablante o el significado de las palabras en una frase.

Las cuatro cualidades subjetivas que se utilizan para describir un sonido musical son: intensidad, tono, timbre y duración. Estas dependen de los parámetros físicos como la frecuencia, la presión sonora, el espectro de parciales, la forma de la onda, la envolvente, etc. El tono está determinado principalmente por la frecuencia que se mide en hercios (Hz) y se refiere a la cantidad de veces que se repite un sonido por segundo, pero también puede verse afectado por la presión y la envolvente. La duración física y la duración percibida están relacionadas, pero no son iguales, ya que la duración percibida se refiere al intervalo temporal en que el sonido persiste sin discontinuidad [199].

La intensidad está relacionada con la amplitud de la frecuencia y depende principalmente de la presión sonora que se mide en Pascal aunque también influyen el espectro de parciales (armónicos) y la duración. La presión acústica es la magnitud física que indica la fuerza del sonido y la fuente de acústica determina cómo se propaga el sonido en el espacio. La variación de la intensidad sonora con el tiempo se parece a un montículo en el que hay que distinguir tres fases ataque, estadio cuasiestacionario y decaimiento.

El timbre es la capacidad de distinguir entre dos sonidos de igual frecuencia e intensidad pero emitidos por fuentes diferentes y se debe a que generalmente un sonido no es puro por lo que se relaciona con la calidad del sonido, debido a la forma en que se produce el sonido y de las características de la fuente sonora. Está influenciado por principalmente por el espectro de parciales (armónicos) aunque también depende de la forma de la onda y la envolvente [200].

Podemos ver diferentes formas de analizar y representar la información acústica que captan los oídos humanos, es decir, las características perceptivas del audio. Estos dominios incluyen el dominio del tiempo, el dominio de la transformada de Fourier, el dominio de transformación de wavelet, el dominio espectro-temporal multiescala, el dominio de la imagen y el dominio cepstral.

En el dominio del tiempo, las características incluyen la amplitud pico de cruce por cero y la función de autocorrelación. En el dominio de la transformada de Fourier, las características están relacionadas con la modulación, el brillo, la tonalidad y la sonoridad, y se pueden analizar a corto y largo plazo. En el dominio de transformación de wavelet, las características incluyen coeficientes de orientación del flujo de potencia del kernel, coeficientes de wavelets discretos de frecuencia de Mel y paquetes wavelet perceptuales. En el dominio de la imagen, las características incluyen la característica de imagen del espectrograma y las imágenes de tiempo-croma. En el dominio cepstral, las características se basan en el banco de filtros y en la autorregresión [198].

2.3. Aprendizaje automático

El aprendizaje automático o *Machine Learning* (ML) es una subárea de la IA que se enfoca en el desarrollo de algoritmos y técnicas que permiten a las máquinas "aprender" a partir de datos y mejorar su rendimiento en una tarea específica a medida que reciben más datos, sin necesidad de programación explícita. La cantidad, calidad y relevancia de los datos disponibles es fundamental para entrenar modelos de ML de manera efectiva, ya que, estos sistemas transforman los datos en información, y con esta información pueden tomar decisiones. En esta sección, hablaremos sobre el método de trabajo que suele realizar un científico de datos junto con la existencia del AutoML para algunas tareas sistemáticas, además de los tipos de aprendizaje automático que existen y los algunos clasificadores.

En la Figura 2.3 podemos ver que el proceso de análisis de datos consta de tres etapas principales que son importación y limpieza de datos, exploración y modelado de datos, y comunicación de resultados. Durante estas etapas, el científico de datos trabaja para destilar los datos crudos y extraer información valiosa.

En la etapa de importación y limpieza de datos obtenemos los datos limpios y libres de errores para posteriormente en la etapa de exploración y modelado decidir qué transformaciones aplicar a los datos para crear características que sirvan como indicadores del modelo subyacente. Luego, divide el conjunto de datos en un conjunto de entrenamiento y uno de prueba para determinar su capacidad para generalizar y hacer predicciones precisas en datos no vistos previamente evaluando así la fiabilidad del modelo. A continuación, el científico de datos aplica un proceso iterativo en el que prueba diferentes modelos, y observa y mide los parámetros matemáticos que expresan

su rendimiento. Además, puede variar el conjunto de datos de entrenamiento con nuevas transformaciones para calcular características que mejoren el ajuste del modelo a los datos.

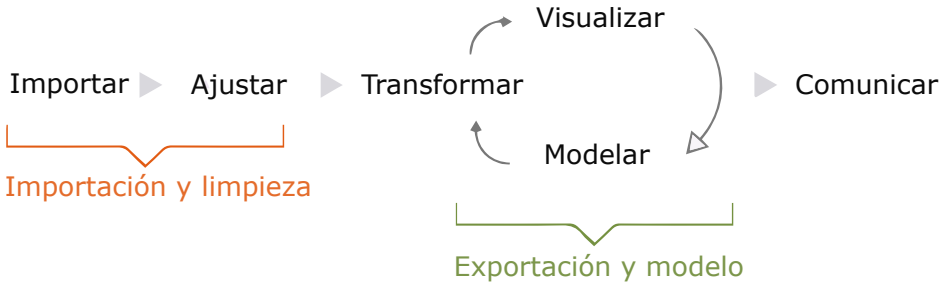


Figura 2.3: Proceso del análisis de datos [201].

Debido al proceso iterativo que realiza el científico de datos un sistema de AutoML puede complementarlo en el proceso de creación de modelos. Algunos de los beneficios que puede aportar son la sugerencia de las mejores técnicas de ML, la generación de modelos optimizados ajustando automáticamente los parámetros, la identificación de características relevantes para el modelo, la generación de visualizaciones para entender el resultado del proceso y la creación de un entorno interactivo para explorar el modelo. Algunos de los sistemas y herramientas AutoML cubren la totalidad del proceso como H2O.ai, Auto-Sklearn y TPOT [202], sin embargo, existen más soluciones y herramientas que cubren parcialmente alguno de los pasos del proceso completo [203].

El proceso de aprendizaje consiste en el análisis de grandes cantidades de datos para encontrar patrones y relaciones que permitan realizar predicciones precisas. Hay diferentes tipos de aprendizaje automático estos son el aprendizaje supervisado, aprendizaje no supervisado, aprendizaje semi-supervisado y aprendizaje por refuerzo por lo que elegiremos entre los distintos tipos de aprendizaje dependiendo de la tarea que queremos abordar y los datos disponibles.

El más común es el aprendizaje supervisado que consiste en aprender a partir de un conjunto de datos etiquetados, en los que se conoce la respuesta correcta. El sistema luego usaría estos datos para aprender a clasificar correctamente nuevas imágenes. Hay dos modelos dependiendo del tipo de etiqueta [204]:

- Los algoritmos de clasificación se usan para predecir etiquetas. Estos algoritmos se usan a menudo para predecir si una imagen contiene un gato o un perro, si un correo electrónico es spam o no, etc. Por ejemplo, si se está tratando de crear un sistema para reconocer imágenes de gatos, se le proporcionaría un conjunto de imágenes etiquetadas como “gato” o “no gato”. Los algoritmos de clasificación más comunes son la regresión logística, los bosques aleatorios, los árboles de decisión, las máquinas de vectores de soporte (SVM), el clasificador Naive Bayes, el método de los K-NN y la red neuronal.

- Los algoritmos de regresión se utilizan para predecir una variable numérica a partir de un conjunto de variables independientes. Estos algoritmos se usan a menudo para realizar predicciones en el ámbito financiero, predecir el consumo energético de edificios, el tiempo de entrega de un producto, etc. Por ejemplo, tomando como entrada un conjunto de características de la casa (e.g., tamaño, ubicación, número de habitaciones, etc.) para aprender cómo estas características afectan el precio de la casa con el objetivo de predecir el precio de una casa dada un conjunto de características. Los algoritmos de regresión más comunes son la regresión lineal, la regresión polinómica, la regresión paso a paso, la regresión Ridge (utiliza una penalización de la norma L_2), la regresión Lasso (utiliza una penalización de la norma L_1), la regresión ElasticNet (utiliza una penalización que combina la norma L_2 y L_1), los CART y los bosques aleatorios.

Por otra parte, el aprendizaje no supervisado consiste en desarrollar sistemas que puedan aprender a partir de datos no etiquetados, sin conocer la salida deseada para cada entrada. Estos sistemas analizan los datos sin etiquetas buscando patrones en ellos sin tener una respuesta correcta. Algunos campos donde se aplican las técnicas de aprendizaje automático no supervisado son el marketing y la biología e incluyen [205]:

- La agrupación en clústeres permite dividir automáticamente el conjunto de datos en grupos según la similitud. Sin embargo, a menudo, el análisis de clústeres sobreestima la similitud entre los grupos y no trata los puntos de datos como individuos. Por esta razón, el análisis de clústeres es una mala elección para aplicaciones como la segmentación y la selección de clientes. Algoritmos utilizados son k-medias, agrupamiento jerárquico y el agrupamiento espacial basado en densidad de aplicaciones con ruido (DBSCAN).

- La detección de anomalías puede descubrir automáticamente puntos de datos inusuales en su conjunto de datos. Esto es útil para identificar transacciones fraudulentas, descubrir piezas de hardware defectuosas o identificar un valor atípico causado por un error humano durante la entrada de datos.

- La minería de asociaciones identifica conjuntos de elementos que ocurren juntos con frecuencia en su conjunto de datos. Los minoristas suelen utilizarlo para el análisis de la cesta, ya que permite a los analistas descubrir productos que a menudo se compran al mismo tiempo y desarrollar estrategias de marketing y comercialización más eficaces.

- Los modelos de variables latentes se usan comúnmente para el preprocesamiento de datos, como reducir la cantidad de características en un conjunto de datos (reducción de dimensionalidad) o descomponer el conjunto de datos en múltiples componentes. Ejemplos de algoritmos de reducción de dimensionalidad son PCA, t-SNE,ICA, LDA o autoencoder.

Una combinación de ambos tipos de aprendizaje es el aprendizaje semi-supervisado que se utiliza cuando hay un conjunto de datos parcialmente etiquetado. El modelo se entrena con ambos tipos de ejemplos, usando los etiquetados para aprender a hacer predicciones, mientras que los no etiquetados se utilizan para encontrar patrones y estructuras subyacentes en los datos. La ventaja es que al tener más datos, aunque no tengan etiquetas, se puede aumentar el rendimiento del modelo en comparación con el aprendizaje supervisado con un conjunto de datos menor [204].

En la detección de fraude en transacciones financieras un ejemplo de aprendizaje semisupervisado podría ser que al tener grandes cantidades de datos de transacciones financieras, pero solo un pequeño porcentaje de ellas son fraudulentas y, por lo tanto, están etiquetadas como tales. En lugar de etiquetar manualmente todas las transacciones, se podría utilizar el aprendizaje semisupervisado para identificar patrones en los datos y etiquetar automáticamente las transacciones no etiquetadas que se parezcan a las transacciones fraudulentas conocidas. Luego, se utilizarían las transacciones etiquetadas manualmente junto con las etiquetadas automáticamente para entrenar un modelo supervisado para detectar fraudes en las transacciones. Esto podría resultar en una detección de fraude más eficiente y precisa con menos necesidad de etiquetar manualmente todas las transacciones.

Por otra parte tenemos el aprendizaje por refuerzo que se enfoca en desarrollar sistemas que puedan aprender a partir de la retroalimentación que reciben al interactuar con el entorno como vemos en la Figura 2.4. Se utiliza en entornos donde hay poca información donde un agente aprende a tomar decisiones en un entorno determinado a través de prueba y error, lo que implica entrenar un modelo mediante la recompensa de los comportamientos deseados o el castigo de los no deseados en función de sus acciones con el objetivo obtener una recompensa general máxima y lograr una solución óptima. El aprendizaje por refuerzo se puede aplicar en una variedad de campos, como la robótica, la optimización de recursos o los sistemas de control, donde un agente debe aprender a tomar decisiones para maximizar una recompensa en un entorno incierto y complejo. Además se ha utilizado ampliamente en juegos para entrenar agentes inteligentes, como AlphaGo, ya que estos juegos ofrecen un entorno claro y bien definido en el que se pueden medir claramente los éxitos y fracasos del agente [206].

El aprendizaje por refuerzo a partir de la retroalimentación humana (RLHF) es un enfoque de aprendizaje por refuerzo que se basa en la retroalimentación de un humano para orientar el aprendizaje del agente. El humano le da al agente retroalimentación positiva o negativa según las acciones que realiza el agente, en vez de usar una función de recompensa predefinida por un experto o definida de forma heurística. RLHF puede ser útil cuando es difícil o costoso definir una función de recompensa, o cuando el entorno de aprendizaje es complejo y cambiante, lo que hace que el uso de una función de recompensa fija sea limitante. Además, RLHF puede ser beneficioso en entornos de aprendizaje donde se deben considerar las preferencias o valores humanos, como en aplicaciones de asistencia sanitaria o chatbots como ChatGPT.

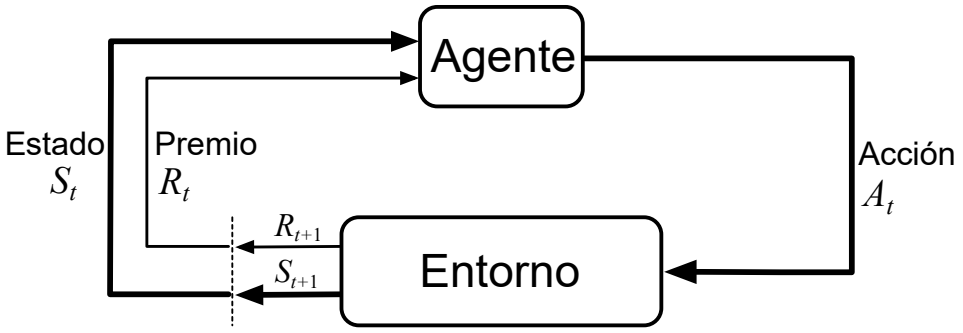


Figura 2.4: Esquema de aprendizaje por refuerzo [207].

Finalmente vamos a realizar una descripción de alto nivel de algunos de los algoritmos mencionados a lo largo de esta sección exceptuando las redes neuronales y autoencoders de los cuales hablaremos en el capitulo 4:

- Regresión logística: este clasificador se utiliza en aprendizaje de máquina supervisado cuando las clases son mutuamente exclusivas para crear una frontera en los datos con el objetivo de separar clases y encontrar patrones ocultos.
- Árboles de decisión: son algoritmos que permiten tomar decisiones basadas en el análisis de un conjunto de reglas y condiciones. El algoritmo conocido como particionamiento recursivo es el proceso paso a paso para construir un árbol de decisión y es la clave para el método estadístico no paramétrico CART [208]. Cada nodo del árbol representa una característica o una pregunta sobre una característica, y cada rama del árbol representa un posible resultado de la pregunta. Cuando se utiliza un nuevo dato, se sigue el camino a través del árbol hasta llegar a una hoja, que indica a qué clase pertenece el dato. Los CART pueden manejar clases no linealmente separables y pueden ser utilizados para realizar predicciones basadas en reglas if-then. Es muy tipico reaclizar una combinación de árboles predictores mediante aprendizaje por conjunto que es una técnica de aprendizaje automático en la que se combinan múltiples modelos para mejorar el rendimiento general. Dos métodos populares de aprendizaje por conjunto son *bagging* y *boosting*:
  - Los bosques aleatorios o *Random Forest* son un ejemplo popular de un método de conjunto de *bagging* en el que se entrenan árboles de decisión en subconjuntos aleatorios de los datos y después promedia sus predicciones para hacer la predicción final. En promedio, un aprendiz compuesto usualmente es mejor que un aprendiz de una sola base por que la variación es menor.
  - *Boosting* construye múltiples modelos débiles de forma secuencial para gradualmente reducir el sesgo o *bias* de un aprendiz compuesto. El aprendiz compuesto puede ajustar los datos bien, por lo que puede causar sobre-adaptación o *overfitting*. GBDT, XGBoost y Adaboost son ejemplos populares de métodos de conjunto de *boosting*. En GBDT, se entrenan árboles de decisión en secuencia, y cada nuevo árbol intenta corregir los errores de los árboles anteriores. XGBoost es una versión mejorada de GBDT con características adicionales como regularización y procesamiento en paralelo. Adaboost pondera los ejemplos de entrenamiento para que el siguiente modelo se centre en los ejemplos que fueron mal clasificados por el modelo anterior.
- Máquinas de vectores de soporte (SVM): estas utilizan una función de kernel para encontrar una frontera de decisión que maximice la separación entre las diferentes clases en el conjunto de datos y son algoritmos que se utilizan para resolver problemas de clasificación y regresión.
- Clasificador bayesiano ingenuo: se basa en el teorema de Bayes con algunas hipótesis simplificadoras adicionales para construir un modelo de predicción a partir de un conjunto de datos que estima la probabilidad de que un nuevo ejemplo pertenezca a cada clase posible. Recibe el apelativo de ingenuo o *naive* a causa de estas simplificaciones, que se suelen resumir en la hipótesis de independencia entre las variables predictoras.

- k vecinos más próximos (K-NN): es un algoritmo de aprendizaje supervisado no paramétrico que generalmente utiliza la distancia euclidiana para medir la similitud entre los ejemplos de entrenamiento y un nuevo ejemplo. Basándose en la mayoría de las categorías de los k vecinos más cercanos asigna la etiqueta que más se repite entre los vecinos cercanos al nuevo punto de datos.
- Regresión lineal: es una técnica de aprendizaje supervisado que se utiliza para modelar la relación entre una variable dependiente Y continua con una o más variables independientes o regresoras X generando un hiperplano de parámetros que en el caso más sencillo, con una sola variable explícita, es una recta. Por lo tanto, esta técnica de análisis de datos en el caso más sencillo predice el valor de datos desconocidos mediante el uso de otro valor de datos relacionado y una línea recta.
- Análisis discriminante lineal (LDA): es una generalización del discriminante lineal de Fisher utilizada para separar grupos de observaciones buscando encontrar una combinación lineal de características que maximice la separación entre las clases. El LDA se puede usar para reducir la dimensionalidad de un conjunto de datos y también se puede utilizar para predecir la pertenencia a una categoría de una observación basada en un conjunto de predictores realizando un trabajo de clasificación.

Como hemos visto estos son solo algunos ejemplos, y hay muchos otros tipos de algoritmos de ML disponibles por lo que es importante tener en cuenta que cada algoritmo tiene sus propias fortalezas y debilidades, y es importante evaluar cuál es el más adecuado para cada caso de uso específico.

Capítulo 3

TÉCNICAS PARA EVALUAR EL RENDIMIENTO DE UN CLASIFICADOR

La evaluación del rendimiento de un clasificador es una tarea esencial en el campo de la inteligencia artificial y el aprendizaje automático. En este capítulo vamos a ver las técnicas para evaluar el funcionamiento de un sistema, que hemos utilizado en este trabajo para evaluar el sistema de detección y clasificación propuesto.

3.1. Descripción de las métricas

En esta sección hablaremos de los parámetros con los que hemos medido de forma cuantificable nuestra red neuronal para rastrear y evaluar su comportamiento a la hora de predecir las clases con las que previamente ha sido entrenada.

La matriz de confusión es una herramienta comúnmente utilizada en el análisis de resultados de modelos de clasificación en ML y será La base para cuantificar el comportamiento de nuestra red neuronal. Muestra la cantidad de veces que el modelo clasificó correctamente y de manera incorrecta a cada clase proporcionando así un aspecto visual de cómo un modelo de aprendizaje automático comete errores sistemáticos en sus predicciones para los modelos de clasificación. La palabra "confusión" en el nombre proviene de un modelo "confuso" o de un mal etiquetado de muestras [209], por lo que es una herramienta excelente que nos permite visualizar el desempeño de un algoritmo de Inteligencia Artificial de aprendizaje supervisado como es nuestro caso.

El número de predicciones de cada clase es representado en las columnas y las instancias de la clase real en las filas. Uno de los beneficios de las matrices de confusión es que facilitan ver si el sistema está confundiendo dos clases. Por ejemplo, si estamos construyendo un modelo de clasificación binaria que predice si un correo electrónico es spam o no spam, entonces la matriz de confusión podría tener dos filas y dos columnas. La fila superior correspondería a los correos electrónicos que fueron etiquetados como spam, y la fila inferior correspondería a los correos electrónicos que fueron etiquetados como no spam. La columna izquierda correspondería a los correos electrónicos que el modelo predijo como spam, y la columna derecha correspondería a los correos electrónicos que el modelo predijo como no spam.

En un sistema multiclase, para cuantificar el comportamiento de nuestro clasificador en cada clase, debemos dividir nuestra matriz de confusión en regiones. Para entender estas regiones nos vamos a apoyar en la Figura 3.1, donde podemos apreciar con claridad como se divide la matriz en cuatro regiones de interés, siendo estas:

- **Verdaderos Positivos (VP)**, en esta región encontraremos las veces que el modelo ha predicho correctamente un resultado positivo, es decir, un caso en el que el modelo ha predicho correctamente que un elemento pertenece a una determinada clase. Por ejemplo una persona que tiene COVID-19 y el modelo lo clasificó como COVID-19 positivo.

- **Falsos Negativos (FN)**, en esta región encontraremos las veces que el modelo ha predicho incorrectamente un resultado negativo cuando en realidad era positivo, es decir, un caso en el que el modelo ha predicho inco-

rrectamente que un elemento no pertenece a una cierta clase por lo que no ha tenido éxito en su predicción. Por ejemplo una persona que tiene COVID-19 y el modelo lo clasificó como COVID-19 negativo.

- **Falsos Positivos (FP)**, en esta región encontraremos las veces que el modelo ha predicho un resultado positivo cuando en realidad era negativo, es decir, un caso en el que el modelo ha predicho incorrectamente que un elemento pertenece a una cierta clase, y por lo tanto no era de esa clase. Los falsos positivos son "falsas alarmas" de un clasificador. Por ejemplo una persona que no tiene COVID-19 y el modelo lo clasificó como COVID-19 positivo.

- **Verdaderos Negativos (VN)**, en esta región encontraremos las veces que el modelo ha predicho un resultado negativo y ha acertado, es decir, un caso en el que el modelo ha predicho correctamente que un elemento no pertenece a una cierta clase. Los verdaderos negativos son "rechazos correctos" de un clasificador. Por ejemplo una persona que no tiene COVID-19 y el modelo lo clasificó como COVID-19 negativo.

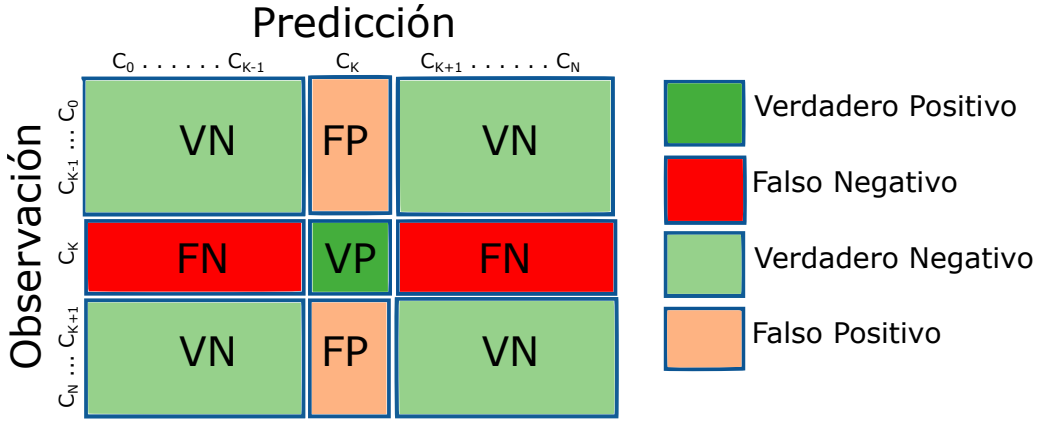


Figura 3.1: Regiones de la matriz de confusión [210].

Observamos en la Figura 3.2 el termino *Gold Standard* (GS) o test de referencia, este se refiere a una medida de referencia que se utiliza como punto de comparación para evaluar el rendimiento de un modelo. Por ejemplo, si estamos evaluando un modelo que detecta una enfermedad, el GS podría ser el resultado de una prueba médica confiable, es decir, que tienen la máxima fiabilidad a la hora de diagnosticar una determinada enfermedad. Esto no conlleva que la prueba posea la máxima fiabilidad en términos absolutos, por tanto, podría decirse que el estatus de *Gold Standard* se aplica a aquellos tests de fiabilidad máxima dentro de una serie de condiciones específicas.

Además también debemos señalar que llamamos test positivo a la cantidad total de casos en los que el modelo ha predicho que un elemento pertenece a una cierta clase y test negativo cuando el modelo ha predicho que un elemento no pertenece a una cierta clase, mientras que llamamos condición positiva a la cantidad total de casos en los que un elemento pertenece a una cierta clase y condición negativa cuando un elemento no pertenece a una cierta clase.

Para cuantificar el comportamiento de un clasificador con cada clase en un sistema multiclase, entendiendo lo que representa la matriz de confusión y conociendo sus regiones de interés, se utiliza una tabla de contingencia como la mostrada en la Figura 3.2. Esta tabla permite calcular diversas métricas para medir el rendimiento del clasificador. Además en esta tabla observamos cuales son las métricas habituales y como se calculan.

Entre las métricas más comúnmente utilizadas en la literatura médica sobre estudios diagnósticos, solo vamos a utilizar cinco de las más usadas que detallamos a continuación:

- **Exactitud (*Accuracy*, ACC)**, se refiere a la proporción de predicciones correctas que ha realizado el modelo en comparación con el GS o también se puede entender como la fracción de predicciones correctas que hizo

el modelo en comparación con el número total de predicciones realizadas. Para funcionar mejor en conjuntos de datos desequilibrados se puede utilizar la exactitud balanceada BA.

- **Sensibilidad (*Sensitivity*, SEN)**, también denominada razón o ratio de verdaderos positivos (*True Positive Rate*, TPR), se define como el número de veces que el resultado del test es positivo y la condición es verdadera sobre el número total de veces que la condición es verdadera en el GS. Por lo que se refiere a la habilidad del modelo para detectar correctamente una determinada clase. Por ejemplo, en el caso de un modelo que detecta una enfermedad, la sensibilidad se refiere a la habilidad del modelo para detectar correctamente la enfermedad. En un paciente determinado, si aplicamos un examen altamente sensible (identifica muy bien a los enfermos) y obtenemos un resultado negativo, podemos descartar razonablemente la enfermedad. Su métrica complementaria es la tasa de falsos negativos (FNR) que se calcula de manera similar, pero en lugar de contar el número de veces obtenemos un verdadero positivo, se cuenta el número de veces que obtenemos un falso negativo.

- **Especificidad (*Specificity*, SPE)**, también denominada razón o ratio de verdaderos negativos (*True Negative Rate*, TNR), se define como el número de veces que el resultado del test es negativo y la condición es falsa sobre el número total de veces que la condición es falsa en el GS. Por lo que se refiere a la habilidad del modelo para identificar correctamente cuando una determinada clase no está presente. Por ejemplo, en el caso de un modelo que detecta una enfermedad, la especificidad se refiere a la habilidad del modelo para identificar correctamente a las personas que no tienen la enfermedad. Si un paciente tiene un resultado positivo en un test altamente específico podemos confirmar la enfermedad. En sentido estricto nos dice qué tan bueno es el test en excluir a los sanos. Su métrica complementaria es la tasa de falsos positivos (FPR) que se calcula de manera similar, pero en lugar de contar el número de veces obtenemos un verdadero negativo, se cuenta el número de veces que obtenemos un falso positivo.

- **Valor predictivo positivo (*Positive Predictive Value*, PPV)**, también denominado precisión, se define como la cantidad de predicciones positivas que el modelo hizo correctamente para una clase particular en comparación con el número total de predicciones positivas que hizo para esa clase por lo que se refiere a la probabilidad de que una persona tenga la enfermedad si el modelo ha predicho correctamente que la tiene. Su métrica complementaria es la tasa de descubrimiento falso (FDR) que se calcula de manera similar, pero en lugar de contar el número de veces obtenemos un verdadero positivo, se cuenta el número de veces que obtenemos un falso positivo.

- **Valor predictivo negativo (*Negative Predictive Value*, NPV)**, se define como la cantidad de predicciones negativas que el modelo hizo correctamente para una clase particular en comparación con el número total de predicciones negativas que hizo para esa clase por lo que se refiere a la probabilidad de que una persona no tenga la enfermedad si el modelo ha predicho que no la tiene. Su métrica complementaria es la tasa de omisión falsa (FOR) que se calcula de manera similar, pero en lugar de contar el número de veces obtenemos un verdadero negativo, se cuenta el número de veces que obtenemos un falso negativo.

Si bien los valores predictivos proporcionan información clínicamente relevante sobre la probabilidad de que un paciente tenga o no una condición en función del resultado del test, estos valores están estrechamente relacionados con la prevalencia de la enfermedad en la población estudiada, y por lo tanto, no son independientes de ella. Debido a esta característica sólo es utilizable si nos enfrentamos a pacientes similares a aquellos en que se realizó el estudio, ya que, varían enormemente dependiendo de la prevalencia o riesgo basal de la condición.

Lo anterior no ocurre con la sensibilidad y especificidad, ya que no varían con la prevalencia de la condición, aunque a diferencia de los valores predictivos no nos hablan de la probabilidad que tiene un paciente de presentar la enfermedad de interés [212]. Esto ha hecho que constituyan una de las formas más frecuentes de expresar el rendimiento de un test aunque ambas son importantes para interpretar los resultados de un estudio. Además hay otras métricas como el umbral de prevalencia (PT) [213] que es el nivel de prevalencia en la curva de detección por debajo del cual las pruebas de detección caen más precipitadamente [214] o la razón de verosimilitud positiva (PLR) y negativa (NLR) que indican cuán bien un modelo puede diferenciar entre dos clases basadas en la sensibilidad y especificidad, que a su vez sirven para calcular la razón de probabilidades diagnósticas (DOR).

		CONDICIÓN Determinada por el "Gold Standard"			
	POBLACIÓN TOTAL VP+VN+FP+FN	CONDICIÓN POSITIVA VP+FN	CONDICIÓN NEGATIVA FP+VN	PREVALENCIA $P=\frac{\text{CONDICIÓN POSITIVA}}{\text{POBLACIÓN TOTAL}}$	UMBRAL DE PREVALENCIA $PT=\frac{\sqrt{FPR}}{\sqrt{TPR}+\sqrt{FPR}}$
RESULTADO DE LA PRUEBA	TEST POSITIVO VP+FP	Acierto Verdadero Positivo VP	Error Tipo I o subestimación Falsa alarma Falso Positivo FP	Precisión Valor Predictivo Positivo $PPV=\frac{VP}{\text{TEST POSITIVO}}$	Tasa de Descubrimiento Falso $FDR=\frac{FP}{\text{TEST POSITIVO}}$
	TEST NEGATIVO FN+VN	Error Tipo II o sobreestimación Error Falso Negativo FN	Rechazo correcto Verdadero Negativo VN	Tasa de Omisión Falsa $FOR=\frac{FN}{\text{TEST NEGATIVO}}$	Valor Predictivo Negativo $NPV=\frac{VN}{\text{TEST NEGATIVO}}$
	EXACTITUD ACCURACY (ACC) $ACC=\frac{VP+VN}{\text{POBLACIÓN TOTAL}}$	Tasa de aciertos Recuperación, Exhaustividad Sensibilidad (SEN), Recall Tasa de Verdaderos Positivos $TPR=\frac{VP}{\text{CONDICIÓN POSITIVA}}$	Caída, Fall-Out Probabilidad de falsa alarma Tasa de Falsos Positivos $FPR=\frac{FP}{\text{CONDICIÓN NEGATIVA}}$	Razón de Verosimilitud Positiva $PLR=\frac{TPR}{FPR}=\frac{SEN}{1-SPE}$	Razón de Probabilidades Diagnósticas $DOR=\frac{PLR}{NLR}$
	EXACTITUD BALANCEADA BALANCED ACCURACY (BA) $BA=\frac{TPR+TNR}{2}=\frac{SEN+SPE}{2}$	Tasa de error Tasa de Falsos Negativos $FNR=\frac{FN}{\text{CONDICIÓN POSITIVA}}$	Selectividad Especificidad (SPE) Tasa de Verdaderos Negativos $TNR=\frac{VN}{\text{CONDICIÓN NEGATIVA}}$	Razón de Verosimilitud Negativa $NLR=\frac{FNR}{TNR}=\frac{1-SEN}{SPE}$	Valor-F $F_\beta=(1+\beta^2)\cdot\frac{PPV\cdot TPR}{(\beta^2\cdot PPV)+TPR}$ $F_1=2\cdot\frac{\text{Precisión}\cdot SEN}{\text{Precisión}+SEN}$

Figura 3.2: Tabla de contingencia y métricas habituales [211].

En la visión por computadora la métrica de evaluación que usamos para medir la efectividad de un detector de objetos, independientemente de la naturaleza del mismo es la intersección sobre unión (IoU). Es una medida de similitud entre dos conjuntos de datos. En el contexto de la detección de objetos en imágenes, IoU se utiliza para medir la superposición entre una caja delimitadora predicha o máscara de segmentación(SM) y una caja delimitadora verdadera o *ground truth* (GT). La fórmula para IoU es el área de intersección dividida por el área de unión de las dos cajas delimitadoras. Además como vemos en la Ecuación 3.1 podemos expresarlo en términos de VP, FN y FP, entendiendo las áreas como números de pixeles.

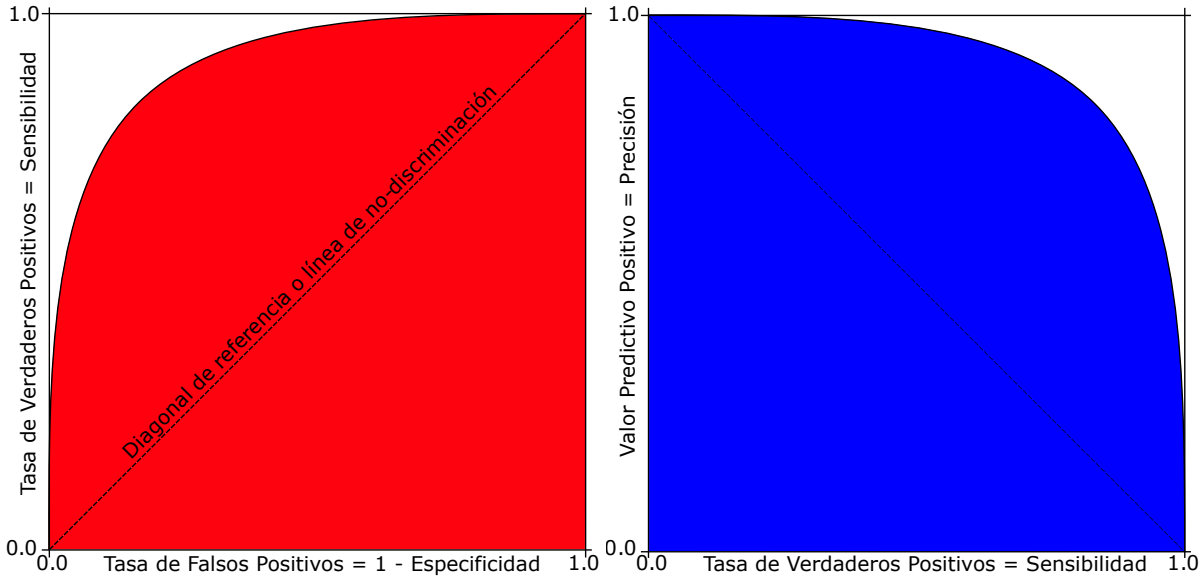
$$IoU = \frac{A \cap B}{A \cup B} = \frac{A \cap B}{GT + SM - A \cap B} = \frac{A \cap B}{A \cap B + (GT - A \cap B) + (SM - A \cap B)} = \frac{VP}{VP + FN + FP} \tag{3.1}$$

3.2. Curva ROC

En esta sección, analizaremos en profundidad la curva característica operativa del receptor (*Receiver Operating Characteristic*, ROC), una herramienta fundamental en la evaluación cuantificable de nuestro clasificador a la hora de predecir las clases con las que previamente ha sido entrenada.

La curva ROC es la representación gráfica que muestra la relación entre la tasa de verdaderos positivos (*True Positive Rate*, TPR) correspondiendo el eje Y y la tasa de falsos positivos (*False Positive Rate*, FPR) correspondiendo el eje X, ambos ejes incluyen valores entre 0 y 1 (0 % a 100 %) para diferentes umbrales de clasificación, una curva ROC genérica la podemos ver en la Figura 3.3a. Por lo tanto, con la curva ROC se ilustra cada uno de los posibles puntos de corte, correspondiendo el eje Y a la sensibilidad y el eje X a la métrica complementaria de la especificidad, de un test diagnóstico cuya escala de medición es continua. Una curva ROC ideal se encontraría en la esquina superior izquierda del gráfico, lo que indicaría una alta tasa de verdaderos positivos y una baja tasa de falsos positivos. Un modelo que produce una curva ROC que sigue la diagonal del gráfico, llamada diagonal de referencia o línea de no-discriminación, desde la esquina inferior izquierda (punto [0, 0]) hasta la esquina superior derecha (punto [1, 1]) se considera un modelo aleatorio [215], ya que está produciendo resultados que no son mejores que una moneda al azar.

Además, el análisis de la curva ROC también proporciona herramientas para seleccionar modelos óptimos independientemente del costo de la distribución de las clases, pudiendo así descartar los modelos subóptimos, esto es especialmente útil en problemas de clasificación donde los costos asociados con las decisiones de clasificación pueden variar significativamente. Por otro lado, la curva ROC es una herramienta muy útil para evaluar el rendimiento de un clasificador binario independientemente de la distribución de las clases en la población. Al representar la relación entre la tasa de verdaderos positivos y la tasa de falsos positivos, permite comparar diferentes modelos en diferentes niveles de discriminación. Esto es especialmente importante en problemas de diagnóstico médico, donde la prevalencia de una enfermedad puede variar significativamente entre diferentes poblaciones.



(a) Curva ROC.

(b) Curva PR.

En nuestro caso es interesante usar la curva ROC ya que tenemos un *dataset* muy balanceado pero en el caso de *datasets* no balanceados, es decir, cuando la clase positiva ocurre pocas veces, la curva PR sería la mejor opción [216]. En la curva PR se traza la relación entre la precisión y la coincidencia o sensibilidad a medida que cambia el umbral de decisión [209] como observamos en la Figura 3.3b.

Las curvas ROC también se denominan curvas de rendimiento diagnóstico, ya que, se utilizan las curvas ROC para elegir entre dos pruebas diagnósticas diferentes. Estas curvas son una medida global e independiente del punto de corte y se utilizan para medir el rendimiento diagnóstico [217]. El análisis en base a curvas ROC proporciona un método estadístico para determinar la exactitud diagnóstica de estos tests que utilizan escalas continuas. Debido a que se puede establecer un punto de corte en el que se alcanza la sensibilidad y especificidad más alta, para formular el diagnóstico de una determinada enfermedad [215] muchos tests diagnósticos reportan sus resultados de esta forma. Además también es útil para evaluar la capacidad discriminativa del test diagnóstico, es decir, su capacidad de diferenciar correctamente a los pacientes enfermos de los pacientes sanos y comparar la capacidad discriminativa de dos o más tests diagnósticos que expresan sus resultados como escalas continuas, lo que permite seleccionar el test que mejor se adapte a las necesidades del problema en cuestión.

La determinación del punto de corte óptimo en una escala continua es clave para maximizar la precisión del diagnóstico. Una forma de identificar este punto es a través del índice de Youden [218], que se calcula según la fórmula $(sensibilidad + especificidad - 1)$, es decir, como la diferencia entre la tasa de verdaderos positivos y la tasa de falsos negativos. Gráficamente, el punto de corte con el mayor índice de Youden corresponde al punto de la curva ROC más cercano al ángulo superior-izquierdo (0,1), es decir, con la sensibilidad y especificidad más altas. Sin embargo, es importante tener en cuenta que el punto de corte con el mayor índice de Youden no necesariamente determina la sensibilidad o especificidad máximas que podría alcanzar el test, ya que estas pueden ser logradas en puntos de corte diferentes.

En algunos casos, se pueden requerir características distintas dependiendo del propósito del test, como una

sensibilidad alta para encontrar enfermedades (e.g., en los cribados) o una especificidad alta para asegurar un diagnóstico (e.g., en la confirmación de enfermedades). En estos casos, no siempre es recomendable utilizar el punto de corte identificado por el índice de Youden. Es importante conocer los valores de sensibilidad y especificidad determinados por diferentes puntos de corte y escoger el que mejor se ajuste al objetivo específico.

La curva ROC se utiliza a menudo junto con la medida de desempeño conocida como área bajo la curva ROC (*Area Under the Curve*, AUC) es una medida única e independiente de la prevalencia de la enfermedad en estudio que permite evaluar la capacidad discriminativa de un test diagnóstico, refiriéndose a su habilidad para distinguir correctamente entre pacientes sanos y enfermos a lo largo de todo el rango de puntos de corte posibles [215], este área en la Figura 3.3a es de color rojo. Un valor AUC de 1 indica un rendimiento perfecto, mientras que un valor de 0.5 indica un rendimiento al azar. Es recomendable elegir el test que presente un mayor área bajo la curva ROC, ya que, indica la probabilidad de que el diagnóstico realizado a un enfermo sea más correcto que el de una persona sana escogida al azar.

La elección se realiza comparando el área bajo la curva (AUC) de ambas pruebas. Como guía para interpretar las curvas ROC, se han establecido diferentes intervalos de AUC como son test malo ($0,5 \leq AUC < 0,6$), test regular ($0,6 \leq AUC < 0,75$), test bueno ($0,75 \leq AUC < 0,9$), test muy bueno ($0,9 \leq AUC < 0,97$) y test excelente ($0,97 \leq AUC < 1$) [217].

3.3. Validación cruzada

En secciones anteriores, aprendimos sobre los diferentes métodos de evaluar la capacidad discriminativa de nuestro clasificador. Sin embargo, hay un problema con la evaluación del rendimiento un modelo utilizando solo los datos de entrenamiento: es posible que el modelo tenga un alto rendimiento en los datos de entrenamiento, pero un bajo rendimiento en los datos no vistos anteriormente. Para superar este problema, en esta sección, introduciremos el concepto de validación cruzada, sus ventajas y desventajas, y algunos de los tipos de validación cruzada que hay.

Comenzamos definiendo la validación simple que consiste en dividir el conjunto de datos en dos grupos: uno para entrenar el modelo y otro para evaluarlo. Esta es una técnica sencilla y rápida, pero puede resultar menos precisa que la validación cruzada, ya que no se aprovecha al máximo el conjunto de datos disponible. La validación cruzada es una técnica importante en el área del aprendizaje automático que permite evaluar el rendimiento de un modelo de forma más precisa. Se realiza dividiendo el conjunto de datos en varios grupos (llamados *folds*), y se entrena el modelo varias veces, cada vez usando un grupo diferente como conjunto de datos de prueba y el resto como conjunto de datos de entrenamiento y validación [219].

En este punto cabe destacar que el entrenamiento, prueba y validación son tres etapas importantes en el desarrollo de un modelo de aprendizaje automático. Durante la etapa de entrenamiento, se utilizan los datos de entrenamiento para "entrenar" al modelo. El modelo aprende a realizar predicciones a partir de estos datos. Una vez que el modelo ha sido entrenado, se utilizan los datos de prueba para evaluar su rendimiento. Los datos de prueba son independientes de los datos de entrenamiento y permiten evaluar cómo el modelo se desempeñará con nuevos datos. La validación es una etapa intermedia entre el entrenamiento y la prueba. Se utiliza para evaluar el rendimiento del modelo con diferentes configuraciones o hiperparámetros. Por ejemplo, se pueden probar diferentes valores de un parámetro para ver cómo afecta al rendimiento del modelo. La motivación de esta división está en validar el algoritmo sobre un conjunto de datos diferente del empleado para estimar sus parámetros.

La validación cruzada tiene desventajas como requerir más tiempo y recursos computacionales que otras técnicas de evaluación, ya que se entrena el modelo varias veces y puede resultar menos precisa que otras técnicas en conjuntos de datos muy pequeños. Aunque es útil porque permite evaluar el rendimiento del modelo en un conjunto de datos que no se ha utilizado para entrenarlo, lo que es importante para evitar el sobreajuste del modelo, ya que se tiene en cuenta la variabilidad en la división de los datos, proporcionando así una medida más precisa de la precisión del modelo, es más eficiente que otras técnicas de evaluación, ya que se aprovecha al máximo el conjunto de datos disponible y permite obtener una medida más precisa del rendimiento del modelo, ya que se entrena varias veces y se promedian los resultados obtenidos.

De esta manera, se puede evaluar el rendimiento del modelo en diferentes conjuntos de datos y obtener una medida más precisa de su desempeño general, ya que, el rendimiento del modelo se calcula como la media de las métricas obtenidas en cada iteración. Hay dos formas diferentes de calcular el rendimiento de un modelo de

aprendizaje automático en una tarea de clasificación que son el promediado macro y micro.

El promediado macro se calcula promediando las métricas de rendimiento en un nivel de grupo, es decir, se calcula la métrica para cada grupo y luego se promedian como vemos en la Ecuación 3.2, siendo B es la métrica que se desea calcular, q es el número total de clases y w_λ es el peso asignado al grupo λ. Este enfoque generalmente asigna el mismo peso a todas los grupos de forma igualitaria con $w_\lambda = 1$, aunque si se desea reflejar que algún grupo es más importantes que otro puede aplicarse en el cálculo de la métrica modificando el peso asignado a la grupo λ.

$$B_{macro} = \frac{1}{q}\sum_{\lambda=1}^{q} w_\lambda \cdot B(TP_\lambda, FP_\lambda, TN_\lambda, FN_\lambda) \tag{3.2}$$

El promediado micro, por otro lado, promedia las métricas de rendimiento en un nivel global, se calcula sumando el número total de verdaderos positivos (VP), verdaderos negativos (VN), falsos positivos (FP) y falsos negativos (FN) y dividiéndolo por el número total de ejemplos en el conjunto de datos.

$$B_{micro} = B\left(\sum_{\lambda=1}^{q} TP_\lambda, \sum_{\lambda=1}^{q} FP_\lambda, \sum_{\lambda=1}^{q} TN_\lambda, \sum_{\lambda=1}^{q} FN_\lambda\right) \tag{3.3}$$

En general, el promediado macro es más adecuado cuando se desea tener una visión equitativa del rendimiento del modelo en cada grupo, mientras que el promediado micro es más adecuado cuando se desea tener una visión global del rendimiento del modelo. Algunos creen que el rendimiento que se obtiene con promediado micro es un tanto engañoso porque los temas más frecuentes son más pesados en el promedio y otros creen que los temas deberían contar de manera proporcional a su frecuencia y, por lo tanto, se inclinan hacia el promediado micro [220]. En nuestro caso usamos promediado macro con $w_\lambda = 1$.

Existen diferentes tipos de validación cruzada, según el número de grupos en los que se dividen los datos y la forma en que se realiza la división. Algunos de los tipos más comunes de validación cruzada son [221]:

- Validación cruzada K-fold: en este tipo de validación cruzada, se dividen los datos en K grupos de igual tamaño (si es posible), y el modelo se entrena K veces cada vez usando un grupos distintos. Se usan K-1 grupos como datos de entrenamiento y el grupo restante como datos de prueba. En la Figura 3.4 vemos un ejemplo con 5 grupos.

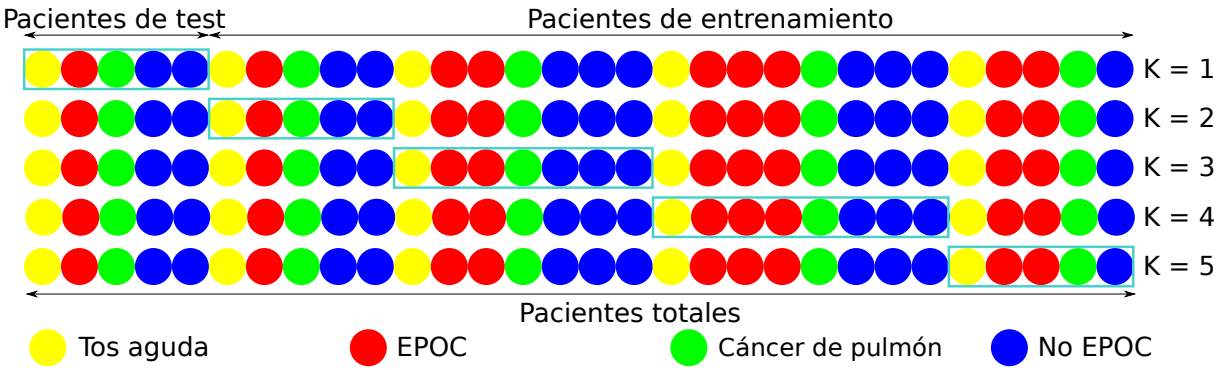


Figura 3.4: Ejemplo de *5-fold cross validation*.

- Validación cruzada dejando uno fuera (LOO): este tipo de validación cruzada es equivalente a la estrategia K-fold siendo K el número de datos en el conjunto. Por lo que cada vez se utiliza un solo dato como conjunto de datos de prueba y el resto como conjunto de datos de entrenamiento [222]. LOO se utiliza cuando se tiene un conjunto de datos pequeño y se desea una evaluación más precisa del rendimiento del modelo.

- Validación cruzada estratificada: es similar a la validación cruzada K-fold, pero se asegura de que cada grupo contenga una proporción similar de datos de cada clase, lo que puede ser importante en problemas de clasificación con desequilibrio de clases. Sin embargo, esta técnica puede requerir más tiempo y recursos computacionales que la validación cruzada K-fold.
- Validación cruzada aleatoria: en este tipo de validación cruzada, se dividen los datos de manera aleatoria en varios grupos, y se entrena el modelo varias veces, cada vez usando un grupo distinto como conjunto de datos de prueba y el resto como conjunto de datos de entrenamiento. En la validación cruzada aleatoria, los datos se dividen de manera aleatoria en los grupos o "folds". La validación cruzada aleatoria es más simple y rápida de implementar, ya que sólo implica dividir los datos en grupos aleatorios. Sin embargo, el rendimiento del modelo puede variar dependiendo de cómo se dividan los datos, por lo que puede ser menos preciso que la validación cruzada K-fold.

En nuestro caso, usamos la validación cruzada K-fold dividiendo los datos en 3, 5 y 10 grupos a nivel de paciente sin solapamiento, para asegurarnos de que los grupos de entrenamiento y prueba no tienen los mismos pacientes. Esto es una buena práctica para evaluar el rendimiento de un modelo de aprendizaje automático. Al utilizar diferentes valores de k, se pueden obtener diferentes estimaciones del error y comparar cómo el rendimiento del modelo varía con diferentes configuraciones. Además, utilizar diferentes valores de k permite obtener una visión más completa del rendimiento del modelo en diferentes configuraciones y ayuda a asegurar que el modelo generalice bien a datos desconocidos.

Capítulo 4

REDES NEURONALES

En este capítulo, exploraremos con más detalle el funcionamiento de las redes neuronales y cómo se entrenan. Vamos a profundizar en el concepto de *Deep Learning*, el cual es un subcampo del *Machine Learning* que se enfoca en el entrenamiento de redes neuronales artificiales para imitar el comportamiento del cerebro humano, hablaremos de cuáles son las principales arquitecturas de redes neuronales utilizadas en el *Deep Learning* y algunas de las aplicaciones prácticas de esta técnica.

Como ya se explicó en la sección 2.1, la Inteligencia Artificial es un concepto más amplio que abarca a la rama del *Machine Learning*, y dentro del *Machine Learning* encontramos al *Deep Learning* como un conjunto de algoritmos avanzados que han demostrado ser muy efectivos en tareas de reconocimiento de patrones, clasificación y procesamiento del lenguaje natural.

4.1. Funcionamiento básico de una neurona

Las redes neuronales son sistemas de procesamiento de datos que están inspirados en el cerebro humano y consisten en una red de neuronas artificiales. Estas neuronas artificiales son unidades básicas de procesamiento que están diseñadas para imitar el comportamiento de las neuronas biológicas del cerebro, sin tener en cuenta el entorno biológico húmedo, formado por iones, biomoléculas y neurotransmisores.

Tanto las neuronas biológicas como las neuronas artificiales son capaces de procesar información y realizar cálculos, y pueden ser activadas o desactivadas según la entrada recibida. Sin embargo, a pesar de las similitudes básicas, es importante tener en cuenta que las neuronas artificiales son una simplificación de las neuronas biológicas, y que la complejidad de la actividad cerebral aún está lejos de ser completamente comprendida.

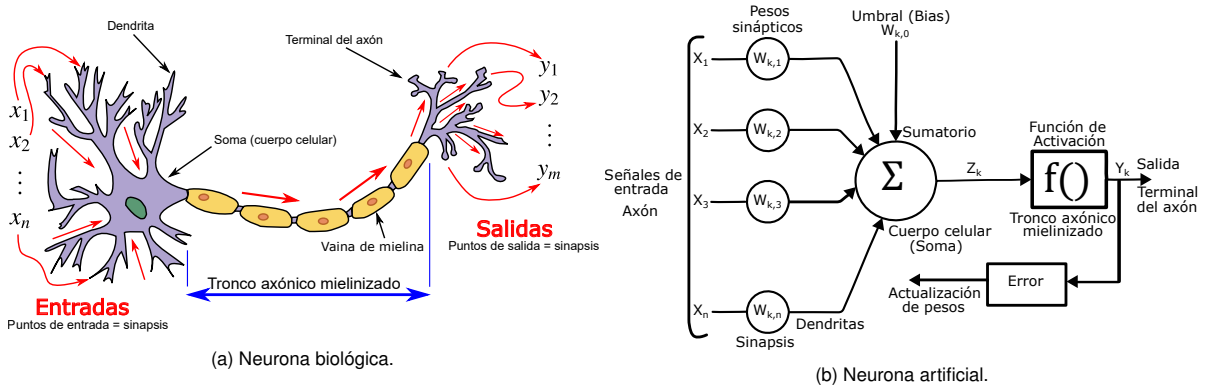
Estas neuronas artificiales están conectadas entre sí con neuronas de la siguiente capa por lo que cada neurona artificial recibe una serie de entradas, estas entradas recibidas por el "terminal del axón" de la anterior neurona se multiplican por un peso que imita la "sinapsis" que realiza la neurona al recibir la señal por la "dendrita", estos pesos pueden ser ajustados durante el entrenamiento de la red neuronal, ponderando de esta forma la importancia de cada entrada para el cálculo de la neurona. Posteriormente en el "cuerpo" de nuestra neurona suma todas las señales recibidas y añade un umbral, este umbral o *bias* nos permite ajustar la sensibilidad de la función de activación que veremos a continuación. Finalmente en el "tronco axónico" se aplica una función de activación, la cual se encarga de decidir si se activa o no la neurona y como se activa, para producir una salida, este proceso lo podemos ver en la Figura 4.1.

Matemáticamente y a modo de resumen de lo expuesto anteriormente, en la Ecuación 4.1 podemos expresar la salida de una neurona como:

$$\hat{y} = f(z) = f(\sum_{i=1}^{n} x_i w_i + w_0) = f(XW^T) \tag{4.1}$$

Donde $X = [x_n, \ldots, x_i, \ldots, x_1, 1]$ es el vector de las n entradas o estímulos (x_i) que recibe la neurona y $W = [w_n, \ldots, w_i, \ldots, w_1, w_0]$ es el vector de los n pesos (w_i) asociados a cada entrada que recibe la neurona,

equivalente a las conexiones sinápticas de una neurona real, siendo w_0 el umbral o *bias* aplicado por la neurona y f la función de activación.

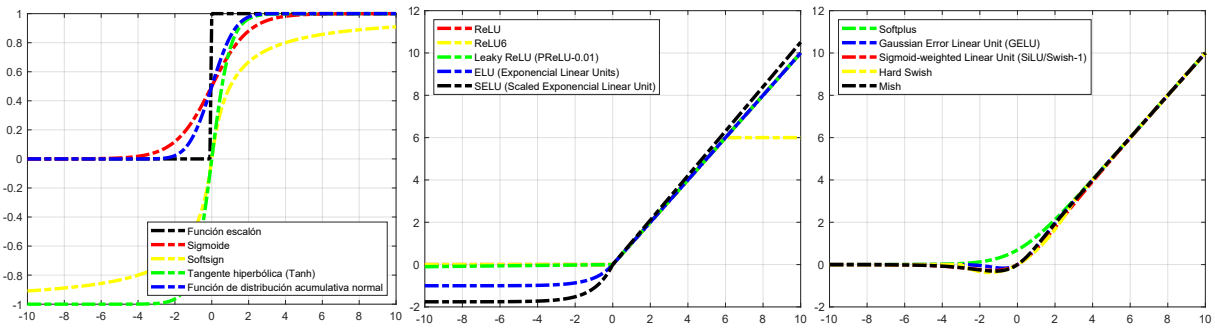


(a) Neurona biológica.

(b) Neurona artificial.

Figura 4.1: Similitudes entre la neurona biológica y la neurona artificial.

La función de activación es esencial para introducir no linealidad en la red y permitir la captura de relaciones no lineales entre las variables de entrada y salida. Aunque el uso de una función de activación introduce complejidad adicional en la red, es esencial para permitir el aprendizaje de patrones no lineales y mejorar el rendimiento de la red. Por lo que en el caso de no utilizar una función de activación en nuestra red neuronal, estaríamos limitando la capacidad de la red a transformaciones lineales de las entradas. En este caso, la red sería esencialmente un modelo de regresión lineal y no sería capaz de capturar patrones complejos en los datos. Por el mismo motivo ya no se usan las funciones de activación lineales, ya que, la salida de las funciones no estará confinada entre ningún rango y la suma de diferentes funciones lineales sigue siendo una función lineal acotando así la activación de la neurona [223].



(a) Funciones de activación acotadas en rango.

(b) Familia ReLU.

(c) Funciones de activación avanzadas.

Figura 4.2: Funciones de activación mas comunes.

Una función de activación en una red neuronal debe tener ciertas características deseables para asegurar un entrenamiento efectivo y un buen rendimiento del modelo como las que vemos en la Figura 4.2. En primer lugar, la función no debe saturarse en el rango de valores comunes de entrada para evitar dificultades en el aprendizaje. Además, se busca una baja fuga de gradiente, evitando funciones con gradientes muy pequeños, para que el gradiente de cada capa afecte correctamente a la capa anterior y permita una propagación hacia atrás efectiva, lo opuesto sería el problema del gradiente explosivo debido a gradientes muy grandes. También es importante que la función de activación esté centrada en cero para evitar que los gradientes se desplacen en una dirección particular. La función debe ser económicamente eficiente, lo que significa que el gasto computacional no debe ser excesivo. Además, es necesario que la función sea diferenciable para permitir el entrenamiento mediante el algoritmo de descenso de gradiente y que sea monótona para evitar puntos de inflexión y dificultades en la convergencia del

modelo. En general, la elección de la función de activación dependerá del problema específico y de la arquitectura de la red neuronal, y es importante evaluar diferentes opciones y comparar su rendimiento [224].

Comenzamos analizando las funciones acotadas en rango que vemos en la Figura 4.2a. En primer lugar, vemos la función *sigmoidea* o logística que tiene rango $(0, 1)$ que limita su salida, es monótona creciente lo que ayuda a la convergencia, continuamente diferenciable (C^∞), asimemtrica y no pasa por el origen, mientras que su primera derivada tiene rango $(0, 0.25)$, es continua no monótona y solventa los problemas de neuronas muertas y gradiente explosivo pero tiene problemas de saturación y fuga de gradiente ya que la derivada de la función sigmoidea está cerca de 0 en posiciones alejadas del punto central, por lo que cuando la red es muy profunda, cada vez más gradientes de retropropagación caen en el área de saturación para que el módulo de gradiente se vuelva más pequeño. Generalmente, si la red sigmoidea tiene mas de cinco capas, el gradiente se degrada a 0, lo que es difícil de entrenar. Además es costosa computacionalmente al tener operaciones exponenciales.

$$f(z) = \sigma(z) = \frac{1}{1+e^{-z}}, f'(z) = \frac{\partial \sigma(z)}{\partial z} = \sigma(z) \cdot (1 - \sigma(z)) = \frac{e^{-z}}{(1+e^{-z})^2} \tag{4.2}$$

La función tangente hiperbólica (*Tanh*) tiene rango $(-1, 1)$, es monótona creciente, continuamente diferenciable (C^∞), antisimemtrica y pasa por el origen, lo cual, se trata de una mejora respecto a la anterior ya que el rango está centrado en el cero y eso permite que, durante el aprendizaje, se pueda corregir los pesos en direcciones distintas (aumentando o disminuyendo su valor individual). Mientras que su primera derivada tiene rango $(0, 1]$, es continua no monótona y solventa los problemas de neuronas muertas y gradiente explosivo pero tiene problemas de saturación y aunque la pendiente en esta función es mucho más acentuada y con ello el gradiente producido sigue presentando fuga de gradiente.

$$f(z) = \tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1} = \frac{1 - e^{-2z}}{1 + e^{-2z}} = \frac{2}{1 + e^{-2z}} - 1 \tag{4.3}$$

$$f'(z) = \frac{\partial \tanh(z)}{\partial z} = \frac{\partial}{\partial z}\frac{\sinh(z)}{\cosh(z)} = \frac{\cosh^2(z) - \sinh^2(z)}{\cosh^2(z)} = 1 - \tanh^2(z) = \frac{1}{\cosh^2(z)} \tag{4.4}$$

La función *softsign* [225] se satura más lentamente que la función *tanh*, tiene rango $(-1, 1)$, es monótona creciente, continuamente diferenciable (C^1), antisimemtrica y pasa por el origen, mientras que su primera derivada tiene rango $(0, 1]$, es continua no monótona y solventa los problemas de neuronas muertas y gradiente explosivo pero tiene problemas de saturación y fuga de gradiente por lo que vemos que este problema es inevitable en las funciones de rango acotado. La función escalón y la función de distribución acumulativa normal no las analizamos individualmente ya que actualmente se usan poco pero tienen influencia en funciones de activación como la GELU o las de la familia ReLU.

$$f(z) = \frac{z}{1+|z|}, f'(z) = \frac{\partial f(z)}{\partial z} = \frac{1}{(1+|z|)^2} \tag{4.5}$$

La función unidad lineal rectificada (ReLU) o función rampa [226] tiene rango $[0, \infty)$, es monótona creciente, no es continuamente diferenciable (C^0), es asimemtrica y pasa por el origen, además vemos que comparada con las funciones sigmoide y tanh que implican la exponenciación, la ReLU se puede implementar más fácilmente reduciendo así la carga de trabajo de computación, pero no hay límite superior, por lo que el entrenamiento es relativamente fácil de divergir. Mientras que su primera derivada, que es la función escalón, tiene rango $([0, 1]$, no es continua ni monótona y solventa los problemas de fuga de gradiente, gradiente explosivo y saturación, aunque tiene el importante problema de neuronas muertas.

$$f(z) = \text{máx}(0, z) = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases}, f'(z) = \frac{\partial f(z)}{\partial z} = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases} \tag{4.6}$$

El problema del gradiente de fuga lo mitiga eficazmente, ya que, el gradiente ReLU es insaturado a diferencia de las funciones acotadas en rango que cuando están cerca del área de saturación (lejos del centro de la función), la transformación es demasiado lenta y la derivada está cerca de 0. Sin embargo tiene dos problemas importantes, comenzamos con el problema de neuronas muertas que ocurre cuando las entradas se acercan a cero o son negativas,

ya que, el gradiente se vuelve cero y la red no puede realizar una propagación hacia atrás, por lo que, las neuronas que entran en esta situación no puede aprender y no juegan ningún papel, esto hace que con el tiempo harán que parte de la red no haga nada. Continuamos con el problemas de la continuidad que hace que no sea diferenciable en $x = 0$ y una derivada se define forzosamente en este punto, además, la superficie definida en el punto cero no es lo suficientemente suave en algunos problemas de regresión.

$$f(z) = \text{mín}(\text{máx}(0, z), 6) = \begin{cases} 6 & z \geq 6 \\ z & 0 \leq z < 6 \\ 0 & z < 0 \end{cases}, f'(z) = \frac{\partial f(z)}{\partial z} = \begin{cases} 0 & z \geq 6 \\ 1 & 0 \leq z < 6 \\ 0 & z < 0 \end{cases} \tag{4.7}$$

Dentro de la familia ReLU hay distintas funciones de activación que buscan mitigar los problemas de los que hemos hablado, por ejemplo, ReLU6 [227] pone un limite superior para que el entrenamiento converja mas fácilmente, según los autores, empuja al modelo a aprender representaciones más dispersas en las capas del principio. Pero la mayoría intentan mitigar el problema de las neuronas muertas modificando el lado negativo como es el caso de Leaky ReLU [228] donde α es un hiperparámetro generalmente establecido en 0.01 que indica el valor de la pendiente en el lado negativo que asegura que la neurona nunca muera, resolviendo el problema de ReLU moribundo. Parametric ReLU [229] es una variación de Leaky ReLU, donde α en lugar de ser un hiperparámetro debe ser aprendido durante el entrenamiento [230].

$$f(z) = \text{máx}(\alpha z, z) = \begin{cases} z & z \geq 0 \\ \alpha z & z < 0 \end{cases}, f'(z) = \frac{\partial f(z)}{\partial z} = \begin{cases} 1 & z \geq 0 \\ \alpha & z < 0 \end{cases} \tag{4.8}$$

Si bien las LReLU y las PReLU tienen valores negativos para mitigar el problema de las neuronas muertas, no garantizan un estado de desactivación resistente al ruido. Las ELU [231] se saturan a un valor negativo con entradas más pequeñas y, por lo tanto, disminuyen la variación y la información propagadas hacia adelante. La función de activación SELU [232] que vemos en la Ecuación 4.9 es una variación de ELU que induce propiedades de auto-normalización donde se escala por un valor λ, por lo que la función de activación ELU es el caso en el que $\lambda = 1$.

$$f(z) = \text{máx}(\lambda\alpha \cdot (e^z - 1), \lambda z) = \lambda \begin{cases} z & z \geq 0 \\ \alpha \cdot (e^z - 1) & z < 0 \end{cases}, f'(z) = \frac{\partial f(z)}{\partial z} = \lambda \begin{cases} 1 & z \geq 0 \\ \alpha e^z & z < 0 \end{cases} \tag{4.9}$$

También se he intentado mitigar el problema de la continuidad que presenta ReLU en $x = 0$ con funciones de activación que tienen una geometría similar a las ReLU, pero que son derivables en todos los puntos. Como es el caso de la función de activación *softplus* [233] que tiene rango $(0, \infty)$, es monótona creciente, continuamente diferenciable (C^∞), asimemtrica y no pasa por el origen, mientras que su primera derivada tiene rango $(0, 1)$, es monótona creciente, continua y solventa los problemas de neuronas muertas y gradiente explosivo pero tiene problemas de saturación y fuga de gradiente. Vemos en la Ecuación 4.10 que esta función tiene un cálculo más complejo pero, tiene una derivada continua y define una superficie curva lisa [234].

$$f(z) = \ln(1 + e^z), f'(z) = \frac{\partial f(z)}{\partial z} = \frac{1}{1 + e^{-z}} = \sigma(z) \tag{4.10}$$

En las siguientes funciones de activación además de ser continua y derivable, buscaremos que pase por cero, aunque esto va a hacer que sea constitucionalmente mas costoso. Comenzamos hablando de la función Swish [235] que multiplica x por la distribución logística $Logistica(\mu = 0, s = 1)$ como vemos en la Ecuación 4.11, donde se puede apreciar que se trata de una función no-monótona, una característica contrario a todas las anteriores. El tramo no monótono le otorga una mayor expresividad a la función mejorando el flujo del gradiente, lo cual es clave durante el arranque con la pre-inicialización aleatoria de pesos, ya que muchos hacen caer los inputs en la zona negativa y éstos se verían obviados en un ReLU por ser anulados directamente; de esta forma, en cambio, existe gradiente y se puede aún aprender y, quizás, salir de la zona negativa.

$$f(z) = z \cdot \sigma(\beta z) = \frac{z}{1 + e^{-\beta z}} \tag{4.11}$$

En esta función el hiperparametro β es aprendido durante el entrenamiento, pero en el caso de utilizar $\beta = 1$, estaríamos utilizando la función SiLU o Swish-1 [236]. También existe la función Hard Swish [237] que vemos en la Ecuación 4.12 donde busca simplificar la función Swish para que sea menos costosa constitucionalmente utilizando ReLU6 para realizar una aproximación por partes.

$$f(z) = z \cdot \frac{ReLU6(z+3)}{6} = \begin{cases} z & z \geq 3 \\ \frac{z \cdot (z+3)}{6} & -3 \leq z < 3 \\ 0 & z < -3 \end{cases}, f'(z) = \frac{\partial f(z)}{\partial z} = \begin{cases} 1 & z \geq 3 \\ \frac{2z+3}{6} & 3 \leq z < -3 \\ 0 & z < -3 \end{cases} \tag{4.12}$$

ReLU es un mapa cero determinista o un mapa de identidad, según el valor de entrada. Pero este determinismo de la ReLU es limitante. Algunas estrategias de regularización de redes neuronales, como dropout [238] y zoneout [239], actúan independientemente del valor de una neurona. Dropout puede actuar como un mapa cero estocástico estableciendo aleatoriamente los valores de las neuronas en cero, y zoneout puede actuar como un mapa de identidad estocástico conservando aleatoriamente los valores de las neuronas calculados previamente. La estocasticidad de abandono y zona libre puede permitir un pseudoconjunto de redes con ancho estocástico o profundidad estocástica, respectivamente, y esto conduce a mejoras de rendimiento notables. GELU [240] une estos conceptos acortando la brecha entre las no linealidades y los regularizadores estocásticos al encapsular el regularizador estocástico que depende de los valores de entrada en una función de activación determinista. Podemos ver que GELU multiplica x por la distribución normal $\mathcal{N}(\mu = 0, \sigma^2 = 1)$ como vemos en la Ecuación 4.13 y una de las simplificaciones para hacer GELU computacionalmente mas eficiente es utilizar Swish-1.702.

$$f(z) = z \cdot P(Z \leq z) = z \cdot \Phi(z) = z \cdot \frac{1}{2}\left[1 + erf\left(\frac{z}{\sqrt{2}}\right)\right] \approx \frac{1}{2} z(1 + \tanh\left[\sqrt{\frac{2}{\pi}}(z + 0.044715z^3)\right]) \approx z \cdot \sigma(1.702z) \tag{4.13}$$

Por último hablaremos de Mish [241] que es una función de activación no monótona autoregulada inspirada en la propiedad de activación automática de Swish que como vemos en la Ecuación 4.14 combina las funciones de activación softplus y tanh.

$$f(z) = z \cdot \tanh(\ln(1 + e^z)) = z \cdot \tanh(softplus(z)) \tag{4.14}$$

Además, no nos podemos olvidar de la función *softmax* o función exponencial normalizada, que se utiliza a menudo en la capa de salida para clasificar datos en varias categorías. La función softmax cuya expresión vemos en la Ecuación 4.15 toma un conjunto de valores de entrada y devuelve un conjunto de valores de salida que suman 1. Cada valor de salida representa la probabilidad de que la entrada pertenezca a una determinada categoría. Por ejemplo, si se tiene una red neuronal que se utiliza para clasificar imágenes en diferentes categorías de animales, la función softmax en la capa de salida podría devolver valores de salida que indiquen la probabilidad de que la imagen sea un gato, un perro, un pájaro, etc. Esto permite que la red neuronal produzca una salida probabilística en lugar de una salida binaria o discreta. La función softmax se calcula mediante la aplicación de la función exponencial a cada valor de entrada y luego normalizando los resultados para que sumen 1.

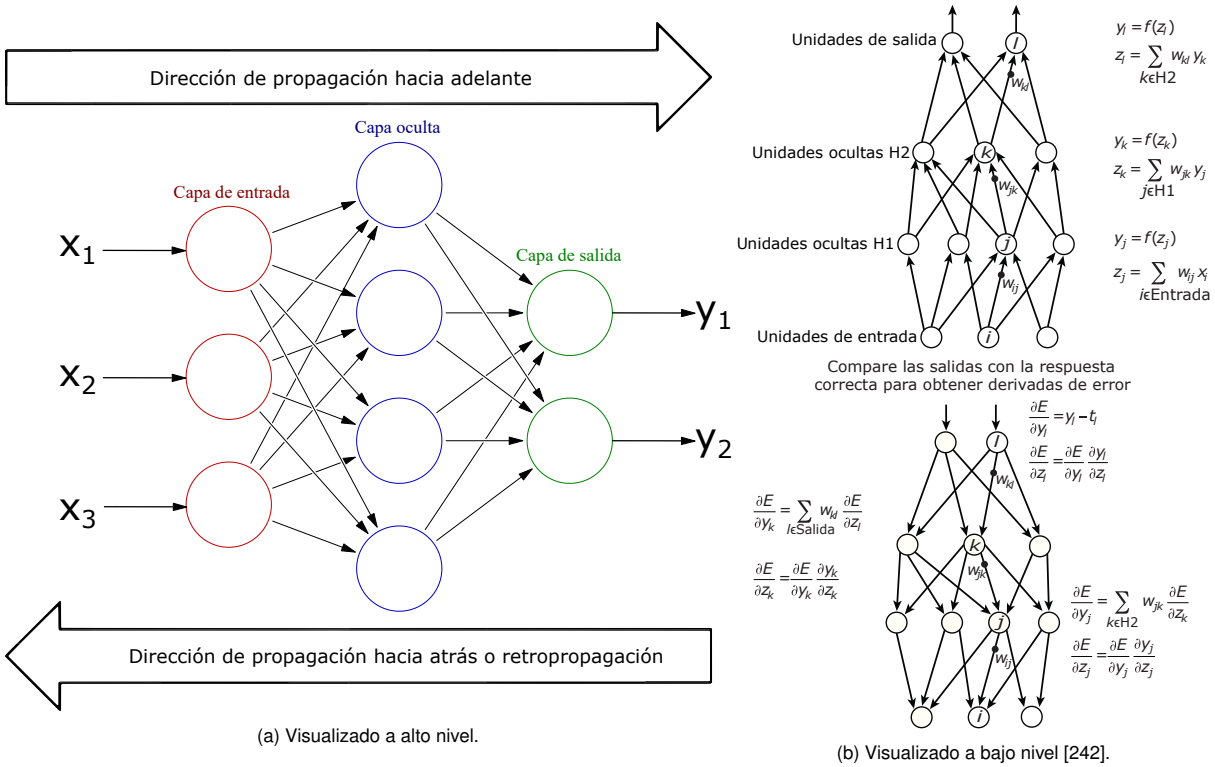
$$f(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad \forall i \in [1, N] \tag{4.15}$$

4.2. Entrenamiento de una red neuronal

El proceso de entrenamiento de una red neuronal es el proceso de ajustar los pesos en las conexiones entre las neuronas en la red para que pueda aprender a realizar una tarea específica y se lleva a cabo en varios pasos. En primer lugar, se realiza la inicialización de los pesos y sesgos de la red de forma aleatoria. A continuación, se introduce una entrada en la red y se propaga hacia adelante a través de las capas de entrada, ocultas y de salida para producir su salida.

El algoritmo de propagación hacia adelante es un proceso utilizado en redes neuronales artificiales para calcular las salidas de una red dada una entrada específica. La propagación hacia adelante implica el paso de datos de entrada a través de la red neuronal, a través de cada una de sus capas, hasta llegar a la capa de salida. En cada capa, los datos son procesados por las neuronas utilizando una función de activación y unos pesos específicos asociados a cada conexión entre neuronas. La salida de cada neurona en una capa se utiliza como entrada para la siguiente capa, hasta que se llega a la capa de salida, donde se obtienen las salidas finales de la red. El algoritmo de propagación hacia adelante es esencial para el entrenamiento de redes neuronales, ya que permite calcular las salidas de la red para un conjunto dado de datos de entrada.

Luego, se calcula la función de coste comparando la salida de la capa de salida con la salida objetivo. El siguiente paso es la retropropagación, que se utiliza para propagar los errores hacia atrás a través de la red utilizando el algoritmo de retropropagación de errores. Los errores en las capas ocultas se calculan propagando hacia atrás los errores de la capa de salida utilizando la matriz de pesos de la capa correspondiente.



(a) Visualizado a alto nivel.

(b) Visualizado a bajo nivel [242].

Figura 4.3: Entrenamiento de una red neuronal.

Posteriormente, se actualizan los pesos utilizando un algoritmo de optimización como el algoritmo de descenso de gradiente, el cual ajusta los pesos en las conexiones entre las neuronas en la dirección en la que la función de coste disminuye más rápidamente de manera que se minimice el error entre las salidas de la red y las salidas esperadas. Este proceso se repite para cada entrada en el conjunto de entrenamiento.

Finalmente, la red se evalúa utilizando un conjunto de validación. Si el rendimiento de la red es satisfactorio, se puede utilizar para hacer predicciones sobre nuevos datos. Si el rendimiento no es satisfactorio, se ajustan los parámetros y se repiten los pasos anteriores hasta que se alcance un rendimiento satisfactorio. Podemos ver que el proceso de entrenamiento de una red neuronal es un proceso iterativo que busca ajustar los pesos y parámetros de la red para minimizar la función de coste y mejorar la precisión de las predicciones.

4.2.1. Función de coste y algoritmo de *backpropagation*

Como hemos visto después de la propagación hacia adelante es necesario evaluar la salida predicha por nuestra red ($\hat{y}$) en comparación con el valor real de los datos (y) que es la salida esperada. La evaluación entre $\hat{y}$ e y se realiza a través de una función de coste. Las dos funciones de coste mas importantes son el error cuadrático medio (MSE) y la entropía cruzada categórica (*categorical cross-entropy*). La función de error de entropía cruzada se define como vemos en la Ecuación 4.16 es una función de pérdida ampliamente utilizada para problemas de clasificación y representa la distancia entre dos distribuciones de probabilidad, en este caso la distribución real de etiquetas y la distribución predicha por el modelo.

$$C(y,\hat{y}) = -\sum_{i=1}^{n} y \ln \hat{y}, \frac{\partial C(y,\hat{y})}{\partial w_j} = \frac{\partial C}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j} = \frac{\partial C}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_j} = -\sum_{i=1}^{n}\frac{y}{\hat{y}}\frac{\partial f(z_i)}{\partial z_i}x_{ij} \tag{4.16}$$

Donde n son las neuronas en la capa de salida. Por lo general, la distribución "verdadera" (la que el algoritmo de ML está tratando de igualar) se expresa en términos de una distribución *one-hot*. Por ejemplo, si estás trabajando en un problema de clasificación con tres clases y tu modelo predice una distribución de probabilidad [0.1, 0.2, 0.7] para una instancia dada, entonces la distribución "verdadera" podría ser [0, 0, 1], lo que significa que la instancia pertenece a la tercera clase.

One-hot encoding es una técnica de codificación que se utiliza para representar variables categóricas como vectores binarios. En esta técnica, cada categoría se convierte en un vector binario con todas las entradas iguales a cero excepto la entrada correspondiente a la categoría, que se marca con un 1. Por ejemplo, si tienes una variable categórica "color" con tres categorías: rojo, verde y azul. Entonces, el vector *one-hot* para la categoría "verde" sería [0, 1, 0]. La codificación *one-hot* se utiliza a menudo en problemas de clasificación, ya que permite a los modelos de aprendizaje automático tratar cada categoría de forma independiente y evita la asignación de un orden a las categorías, lo que podría resultar en una interpretación errónea de los datos.

Mientras que la función de error cuadrático medio, coste cuadrático o coste L_2 se define como vemos en la Ecuación 4.17 y generalmente se utiliza para resolver el problema de regresión.

$$C(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2, \frac{\partial C(y,\hat{y})}{\partial w_j} = \frac{\partial C}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w_j} = \frac{\partial C}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_j} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i-y_i)\frac{\partial f(z_i)}{\partial z_i}x_{ij} \tag{4.17}$$

De forma general podemos ver aplicando la regla de la cadena hacemos la derivada de la función de coste respecto a la salida de la red neuronal ($\frac{\partial C}{\partial \hat{y}_i}$), la derivada de la función de activación ($\frac{\partial f(z_i)}{\partial z_i}$) y la derivada del sumatorio que entra a la función de activación respecto a los pesos ($\frac{\partial z_i}{\partial w_{ij}} = x_{ij}$) que en el caso del umbral o *bias* de la neurona será $x_{i0} = 1$. Esto será importante para el algoritmo de descenso de gradiente que veremos mas tarde u otros optimizadores, ya que se modifican los pesos gracias a estas derivadas como vemos a continuación.

$$\Delta w_{ij} = -\alpha\frac{\partial C}{\partial w_{ij}} = -\alpha\frac{\partial C}{\partial w_{ij}} = -\alpha\frac{\partial C}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial w_{ij}} = -\alpha\frac{\partial C}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial z_i}\frac{\partial z_i}{\partial w_{ij}} = -\alpha\frac{\partial C}{\partial \hat{y}_i}\frac{\partial f(z_i)}{\partial z_i}x_{ij} = -\alpha\delta_i x_{ij} \tag{4.18}$$

Las señales se propagan en dirección hacia adelante y los errores se propagan en dirección hacia atrás. Los errores en las capas de entrada, oculta y salida se acumulan para generar el error en la función de pérdida. En el caso de calcular los errores aplicando MSE, podemos expandir como vemos en la Ecuación 4.19 el error en la capa oculta y en la capa de entrada.

$$C(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f^L(\sum_{j=1}^{m}x_{ij}^L w_{ij}^L + w_{i0}^L))^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - f^L(\sum_{j=1}^{m}f^{L-1}(\sum_{k=1}^{p}x_{jk}^{L-1}w_{jk}^{L-1} + w_{j0}^{L-1})w_{ij}^L + w_{i0}^L))^2 \tag{4.19}$$

Donde L es el número de capas de la red neuronal, n son neuronas de la capa de salida, m las neuronas de la capa oculta y p las neuronas de la capa de entrada. De forma genérica y expresado de forma matricial la expresión sería

$$C(y, \hat{y}) = C(y, f^L(W^L f^{L-1}(W^{L-1} \dots f^2(W^2 f^1(W^1 x)) \dots))) \tag{4.20}$$

Como podemos ver para calcular $\Delta w_{ij} = -\eta \frac{\partial C}{\partial w_{ij}}$ en las primeras capas de redes neuronales muy profundas se necesitaría demasiada capacidad de computo, por lo que es necesario un algoritmo que realice las derivadas de forma más eficiente. Este algoritmo es el algoritmo retropropagación o *backpropagation* que explicaremos a continuación para realizar la derivada total de la función de coste respecto de las entradas utilizando la regla de la cadena.

$$\frac{dC(y, \hat{y})}{dx} = \frac{dC}{d\hat{y}^L} \circ \frac{d\hat{y}^L}{dz^L} \cdot \frac{dz^L}{d\hat{y}^{L-1}} \circ \frac{d\hat{y}^{L-1}}{dz^{L-1}} \cdot \frac{dz^{L-1}}{d\hat{y}^{L-2}} \circ \dots \circ \frac{d\hat{y}^1}{dz^1} \cdot \frac{\partial z^1}{\partial x^1} \tag{4.21}$$

Donde $(A \circ B)_{ij} = (A \odot B)_{ij} = (A)_{ij}(B)_{ij}$ es el producto de Hadamard o producto de Schur, es decir, un producto de elementos. Además sabiendo que $\frac{d\hat{y}^L}{dz^L} = \frac{df(z^L)}{dz^L} = (f^L)'$ y que $\frac{dz^L}{d\hat{y}^{L-1}} = \frac{dz^L}{dx^L} = W^L$ podemos reducir la expresión de la siguiente forma:

$$\frac{dC(y, \hat{y})}{dx} = \frac{dC}{d\hat{y}^L} \circ (f^L)' \cdot W^L \circ (f^{L-1})' \cdot W^{L-1} \circ \dots \circ (f^1)' \cdot W^1 \tag{4.22}$$

Transponemos las matrices e invertimos el orden de multiplicación, ya que, el gradiente (∇) es la transposición de la derivada de la salida en términos de la entrada, pero las entradas son las mismas

$$\nabla_x C = (W^1)^T \cdot (f^1)' \circ \dots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} C \tag{4.23}$$

Llegados a esta expresión nos damos cuenta de su recursividad, para lo cual vamos a utilizar el termino auxiliar δ^l, interpretado como el "error a nivel l" y definido como el gradiente de los valores de entrada en el nivel l.

$$\delta^l := (f^l)' \circ (W^{l+1})^T \circ \dots \circ (W^{L-1})^T \cdot (f^{L-1})' \circ (W^L)^T \cdot (f^L)' \circ \nabla_{a^L} C \tag{4.24}$$

Por lo que el gradiente de los pesos en la capa l es entonces $\nabla_{W^l} C = \delta^l (\hat{y}^{l-1})^T = \delta^l (x^l)^T$, calculando δ^l fácilmente de forma recursiva, yendo de derecha a izquierda, como $\delta^{l-1} := (f^{l-1})' \circ (W^l)^T \cdot \delta^l$.

Observamos por tanto que *backpropagation* es un algoritmo utilizado en el aprendizaje supervisado de redes neuronales artificiales para calcular de manera efectiva las derivadas parciales de una función de coste con respecto a cada uno de los pesos en la red. Esto se hace utilizando la regla de la cadena para calcular las derivadas parciales de forma recursiva a través de la red, comenzando en la salida y trabajando hacia atrás, hacia la entrada. Una vez que se han calculado estas derivadas, se pueden utilizar para actualizar los pesos de las conexiones entre las neuronas de la red de manera que se minimice la función de coste mediante optimizadores como el algoritmo de descenso de gradiente.

4.2.2. Optimizadores

El propósito del optimizador es proporcionar una regla de actualización de los pesos que modifica iterativamente los pesos en función de la función de coste de manera que la salida de la red sea lo más precisa posible para un conjunto dado de datos de entrada y salida deseados hasta alcanzar el punto óptimo de la red neuronal. Existen varios métodos de optimización de los cuales hablaremos en esta sección.

Descenso del gradiente

El descenso del gradiente (GD) es un algoritmo de optimización que vemos en la Ecuación 4.25 utilizado en el entrenamiento de redes neuronales y otros modelos de aprendizaje automático.

$$w_{t+1} = w_t - \alpha \nabla_{w_t} C(w_t) \tag{4.25}$$

Donde w_t es el vector de pesos del modelo en la iteración t, $\alpha > 0$ es la tasa de aprendizaje (*learning rate*) que controla el tamaño de los pasos en la dirección del gradiente y $\nabla_{w_t} C(w_t)$ es el gradiente de la función de pérdida $C(y, \hat{y}(w_t))$ evaluada en el punto w_t.

El algoritmo se basa en el concepto matemático del gradiente, que es una medida de cuánto cambia una función en una dirección determinada. Durante el entrenamiento de una red neuronal, se utiliza el gradiente descendente para ajustar los pesos en las conexiones entre las neuronas de manera que se minimice el error entre las salidas de la red y las salidas esperadas. Para hacer esto, se calcula el gradiente del error en relación con los pesos de la red y se ajustan los pesos en la dirección del gradiente, lo que reduce el error. El descenso del gradiente es un algoritmo iterativo, lo que significa que se repite varias veces durante el entrenamiento de la red neuronal, lo que permite que la red aprenda a realizar la tarea deseada.

Dependiendo de las muestras seleccionadas para cada entrenamiento tenemos tres tipos de descenso del gradiente que son:

- Descenso del gradiente por lotes (BGD): utiliza todas las muestras del conjunto de entrenamiento (m en total) en cada iteración para actualizar el parámetro de peso en función del valor de gradiente en el punto actual. Con respecto a las otras variantes, tiene una gran estabilidad y precisión, pero es más lento y requiere más memoria, por lo que consume más recursos informáticos. La fórmula es:

$$w_{t+1} = w_t - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla_{w_t} C_i(w_t)$$

- Descenso del gradiente estocástico (SGD): selecciona una única muestra aleatoria en cada iteración para actualizar el parámetro de peso en función del valor de gradiente en el punto actual. Con respecto a las otras variantes, es más rápido y menos costoso pero más ruidoso y menos estable. Tal inestabilidad incluso causa desplazamiento inverso cuando la función de pérdida disminuye al punto más bajo. La fórmula es:

$$w_{t+1} = w_t - \alpha \nabla_{w_t} C_i(w_t)$$

- Descenso del gradiente por mini-lotes (MBGD): combina las características de BGD y SGD, por lo que en cada iteración selecciona un subconjunto pequeño y aleatorio de datos con n muestras en cada iteración para actualizar el parámetro de peso. De tal forma que es un compromiso entre los dos anteriores que combina velocidad y precisión, reduciendo la varianza y mejorando la convergencia. La fórmula es:

$$w_{t+1} = w_t - \alpha \frac{1}{n} \sum_{i=b}^{b+n-1} \nabla_{w_t} C_i(w_t)$$

Donde $\nabla_{w_t} C_i(w_t)$ es el gradiente de la función de pérdida $C(y, \hat{y}(w_t))$ evaluada en el punto w_t para el i-ésimo ejemplo del conjunto de datos, m es el tamaño del conjunto de entrenamiento, $n \in [0, m - b]$ es el tamaño del mini-batch y $b \in [0, m - n]$ es donde comienza el mini-batch.

Además se puede añadir un vector de momento para mejorar el algoritmo de GD que pueden acelerar la convergencia y evitar los mínimos locales al dar más peso a la dirección del movimiento anterior mejorando el rendimiento del algoritmo aunque requieren más memoria para almacenar el vector de momento y más hiperparámetros para ajustar. Este es el caso de [243]:

- Descenso del gradiente con momento clasico (GD-CM) [244]: es el método clásico de momento que actualiza los parámetros usando una combinación del gradiente actual y el gradiente anterior. La fórmula es:

$$w_{t+1} = w_t + v_{t+1} = w_t + \mu v_t - \alpha \nabla_{w_t} C(w_t)$$

- Descenso del gradiente con gradiente acelerado de Nesterov (GD-NAG) [245]: es una variante del CM que usa el gradiente calculado después del subpaso de momento en lugar del gradiente actual. La fórmula es:

$$w_{t+1} = w_t + v_{t+1} = w_t + \mu v_t - \alpha \nabla_{w_t} C(w_t + \mu v_t)$$

Donde $\mu \in [0, 1]$ es el factor de momento que controla la influencia del gradiente anterior en la dirección de actualización y v_t es el vector de momento en la iteración t. La ventaja de NAG sobre CM es que puede corregir mejor la dirección del movimiento si se aleja demasiado del mínimo.

AdaGrad

En el método AdaGrad [246], se divide α de cada paso por la norma L_2 de todos los gradientes anteriores, esto ralentiza el aprendizaje a lo largo de las dimensiones que ya han cambiado significativamente y acelera el aprendizaje a lo largo de las dimensiones que solo han cambiado levemente, estabilizando la representación del modelo de características comunes y permitiéndole "ponerse al día" rápidamente con su representación de características raras.

Este método es especialmente útil para datos dispersos. Sin embargo, el denominador acumula los gradientes al cuadrado de cada iteración a partir del comienzo del entrenamiento. Dado que cada término es positivo, esta suma acumulada continúa creciendo a lo largo del entrenamiento, reduciendo efectivamente la tasa de aprendizaje en cada dimensión. Después de muchas iteraciones, esta tasa de aprendizaje se volverá infinitesimalmente pequeña y el algoritmo puede detenerse prematuramente.

$$w_{t+1} = w_t + \Delta w_t = w_t - \frac{\alpha}{\sqrt{n_t + \epsilon}} g_t \tag{4.26}$$

Donde la tasa de aprendizaje es α, el gradiente actual es $g_t = \nabla_{w_{t-1}} C_t(w_{t-1})$, el gradiente acumulado es $n_t = n_{t-1} + g_t^2$, siendo $g_t^2 = g_t \circ g_t$ y ϵ es una pequeña constante y se establece en aproximadamente 10^{-7} para la estabilidad numérica evitando la división por cero.

AdaDelta

La idea de AdaDelta [247] se derivó de AdaGrad con el fin de mejorar los dos principales inconvenientes del método: 1) la disminución continua de las tasas de aprendizaje a lo largo del entrenamiento y 2) la necesidad de una tasa de aprendizaje global seleccionada manualmente.

Este algoritmo trata de resolver el problema de AdaGrad usando una media móvil exponencial (EMA) del cuadrado del gradiente en lugar del cuadrado del gradiente acumulado. Esto hace que la tasa de aprendizaje no dependa del número total de iteraciones, sino solo de las más recientes. También elimina la necesidad de una tasa de aprendizaje inicial y usa la media móvil exponencial del cambio en los parámetros para escalar el gradiente.

$$w_{t+1} = w_t + \Delta w_t = w_t - \frac{RMS[\Delta w]_{t-1}}{RMS[g]_t} g_t = w_t - \frac{\sqrt{E[\Delta w^2]_{t-1} + \epsilon}}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{4.27}$$

Donde la tasa de aprendizaje global en este caso son las actualizaciones acumuladas que se definen como $E[\Delta w^2]_t = \mu E[\Delta w^2]_{t-1} + (1-\mu)\Delta w_t^2$, el gradiente actual es $g_t = \nabla_{w_{t-1}} C_t(w_{t-1})$, el gradiente acumulado es $E[g^2]_t = \mu E[g^2]_{t-1} + (1-\mu) g_t^2$, es decir, es la EMA del cuadrado del gradiente y ϵ es una pequeña constante y se establece en aproximadamente 10^{-7} para la estabilidad numérica evitando la división por cero.

RMSProp

RMSProp [248] es similar a AdaDelta en el sentido de que también utiliza la media móvil exponencial del cuadrado del gradiente y fue propuesto por Geoffrey Hinton en su curso sobre redes neuronales en Coursera. Aunque RMSProp utiliza una tasa de aprendizaje global que le permite controlar mejor el ritmo de decrecimiento de la tasa de aprendizaje.

$$w_{t+1} = w_t + \Delta w_t = w_t - \frac{\alpha}{RMS[g]_t} g_t = w_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{4.28}$$

Adam

Adam [249] este algoritmo combina las ideas de RMSProp y AdaGrad, usando medias móviles exponenciales tanto del gradiente como del cuadrado del gradiente. Además, introduce un sesgo correctivo para evitar que las medias estén sesgadas hacia cero al principio. De tal forma que está diseñado para combinar las ventajas de AdaGrad que funciona bien con gradientes dispersos, y RMSProp que funciona bien en entornos en línea y no estacionarios. Por lo tanto, algunas de las ventajas de Adam son que las magnitudes de las actualizaciones de los

parámetros son invariantes al reescalado del gradiente, sus tamaños de paso están aproximadamente acotados por el hiperparámetro del tamaño de paso, no requiere un objetivo estacionario, funciona con gradientes dispersos y realiza naturalmente una forma de enfriamiento del tamaño de paso.

El algoritmo iterativo matemático de Adam es bastante complejo por lo que trataremos de desarrollarlo a continuación:

1) Se calcula la estimación del primer momento corregido por el sesgo actualizándolo mediante la expresión:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} = \frac{\beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t}{1-\beta_1^t} \tag{4.29}$$

2) Se calcula la estimación del segundo momento corregido por el sesgo actualizándolo mediante la igualdad:

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} = \frac{\beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2}{1-\beta_2^t} \tag{4.30}$$

3) Se actualizan los parametros mediante la expresion:

$$w_t = w_{t-1} - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4.31}$$

Donde α es el tamaño de paso con el valor predeterminado de $\alpha = 0.001$, $\beta_1, \beta_2 \in [0, 1)$ son las tasas de decaimiento exponencial para las estimaciones de los momentos $\hat{m}_t$ y $\hat{v}_t$ con el valor predeterminado de $\beta_1 = 0.9$ y $\beta_2 = 0.999$, el gradiente con respecto a los pesos en el *timestep* t es $g_t = \nabla_{w_{t-1}} C_t(w_{t-1})$ y ϵ es una pequeña constante y se establece con el valor predeterminado de 10^{-8} para la estabilidad numérica evitando la división por cero.

AdaMax

AdaMax [249] este algoritmo es una variante de Adam que usa la norma L_∞ en lugar de la norma L_2 para estimar el cuadrado del gradiente. Esto hace que sea más robusto a gradientes dispersos o ruidosos.

$$w_t = w_{t-1} - \frac{\alpha}{1-\beta_1^t} \cdot \frac{m_t}{u_t} = \theta_{t-1} - \frac{\alpha}{1-\beta_1^t} \cdot \frac{\beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t}{max(\beta_2 \cdot u_{t-1}, |g_t|)} \tag{4.32}$$

Donde α es el tamaño de paso con el valor predeterminado de $\alpha = 0.002$, $\beta_1, \beta_2 \in [0, 1)$ son las tasas de decaimiento exponencial para las estimaciones de los momentos m_t y u_t con el valor predeterminado de $\beta_1 = 0.9$ y $\beta_2 = 0.999$, el gradiente con respecto a los pesos en el *timestep* t es $g_t = \nabla_{w_{t-1}} C_t(w_{t-1})$ y $\frac{\alpha}{1-\beta_1^t}$ es la tasa de aprendizaje con el término de corrección de sesgo para el primer momento.

Nadam

Nadam [250] este algoritmo es una combinación de Adam y NAG. El algoritmo iterativo matemático de Nadam es bastante complejo por lo que trataremos de desarrollarlo a continuación:

1) Se calcula la estimación del primer momento corregido por el sesgo actualizándolo mediante la expresión:

$$\bar{m}_t = (1-\mu_t)\hat{g}_t + \mu_{t+1}\hat{m}_t = \frac{(1-\mu_t)g_t}{1-\prod_{i=1}^{t}\mu_i} + \frac{\mu_{t+1}m_t}{1-\prod_{i=1}^{t+1}\mu_i} = \frac{(1-\mu_t)g_t}{1-\prod_{i=1}^{t}\mu_i} + \frac{\mu_{t+1}(\mu_t m_{t-1} + (1-\mu_t)g_t)}{1-\prod_{i=1}^{t+1}\mu_i} \tag{4.33}$$

2) Se calcula la estimación del segundo momento corregido por el sesgo actualizándolo mediante la igualdad:

$$\hat{n}_t = \frac{\nu n_t}{1-\nu^t} = \nu \cdot \frac{\nu \cdot n_{t-1} + (1-\nu) \cdot g_t^2}{1-\nu^t} \tag{4.34}$$

3) Se actualizan los parametros mediante la expresion:

$$w_t = w_{t-1} - \alpha_t \frac{\bar{m}_t}{\sqrt{\hat{n}_t} + \epsilon} \tag{4.35}$$

Donde α_t es el tamaño de paso en el *timestep* t, $\mu_t, \nu \in [0, 1)$ son las tasas de decaimiento exponencial para las estimaciones de los momentos $\bar{m}_t$ y $\hat{n}_t$, el gradiente con respecto a los pesos en el *timestep* t es $g_t = \nabla_{w_{t-1}} C_t(w_{t-1})$ y ϵ es una pequeña constante y se establece para la estabilidad numérica evitando la división por cero.

4.2.3. Regularización

La regularización es una forma de modificar el algoritmo de aprendizaje automático para reducir el error de generalización, es decir, el error que comete el modelo al predecir nuevos datos que no ha visto antes. La regularización busca encontrar un equilibrio entre el sesgo y la varianza del modelo que podemos ver en la Figura 4.4, evitando que se ajuste demasiado a los datos de entrenamiento (*overfitting* o sobreajuste) o que sea demasiado simple y no capte las características relevantes de los datos (*underfitting* o subajuste). Cuando el modelo tiene un sesgo bajo y una varianza alta, o cuando tenemos una pequeña cantidad de datos tiende a sobreajustarse, por lo no podrá generalizar correctamente las estimaciones en los datos de prueba. La regularización se puede aplicar a diferentes tipos de modelos y algoritmos de aprendizaje automático.

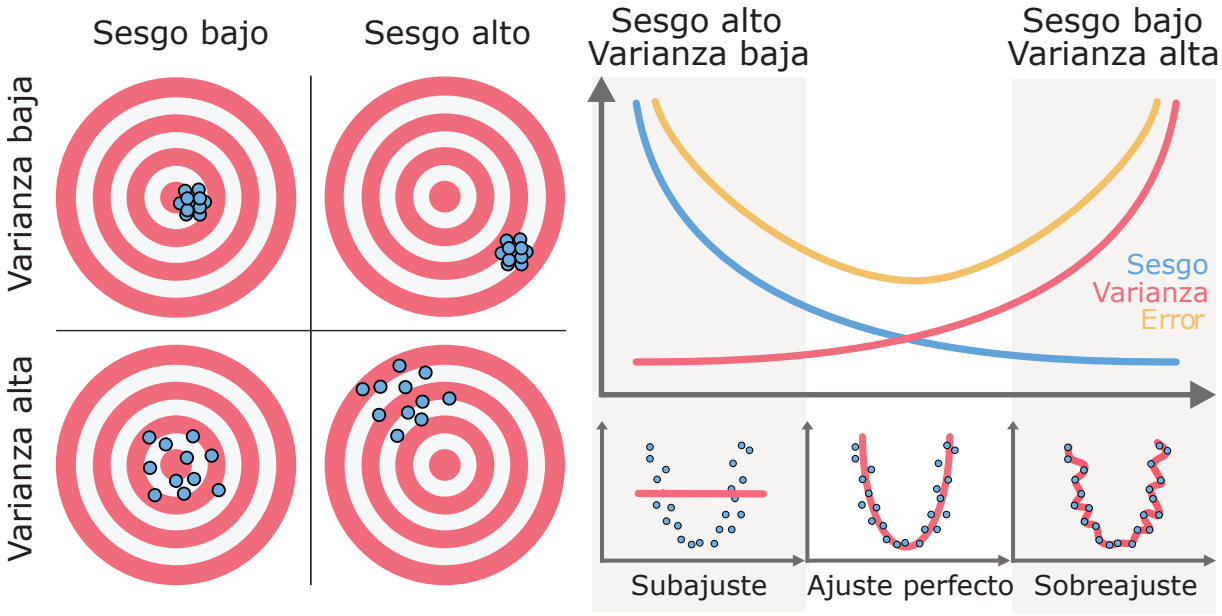


Figura 4.4: Sesgo y varianza en el entrenamiento de una red neuronal [251].

El sesgo mide la diferencia entre la predicción promedio del modelo y el valor real que se quiere predecir, es decir, el sesgo de la función estimada nos dice el error sistemático que comete el modelo para predecir los valores al ignorar parte de la información de los datos. La varianza mide la variabilidad de las predicciones promedio del modelo para un conjunto de datos dado, es decir, la varianza de la función estimada es el error aleatorio que comete el modelo al ser sensible a pequeños cambios en los datos. Un alto sesgo indica que el modelo es demasiado simple y no se ajusta bien a los datos (subajuste), lo que implica un alto error tanto en los datos de entrenamiento como en los de prueba. Una alta varianza indica que el modelo es demasiado complejo y se ajusta demasiado a los datos (sobreajuste) comenzando a modelar el ruido en la entrada, lo que implica un bajo error en los datos de

entrenamiento pero un alto error en los de prueba. El objetivo es encontrar un equilibrio entre el sesgo y la varianza que minimice el error total, que es la suma del sesgo al cuadrado más la varianza más el ruido irreducible.

Vamos a hablar sobre algunos de los tipos de regularización que existen como son modificar la función de pérdida, modificar el muestreo de datos, cambiar el enfoque de entrenamiento o combinar varias técnicas de aprendizaje automático en un modelo predictivo.

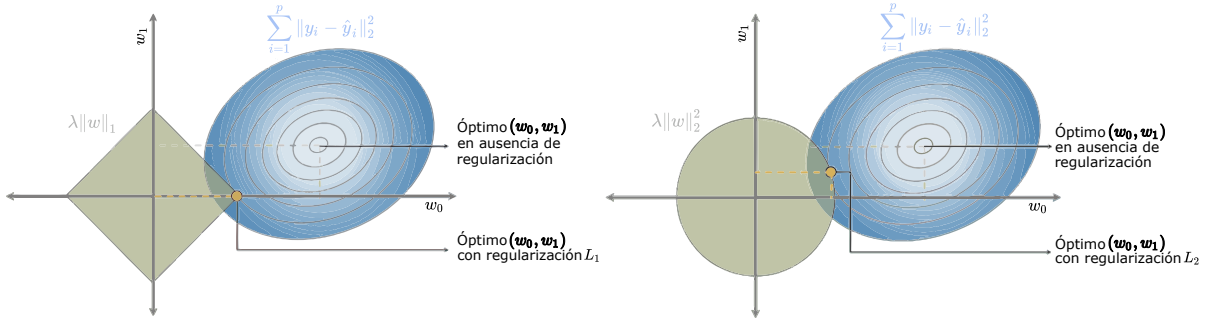
Modificar la función de pérdida

Para modificar la función de perdida se suele introducir una penalización por la complejidad del modelo en la función de coste de la siguiente forma $Coste(w) = Error(w) + \lambda Complejidad(w)$, donde $\lambda \geq 0$ es una constante que regula la importancia de la complejidad en el coste total proporcional al sesgo del modelo e inversamente proporcional a la varianza del modelo. Ahora vamos a mencionar algunas de las penalizaciones mas importantes.

El decaimiento de los pesos (*Weight decay*) es la penalización L_2, también llamada penalización Ridge o Prior gaussiano es una técnica de regularización que penaliza los pesos grandes en la función de costo. La penalización se agrega a la función de costo como un término que es proporcional a la norma L_2 de los pesos. La norma L_2 de un vector de pesos es simplemente la raíz cuadrada de la suma de los cuadrados de los pesos, de forma que $Complejidad(w) = \|w\|_2$, además podemos ver la forma general para la norma L_p en la Ecuación 4.36. Al penalizar los pesos grandes, la regularización L_2 fuerza al modelo a tener pesos más pequeños y reduce la complejidad del modelo, lo que a su vez reduce la posibilidad de sobreajuste.

$$Complejidad(w) = \|w\|_p = \left(\sum_{i=1}^{N} |w_i|^p \right)^{1/p} \tag{4.36}$$

La regularización L_1 también llamada penalización LASSO o Prior laplaciano es similar a la regularización L_2, pero penaliza los pesos grandes de una manera diferente. En lugar de penalizar los pesos grandes proporcionalmente a su cuadrado, la regularización L_1 los penaliza proporcionalmente a su valor absoluto, de forma que $Complejidad(w) = \|w\|_1$. Esto tiene el efecto de introducir escasez en los pesos, lo que significa que más pesos se establecerán en cero. La introducción de ceros en los pesos reduce aún más la complejidad del modelo, lo que puede ayudar a prevenir el sobreajuste.



(a) Regularización L_1.

(b) Regularización L_2.

Figura 4.5: Técnicas para modificar la función de perdida.

Por otro lado, Elastic Net es una técnica de regularización que combina la regularización L_1 y L_2 en la función de costo. Esto significa que la función de costo incluye tanto el término proporcional a la norma L_1 de los pesos como el término proporcional al cuadrado de la norma L_2 de los pesos. Elastic Net combina las ventajas de ambas técnicas de regularización, lo que puede ayudar a mejorar la precisión predictiva y a seleccionar las variables más importantes en modelos de aprendizaje automático.

La regularización L_∞ es una técnica de regularización que penaliza los pesos grandes en la función de costo, proporcional a la norma L_∞ de los pesos. La norma L_∞ de un vector de pesos es simplemente el valor absoluto

del peso más grande en el vector como podemos ver en la Ecuación 4.37. La regularización L_∞ puede ser útil en situaciones donde se desea una regularización más fuerte en algunos pesos en particular.

$$Complejidad(w) = \lim_{p\to\infty} \|w\|_p = \|w\|_\infty = \max(|w_1|, |w_2|, \ldots, |w_N|) \tag{4.37}$$

La entropía es una medida de la incertidumbre en una distribución de probabilidad. En el aprendizaje automático, la entropía se utiliza a menudo para evaluar la calidad de una distribución de probabilidad generada por un modelo. La entropía se utiliza en particular en los modelos de clasificación, donde se desea que el modelo genere una distribución de probabilidad que sea lo más cercana posible a una distribución uniforme, de forma que $Complejidad(w) = -\sum_{i=1}^{N} \hat{y}_i \log(\hat{y}_i)$. Forzar una distribución uniforme a través de la entropía puede ayudar a evitar el sobreajuste y mejorar la capacidad de generalización del modelo [251].

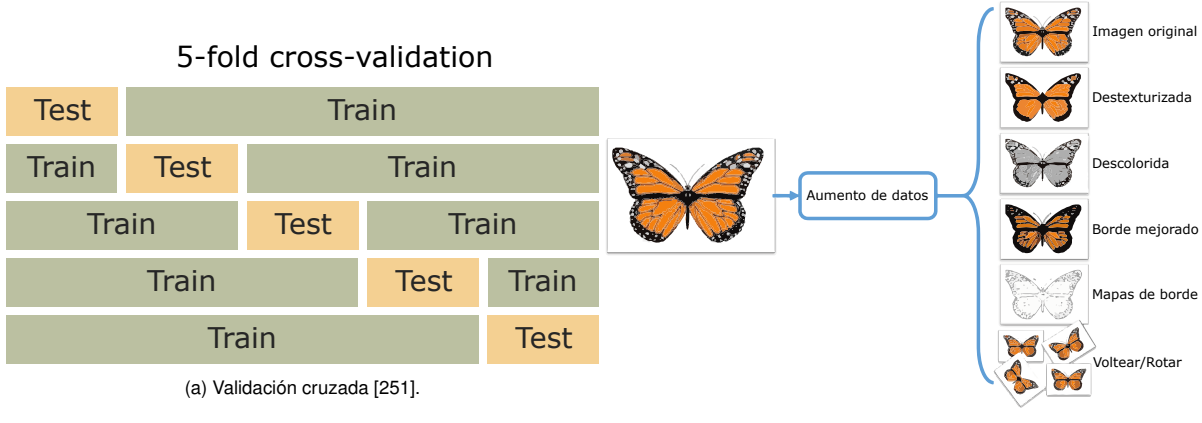
Se puede realizar una modificación en la función de perdida a la hora de aplicar el algoritmo de optimización, como hemos visto en la sección 4.2.2 la mayoría aplican la penalización L_2, aunque AdaMax por ejemplo aplica la penalización L_∞

Modificar el muestreo de datos

Hay dos formas principales de modificar el muestreo de datos la primera es mediante la validación cruzada que vimos en la sección 3.3 y la segunda mediante *data augmentation* que se refiere a la técnica de aumentar el conjunto de datos disponibles mediante la creación de nuevos datos a partir de los datos existentes.

El aumento de datos es una técnica útil para mejorar el rendimiento y la generalización de los modelos de ML, especialmente cuando el conjunto de datos de entrenamiento es pequeño. Al generar datos sintéticos a partir de los datos existentes, se pueden crear más variaciones y diversidad en los datos de entrenamiento, lo que ayuda a evitar el sobreajuste y mejora la capacidad del modelo para generalizar a datos nuevos y no vistos.

En el caso de aplicarse en visión por computadora para generar nuevas variaciones de imágenes de entrenamiento, se pueden realizar rotaciones, cambios en la iluminación y perspectivas, recortes, distorsión de las imágenes u otros cambios como vemos en la Figura 4.6. Mientras que si se aplica en PLN, las técnicas de aumento de datos que pueden aplicarse para generar nuevas variaciones de texto de entrenamiento, serían reemplazo de sinónimos, inserción aleatoria, eliminación aleatoria e incrustaciones de palabras, lo que da lugar a cambios de palabras, eliminación de palabras y agregación de palabras.



(a) Validación cruzada [251].

(b) *Data augmentation* [252].

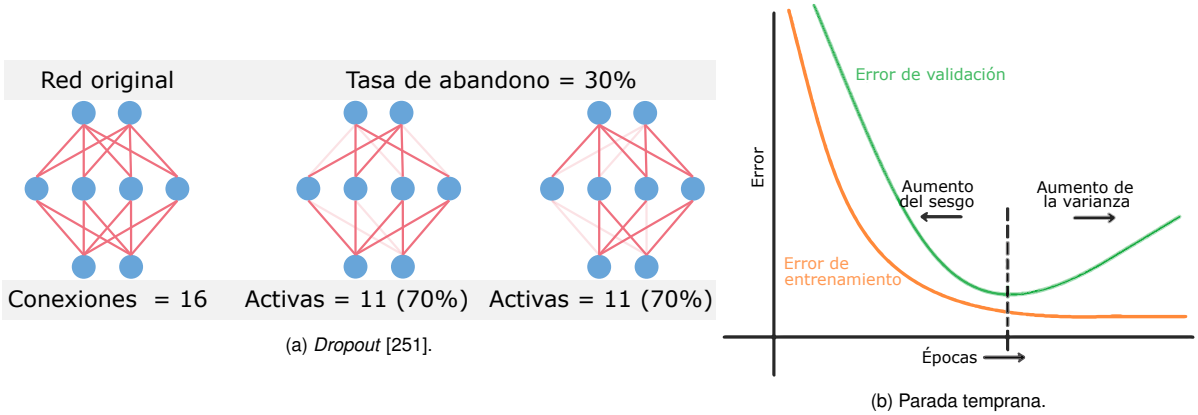
Figura 4.6: Técnicas para modificar el muestreo de datos.

Cambiar el enfoque de entrenamiento

También podemos cambiar el enfoque del entrenamiento haciendo normalización por lotes (*batch normalization*), *dropout*, *zoneout*, parada temprana (*early stopping*) o inyectando ruido para reducir el error de generalización.

A menudo se introduce ruido en las entradas como una estrategia de aumento del conjunto de datos, pero se puede agregar ruido aleatorio a otras partes de la red durante el entrenamiento. Por ejemplo se puede alterar los pesos de la red agregando ruido aleatorio a los pesos cuando está aprendiendo para hacer la red más robusta a pequeñas variaciones en los pesos, encontrando puntos que no son simplemente mínimos, sino mínimos rodeados de regiones planas, el ruido agregado a los pesos puede interpretarse como una forma más tradicional de regularización. Además también se puede inyectar ruido en las salidas, siendo un ejemplo de inyección de ruido en las salidas el suavizado de etiquetas, ya que en el conjunto de datos del mundo real, podemos esperar cierta cantidad de errores en las etiquetas de salida y una forma de remediar esto es modelar explícitamente el ruido en las etiquetas.

La deserción, abandono o *dropout* [238] es una técnica introducida por primera vez con dicho nombre por Geoffrey Hinton en 2012 para abordar el problema de sobreajuste en redes neuronales profundas con una gran cantidad de parámetros, estas grandes redes también son lentas de usar, lo que dificulta lidiar con el sobreajuste al combinar las predicciones de diferentes grandes redes en el momento de la prueba. La idea clave es eliminar aleatoriamente unidades (junto con sus conexiones) de la red neuronal durante el entrenamiento. Esto evita que las unidades se adapten demasiado. La deserción se puede implementar fácilmente durante el entrenamiento seleccionando aleatoriamente nodos para que se descarten con una probabilidad determinada (e.g. 0,1 %) en cada ciclo de actualización de peso, entrenando así diferentes redes "delgadas". Sólo se usa durante el entrenamiento de un modelo, por lo que en el momento de la prueba, es fácil aproximar el efecto de promediar las predicciones de todas estas redes diluidas simplemente usando una sola red no diluida que tiene pesos más pequeños. Esta técnica es simple y computacionalmente barata, además se puede utilizar en otros modelos que no son de aprendizaje profundo, sin embargo, es menos eficaz cuando los datos de entrenamiento son insuficientes.



(a) *Dropout* [251].

(b) Parada temprana.

Figura 4.7: Técnicas para cambiar el enfoque de entrenamiento.

Al igual que el *dropout*, *zoneout* [239] utiliza ruido aleatorio para entrenar un pseudo-conjunto y mejorar la generalización pero preservando en lugar de eliminar unidades ocultas. En cada paso de tiempo, *zoneout* obliga estocásticamente a algunas unidades ocultas a mantener su valores anteriores de las unidades ocultas de la red para mejorar la propagación del gradiente y la información del estado a través del tiempo.

La normalización por lotes (*batch normalization*) [253] es una técnica que normaliza las entradas a cada capa de una red neuronal para que tengan una media de 0 y una varianza 1. Ayuda a estabilizar el aprendizaje y acelera el entrenamiento del modelo, lo que puede mejorar significativamente el rendimiento y la precisión. También reduce los problemas debidos a una mala inicialización de parámetros, por lo que la inicialización de parámetros es más fácil y se pueden utilizar tasas de aprendizaje más altas. Además agrega ruido que reduce el sobreajuste con un efecto de regularización, por lo tanto, cuando se aplica la normalización por lotes, se debe utilizar menos *dropout*, ya que el *dropout* en sí mismo agrega ruido.

Parada temprana (*early stopping*) es una técnica utilizada para prevenir el sobreajuste. Se detiene el entrenamiento del modelo antes de que alcance su punto máximo, lo que ayuda a evitar que el modelo se ajuste demasiado a los datos de entrenamiento y mejora la capacidad de generalización del modelo. Mientras se construye el modelo, se evalúa en el conjunto de datos de validación después de cada época durante el entrenamiento. Si la precisión

del modelo en el conjunto de datos de validación comienza a degradarse (e.g., la pérdida comienza a aumentar o la precisión comienza a disminuir), el proceso de entrenamiento se detienen y realiza una detención temprana.

Combinar varias técnicas de ML

Los métodos de conjunto o *ensembling* combinan múltiples modelos débiles en un modelo predictivo fuerte para producir una predicción final. Al combinar los resultados de varios modelos, se puede mejorar el rendimiento, la precisión y la robustez del modelo, además de reducir el sesgo, la varianza y el riesgo de sobreajuste. Hay muchos métodos diferentes para ensamblar, los dos más comunes son *bagging* y *boosting* pero añadiremos un tercero que es *stacking* [251].

El *bagging* entrena N modelos débiles diferentes (generalmente del mismo tipo, homogéneos) con N subconjuntos no superpuestos del conjunto de datos de entrada en paralelo. En la fase de prueba, se evalúa cada modelo. La etiqueta con el mayor número de predicciones se selecciona como predicción, es decir, por votación simple. Los métodos de embolsado reducen la varianza de la predicción, debido a que una forma de reducir la varianza de una estimación es promediar varias estimaciones, por lo que combina a todos los aprendices débiles para "suavizar" sus predicciones.

El *boosting* entrena N modelos débiles diferentes (generalmente del mismo tipo, homogéneos) con el conjunto de datos completo en un orden secuencial. Los *datapoints* clasificados incorrectamente con el modelo débil anterior reciben más pesos para que puedan ser clasificados correctamente por el siguiente modelo débil, haciendo que cada modelo débil en la secuencia se enfoque en aprender de los errores del anterior pudiendo convertir a los estudiantes débiles en estudiantes fuertes. En la fase de prueba, se evalúa cada modelo y, en función del error de prueba de cada modelo débil, se realiza una votación ponderada para obtener la predicción. Los métodos de refuerzo disminuyen el sesgo de la predicción. El *bagging* utiliza modelos base complejos e intenta "suavizar" sus predicciones, mientras que el impulso utiliza modelos base simples e intenta "aumentar" su complejidad agregada.

El *stacking* entrena N modelos débiles diferentes (generalmente de diferentes tipos, heterogéneos) con uno de los dos subconjuntos del conjunto de datos en paralelo. Una vez que los alumnos débiles están capacitados, se utilizan para entrenar a un metaaprendiz para combinar sus predicciones y llevar a cabo la predicción final utilizando el otro subconjunto. En la fase de prueba, cada modelo predice su etiqueta, este conjunto de etiquetas se envía al metaaprendiz que genera la predicción final.

4.3. Aprendizaje profundo

El aprendizaje profundo (DL) es una rama del aprendizaje automático que utiliza redes neuronales artificiales (ANN) profundas para aprender a partir de grandes conjuntos de datos y un proceso iterativo de ajuste de parámetros. El aprendizaje profundo se considera una forma avanzada de aprendizaje automático y ha dado lugar a muchas aplicaciones prácticas en diversos campos. Estas redes neuronales son capaces de aprender representaciones abstractas de los datos y realizar tareas complejas como el reconocimiento de patrones en imágenes y procesamiento del lenguaje natural (NLP)).

En contraste con el aprendizaje automático convencional, donde esas características suelen estar diseñadas a mano con el objetivo de reducir el tamaño de los datos, que se requiere para describir el problema de predicción en cuestión. Posteriormente, las características calculadas se utilizan como entrada para que el algoritmo entrene y permita el aprendizaje de un patrón representado en los datos. Esto tiene claras ventajas. Primero, una representación condensada de los datos requiere menos memoria, menos poder de cómputo y, en última instancia, reduce el riesgo de sobreajustar el modelo a las muestras de entrenamiento, lo que resulta en una mala generalización a nuevas muestras no vistas. Sin embargo, todo esto conlleva el riesgo de perder información valiosa y limitar el poder de representación del algoritmo de aprendizaje.

Por el contrario, el aprendizaje profundo, una clase específica de aprendizaje automático, cambia la complejidad de la ingeniería de funciones artesanal hacia la optimización del modelo y el uso de la potencia de cálculo y los datos disponibles. Una red neuronal grande y profunda tiene muchas más capas y muchos más nodos en cada capa, lo que da como resultado exponencialmente muchos más parámetros para ajustar. Sin suficientes datos, no podemos aprender los parámetros de manera eficiente. Sin ordenadores potentes, el aprendizaje sería demasiado lento e insuficiente.

En un conjunto de datos pequeño, los algoritmos tradicionales de ML (Regresión, Random Forests, SVM, etc.) o el aprendizaje estadístico hacen un gran trabajo, pero una vez que la escala de datos va creciendo, las redes neuronales profundas superan a los demás. En parte porque, en comparación con un modelo de ML tradicional, un modelo de red neuronal tiene muchos más parámetros y tiene la capacidad de aprender patrones no lineales complicados. Por lo tanto, esperamos que el modelo elija las características más útiles por sí mismo sin demasiada ingeniería de características manual involucrada por expertos. Esto se ilustra en la Figura 4.8 que presenta la relación entre la cantidad de datos disponibles y el rendimiento del modelo, propuesto en el libro blanco de la Asociación Canadiense de Radiólogos sobre inteligencia artificial en radiología.

Las arquitecturas de aprendizaje profundo han venido a reforzar el estado del arte de varios dominios, como el reconocimiento de voz y la clasificación de imágenes. Se basan en redes neuronales artificiales y se pueden caracterizar como un modelo que consiste en una cascada de múltiples capas de procesamiento de información no lineal. A diferencia de los algoritmos tradicionales de aprendizaje automático, las redes neuronales profundas se construyen a partir de muchas capas interconectadas de neuronas, lo que les permite aprender de manera más precisa y eficiente.

Aunque algunos algoritmos de detección de tos han utilizado arquitecturas de redes neuronales, se han centrado principalmente en la ingeniería de características y han tenido arquitecturas bastante superficiales debido a la dificultad de recopilar grandes cantidades de datos de sujetos reales, lo que da como resultado menos muestras de datos y limita la escala a una red más profunda [24].

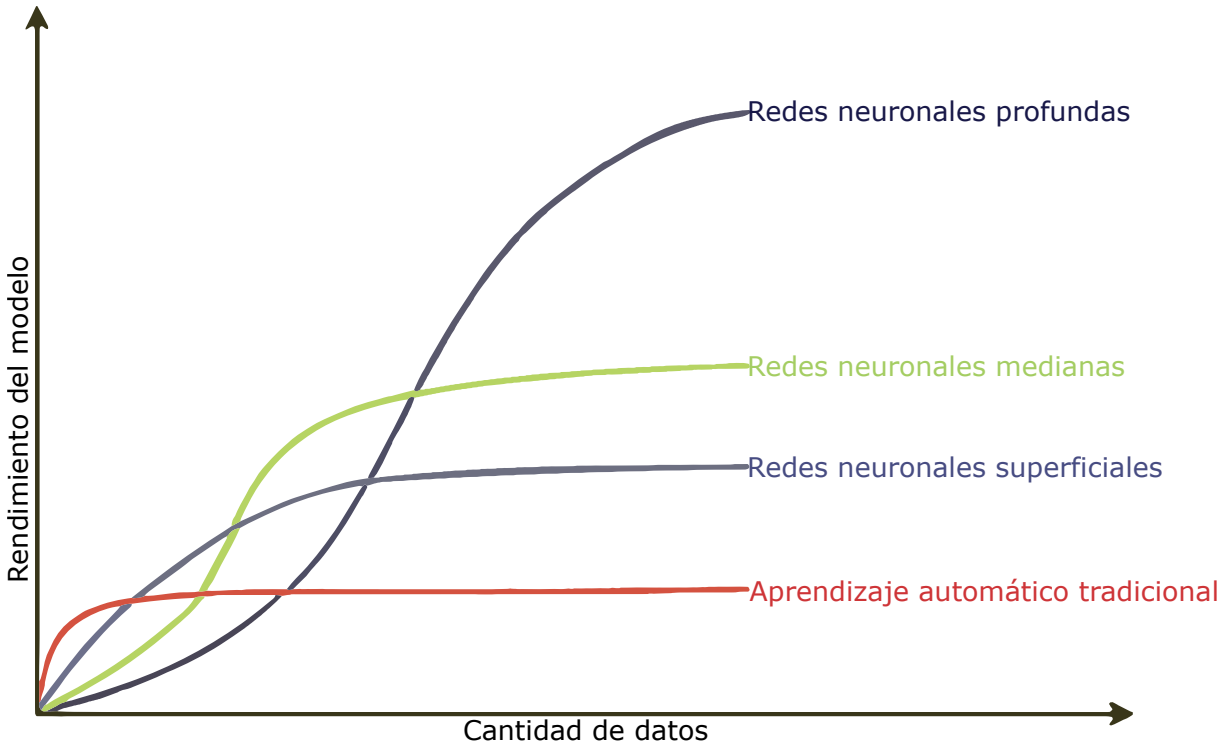


Figura 4.8: Gráfico que ilustra el impacto de los datos disponibles en el rendimiento de los algoritmos tradicionales de aprendizaje automático (que se basan en características hechas a mano) y redes neuronales con pocas capas (superficiales), moderadas (medias) o grandes (profundas) [254].

Hay varios modelos diferentes de redes neuronales profundas que se utilizan en diferentes contextos y para abordar diferentes tipos de problemas y tareas. Algunos de los modelos más comunes incluyen el perceptrón multicapa, las redes neuronales recurrentes, redes neuronales convolucionales, redes neuronales recurrentes convolucionales, redes generativas adversarias, los *transformers* y *autoencoders*, de los que hablaremos a continuación. Para finalmente hablar sobre el aprendizaje por transferencia que es un tipo de aprendizaje importante en el apren-

dizaje profundo.

4.3.1. Perceptrón multicapa

El perceptrón (o neurona de McCulloch-Pitts) fue inventado en 1943 por McCulloch y Pitts [52] y la primera implementación fue una máquina construida en 1958 en el Laboratorio Aeronáutico de Cornell por el psicólogo cognitivo Frank Rosenblatt [255] financiada por la Oficina de Investigación Naval de los Estados Unidos. El perceptrón original es un clasificador binario cuyo algoritmo utilizaba el aprendizaje supervisado, es decir, un algoritmo de clasificación que hace sus predicciones basadas en una función predictora lineal que combina un conjunto de pesos con el vector característico.

Si bien se garantiza que el algoritmo de perceptrón convergerá en alguna solución en el caso de un conjunto de entrenamiento linealmente separable, aún puede elegir cualquier solución y los problemas pueden admitir muchas soluciones de calidad variable. El perceptrón de estabilidad óptima, hoy en día más conocido como la máquina vectorial de soporte lineal (SVM), fue diseñado para resolver este problema [256].

A medida que se desarrollaron nuevos algoritmos de entrenamiento y se crearon nuevos tipos de redes neuronales, el perceptrón original fue mejorado y expandido. Uno de los avances más importantes fue el desarrollo del algoritmo de backpropagation [67] en 1986, que permitió entrenar redes neuronales más complejo como el perceptrón multicapa.

Un perceptrón multicapa (MLP) es un tipo de red neuronal artificial (ANN) *feedforward* en la que las señales viajan en una sola dirección desde la entrada hasta la salida, un ejemplo de este tipo de red lo podemos ver en la Figura 4.3. A veces MLP hace referencia a cualquier red neuronal feedforward (FNN) pero estrictamente se refiere a redes compuestas por múltiples capas de perceptrones. Estas redes se construyen con una o más capas ocultas de neuronas entre la capa de entrada y la capa de salida, siendo capaz resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón simple, por lo que les permite aprender representaciones más complejas y abstractas de los datos.

Los perceptrones multicapa se utilizan comúnmente para resolver problemas de clasificación y regresión y se entrenan utilizando el algoritmo de *backpropagation*. Estos modelos pueden ser muy efectivos en muchas tareas diferentes, pero también pueden ser sensibles a los ruidos en los datos y requieren un gran número de datos de entrenamiento para funcionar adecuadamente.

En la actualidad, los perceptrones multicapa y las redes neuronales *feedforward* en general siguen siendo una herramienta importante en el campo de la inteligencia artificial y se utilizan en una variedad de tareas y problemas.

4.3.2. Redes neuronales recurrentes

Las redes neuronales recurrentes (RNN) son un tipo de red neuronal recursivas con una la estructura de una cadena lineal en la que las señales pueden retroceder y ciclar en la red, lo que permite que la red tenga memoria a corto plazo que se actualiza en cada paso temporal, lo que les permite recordar y utilizar información de entradas anteriores al procesar nuevas entradas. Estas redes se utilizan comúnmente para procesar secuencias de datos, como texto o audio, en tareas de procesamiento del lenguaje natural o generación de secuencias que requieren la comprensión de contexto y dependencia temporal.

El procesamiento del lenguaje natural se refiere a la disciplina de la inteligencia artificial que se encarga del desarrollo de sistemas capaces de procesar y comprender el lenguaje humano. Esta una disciplina es clave, ya que el lenguaje es una de las principales formas en que los seres humanos nos comunicamos y transmitimos información. Los sistemas de procesamiento del lenguaje natural pueden ayudar a facilitar la comunicación entre humanos y máquinas, lo que puede tener aplicaciones en una amplia variedad de campos, como la atención al cliente, la traducción automática, la generación de texto, la extracción de información o la identificación de emociones en el texto.

En el entrenamiento de las RNN se utilizan específicamente dos variantes del algoritmo de retropropagación (*backpropagation*, BP) que están diseñadas para tratar las particularidades de estas redes. La retropropagación a través de la estructura (BPTS) [257] es una técnica basada en gradientes para entrenar redes neuronales recursivas (un superconjunto de redes neuronales recurrentes) en el que se calculan los gradientes a través de toda la red, incluyendo las conexiones entre capas, a diferencia de en la retropropagación tradicional donde se calculan los

gradientes a través de cada capa de la red de forma independiente [258]. Mientras que la retropropagación a través del tiempo (BPTT) [259] es una técnica basada en gradientes para entrenar ciertos tipos de redes neuronales recurrentes donde el algoritmo desenrolla una red neuronal recurrente a través del tiempo, convirtiéndola en una red *feedforward* y utiliza la retropropagación para calcular los gradientes necesarios para actualizar los pesos. Tiende a ser significativamente más rápido para entrenar redes neuronales recurrentes que las técnicas de propósito general, pero tiene dificultades con el óptimo local.

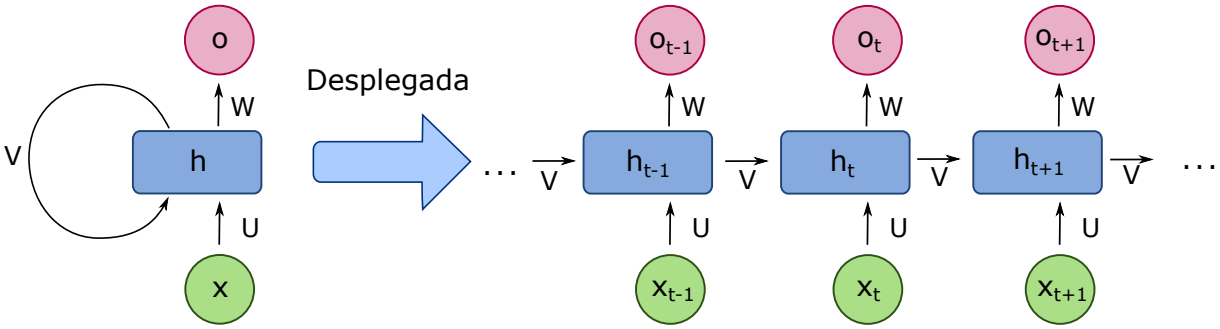


Figura 4.9: Red neuronal recurrente básica comprimida (izquierda) y desplegada (derecha).

Además, en lugar de utilizar una representación one-hot para las palabras, que es muy dispersa y requiere una gran cantidad de memoria, como entrada para una RNN se utilizan los word embeddings que permiten una forma densa de representar palabras o frases del lenguaje natural con vectores de números reales en un espacio vectorial de baja dimensión que se puede aprender a partir de los datos. Los word embeddings se construyen utilizando técnicas de aprendizaje automático que permiten asignar un vector a cada palabra en un corpus de texto y conceptualmente implica el encaje matemático de un espacio vectorial con las palabras del lenguaje natural a través del uso de algoritmos, lo que permite mejorar el descubrimiento de conocimiento, ya que, pueden capturar relaciones semánticas y sintácticas entre las palabras, mientras que el one-hot-encoding no puede hacerlo.

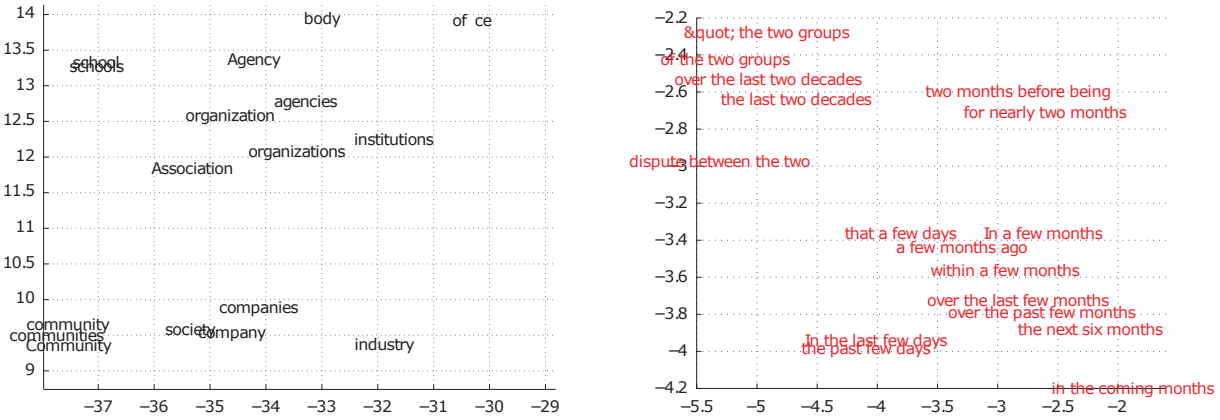


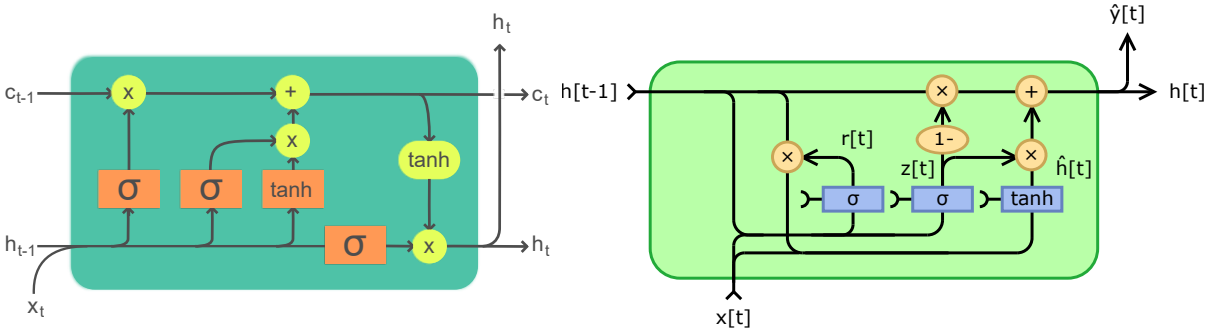
Figura 4.10: Visualización de vectores de palabras aprendidas. A la izquierda hay una ilustración de las representaciones de palabras para su visualización en 2D. A la derecha hay una representación 2D de frases aprendidas por una red neuronal recurrente de codificador-decodificador de inglés a francés. Se puede observar que palabras o secuencias de palabras semánticamente similares se asignan a representaciones cercanas [242].

Por otro lado, se han desarrollado celdas especiales como las unidades de memoria a largo plazo (LSTM) [73] y las unidades recurrentes cerradas (GRU) [260] que se utilizan para mejorar su capacidad de memoria a largo plazo y hacerlas más fáciles de entrenar. Estas dos celdas especiales buscan resolver el problema del gradiente de

fuga, donde el gradiente de la función de pérdida decae exponencialmente con el tiempo, lo que dificulta que las RNN puedan aprender dependencias a largo plazo.

Las unidades LSTM utilizan compuertas para controlar el flujo de información en la red, permitiendo que se elimine o agregue información a una "célula de memoria". Estas compuertas permiten un mejor control sobre el flujo de gradiente y la conservación de las "dependencias de largo plazo", permitiendo que la red olvide información irrelevantes o poco útil y conserve solo la información importante a medida que se procesan las secuencias de datos.

Las unidades GRU son una versión simplificada de las LSTM que utilizan un conjunto de puertas para controlar el flujo de información, pero no tienen celdas de memoria separadas como las LSTM. Esto las hace más fáciles de entrenar y más rápidas de ejecutar que las LSTM, aunque pueden ser menos precisas en algunas tareas, ya que tiene menos parámetros. En cuanto a la arquitectura, GRU tiene menos unidades internas que LSTM y utiliza dos puertas (puerta de actualización y puerta de reinicio) en lugar de tres (puerta de entrada, puerta de salida y puerta de olvido) como en LSTM.



(a) Celda *long short-term memory* (LSTM).

(b) Celda *gated recurrent unit* (GRU).

Figura 4.11: Principales variantes de de las redes neuronales recurrentes (RNN).

Finalmente añadimos unas últimas variantes como son las redes neuronales recurrentes bidirecionales [261] y los modelos de secuencia a secuencia (Seq2Seq). Los modelos bidirecionales procesan la entrada en ambas direcciones, hacia adelante y hacia atrás, esto permite que la red tenga acceso a información futura y pasada al mismo tiempo. Mientras que los modelos Seq2Seq vinculan dos RNN, un codificador y un decodificador. El codificador resume la entrada en una variable de contexto, también llamada estado. Luego, este contexto se decodifica y se genera la secuencia de salida.

4.3.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal artificial que se inspira en la organización y funcionamiento del cerebro humano y se utiliza principalmente para procesamiento de imágenes en tareas de visión por computadora, como la clasificación de imágenes, la detección de objetos y la segmentación de imágenes.

Aunque las CNN no son una réplica exacta del cerebro humano, tienen algunas similitudes en su estructura y funcionamiento. Por ejemplo, las CNN tienen capas de neuronas que procesan información de manera jerárquica y aprenden a reconocer patrones complejos en los datos de entrada como vemos en la Figura 4.12. De manera similar, el cerebro humano tiene diferentes regiones que procesan información de manera jerárquica y aprenden a reconocer patrones complejos en los estímulos sensoriales.

Además, las conexiones residuales se inspiran en la corteza visual del cerebro humano como vemos en la Figura 4.13 y se utilizan en las redes neuronales profundas para resolver el problema de la desaparición de los gradientes. En lugar de aprender una representación directa de la entrada, la red aprende una representación residual que se suma a la entrada para producir la salida. Esto permite que la red aprenda una representación más profunda y precisa de los datos para mejorar la precisión y la eficiencia de la red.

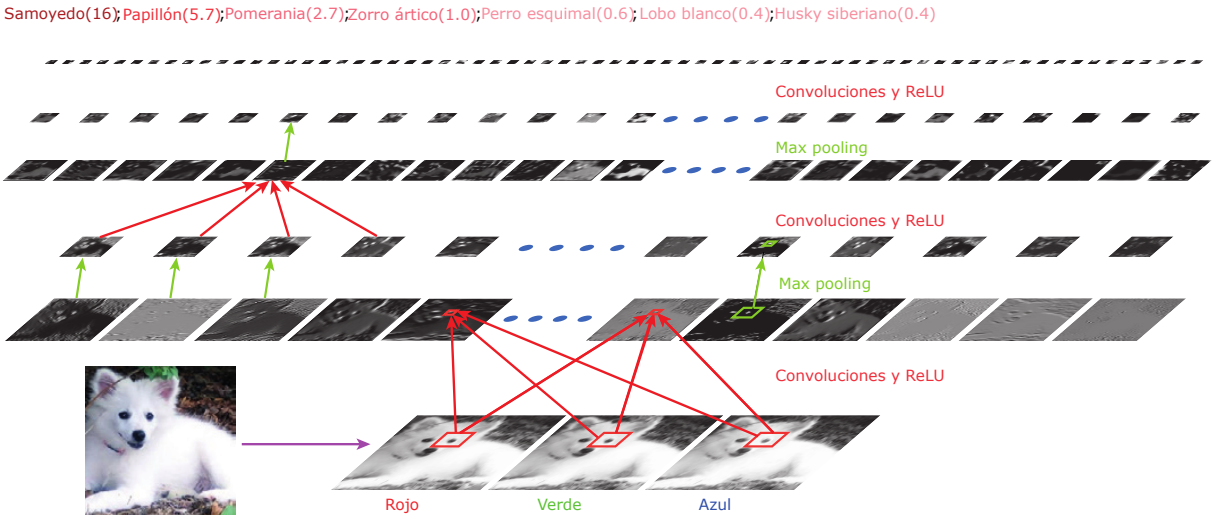
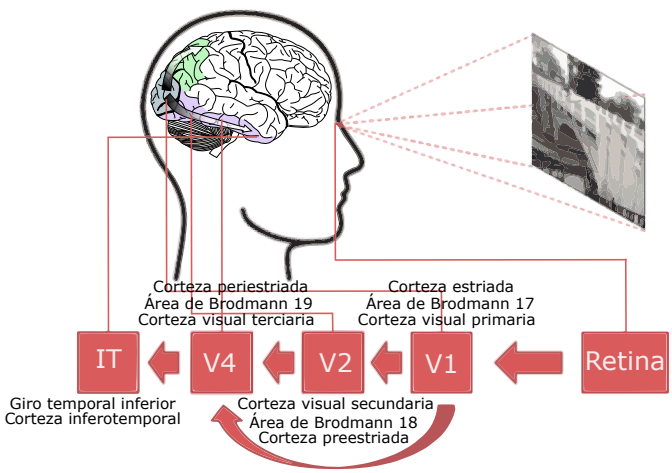
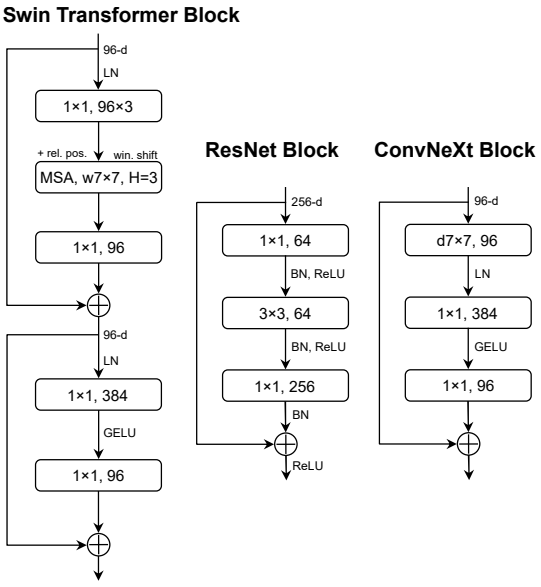


Figura 4.12: Dentro de una red convolucional cada imagen rectangular es un mapa de características que corresponde a la salida de una de las características aprendidas, detectada en cada una de las posiciones de la imagen. La información fluye de abajo hacia arriba, con características de nivel inferior que actúan como detectores de bordes orientados, y se calcula una puntuación para cada clase de imagen en la salida [242].



(a) La franja dorsal (verde) y la franja ventral (púrpura) tienen su origen en la corteza visual primaria del sistema de visión humano. La ruta ventral comienza en V1, va a través del área visual V2, luego a través del área visual V4, y a la corteza temporal inferior. La franja ventral es llamada a menudo como la «ruta qué», está asociada a la forma reconocimiento y representación del objeto, también está asociada con el almacenamiento de la memoria de largo término. V1: detección de bordes, etc. V2: Extrae propiedades visuales simples como la orientación, la frecuencia espacial o el color. V4: Detectar características de objetos de complejidad intermedia como formas geométricas simples. IT: Reconocimiento de objetos según el color y la forma del objeto y comparar esa información procesada con los recuerdos almacenados [262].



(b) Diseños de bloques para ResNet, Swin Transformer y ConvNeXt. El bloque de Swin Transformer es más sofisticado debido a la presencia de múltiples módulos especializados y dos conexiones residuales. Para simplificar, notamos las capas lineales en los bloques MLP del *Transformer* como "convoluciones 1x1" ya que son equivalentes [263].

Figura 4.13: El diseño de los bloques residuales está inspirado en que V4 recibe información directamente de V1 en la ruta ventral del visual humano.

Las CNN se construyen con una serie de capas diferentes que trabajan juntas para extraer características y patrones visuales de los datos de imagen. Algunas de las capas típicas que se utilizan en las CNN son la capa de entrada, capa de convolución, capa de *pooling* y capa completamente conectada. La capa de entrada recibe los

pixeles de la imagen de entrada conociendo el alto, ancho y profundidad que será 1 para blanco y negro o 3 para RGB.

Las capas de convolución son capas de neuronas que realizan operaciones matemáticas de convolución sobre los datos de entrada para extraer características y patrones visuales específicos en los datos de imagen. Procesará la salida de neuronas que están conectadas en "regiones locales" de entrada (es decir pixeles cercanos), calculando el producto escalar entre sus pesos (valor de pixel) y un *kernel* en una pequeña región a la que están conectados en el volumen de entrada. El volumen de salida será la cantidad de filtros que decidamos.

Los parámetros que definen una capa de convolución son el tamaño del kernel, el *stride* y el *zero padding*. El tamaño del kernel es el tamaño de la ventana que se desliza sobre la imagen de entrada para realizar la convolución. El *stride* es el número de píxeles que se mueve la ventana en cada paso. El *zero padding* es la cantidad de ceros que se agregan alrededor de los bordes de la imagen de entrada para que al realizar la convolución, no se eliminarán los bordes de la imagen original y adicionalmente el tamaño resultante será mayor que aquel obtenido sin el *padding*.

Para calcular el tamaño de la salida de la convolución de la figura 4.14 se utiliza la fórmula $\frac{n+2p-f}{s}+1$, donde n es el tamaño de la entrada, p es el *zero padding*, f es el tamaño del filtro y s es el *stride*. Por lo tanto, el tamaño de la salida de la convolución la calculamos sustituyendo los valores en la fórmula anterior $\frac{5+2*1-3}{2}+1 = {}^{4}/_{2}+1 = 3$ y vemos que sería una matriz bidimensional de 3x3.

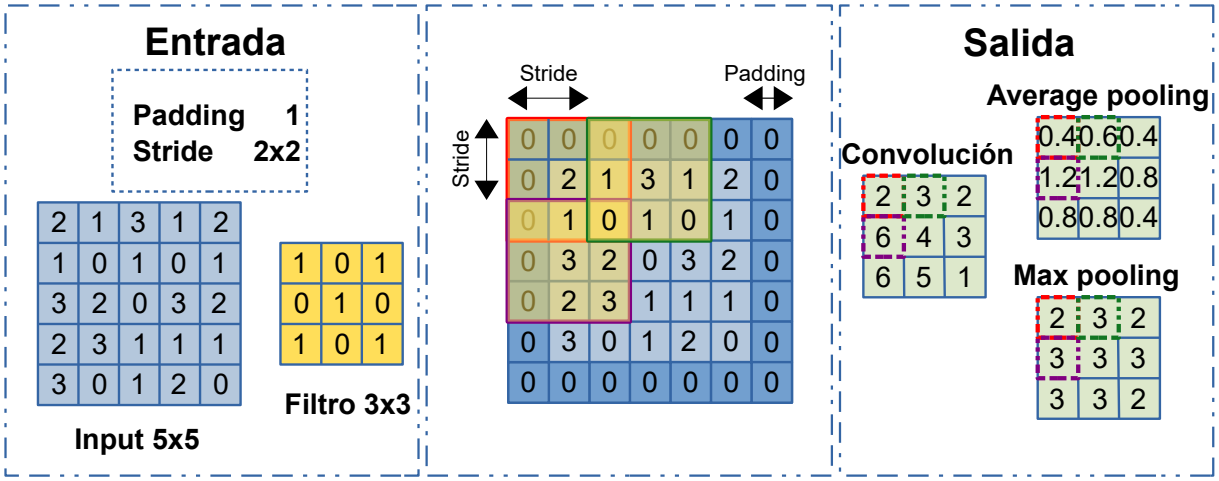


Figura 4.14: Operación de convolución, *average pooling* y *max pooling* en un caso donde la entrada es una matriz bidimensional de 5x5 con un tamaño de filtro de 3x3, un *stride* de 2x2 y un *zero padding* de 1 [47].

También se aplicará una función de activación en los elementos de la matriz como la ReLU o GELU. Las capas de *pooling* agrupan los valores de los datos de salida de las capas de convolución para reducir su dimensionalidad y hacer que la red sea más robusta ante pequeñas variaciones en los datos de entrada, haciendo una reducción en las dimensiones alto y ancho, pero se manteniendo la profundidad.

En la capa de *pooling*, se divide la imagen en regiones y se aplica una función de agregación a cada región. Los parámetros que definen una capa de *pooling* son los mismos que en la capa de convolución. La función de agregación puede ser una función promedio o una función máxima. La función promedio (*Average Pooling*) calcula el promedio de los valores en cada región, mientras que la función máxima (*Max Pooling*) calcula el valor máximo en cada región.

Finalmente se añaden varias capas completamente conectadas, estas capas son capas densas de neuronas formando una red de neuronas *feedforward* que reciben los datos de salida de las capas anteriores después de múltiples rondas de procesamiento de capa convolucional y capa de agrupación. Generalmente se usan de 1 a 2 capas completamente conectadas para dar el resultado de clasificación final, donde la última capa tendrá una cantidad de neuronas equivalente a las clases de objetos que queremos clasificar con una función de activación *softmax*.

Hay varios tipos diferentes de CNN que se utilizan en diferentes contextos y para resolver diferentes tipos de problemas. Las redes neuronales convolucionales tradicionales como hemos visto son modelos de CNN que

constan de una secuencia de capas de convolución, *pooling* y capas completamente conectadas, estas redes se utilizan comúnmente para tareas de clasificación de imágenes.

También tenemos las redes neuronales totalmente convolucionales (FCN), que son redes convolucionales sin capas completamente conectadas. La estructura de las FCN se asemeja a la de un autoencoder, donde el proceso de codificación de una imagen se realiza a través del uso de capas convolucionales y de pooling, que reducen la resolución de la imagen y extraen características.

Luego, estas características se procesan a través capas de convolución transpuesta (también conocidas como capas deconvolucionales) para generar una salida por píxel de la misma forma que la entrada original, en vez de que unas capas completamente conectadas que nos darían la clasificación de un objeto. Estas redes se utilizan comúnmente para tareas de segmentación de imágenes, ya que su arquitectura permite la generación de mapas de características de la misma resolución que la imagen de entrada.

Luego tenemos las redes neuronales convolucionales en cadena o *Chain Convolutional Neural Networks*, que son redes convolucionales que tienen una arquitectura de cadena en lugar de una arquitectura de grafo. En estas redes, cada capa convolucional está conectada solo a la capa inmediatamente anterior, lo que permite una propagación de información más eficiente y una menor complejidad computacional a diferencia de las redes neuronales convolucionales con saltos residuales que utilizan estos saltos residuales para mejorar la capacidad de la red para aprender representaciones complejas y abstractas de los datos de imagen.

Finalmente podemos ver las redes neuronales convolucionales atencionales o *Attentional Convolutional Neural Networks*, que son una variante de las CNN que utilizan una función de atención interna para asignar pesos a las características de la imagen, lo que les permite enfocarse en regiones específicas de la imagen de entrada y reducir el ruido en las características menos relevantes. Ademas de las redes neuronales convolucionales deformables o *Deformable Convolutional Neural Networks*, que son una variante de las CNN que utilizan módulos de deformación que permiten que la rejilla de convolución se ajuste deformando las operaciones convolucionales en función de la información de la imagen de entrada a diferencia de las CNN convencionales, que aplican convoluciones rígidas a una rejilla regular de píxeles.

4.3.4. Redes neuronales recurrentes convolucionales

Las redes neuronales recurrentes convolucionales son un tipo de algoritmo de aprendizaje profundo que ha revolucionado el campo de la inteligencia artificial en los últimos años. Estas redes se construyen con una secuencia de capas de convolución y pooling que procesan los datos de imagen, seguidas de una o más capas de RNN que procesan las secuencias de datos. Este tipo de redes que combinan características de redes CNN y RNN se conocen comúnmente como redes neuronales híbridas o mixtas. En general, todos estos términos se refieren a la misma idea de combinar diferentes tipos de redes neuronales para obtener modelos más complejos y potentes.

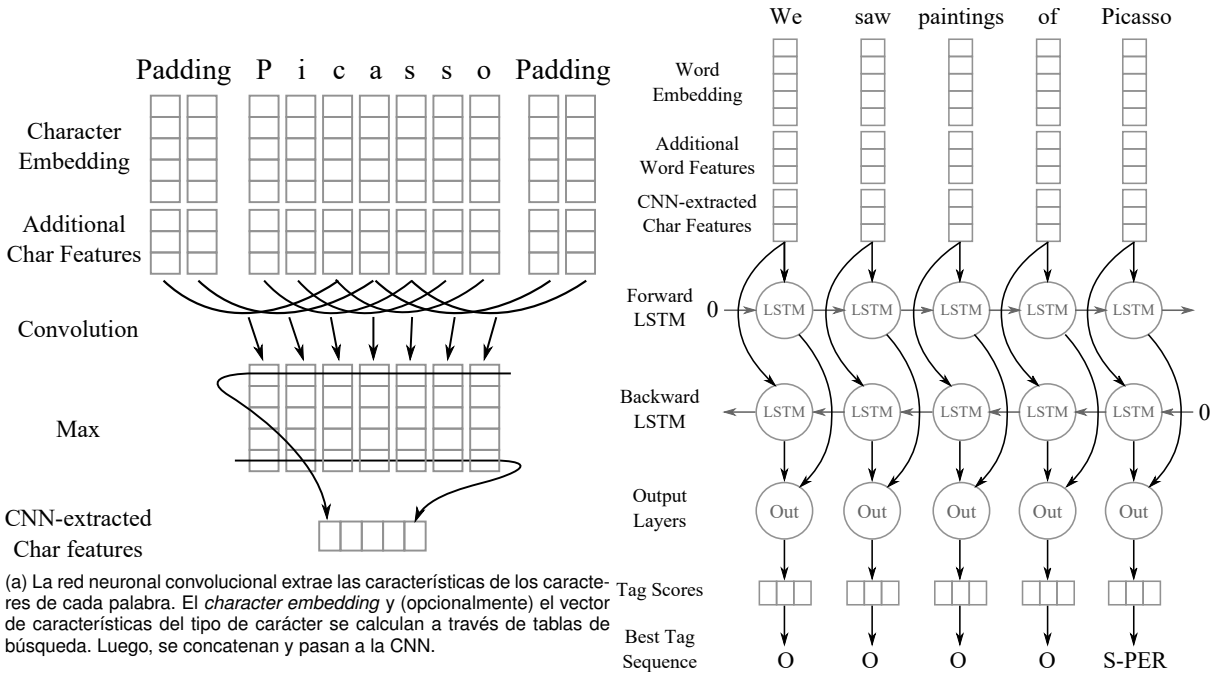
Las redes CNN se caracterizan por su capacidad para extraer características de imágenes y utilizarlas para realizar inferencias. Esto se logra mediante el uso de capas de convolución, que son capas especiales diseñadas para extraer patrones de características de las imágenes de entrada. Las redes RNN, por su parte, son adecuadas para trabajar con secuencias de datos de cualquier tipo y se caracterizan por su capacidad de recordar información a largo plazo. Esto se logra mediante el uso de unidades recurrentes, que son capas especiales que permiten que la red mantenga un estado interno que le permite recordar información a largo plazo.

Al combinar estas dos arquitecturas, es posible obtener modelos que sean capaces de extraer características de imágenes y utilizarlas para realizar inferencias sobre secuencias de datos. Estos modelos combinados se utilizan comúnmente en tareas que requieren la comprensión de contexto y la dependencia temporal, como la generación de texto a partir de imágenes o el procesamiento de audio y video.

Existen diferentes tipos de redes que combinan las características de las redes CNN y RNN capaces de procesar tanto datos de imagen como secuencias de datos. Algunos de los tipos más comunes son las redes neuronales convolucionales con capas GRU (C-GRU), las redes neuronales convolucionales con capas LSTM (C-LSTM) y las redes neuronales convolucionales con capas LSTM bidireccionales (C-BiLSTM).

Las C-GRU constan de una secuencia de capas de convolución y pooling que procesan los datos de imagen, seguidas de una o más capas RNN con celdas GRU que procesan las secuencias de datos. Mientas que las C-LSTM en vez de utilizar celdas GRU usa celdas LSTM y las C-BiLSTM utiliza celdas BiLSTM en las que a diferencia de la celda LSTM se toma en cuenta tanto las secuencias de datos en el orden original como en el orden

inverso, procesando la secuencia de entrada en ambas direcciones, lo que puede mejorar su capacidad para realizar inferencias sobre la secuencia.



(a) La red neuronal convolucional extrae las características de los caracteres de cada palabra. El *character embedding* y (opcionalmente) el vector de características del tipo de carácter se calculan a través de tablas de búsqueda. Luego, se concatenan y pasan a la CNN.

(b) Red C-BiLSTM desenrollada.

Figura 4.15: El C-BiLSTM (desenrollado) para etiquetar entidades con nombre. Múltiples tablas buscan vectores de características a nivel de palabra. La CNN extrae un vector de características de longitud fija de las características a nivel de carácter. Para cada palabra, estos vectores se concatenan y alimentan a la red BiLSTM y luego a las capas de salida [264].

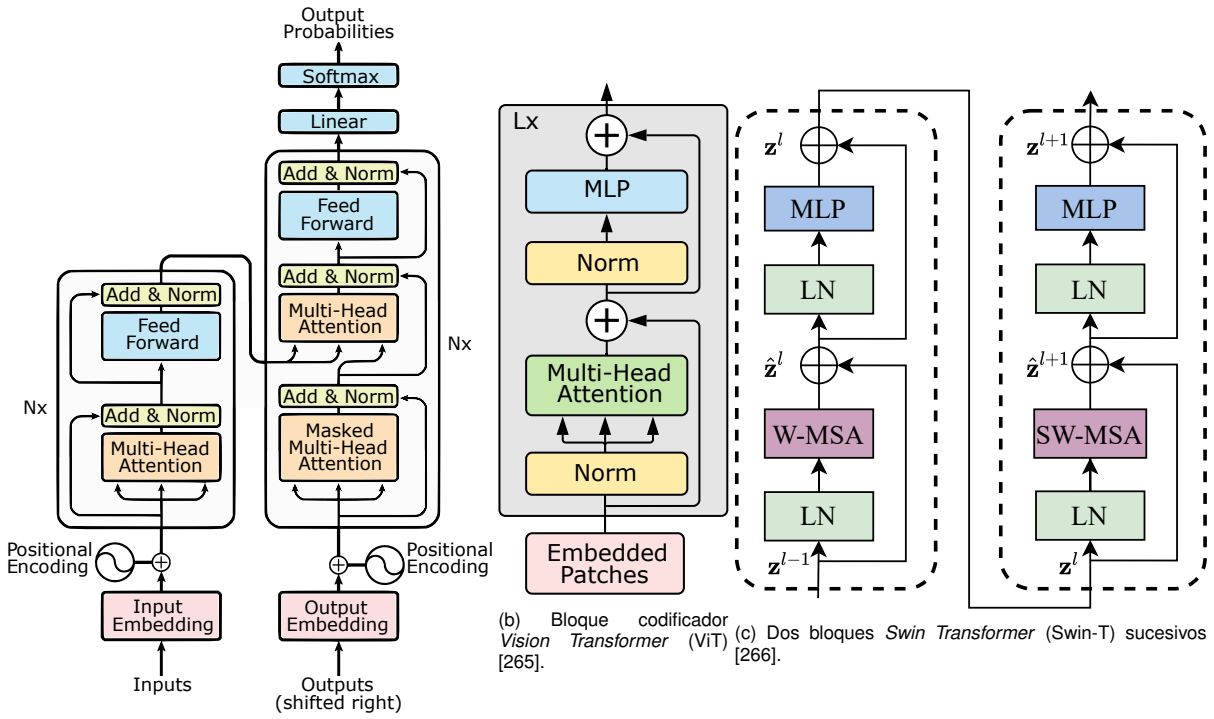
4.3.5. *Transformers*

El *transformer* es un modelo de aprendizaje profundo desarrollado en 2017 por el grupo de investigación en aprendizaje automático de Google Research. Se caracteriza por utilizar una estructura de atención y ha demostrado buen rendimiento en tareas de procesamiento del lenguaje natural, como la traducción automática o la generación de texto.

Una de las principales ventajas de la arquitectura *transformer* es que permite a la red procesar secuencias de entrada en paralelo, lo que reduce el tiempo de entrenamiento y mejora el rendimiento en comparación con las arquitecturas RNN que procesan secuencias de entrada de manera secuencial. Además, la estructura de atención utilizada en la arquitectura *transformer* permite que la red tenga una memoria a largo plazo y sea capaz de realizar inferencias sobre secuencias de entrada de longitud variable.

Recientemente se han aplicado a la visión por computadora, sustituyendo las convoluciones, por una combinación de capas de atención y capas completamente conectadas para procesar las imágenes. En tareas de visión por computadora como la clasificación de imágenes y la detección de objetos han demostrado ser muy eficaces, ya que al basarse en la atención, permite a la red centrarse en partes específicas de la entrada para realizar una tarea determinada. Además, los transformadores son capaces de manejar entradas de longitud variable, lo que los hace ideales para tareas como el procesamiento de videos.

Un ejemplo es la arquitectura Swin Transformer (Swin-T) que es una versión mejorada del Vision Transformer (ViT) y utiliza una auto-atención multi-cabeza basada en ventanas (W-MSA) y una auto-atención multi-cabeza basada en ventanas desplazadas (SW-MSA) en lugar de las convoluciones tradicionales. Ademas utiliza capas de normalización (LN) para mejorar la eficiencia de las capas de auto-atención multi-cabeza (MSA).



(a) Arquitectura del modelo *Transformer* [142].

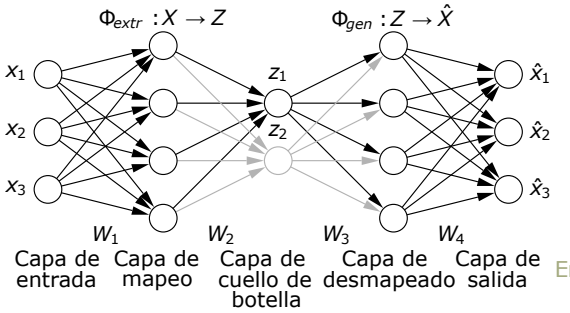
(b) Bloque codificador *Vision Transformer* (ViT) [265].

(c) Dos bloques *Swin Transformer* (Swin-T) sucesivos [266].

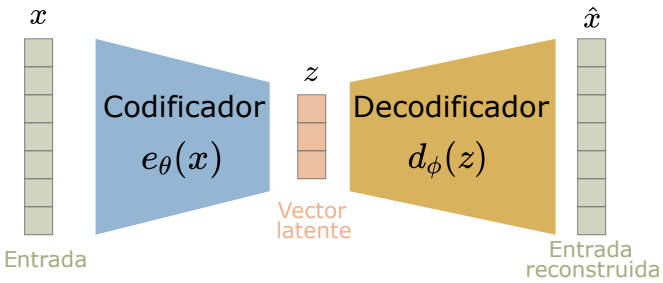
Figura 4.16: Modelo *Transformer* junto con dos bloques de *Transformers* utilizados en visión por computadora.

4.3.6. *Autoencoders* y modelos generativos

Los *autoencoders* (AE) aprenden a reconstruir sus propias entradas, de esta forma, la red neuronal puede extraer características relevantes de los datos y comprimirlos en una representación compacta y eficiente de los datos con menor dimensión. Esta representación se llama código latente y puede ser utilizada para generar nuevos datos similares a los originales. Estos modelos constan de dos partes: un *encoder* que codifica los datos de entrada en una representación compacta, y un *decoder* que decodifica la representación compacta en datos de salida.



(a) Arquitectura básica de un *autoencoder* [267].



(b) Diagrama de bloques de un *autoencoder* [251].

Figura 4.17: *Autoencoder*.

Este tipo de redes se entrenan utilizando una técnica de aprendizaje no supervisado, lo que significa que no se proporciona ninguna etiqueta de clase o información de salida deseada. En su lugar, se utiliza la función de pérdida de la Ecuación 4.38 que mide la similitud entre los datos de entrada y salida para guiar el proceso de entrenamiento.

$$\textit{Pérdida} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(e_\theta(x))\|_2 \tag{4.38}$$

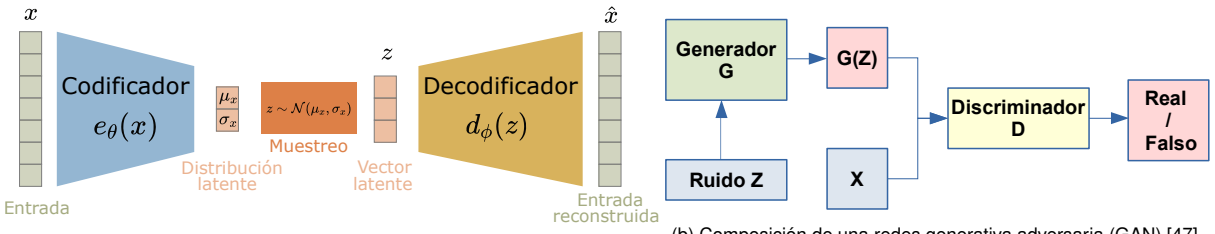
Comúnmente se utilizan para tareas de reducción de dimensionalidad y extracción de características, y pueden ser utilizados como piezas fundamentales en otros tipos de modelos de redes neuronales. Hay varios tipos de *autoencoders*, incluyendo los *autoencoders* lineales, los *autoencoders* esparcidos, los *autoencoders* denoising, los *autoencoders* variacionales y los *autoencoders* adversariales. Cada tipo de *autoencoder* tiene una arquitectura diferente y se utiliza para diferentes aplicaciones.

Los *autoencoders* lineales son el tipo más simple de *autoencoder* y se utilizan para la reducción de la dimensionalidad. Los *autoencoders* esparcidos son una variante de los *autoencoders* lineales que se utilizan para la reducción de la dimensionalidad en datos dispersos. Los *autoencoders denoising* son una variante de los *autoencoders* que se utilizan para eliminar el ruido de los datos. Los *autoencoders* variacionales (VAE) son una variante de los *autoencoders* que se utilizan para generar nuevos datos a partir de un conjunto existente, utilizando técnicas de inferencia bayesiana. Los *autoencoders* adversariales (AAE) son una técnica de aprendizaje profundo que combina los *autoencoders* con las redes generativas adversarias (GAN) [268] y se utilizan en aplicaciones como la clasificación semisupervisada, la reducción de la dimensionalidad y la visualización de datos.

Los modelos generativos son una clase de modelos que se utilizan para generar datos nuevos a partir de un conjunto de datos existente. Estos modelos permiten a una máquina "aprender" los patrones que existen en los datos con los que son entrenados y a partir de dicho aprendizaje, son capaces de generar datos similares que en algunos casos pueden ser casi tan "reales" como los que se utilizaron inicialmente para su entrenamiento. Por ejemplo, un modelo generativo podría crear nuevas imágenes de rostros humanos o nuevos textos a partir de un corpus. Una forma de crear modelos generativos es utilizando autoencoders. Para ello, se entrena un autoencoder con los datos de interés y luego se utiliza el código latente como entrada para generar nuevos datos. El código latente puede ser modificado o muestreado aleatoriamente para obtener diferentes resultados.

Los VAE son el modelo generativo más común desarrollado a partir del AE y utilizan una función de pérdida más compleja que los AE. En este caso hay la función de pérdida es la suma de la pérdida de reconstrucción y la pérdida de similitud. La pérdida de reconstrucción que vemos en la Ecuación 4.39 mide la diferencia entre la entrada original y la salida reconstruida del VAE, donde $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. La pérdida de similitud mide la similitud entre la distribución latente de la entrada original y la distribución latente de la salida reconstruida del VAE mediante la divergencia KL expresada con la formula $D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \| \mathcal{N}(\mathbf{0}, \mathbf{I})))$ para asegurarse de que ambas distribuciones sean similares.

$$\textit{Pérdida de reconstrucción} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2 \tag{4.39}$$



(a) Diagrama de bloques de un *autoencoder* variacional [251].

(b) Composición de una redes generativa adversaria (GAN) [47].

Figura 4.18: Redes generativas más importantes.

Las redes neuronales generativas adversarias (GAN) [268] son un tipo de red neuronal que consta de dos partes: un generador y un discriminador. El generador toma un conjunto de números aleatorios como entrada y genera datos sintéticos que se parecen a los datos reales. El discriminador, por otro lado, toma datos reales y sintéticos como entrada y trata de distinguir entre ellos. El objetivo del generador es engañar al discriminador para que crea que los datos sintéticos son reales. A diferencia de los VAE, las GAN no requieren una función de pérdida explícita y pueden generar muestras más realistas.

4.3.7. Aprendizaje por transferencia

El aprendizaje por transferencia o *transfer learning* (TL) es una técnica de aprendizaje automático que se utiliza para transferir el conocimiento de un sistema de aprendizaje a otro. Habitualmente una red neuronal ya pre-entrenada con una gran cantidad de datos que es capaz de resolver un determinado problema transfiere dicho conocimiento a otra red neuronal que busca resolver otro problema diferente pero relacionado para mejorar el rendimiento en dicha tarea.

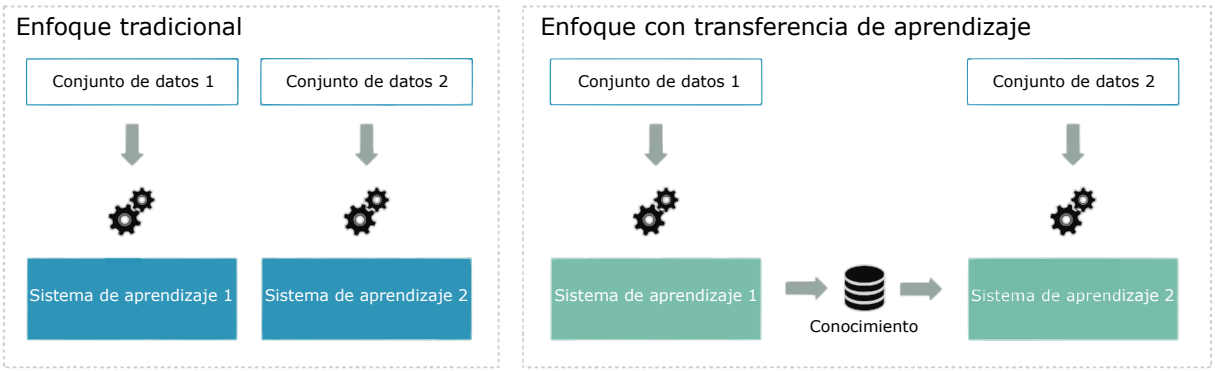


Figura 4.19: Enfoque tradicional vs. enfoque de *Transfer Learning*.

Es una técnica muy útil cuando se dispone de pocos datos para entrenar un modelo desde cero, y en tareas de visión por computadora y procesamiento de lenguaje natural como el análisis de sentimientos debido a la gran cantidad de potencia computacional requerida. Por lo que ha tenido un gran éxito con el crecimiento del *deep learning*, ya que en lugar de necesitar entrenar una red neuronal desde cero, que implica precisar de una gran cantidad de datos y requerir mucho tiempo (días o semanas) de computación para entrenar, mediante el TL se puede partir desde una red pre-entrenada.

Según el dominio, la tarea y los datos que tengamos, podemos aplicar diferentes estrategias y técnicas de TL. Estas se pueden agrupar en tres categorías, según los algoritmos tradicionales de ML que se usen:

- Transferencia de aprendizaje inductiva (*inductive TL*): utilizando un modelo pre-entrenado en un campo similar pero para una tarea diferente como punto de partida y ajustándolo para adaptarse a la nueva tarea. Por ejemplo, un modelo entrenado para reconocer animales en imágenes se podría utilizar como punto de partida para entrenar un modelo para reconocer perros en imágenes.
- Transferencia de aprendizaje no supervisado (*unsupervised TL*): utilizando un modelo pre-entrenado en un campo similar pero para una tarea diferente, sin necesidad de datos etiquetados, para entrenar un modelo y posteriormente ajustalo para la tarea de interés. Por ejemplo, el Self-taught clustering es un método que permite realizar el clustering de pequeñas colecciones de datos objetivo no etiquetados, con la ayuda de una gran cantidad de datos fuente no etiquetados.
- Transferencia de aprendizaje transductivo (*transductive TL*): utilizando un modelo pre-entrenado en un campo similar pero para una tarea diferente en términos de datos o de distribuciones de probabilidad marginales. En lugar de ajustar el modelo previo para la tarea de interés, se utiliza para mejorar el modelo proporcionando información adicional en una tarea específica. Por ejemplo, un modelo previamente entrenado y testeado en una tarea de etiquetado morfosintáctico de palabras o etiquetado gramatical (*POS Tagger*) con datos de actualidad como los del Wall Street Journal se podría adaptar a datos extraídos de redes sociales, cuyo contenido es diferente pero se asemeja al de los periódicos.

De la pregunta "¿qué transferir?" y las técnicas que se han mencionado antes, se derivan cuatro enfoques. Estos enfoques nos ayudan a analizar los aspectos específicos de cada problema y a usar las metodologías más apropiadas:

- Transferencia de instancia: utilizando ejemplos específicos de la tarea fuente para entrenar un modelo para una tarea relacionada. Es decir, se utilizan instancias o ejemplos de un conjunto de datos para entrenar un modelo para otra tarea relacionada. Este enfoque puede ser útil cuando hay una superposición significativa entre las características de la tarea de origen y las de la tarea de destino.
- Transferencia de representación de características: extrayendo características relevantes de la tarea fuente y transfiriéndolas al modelo para la tarea objetivo. Por lo tanto, el modelo de la tarea de destino aprende de las características importantes que se han extraído previamente en la tarea de origen. Este enfoque puede ser útil cuando las características importantes de la tarea de origen son relevantes para la tarea de destino.
- Transferencia de parámetros: transfiriendo los parámetros de un modelo pre-entrenado de la tarea fuente al modelo para la tarea objetivo. Los parámetros e hiperparámetros de los modelos incluyen pesos de las capas de una red neuronal, vectores de soporte en SVM, etc. Si las tareas son similares, es probable que los parámetros del modelo de origen sean útiles en el modelo de destino.
- Transferencia de conocimiento relacional: transfiriendo el conocimiento relacionado con las relaciones entre instancias en la tarea fuente a la tarea objetivo. Por ejemplo, si el conocimiento adquirido en la tarea de origen incluye información sobre la relación entre diferentes objetos, esta información se puede transferir a la tarea de destino para ayudar al modelo a aprender las relaciones relevantes.

Cuando se utiliza un modelo pre-entrenado, el primer paso es quitar el clasificador que tiene por defecto. Después se le añade uno nuevo que se ajuste a los objetivos propios, y por último se hacen ajustes siguiendo una de estas estrategias:

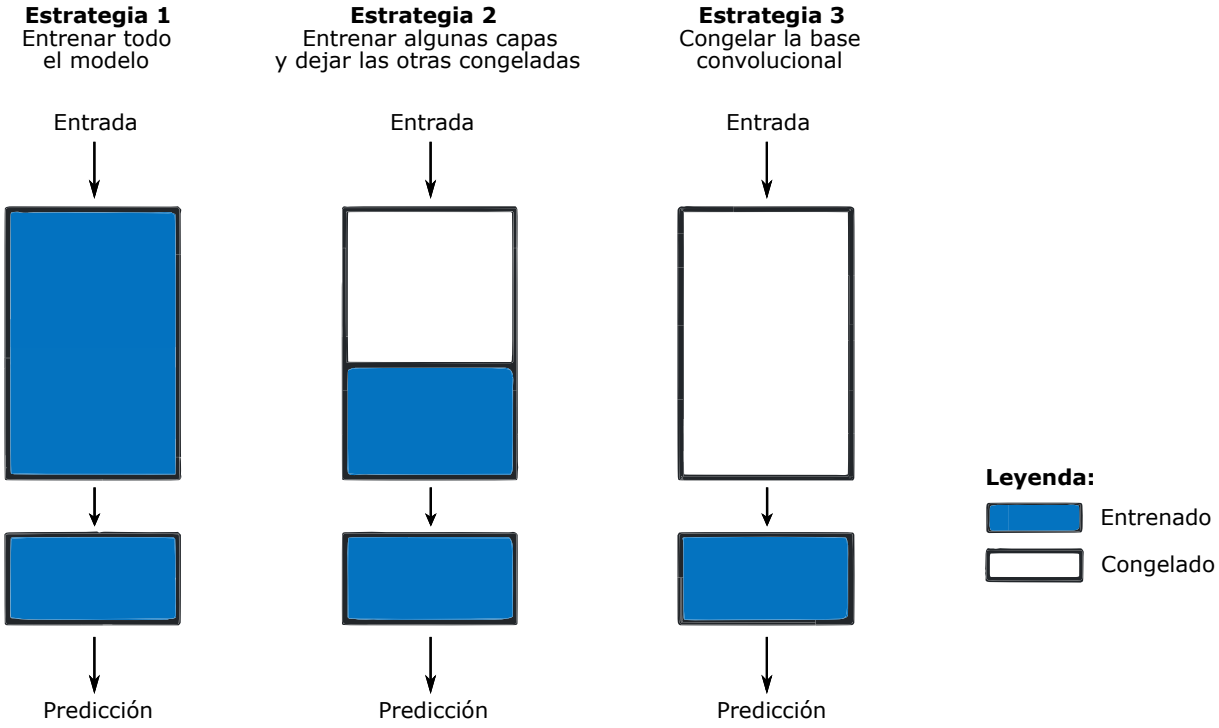


Figura 4.20: Estrategias de ajuste para modelos pre-entrenados de CNN mediante *transfer learning* [269].

- Estrategia 1: El modelo completo se somete a entrenamiento. Se emplea la misma arquitectura que el modelo pre-entrenado y se entrena con su conjunto de datos. El modelo parte de cero, así que hará falta un conjunto de datos amplio y mucha capacidad de cómputo.

- Estrategia 2: En lugar de entrenar toda la red, se realiza un ajuste fino (*fine tuning*). En este proceso los pesos de las capas inferiores que captan características generales se congelan y solo se ajustan los pesos de las capas superiores que características específicas para adaptarse a la nueva tarea. Por lo general, si los datos son pocos y los parámetros muchos, se congelan más capas, aunque si los datos son muchos y los parámetros pocos, se pueden entrenar más capas para la nueva tarea.
- Estrategia 3: Un extremo del balance entrenamiento/congelación denominado extracción de características (*feature extraction*) es el caso de mantener la base convolucional intacta con los pesos ya optimizados. La idea principal es usar la base convolucional tal como está para extraer características interesantes de nuevos datos que serán procesadas a través de un nuevo clasificador entrenado desde cero. Esta técnica puede ser útil si se tiene poca capacidad de cómputo, el conjunto de datos es pequeño y/o el problema que se quiere resolver es muy parecido al que resuelve el modelo pre-entrenado.

Finalmente podemos ver que hay dos enfoques posibles para abordar el aprendizaje por transferencia en el aprendizaje profundo, el enfoque de modelado y el enfoque pre-entrenado. El enfoque de modelado implica desarrollar un modelo para resolver una tarea y luego reutilizarlo para resolver otra tarea relacionada, mientras que el enfoque pre-entrenado implica utilizar un modelo previamente entrenado y adaptarlo a la tarea en cuestión. Ambos enfoques pueden ser efectivos según las circunstancias y los datos disponibles.

En el caso de elegir el enfoque pre-entrenado para el reconocimiento y clasificación de imágenes, se pueden utilizar modelos como VGG, ResNet e Inception [270]. Mientras que para audio, se puede utilizar una red neuronal pre-entrenada en la tarea de clasificación de audio como VGGish [271].

Capítulo 5

ESTADO DEL ARTE

5.1. Detección de tos utilizando *smartphones*

Los sistemas actuales utilizados para monitorizar la tos procesan la señal de audio mediante algoritmos de inteligencia artificial basados generalmente en el análisis de características extraídas de ventanas de señal. Existen propuestas que presentan valores aceptables de sensibilidad y especificidad [13], [272]-[274] en la detección. La mayor parte de estos sistemas implementan un módulo de reconocimiento de patrones que clasifica una serie de características en el dominio temporal y/o espectral extraídas de señales de audio. Entre los conjuntos de características más comunes se encuentran aquéllos típicamente utilizados en análisis de señales de voz como los MFCC (*Mel Frequency Cepstral Coefficients*) [275] y los GTCC (*GammaTone Cepstral Coefficients*) [276]. Estos coeficientes espectrales se basan en un análisis logarítmico del espectro de la señal para emular el funcionamiento del sistema auditivo humano.

La mayoría de los detectores de tos propuestos adolecen de algunas limitaciones que los hacen inadecuados para la monitorización en situaciones de la vida real. Algunos se basan en complejos sistemas de grabación (micrófonos de bajo ruido, preamplificadores, etc.) y sólo se han probado en entornos tranquilos y controlados [274]. Otros se centran en una población muy específica [277], [278] (bebés, pacientes con una patología, etc.) y, por tanto, presentan falta de generalización. Por otra parte, algunos métodos fueron concebidos para resolver un problema más amplio que la detección de la tos [279] y no logran un rendimiento óptimo. Por último, algunos enfoques no se han diseñado con la eficiencia en mente (grandes conjuntos de características o muchos clasificadores [280] algoritmos iterativos [281], etc.) y pueden no ser aconsejables en situaciones de tiempo real. Además de las limitaciones mencionadas, estos sistemas a medida pueden considerarse caros e incómodos (es decir, no se pueden llevar durante la actividad diaria) en un momento en que la telemedicina se ha orientado hacia sensores genéricos genéricos y fácilmente disponibles.

Los avances en la tecnología de teléfonos inteligentes o *smartphones* permitirían, emplear estos dispositivos de uso diario como sistemas inteligentes para la monitorización de la tos, ya que llevan incorporados una serie de sensores capaces de capturar los sonidos correspondientes a eventos de tos y los movimientos relacionados con ella. Además, la capacidad computacional de estos dispositivos es cada vez mayor y, al mismo tiempo, cuentan con conectividad en tiempo real para descargar operaciones complejas a sistemas de mayor rendimiento. Las soluciones *mhealth* (aplicaciones móviles para la salud) ya exploradas para la detección y el análisis de tos no permiten su uso en tiempo real para monitorización. Algunas, como [22], o las concebidas a partir de los trabajos en [282]-[284] (ver [285]) funcionan de forma que el usuario ha de sostener el dispositivo, ejecutar la app y toser de forma que el móvil sólo adquiera el sonido de la tos con bajo nivel de ruido ambiente. Si el usuario pudiese llevar el dispositivo en el bolsillo o bolso de mano, con la app funcionando en segundo plano con funcionalidad total, el impacto en su vida diaria sería mínimo, reduciendo la percepción que el paciente pudiera tener de la medicalización de su vida. Una solución de este tipo involucra retos importantes a la hora de aprovechar la funcionalidad de los *smartphones* para su uso como dispositivos médicos [286]:

- El primer reto reside en la necesidad de realizar la detección y el análisis de audio en entornos en los que el

ruido de fondo es alto, ya sea por el propio ruido ambiente, o incluso por el roce de la tela del bolsillo o el bolso con el micrófono, que produciría un ruido de fricción de más potencia que la señal de interés.

- El segundo se relaciona con la necesidad de mantener el resto de funcionalidades del dispositivo sin que la aplicación de monitorización las comprometa. En este sentido, ningún usuario va a querer utilizar una aplicación que reduzca la vida útil de la batería a 2 o 3 horas. La complejidad de las operaciones necesarias para la detección y el análisis de la tos debe por tanto ser tenida en cuenta.

Las primeras propuestas basadas en técnicas de aprendizaje profundo o *deep learning* aplicadas al problema de la detección de tos realizan extracción automática de características mediante redes neuronales convolucionales (CNN) [287], [288] y recurrentes (RNN) [288] profundas, si bien el punto de partida es generalmente la representación 2D en tiempo-frecuencia de las señales de audio, bien de forma directa o transformada a un espacio de coeficientes perceptivos como los MFCCs. Los resultados de estos trabajos resultaron prometedores y en muchos casos superaban a métodos clásicos en los que el conjunto de características estaba predefinido. Sin embargo, tal y como demostraron en [31], estas técnicas presentan un problema de generalización motivado en parte por la necesidad de bases de datos de gran tamaño para su entrenamiento. Además, la complejidad de este tipo de sistemas es bastante grande, existiendo numerosas posibilidades de optimización para estas redes que pueden aprovecharse de cara a hacer el análisis en tiempo real en el *smartphone* viable sin consumir la batería en pocas horas. En la mayoría de los casos estos métodos requieren utilizar GPUs para desplegarlos. En el caso de los más modernos *smartphones* es posible utilizar su GPU, pero con consumos muy elevados, por lo que la optimización en este caso resulta más importante si cabe.

En los dos últimos años, ha habido una explosión del uso de métodos basados en *deep learning* para resolver el problema de la detección de tos y su análisis con fines diagnósticos [289]-[291]. Esto se ha debido fundamentalmente a la necesidad de contar con herramientas diagnósticas y de cribado útiles para la COVID-19. Los métodos presentados hasta la fecha presentan en general buenos resultados, tanto a nivel de detección [289], [290], como de diagnóstico de patologías, en particular para la COVID-19 [290], [291]. Una de las razones de estos buenos resultados es la disponibilidad de bases de datos de sonidos de tos de gran tamaño y diversidad que se han adquirido de forma colaborativa *crowd-sourcing* durante la pandemia. Ejemplos de estas bases de datos son [292]-[294], que cuentan con toses voluntarias forzadas de pacientes con COVID-19 y otras patologías diferentes, además de toses no asociadas a patología específica (tabaquismo). Sin embargo, la aplicabilidad de los métodos entrenados con este tipo de bases de datos al contexto de la monitorización continua con alto ruido de fondo no es inmediata. Es necesario tener en cuenta que existen grandes diferencias entre los patrones de sonido de las toses espontáneas y las forzadas. Además, el entorno de grabación de las bases de datos colaborativas da lugar generalmente a sonidos limpios, sin la contaminación ruidosa que se espera de un *smartphone* transportado en el bolsillo o bolso de mano.

En el caso particular de detección de tos, además de los ya mencionados [287], [288], destacan los trabajos presentado recientemente en [289], [290]. En [289] se propone el uso de una red mixta C-BiLSTM que emplea capas convolucionales y BiLSTM (Bidirectional Long Short Term Memory) para procesar patrones MFCC normalizados, consiguiendo una sensibilidad en detección del 84 % con alta especificidad (99 %). En este caso, la evaluación del método se realiza sobre una base de datos realista con alta contaminación acústica. Sin embargo, el hecho de contar con un conjunto desbalanceado de patrones (muchos más sonidos correspondientes a eventos que no corresponden a episodios de tos), hace que el valor positivo predictivo del sistema sea claramente mejorable. En [290] se utiliza la red YAMNet [295] de clasificación de audio sobre MobileNet [227] para la detección de eventos de tos a partir de ventanas tiempo-frecuencia sobre patrones MFCC en escala logarítmica. La evaluación sin embargo se realiza sobre bases de datos colaborativas con toses voluntarias sin apenas contaminación acústica, reportándose únicamente resultados sobre diagnóstico COVID-19.

5.1.1. Evaluación en un entorno real de una aplicación de monitorización de tos entrenable para teléfonos inteligentes

En septiembre de 2019, Carlos Hoyos-Barceló y colaboradores [296], presentamos *SmartCough* [297]-[299], una aplicación de M-health para teléfonos inteligentes Android que monitoriza las tendencias de tos en pacientes con enfermedades respiratorias. La aplicación está diseñada para optimizar eficientemente el uso de la batería, funcionar con rapidez y ser robusta frente al ruido. Se basa en algoritmos de aprendizaje automático implementados

de manera eficiente y ha sido validada en condiciones de laboratorio. Sin embargo, en una situación real en la que el usuario lleva el teléfono en el bolsillo o en el bolso rara vez se cumplen las condiciones de laboratorio. Por lo tanto, la aplicación cuenta con un módulo de auto-formación que permite una fácil adaptación a nuevos entornos.

SmartCough primero realiza un preporcesado de la señal en el que el audio del micrófono es dividido en tramas de 50 ms con una superposición de 25 ms y luego se calcula su espectrograma. Con el objetivo de mejorar la eficiencia, se aplica un filtro para detectar y descartar fragmentos de datos sin incidentes. Este filtro se basa en un umbral mínimo de energía y se ajusta de manera dinámica en función de la energía del ruido de fondo, con el objetivo de adaptarse rápidamente a entornos cambiantes.

En la extracción de características, para cada espectrograma, se calculan de manera eficiente un conjunto de características basadas en momentos de imagen que se utilizan para identificar ventanas de tos [300]. Para la clasificación, se utiliza un clasificador K-NN eficiente para detectar eventos de tos [301]. La elección de K-NN permite que el sistema aprenda agregando nuevas muestras a la base de datos.

La eficiencia es de capital importancia por este motivo el clasificador K-NN es un motor de aprendizaje automático optimizado para la detección de tos que podría funcionar en tiempo real incluso en dispositivos de gama baja [301] y la extracción de características fue optimizada en [300] para hacer que la aplicación sea compatible con la batería a fin de admitir más de tres días de operación continua. De forma que para la implementación en la vida real no se agote la batería del dispositivo demasiado rápido para un uso práctico y pueda procesar los eventos de tos en el momento en que ocurren.

La base de datos de prueba está compuesta por grabaciones de audio de más de 24 horas de 20 pacientes con diferentes afecciones respiratorias. Se utilizó un dispositivo Android Sony Xperia Z2 para recopilar los datos, almacenando los archivos en formato WAV de 16 bits a 44,1 kHz. Los pacientes fueron instruidos para llevar el dispositivo dentro de sus bolsillos o bolsos durante un día completo y llevar a cabo sus actividades cotidianas, con el fin de obtener muestras de entornos reales y ruidosos que son hostiles a la detección de la tos.

Mediante la herramienta de software Praat [43] creamos un archivo de subtítulos para etiquetar manualmente los sonidos de tos, usamos este archivo de subtítulos y Praat para crear clips de audio separados para cada evento de sonido. Luego pasamos los clips de audio a nuestra aplicación para probar el detector de tos, entrenado con su base de datos predeterminada, y calculamos la sensibilidad y la tasa de falsos positivos(FPR), medido como falsos positivos por hora. La base de datos predeterminada se compone de datos de pacientes de nuestra validación interior anterior [29], aumentada con una colección de clips de audio SFX de Youtube de fuentes comunes de ruido.

Cabe destacar que previamente, se evaluaron los algoritmos implementados en SmartCough en una base de datos sintética [302] y se obtuvo un buen desempeño en una variedad de condiciones de ruido. Posteriormente, se entrenó el sistema con una base de datos de pacientes reales registrados en un entorno clínico, también con buenos resultados en la evaluación [29]. Finalmente en este estudio, los investigadores evaluamos la aplicación con pacientes reales en un entorno al aire libre para probar el rendimiento en entornos reales que son hostiles a la detección de tos.

Los resultados mostraron que la sensibilidad promedio obtenida en condiciones de laboratorio cae significativamente (hasta un 60 %) cuando se utiliza la configuración de línea base. Sin embargo, al activar el módulo de auto-formación integrado, la sensibilidad media aumenta al 85,87 % después de un pequeño paso de formación, con una tasa limitada de falsos positivos (FPR). El rendimiento alcanzado es análogo al obtenido en condiciones de laboratorio, lo que hace que la aplicación sea adecuada para su uso en escenarios de la vida real.

5.2. Segmentación del audio de la tos con *machine hearing*

En esta sección se presenta una descripción de los motores de reconocimiento de patrones recientes para la detección de tos basados en grabaciones de audio. Dicha descripción para entender su funcionamiento incluye módulos como los de preprocesamiento, diseño de características, esquema de clasificación y posprocesamiento.

5.2.1. El conjunto de datos de crowdsourcing de COUGHVID, un corpus para el estudio de algoritmos de análisis de tos a gran escala

En junio de 2021, Lara Orlandic y colaboradores [2], presentan el conjunto de datos de COUGHVID que proporciona más de 25 000 grabaciones de tos [303], en el que, cuatro médicos experimentados etiquetaron más de 2.800 grabaciones para diagnosticar anomalías médicas presentes en la tos. Además aportaron su algoritmo de detección de tos de código abierto.

Todas las grabaciones fueron recopiladas entre el 1 de abril y el 1 de diciembre de 2020 a través de una aplicación web desplegada en un servidor privado ubicado en las instalaciones de la École Polytechnique Fédérale de Lausanne (EPFL), Suiza. A través del micrófono se comienza una grabación de audio de hasta 10 segundos, una vez que se completa la grabación de audio, se muestra un pequeño cuestionario para obtener algunos metadatos sobre la edad, el género y la condición actual del usuario. Pero incluso si el cuestionario no se completa, el audio se envía al servidor. Asimismo, se solicita al usuario permiso para proporcionar su información de geolocalización, la cual no es obligatoria. Cada usuario que cargó su sonido de tos en el conjunto de datos de COUGHVID presumiblemente usó un dispositivo diferente, lo que podría introducir una variación en la calidad de la grabación debido a los diferentes *hardware* y *software* de grabación de cada dispositivo.

Hubo un poco más de participantes masculinos que femeninos (66,0 % y 33,5 %,respectivamente), y la mayoría de los participantes no tenían afecciones respiratorias preexistentes (83,6 %). Los porcentajes de participantes sanos, sintomáticos de COVID-19 y positivos de COVID-19 fueron del 78 %, 16,1 % y 5,8 %, respectivamente. La edad promedio de las grabaciones fue de 36,5 años con una desviación estándar de 13,9 años.

Primero seleccionaron al azar un conjunto de 215 grabaciones de la base de datos. Este proceso dio como resultado una muestra casi equilibrada de 121 sonidos de tos y 94 sonidos que no son de tos, incluidos el habla, la risa, el silencio y varios ruidos de fondo. Estas grabaciones fueron preprocesadas primero por un filtrado de paso bajo de frecuencia de corte 6 kHz, para posteriormente diezmar la resolución a 12 kHz. A continuación, se extrajeron de cada grabación 68 características de audio comúnmente utilizadas para la clasificación de la tos [304], siendo estas, 26 características MFCC [305], 19 características Energy Envelope Peak Detection (EEPD) [306], 8 características Power Spectral Density (PSD) [31] y 15 características descritas por Pramono y Sharma [305], [307].

Desarrollaron un clasificador E**x**treme **G**radient **Boost**ing (XGBoost) [308] para realizar la clasificación de tos basada en características de audio. Con el fin de evaluar el rendimiento del modelo en varios conjuntos de participantes de prueba y entrenamiento, empleamos un procedimiento de Cross-Validation (CV) de *k-fold* anidado [309], [310]. En primer lugar, el conjunto de datos se dividió en 10 segmentos, cada uno con el mismo número de registros, denominados "*folds* CV externos". En cada *fold* exterior, se usó un segmento para prueba y el resto para entrenamiento. Luego, dentro de cada *fold* de CV externo, los hiperparámetros XGBoost se ajustaron utilizando Estimadores Parzen estructurados en árbol (TPE) [311].

TPE funciona dividiendo aún más los datos de entrenamiento en 10 segmentos, conocidos como "*folds* CV internos", y luego determina qué conjunto de hiperparámetros exhibe la precisión media más alta en los 10 *folds*. Después de cada procedimiento de TPE, probaron el modelo XGBoost ajustado utilizando los datos de prueba de su respectivo *fold* de CV externo y calcularon varias métricas de precisión. Luego promediaron cada métrica de precisión en los 10 *folds* CV externos. Podemos ver que este clasificador de tos muestra un AUC medio del 96,4 %$\pm$3,3 %, lo que indica que el modelo funciona bien y muestra poca variación entre los diferentes grupos de prueba.

Para desarrollar el modelo utilizable final, volvieron a ensamblar todo el conjunto de datos y seleccionaron aleatoriamente el 22 % de las grabaciones para evaluar las capacidades de generalización de nuestro modelo para participantes de prueba invisibles. El 78 % restante de los datos se usó para entrenar el clasificador XGBoost final y determinar un conjunto final de hiperparámetros usando el procedimiento de validación cruzada TPE mencionado anteriormente, esta vez usando un ShuffleSplit CV [312] basado en una permutación aleatoria de 5 veces. El modelo final muestra una precisión del 95,5 % y una sensibilidad del 80,8 %, lo que indica que el modelo elimina con éxito la mayoría de las grabaciones sin tos y mantiene más del 80 % de las grabaciones que realmente contienen tos.

5.2.2. Detección automática de tos a partir de grabaciones de audio realistas utilizando C-BiLSTM con regresión de límites

En febrero de 2022, Mingyu You y colaboradores [289], presentaron un modelo de memoria a largo plazo bidireccional convolucional profundo (C-BiLSTM) con regresión de límites para la detección de tos, donde las partes con y sin tos deben clasificarse y ubicarse. La CNN analiza el espectro de audio, mientras que la BiLSTM predice el *frame* actual haciendo referencia a los *frames* adyacentes.

Diseñaron, recopilaron y etiquetaron un conjunto de datos de audio realista que contiene grabaciones de pacientes con enfermedades respiratorias, llamado Corp Dataset que se publica en línea en el sitio web de MARI Lab [313]. Se incluyen 168 horas de registros con 9969 toses de 42 pacientes diferentes en el hospital de Tongji, China.

Los datos de audio se graban a una frecuencia de muestreo de 44,1 kHz y una tasa de bits de 192 kbps. En el preprocesamiento, el segmento de audio se reduce a 16 kHz. Con ventanas Hamming de 25 ms y superposición de 15 ms, cada segmento de audio de 10 segundos se divide en 998 *frames*. Luego, se calculan las características MFCC [314]-[316] de 13 dimensiones y las derivadas temporales primera y segunda correspondientes.

Las características de audio extraídas de un segmento de 10 segundos se transforman en un espectrograma, que se normaliza y se alimenta a la red profunda C-BiLSTM que contiene una capa de normalización, 5 capas convolucionales y 2 capas BiLSTM. Las dos primeras capas convolucionales tiene 64 núcleos convolucionales, cada uno con el tamaño de 7x1, 7 en el eje temporal y 1 en el eje de características, y las 3 últimas tiene 128 núcleos convolucionales, cada uno con el tamaño de 3x1. Usan relleno a lo largo del eje temporal y establecen el paso en 1 para mantener la longitud del mapa de características sin cambios. La red se entrena utilizando el algoritmo de optimización de Adam con una tasa de aprendizaje inicial de 0,001 y un tamaño de lote de 128.

La red de detección finalmente genera la probabilidad de que cada *frame* pertenezca a un evento de tos mediante el clasificador de puntuación de extremo a extremo. La regresión de límites, como la otra rama de la red, predice límites de tos para cada *frame* para ubicar con precisión los eventos de tos en el mapa de características de la última capa de la red C-BiLSTM.

Este experimento que utiliza C-BiLSTM con regresión de límitesse se ejecuta 3 veces con datos de prueba no superpuestos de 10 pacientes. Para evitar la fuga de información, los datos de prueba se obtienen de 30 pacientes de los 42 pacientes totales. Finalmente, se informan los resultados de prueba promediados de tres ejecuciones para cada experimento.

Los resultados muestran que el sistema con el método de regresión de límites alcanza una sensibilidad del 84,13 %, una especificidad del 99,82 % y una intersección sobre unión (IoU) de 0,89, significativamente superior a otros modelos relacionados.

5.2.3. Hacia la detección de tos móvil independiente del dispositivo con redes neuronales convolucionales

En junio de 2019, Filipe Barata y colaboradores [24], proponen una arquitectura de red neuronal convolucional eficiente y un clasificador basado en conjuntos para reducir la discrepancia entre dispositivos.

En general, se reclutaron 43 participantes sanos (31 mujeres, 12 hombres). Su edad osciló entre 18 y 45 años con un valor medio de 26 (SD: 6). Esto dio como resultado un total de 6737 tos, 3985 risas, 3695 carraspeo, 731 habla y 443 señales de audio de expiración forzada, obteniendo así algunos sonidos identificados en la literatura como ejemplos típicos de sonidos que se confunden con la tos [273], [288].

La configuración del laboratorio siguió una configuración similar ala de [273], donde una persona está sentada (en un ambiente tranquilo) en una mesa en la que se coloca el dispositivo de grabación. Los dispositivos se colocaron sobre la mesa, una vez directamente frente a la silla y una vez desplazados hacia la izquierda 1 m, con una distancia silla-mesa de 15 cm. Los datos recopilados en un entorno de laboratorio siguieron el protocolo de estudio descrito en [317]. Los participantes recibieron instrucciones de toser voluntariamente 20 veces a dos distancias diferentes mientras cinco dispositivos diferentes grababan audio: HTC M8, Samsung S6, Apple iPhone4, tableta Google Nexus 7 y un micrófono de estudio Røde NT1000). Además, se registraron de la misma manera diferentes sonidos de control, como la risa, el carraspeo, la espiración forzada y el habla. Los dispositivos se encendieron para grabar todos los sonidos a la frecuencia de muestreo y la tasa de bits preestablecidas, a saber, 44,1 kHz con 16 bits, respectivamente.

Calcularon la densidad espectral de potencia (PSD) [318] y usaron el PSD del micrófono de estudio como referencia. Calcularon el error cuadrático medio (MSE) de la PSD de la medición de referencia y la medición del teléfono inteligente/tableta sobre todas las toses medidas. Posteriormente, normalizaron los MSE de cada tos con respecto al participante que tosía. Las medias más altas del MSE para las grabaciones de HTC M8 y Google Nexus 7 indican una desviación más alta de las grabaciones del micrófono de estudio Røde NT1000 y, por lo tanto, sugieren una calidad más baja en comparación con las grabaciones de Apple iPhone 4 y Samsung S6.

Redujeron el tamaño de los datos y optimizaron el tiempo de entrenamiento diezmando las señales a 22,05 kHz después de aplicar un filtro anti-aliasing, ya que, los sonidos característicos de la tos y la mayoría de los sonidos enérgicos están asociados con frecuencias inferiores a 10 kHz [319]. Para centrarse en la clasificación de la característica explosiva, calcularon la amplitud máxima de los eventos acústicos extraídos en los marcos de tiempo y, posteriormente, extrajeron 325 ms alrededor del máximo rendimiento de 650 ms de la señal. El marco de tiempo de 650 ms corresponde aproximadamente a la duración total promedio de una tos [319]. La señal extraída se estandarizó empleando la normalización min-max. Posteriormente, se computó un espectrograma escalado en mel con 16 bandas, 112 muestras entre *frames* sucesivos y una FFT [320] de 2048 puntos.

Desarrollaron una CNN inspirada en [321], donde reemplazaron las convoluciones regulares por convoluciones separables en profundidad [322] para reducir el número de cálculos y utilizaron la agrupación global máxima, que combina el máximo de cada mapa de características directamente en las capas de salida, reemplazando las costosas capas totalmente conectadas [323]. Además, utilizaron abandonos con una tasa del 50 %, para evitar el sobreajuste [324]. La red consta de una capa convolucional separable en profundidad con tamaño de filtro 1x7 y 16 canales, una capa convolucional separable en profundidad con tamaño de filtro 1x5 y 64 canales y 3 capas convolucionales separables en profundidad con tamaño de filtro 3x3 y 64 canales, finalizando con una capa completamente conectada de dos salidas. Entre las capas convolucionales hay capas max pooling 1/2 y a la entrada de la capa completamente conectada una capa global max pooling. Es entrenada usando el método de optimización estocástica de Adam [249] la tasa de aprendizaje se encontraba en el rango de [0,001, 0,007] y el tamaño del lote se estableció en 64. Además,aplicamos técnicas que, según se ha informado, tienen un efecto positivo en el descenso de gradiente estocástico,como el recorte de gradiente [325] y la adición de ruido de gradiente [326]. Agregaron ruido de gradiente con una variación de 0,009.

Tenían un set de cinco dispositivos por lo que usaron el método de conjunto de embolsado (bagging). Además los datos se dividieron según la proporción entrenamiento/validación/prueba de 60/15/25, siendo exactamente 27/5/11. En el Escenario I, el entrenamiento se realizó en todos los dispositivos y se probó en participantes no vistos. Dando como resultado CNN con espectrograma de Mel 91.7 % de sensibilidad y 86.7 % de especificidad y con el método de conjunto 91.7 % de sensibilidad y 90.1 % de especificidad.

En el Escenario II, el entrenamiento se realizó en todos los dispositivos excepto en aquel anotado como nombre de columna, luego se evaluó en las grabaciones del dispositivo no visto con participantes no vistos. Podemos observar una caída en el rendimiento medio y un aumento en la desviación estándar para el Escenario II en comparación con el Escenario I.

5.3. Detección de COVID-19 mediante audio de tos

5.3.1. Detección automatizada de tos por COVID-19

En enero de 2022, Alberto Tena y colaboradores [290], buscan realizar una detección automática de COVID-19 rápida y eficiente, solucionando el desafío del coste, el tiempo transcurrido y la cantidad de pruebas que se pueden hacer de forma masiva, utilizando archivos de audio sin procesar con una metodología disponible gratuitamente. Usando el paquete synthpop [327] y basándose en una muestra aleatoria de toses de COVID-19 y no COVID-19 se creó un conjunto de datos sintéticos [328] disponible gratuitamente en línea.

En este trabajo se utilizaron cinco conjuntos de datos, el que la Universidad de Lérida (UdL) recopiló para este estudio con 52 toses aprobado por el CEIm del HUAV, el de la Universidad de Cambridge (UC) [292] con 332 toses, el de Coswara [293] con 288 toses, el de Virufy [294] con 121 toses y el conjunto de datos de tos ferina [305] con 20 toses, lo que hace un total de 813 toses.

Preprocesado: las señales originales se muestrearon a 16 kHz y se almacenaron en L segmentos superpuestos. Cada segmento fue de 0,98s y los segmentos se superpusieron en 0,8575s. Se convirtieron a un espectrograma de

magnitud con 257 intervalos de frecuencia utilizando una transformada de Fourier de tiempo corto (STFT) [329] unilateral con una ventana Hanning (o Hann) [330] periódica de 25 ms con un salto de 10 ms y una transformada de Fourier discreta (DFT) [331] de 512 puntos. Luego, el espectro de magnitud se pasó a través de un banco de filtros espaciados en mel [332], [333] de 64 bandas y se sumaron las magnitudes de cada banda. El audio estaba representado por una matriz de 96x64x1xL,donde 96 es el número de espectros en el espectrograma de mel y 64 es el número de bandas de mel. Finalmente, los espectrogramas de mel se convirtieron a una escala logarítmica.

Red neuronal: La matriz de espectrogramas mel de 96x64x1xL fue la capa de entrada que pasó a través de la red neuronal profunda YAMNet [295] para la identificación automática de las muestras de tos registradas en los archivos de audio. La salida de YAMNet (matriz Lx512) corresponde a puntuaciones de confianza para cada una de las 521 clases de sonido a lo largo del tiempo.

Postprocesamiento: Para detectar la región del evento de sonido, las 521 señales de confianza se pasaron a través de un filtro de media móvil con una longitud de ventana de 7 y cada señal a través de un filtro de mediana móvil con una longitud de ventana de 3. Luego, las señales de confianza se convirtieron en máscaras binarias. Después de realizar varias pruebas, se estableció un umbral de 0,35s porque mostró el mejor rendimiento en la detección de muestras de "tos". Cualquier sonido de menos de 0,5s se descartó para el análisis y las regiones de menos de 0,25s se fusionaron. Las regiones de sonido identificadas que se superpusieron en un 50 % o más se consolidaron en regiones únicas. El tiempo de inicio de la región seleccionado fue el tiempo de inicio más pequeño y el tiempo de finalización de la región seleccionado fue el tiempo de finalización más largo de todos los sonidos del grupo. Luego, las regiones de sonido etiquetadas como "tos"por YAMNET fueron seleccionadas para su análisis. Los límites de estas muestras de tos se seleccionaron usando el algoritmo detectSpeech [334] usando una ventana de Hann con 0.03xSampling rate segundos de salto. Finalmente, los primeros 600 ms de cada muestra de tos identificada se volvieron a muestrear a 8820 Hz y se normalizaron para obtener las representaciones y características de tiempo-frecuencia.

Representación de tiempo-frecuencia: Calcula la distribución de Wigner (WD) [335] posteriormente convoluciona la distribución de Wigner con la exponencial de Choi-Williams, para obtener la distribución de Choi-Williams (CWD) [336]. Entonces, para obtener parámetros estadísticos, la función de densidad $CWD(f,t)$ se normalizó para tener un área igual a 1. Por lo tanto,se puede asociar con una función de densidad de probabilidad conjunta $CWD_N(f,t)$ de las variables de tiempo y frecuencia. Sus distribuciones marginales, que no contienen la interferencia, siguen representando, aunque de forma normalizada, la potencia instantánea y la energía de densidad espectral de la señal original. Por último se obtiene la distribución de densidad de probabilidad conjunta (pD) multiplicando la potencia instantánea y la energía de densidad espectral.

Funciones de tiempo-frecuencia: Se obtienen un total de 39 características a partir de la representación de frecuencia temporal de cada muestra de tos. 28 de ellos correspondieron a la energía espectral instantánea, frecuencia instantánea, frecuencia instantánea de pico e información espectral. Estos se obtuvieron dividiendo el espectro (0–4410 Hz) en 7 bandas de frecuencia: 1, 0–80 Hz; 2, 80–250 Hz; 3,250–550 Hz; 4, 550–900 Hz; 5, 900–1500 Hz; 6, 1500–3000 Hz; 7, 3000–4410 Hz. También se calcularon la frecuencia media del espectro total, las entropías conjunta, instantánea y espectral de Shannon , la curtosis, 3 momentos conjuntos tiempo-frecuencia y 3 momentos conjuntos de las señales marginales de potencia instantánea y densidad espectral.

Posteriormente se redujeron el número de características a través de uno de los siguientes métodos:

- La eliminación recursiva de características (RFE) [337] es un proceso recursivo que clasifica las características según alguna medida de su importancia. En cada iteración, se mide la importancia de la característica y se elimina la menos relevante.

- Utilizando un Autoencoder [338] que reduce la dimensionalidad del conjunto inicial de características mientras se conserva la información en el conjunto de datos original, en este caso las 39 características originales se redujeron a 15.

Modelos de clasificación: Se definieron cinco grupos de sujetos para el análisis, siendo estos sujetos con COVID-19 (C), sujetos con resultado negativo de COVID-19 que no tenían tos (N), sujetos no COVID-19 con tos inespecífica como síntoma (NC), sujetos sin COVID-19 con tos ferina (PT) y todos los sujetos que no eran COVID-19 (NNC). Luego, se realizaron cuatro experimentos de clasificación. Estos consistieron en C frente a N, C frente a NC, C frente a PT y C frente a NNC. Por último se implementaron en R los modelos supervisados

más populares en la clasificación de la tos, es decir, Random Forest (RF), Support Vector Machine (SVM), Linear Discriminant Analysis (LDA), Logistic Regression (LR) y Naïve Bayes (NB).

Las características seleccionadas por RFE para ajustar los modelos obtuvieron mejores resultados en el rendimiento general de los modelos que las características extraídas mediante el Autoencoder. Los modelos de clasificación se comportaron mejor al comparar C vs PT que al comparar C vs N, C vs NC o C vs NNC. En C vs PT, las métricas que funcionaron mejor fueron exactitud (94.81 %), sensibilidad (98.91 %) y F-score (97 %) para RF, precisión (97.13 %) para LR y AUC (97.29 %) para SVM. Este experimento detectó mejor la tos positiva de COVID-19, pero no funcionó tan bien para clasificar la tos ferina (85 % de especificidad para LR y LDA). En cambio, en los otros experimentos, la detección de casos positivos y negativos fue más equilibrada. Esto fue especialmente así en el experimento C vs. NNC, que obtuvo la mejor especificidad (85.09 %) para RF. Este experimento refleja un escenario de caso más real donde la tos de COVID-19 coexiste con tos de diferentes patrones. En los cuatro experimentos de clasificación realizados, RF mostró el mejor rendimiento general.

5.3.2. Detección de COVID-19 a partir de patrones de voz, tos y respiración: conjunto de datos y resultados preliminares

En noviembre de 2021, Vladimir Despotovic y colaboradores [339] proponen dentro del estudio CDCVA recopilar los sonidos emitidos por la boca humana utilizando cinco tareas vocales: fonación sostenida de una vocal/aaaa/, toser (3 veces), inhalar y exhalar profundamente por la boca (3 veces), contar números del 1 al 20 y leer un texto específico. Los datos se recopilan utilizando una plataforma web multilingüe que está diseñada en 8 idiomas (inglés, alemán, francés, español, portugués, árabe, luxemburgués y serbio) para cubrir una amplia gama de participantes potenciales. Además de los datos vocales, también se recopila información sobre la edad, el género, el país de residencia, el idioma nativo, el peso, la altura, los hábitos de fumar y beber del participante, así como los síntomas y comorbilidades relacionados con COVID-19. El estado de salud del participante (positivo o negativo a COVID-19) se determina en función de la autodeclaración confirmada por la prueba estándar RT-qPCR [340] o RAT [341], con la fecha de la prueba. Los participantes también pueden proporcionar la información si actualmente se encuentran en aislamiento domiciliario u hospitalizados.

El conjunto de datos es totalmente colaborativo; por lo tanto, se basa en la confianza con los participantes, además otra dificultad es que debido al uso de diferentes dispositivos, micrófonos, navegadores web y condiciones de grabación para la recopilación de datos, se deben aplicar técnicas de preprocesamiento para unificar las muestras de grabación antes de su uso posterior.

La base de datos contiene un total de 1103 participantes en el estudio, de los cuales el 92,38 % se declaró negativo a COVID-19 (1019) y el 7,62 % como positivo (84). De ahí, se creó un subconjunto equilibrado que incluye tos de todos los participantes positivos a COVID-19 y la misma cantidad de participantes no infectados emparejados por edad y sexo, sin antecedentes médicos de afecciones que afectan la voz, y sin síntomas al COVID-19. El subconjunto equilibrado contiene 82 participantes positivos para COVID-19, de 84 en el conjunto de datos inicial, ya que faltaban señales de tos para dos usuarios. Los participantes positivos COVID-19 tenían entre 14 y 69 años (media 40,11, desviación estándar 12,21), mientras que los participantes no infectados tenían entre 15 y 68 años (media 40,26, desviación estándar 12,1). Para hacer el experimento más realista, el tabaquismo no se tomó como criterio de exclusión del subconjunto balanceado. La prevalencia de tabaquismo fue mayor en el grupo control (28,05 %) que en el grupo positivo a COVID-19 (17,07 %).

La cantidad de toses individuales por muestra de audio varió de 2 a 5. Por lo tanto, decidieron segmentar el subconjunto equilibrado de toses y, en cambio, analizar las señales de tos individuales, lo que llevó a un conjunto de datos que fue compuesto por 496 señales de tos en total, 249 de las cuales fueron producidas por participantes positivos a COVID-19 y 247 por sujetos no infectados.

Todas las instancias de datos de audio se convirtieron al formato de audio WAV con una frecuencia de muestreo de 44,1 kHz y una profundidad de bits de coma flotante de 32 bits. Las grabaciones estéreo se convirtieron a mono antes de su posterior procesamiento y los silencios iniciales y finales se eliminaron de todas las grabaciones de audio. Todas las grabaciones de tos se segmentaron en señales de tos individuales, manteniendo un registro del participante que produjo cada una de las toses individuales. Algunas grabaciones contenían ruido ambiental (e.g., charlas, risas, televisión, ruido de tráfico de fondo, etc.); por lo tanto, se aplicó la reducción de ruido utilizando Audacity v2.4.2 [342] para eliminar el ruido de fondo mediante la activación del ruido espectral.

Se utilizaron cuatro formas diferentes de extracción de características:

- La primera forma fue mediante el GeMaps [343] que es un conjunto minimalista de parámetros de voz seleccionados según su potencial para abordar los cambios fisiológicos durante la producción de voz. Contiene 62 características acústicas que incluyen 18 parámetros de energía/amplitud, frecuencia y espectro, y 6 características temporales, así como funciones estadísticas aplicadas a estos parámetros(media aritmética, desviación estándar, percentil, etc.).

- La segunda forma es mediante el *extended GeMaps* (eGeMaps) que contiene 26 parámetros acústicos adicionales al conjunto de funciones de GeMaps, lo que lleva a un conjunto de 88 funciones acústicas.

- La tercera forma es mediante ComParE este conjunto de funciones se utiliza como parte del desafío de lingüística computacional que se lleva a cabo cada año en la conferencia INTERSPEECH [344] y se utilizan como conjuntos de características de referencia para diversas tareas acústicas. Este conjunto de características acústicas de fuerza bruta esta compuesto por 65 LLD y varias funciones estadísticas aplicadas a estos LLD, lo que da como resultado un total de 6737 características extraídas para cada una de las instancias de datos de longitud variable. Todas las funciones se extraen utilizando el software openSMILE v3.0 [345].

- La cuarta forma es mediante las características *wavelet scaterring* que se utilizan para extraer representaciones de baja varianza de la señal de audio mediante la aplicación de una transformada *wavelet scaterring*, es decir, una serie de descomposiciones de *wavelet* y operadores de módulo. La estructura resultante es similar a una red neuronal convolucional, pero no requiere entrenamiento [346]-[348]. Definieron la red *wavelet scaterring* con dos bancos de filtros de *wavelet*: el primero con 8 *wavelet* por octava y el segundo con 1 *wavelet* por octava, mientras que la escala de tiempo invariante es de 0,5 s. Cada muestra de audio se divide en varias ventanas de tiempo de dispersión a lo largo de 411 rutas en la transformada *scaterring*, lo que lleva a un total de 411 características extraídas por cada ventana de tiempo. Después de extraer las características acústicas o *wavelet scaterring*, solo se usa un subconjunto de las características más relevantes para la tarea de clasificación basada en el criterio de información mutua, que mide la dependencia entre dos variables aleatorias. Es igual a cero si las variables son independientes, mientras que los valores más altos denotan una mayor dependencia [349]. Clasificaron las características en orden descendente de acuerdo con la información mutua y usaron solo el subconjunto de características que tienen la mayor influencia en la tarea de clasificación. Todas las características se estandarizaron antes de su uso posterior, es decir, se eliminó la media y se escalaron a la varianza unitaria.

Proporcionaron una visualización de datos para el subconjunto segmentado de toses utilizando la incrustación de vecinos estocásticos distribuidos en t (t-SNE) [350]. t-SNE genera un espacio de características reducido donde las características similares están representadas en un gráfico bidimensional por puntos cercanos en el espacio y viceversa. Utilizando el conjunto de características de ComParE, como el conjunto más completo de características acústicas, el gráfico podría dividirse en dos regiones: 1) Arriba a la izquierda resaltada con un rectángulo rojo, donde los sujetos diagnosticados positivos a COVID-19 y descritos por puntos rojos están presentes en su mayoría; 2) Abajo a la derecha resaltado con un rectángulo azul para personas no infectadas. Aunque ambas regiones contienen puntos de datos superpuestos, la distribución de datos sugiere que es posible discriminar entre los dos grupos.

Se usó una validación cruzada estratificada de 5 veces para dividir los datos en cinco subconjuntos, cuatro se usan para entrenamiento y el restante para prueba. Cada clase se distribuye aproximadamente por igual entre los subconjuntos, conservando la distribución de clases del conjunto de datos original. El procedimiento se repite 5 veces para obtener estimaciones de rendimiento para cada uno de los *folds* de prueba y luego se promedia sobre todos los *folds*. Dado que el conjunto de datos contiene múltiples señales de tos de cada uno de los participantes, se dividió en *folds* con respecto a los participantes, lo que significa que las toses del mismo hablante no pueden aparecer tanto en el subconjunto de entrenamiento como en el de prueba. El rendimiento del modelo se evaluó utilizando la precisión, la sensibilidad y la especificidad como medidas de rendimiento.

Para explicar los modelos de conjunto (bosques aleatorios, *boosting* y *bagging*) tratando de descubrir los mecanismos exactos que alteran los parámetros acústicos de la tos en personas con COVID-19, analizan las diez características más informativas del conjunto de características acústicas de ComParE. El mejor indicador de la tos de COVID-19 según el criterio de información mutua es la energía del marco de la señal cuadrática media

raíz suavizada por un filtro de media móvil con una longitud de ventana igual a 3. Al ser un sonido explosivo, la tos tiene ráfagas de aumento de energía en un corto intervalo de tiempo que son más evidentes en las señales que producen las personas con COVID-19. Dado que la tos seca es el segundo síntoma más prevalente observado en casi el 40 % de los participantes positivos a COVID-19, se esperan niveles de energía más altos en comparación con la tos forzada de individuos no infectados.

La armonía espectral, es la segunda característica más informativa, describe la estructura armónica de una señal de audio en la que las frecuencias de sonido son múltiplos enteros de la frecuencia fundamental. Por lo general, se observa en el habla sonora, pero un estudio reciente ha demostrado que se pueden observar de 2 a 3 armónicos en la tos forzada de individuos sanos, mientras que no se observa un patrón claro en la estructura de los armónicos en la tos húmeda. El estudio confirma que la armonía espectral también se puede utilizar para distinguir entre la tos de personas positivas a COVID-19 y sujetos no infectados.

La pendiente espectral como tercera característica más informativa está estrechamente relacionada con la velocidad de cierre de la glotis durante la producción de tos, lo que resulta en una pendiente espectral más pronunciada para un cierre más lento de la glotis. La señal de tos tiene un mecanismo fisiológico típico que comienza con la fase inspiratoria, seguida por el cierre de la glotis y el aumento de la presión intratorácica en la fase de compresión, y la apertura repentina de la glotis que produce un flujo de aire espiratorio rápido y un sonido de tos en la fase espiratoria. Varios cierres parciales posteriores de la glotis pueden dar lugar a sonidos sonoros adicionales.

Aunque las características de RASTA [351] no se estudiaron previamente en el procesamiento de señales de tos, 6 de las 10 características más informativas en este estudio están relacionadas con la posición absoluta del valor máximo o mínimo (en marcos), o la longitud máxima del segmento de uno de los 26 bandas auditivas filtradas RASTA, lo que sugiere que es un indicador muy potente de tos producida por personas con COVID-19. El procesamiento RASTA utiliza filtros de paso de banda en las trayectorias temporales de los vectores de características del habla para suprimir los componentes espectrales que cambian más rápido o más lento que en una señal de voz típica; por lo tanto, se puede utilizar para suprimir el ruido aditivo en técnicas de reconocimiento de voz o mejora del habla.

Las características extraídas sirven para alimentar a cuatro modelos de clasificación diferentes:

- El clasificador de bosques aleatorios [352], [353] se utilizó con 250 árboles para realizar la clasificación en participantes positivos y negativos a COVID-19. Los árboles completamente desarrollados y sin podar se utilizan con el índice de Gini [354], [355] como criterio para dividir el nodo. Las características se permutan aleatoriamente en cada división.

- Los clasificadores de árboles de decisión *boosting* [356], [357] y *bagging* [358] están configurados con 250 árboles de decisión con una profundidad máxima igual a cuatro. Finalmente, se utilizó el algoritmo AdaBoost.M1 para el impulso adaptativo con 250 tocones de decisión, es decir, árboles de un nivel con dos nodos de hoja.

- El clasificador perceptrón multicapa (MLP) [359], [360] consta de una capa de entrada con el número de nodos que corresponde al número de características seleccionadas, tres capas ocultas que utilizan la función de activación de la Unidad lineal rectificada (ReLU) con 128, 64 y 16 neuronas respectivamente, y una capa de salida con dos neuronas que corresponden al número de clases para realizar la tarea de clasificación que utiliza la función de activación *softmax*. El algoritmo de optimización de Adam se aplica para actualizar los pesos de la red de forma iterativa durante el entrenamiento, con la entropía cruzada como función objetivo. El número óptimo de capas ocultas, así como el número óptimo de nodos en las capas ocultas, se evalúan utilizando la búsqueda de cuadrícula.

Para ajustar el clasificador de árboles de decisión *bagging*, se utilizaron como hiperparámetros el número de estimadores base (árboles individuales) y el número máximo de muestras para entrenar cada estimador base, dado como la proporción del número total demuestras. Además de estos dos hiperparámetros, para los bosques aleatorios se utiliza la cantidad máxima de características a considerar para cada división, dada como la proporción de la cantidad total de características. El número de estimadores base fue el único hiperparámetro optimizado en el caso del clasificador *boosting*.

El rendimiento se evalúa utilizando tres modelos de conjuntos (bosques aleatorios, *boosting* y *bagging*), redes MLP. Se utilizan tres conjuntos de características estándar para la extracción de características con modelos de

conjunto, es decir, GeMaps, eGeMaps y ComParE. Dado que los conjuntos de funciones de GeMaps y eGeMaps son relativamente pequeños, no se aplicó ninguna selección de funciones. Teniendo en cuenta el tamaño del conjunto de funciones de ComParE, solo el 1 % de las funciones más informativas se mantienen en función del criterio de información mutua. Los resultados obtenidos revelan que aunque los conjuntos minimalistas de características acústicas, como GeMaps y eGeMaps, son capaces de aprender características intrínsecas de la tos, se obtienen resultados sustancialmente mejores utilizando un enfoque de extracción de características de audio de fuerza bruta con características de ComParE, lo que lleva a una precisión y sensibilidad aproximadamente igual a 87 % para bosques aleatorios, mientras que la especificidad sube al 90,87 % en caso de *bagging*. Los bosques aleatorios y el *bagging* superan al *boosting* para todos los conjuntos de características acústicas estándar, lo que puede explicarse por las principales diferencias en el diseño de estos modelos de conjunto. A diferencia de los bosques aleatorios y el *bagging*, donde los árboles se cultivan de forma independiente, en *boosting* cada modelo posterior, se mejora el rendimiento del anterior al aprender de sus clasificaciones erróneas. Por lo tanto, *boosting* es, por naturaleza, más sensible a los valores atípicos que podrían estar presentes en nuestro conjunto de datos debido a la recopilación de datos de colaboración abierta. MLP proporciona en general resultados de rendimiento más bajos que los modelos de conjunto.

La transformada de dispersión de *wavelet*, aunque es similar en estructura a las redes neuronales convolucionales, no aprende los filtros durante la optimización, sino que los aplica a priori, lo que puede ser beneficioso cuando los datos de entrenamiento son limitados, como en nuestro caso. Teniendo en cuenta la naturaleza no estacionaria y estocástica de la tos, la transformada *wavelet* permite descomponer la señal de tos en el dominio del tiempo y la frecuencia, mientras se enfoca en las estructuras de la señal local para capturar de manera eficiente los cambios de frecuencia en cualquier momento [284]. En contraste con las funciones acústicas estándar, el refuerzo proporciona un rendimiento ligeramente mejorado para las funciones de dispersión de *wavelet* en comparación con otros modelos de conjuntos, lo que indica que el filtrado de *wavelet* es resistente al ruido en los datos [361]. Los resultados obtenidos con características de dispersión de *wavelet* y modelos de conjuntos (precisión 88,52 %, sensibilidad 88,75 %,especificidad 89,82 %). Tenga en cuenta que los resultados proporcionados para las funciones de dispersión de *wavelet* se obtienen después de aplicar la selección de funciones con el 10 % de las funciones más informativas según el criterio de información mutua. El uso de una gran cantidad de características irrelevantes agrega incertidumbre a la predicción y reduce el rendimiento general del modelo, sin embargo, una mayor disminución del tamaño de la característica degrada el rendimiento y reduce los costos computacionales del modelo.

También extrajeron incrustaciones de audio profundas usando modelos VGGish [362] y OpenL3 [363]. El extractor de funciones VGGish extrae incrustaciones de audio de 128 dimensiones directamente de *frames* de señal de audio de 960 ms que no se superponen. Cada *frame* se divide en ventanas de 25 ms con una superposición de 10 ms y se calcula la transformada de Fourier de tiempo corto para obtener los espectrogramas integrados en 64 frecuencias espaciadas en mel, lo que da como resultado espectrogramas log-mel de 96 x 64. Las señales de audio de entrada se vuelven a muestrear a 16 kHz para que coincidan con los requisitos de la red VGGish. La red VGGish se compone de una serie de capas convolucionales con función de activación ReLU y capas de agrupación máxima, seguidas de tres capas completamente conectadas. La red está preentrenada en el conjunto de datos a gran escala Audio Set [364] para la clasificación de eventos de audio que se compone de más de 1,7 millones de extractos de 10 s de videos de YouTube (4971 h de audio en total) clasificados en 632 eventos de audio. Eliminaron la capa de salida de la red VGGish preentrenada y agregaron en su lugar una capa completamente conectada con 128 nodos con una función de activación de ReLU y una capa de abandono con una tasa de abandono igual a 0,5 para evitar el sobreajuste. Finalmente, agregaron una capa de salida con 2 nodos para realizar la clasificación.

Para extraer incrustaciones de OpenL3, usaron un modelo preentrenado en el subconjunto ambiental de Audio Set, que incluye sonidos humanos y animales, así como otros sonidos en entornos acústicos naturales. El modelo utiliza una representación de tiempo-frecuencia de espectrograma Mel con 256 bandas Mel y extrae una incrustación de audio de 6144 dimensiones por cada *frame* de señal [363]. Cada fichero de tos se segmenta en fotogramas superpuestos de 1 s con un tamaño de salto de 0,5 s, lo que da lugar a múltiples incrustaciones para cada fichero de tos. La votación por mayoría se utiliza para determinar la etiqueta de un solo fichero de tos, en función de las etiquetas de sus marcos correspondientes. La clasificación se realiza mediante MLP con dos capas ocultas con tamaños de 512 y 128 nodos respectivamente, cada una seguida de una capa de abandono con tasa de abandono igual a 0,5 y finalmente una capa de salida con 2 nodos.

Para entrenar la cantidad de capas ocultas completamente conectadas en las capas superiores de extracción de características, la cantidad de nodos en las capas completamente conectadas, el tamaño del mini lote y la tasa de abandono de las redes OpenL3 y VGGish se ajustaron mediante la búsqueda en cuadrícula.

El modelo VGGish preentrenado existente se entrena utilizando el optimizador Adam con tamaño de mini lote de 64, parámetro de regularización L2 igual a 0.005 y tasa de aprendizaje adaptativo que cae por un factor de 10 si no se observa ninguna mejora después de 3 iteraciones (la tasa de aprendizaje inicial era igual a 0,001). El modelo converge después de 6 épocas y logra una precisión del 76,73 %, una sensibilidad del 80,84 % y una especificidad del 72,86 %, lo que demuestra la capacidad de aprender representaciones de características razonables directamente de las señales de audio y evita la necesidad de una extracción de características artesanal. Sin embargo, el modelo aún no es capaz de alcanzar el rendimiento de vanguardia logrado con la dispersión de *wavelet* o las características de ComParE. Un motivo de un rendimiento más bajo podría ser el hecho de que, debido a los requisitos del modelo VGGish, la señal se redujo a 16 kHz, mientras que la frecuencia de muestreo para las características de dispersión de *wavelet* y los conjuntos de características acústicas estándar fue de 44,1 kHz.

Se obtiene un rendimiento similar con las incrustaciones de tos OpenL3 (precisión 76,65 %, sensibilidad 81,89 %, especificidad 71,12 %). El modelo está entrenado para 50 épocas usando el optimizador Adam con un tamaño de mini lote de 64. Esperaron que al recopilar más datos y combinar diferentes modalidades de voz además de la tos, se puedan aprender mejores incrustaciones de audio, el impacto del aprendizaje transferido será mayor y los modelos de aprendizaje profundo se generalizarán mejor a un conjunto de datos de destino.

5.3.3. Diagnóstico de COVID-19 mediante inteligencia artificial utilizando sólo grabaciones de tos

En septiembre de 2020, Jordi Laguarta y colaboradores [291] presentaron el conjunto de datos, la arquitectura del modelo y el rendimiento de una herramienta de IA de registro de tos forzada de COVID-19 de coste cero, rápida e instantáneamente distribuible de preselección que logra una precisión del 98,5 %, incluida una tasa de detección asintomática del 100 %. Se puede desarrollar un conjunto ortogonal de biomarcadores para diagnosticar COVID-19, Alzheimer [365] y quizás otras afecciones.

A través del motor de grabación del sitio web (opensigma.mit.edu) con el objetivo de crear el conjunto de datos de MIT Open Voice para la discriminación de tos COVID-19 [366] recopilaron grabaciones de audio de tos de duración variable (en promedio 3 toses por sujeto) acompañadas de un conjunto de 10 preguntas de opción múltiple relacionadas con el diagnóstico de la enfermedad e información general del sujeto: edad, sexo, país, región; el resultado del diagnóstico médico realizado, cuándo fue realizado y si la fuente del diagnóstico fue una prueba oficial, la evaluación de un médico o una evaluación personal; y finalmente información sobre síntomas y días desde su aparición. Los síntomas solicitados incluían fiebre, cansancio, dolor de garganta, dificultad para respirar, dolor persistente o presión en el pecho, diarrea y tos.

Recopilaron grabaciones de 2660 sujetos positivos de COVID-19 y una proporción de 1 a 10 de sujetos positivos a control. La grabación estaba disponible en varios navegadores y dispositivos, lo que redujo cualquier posible sesgo específico del dispositivo. Los datos se anonimizaron antes de recopilarlos en su servidor seguro y las muestras se guardaron sin compresión en formato WAV (velocidad de bits de 16 kbps, canal único, códec opus) [1]. Se eliminaron las muestras que no tenían contenido de audio. No se realizó segmentación en las grabaciones de tos utilizadas para entrenar y evaluar.

Usaron todas las muestras positivas de COVID-19 en nuestro conjunto de datos y seleccionamos al azar la misma cantidad de sujetos negativos de COVID-19 para una distribución equilibrada. Solo utilizamos muestras con dos condiciones, primero con diagnóstico en los últimos 7 días y, segundo, con un inicio de síntomas no mayor a 20 días y donde los síntomas continuaron hasta que se tomó la muestra. Los audios de tos forzada y los resultados de diagnóstico del sujeto se utilizaron para entrenar y validar el discriminador de COVID-19. Se utilizaron 4256 sujetos (80 %) para entrenamiento y 1064 (20 %) para validación.

Propusieron una arquitectura que toma una grabación con una o más toses y realiza dos pasos de preprocesamiento con la grabación. En el primer paso cada grabación de tos de entrada se divide en fragmentos de audio de

[1] Cabe destacar el códec opus tiene compresión con perdidas y que con una que velocidad de bits de 16 kbps tiene calidad de banda ancha (*wideband*), ya que esta en el rango 16-20 kbit/s[367]

6 segundos, se rellena según sea necesario, se procesa con el paquete MFCC [368] y en el segundo paso se pasa a través del biomarcador 1 (Degradación Muscular).

La salida de estos pasos se convierte en la entrada de un modelo basado en CNN para generar un diagnóstico de preselección junto con un mapa de prominencia de biomarcadores. La arquitectura CNN se compone de tres ResNet50 [82] en paralelo. La capa de salida del tensor 4D de 7x7x2048 de cada modelo ResNet50 se concatena en paralelo. En los modelos de referencia, estos ResNet50 no están preentrenados. En el modelo de mejor desempeño, están preentrenados para capturar características acústicas en los biomarcadores 2 (Cuerdas vocales), 3 (Sentimiento) y 4 (Pulmones y Tracto Respiratorio). La salida de estos tres tensores concatenados luego se agrupa utilizando una capa 2D de agrupación promedio global, seguida de una capa de red neuronal profundamente conectada de 1024 neuronas (densa) con función de activación ReLU y, finalmente, una capa densa binaria con función de activación sigmoidea. Toda la arquitectura está entrenada en el conjunto de datos de tos COVID-19 para la clasificación binaria. Las diversas salidas de fragmentos de la arquitectura CNN se agregan utilizando esquemas competitivos para generar el mapa de prominencia del sujeto. Los resultados del documento se basan únicamente en las primeras salidas de fragmentos de audio. El trabajo futuro puede mostrar que la agregación no solo puede mejorar la explicabilidad sino también aumentar la precisión del diagnóstico.

La arquitectura de Open Voice Medicine del MIT utiliza los mismos cuatro biomarcadores que probaron anteriormente para la detección de la enfermedad de Alzheimer, los cuales lograron una precisión superior a la más avanzada [369]. Estos cuatro biomarcadores inspirados en las elecciones de la comunidad médica son: degradación muscular, cambios en las cuerdas vocales, cambios en el sentimiento/estado de ánimo y cambios en los pulmones y el tracto respiratorio.

Aplicaron el biomarcador 1 (Degradación Muscular) introduciendo características de degradación y fatiga muscular modificando las señales de entrada para todos los conjuntos de pruebas y trenes con la máscara de Poisson ($M(I_x) = Poiss(\lambda) * I_x$) . La descomposición de Poisson es una distribución común en la naturaleza que se ha propuesto previamente para modelar la degradación muscular. Lo encontraron efectivo ya que la eliminación de este biomarcador duplica aproximadamente la tasa de error en las predicciones oficiales. Para capturar la influencia de la degradación muscular en las predicciones individuales, desarrollaron una métrica de degradación muscular basada en la comparación de la producción con y sin este paso de Poisson inicial. Esta métrica es la relación normalizada de la predicción con y sin la máscara y se incorpora en el mapa de prominencia. Para los negativos de COVID-19, esta métrica se traza directamente; y para los positivos trazaron uno menos esta métrica. La máscara de Poisson aplicada en un punto MFCC de registro de tos, I_x , se calcula multiplicando este valor por una distribución aleatoria de Poisson de parámetros I_x y λ, donde λ es el promedio de todos los valores en el MFCC.

Aplicaron también el biomarcador 2 (Cuerdas vocales) debido a que los sujetos con enfermedades pulmonares suelen tener distintas expresiones de biomarcadores de las cuerdas vocales en comparación con los sanos. Por ejemplo, los estudios han informado que la presión del umbral de fonación, la presión pulmonar mínima necesaria para iniciar y mantener la oscilación de las cuerdas vocales, se correlaciona con la fatiga vocal. Por lo que desarrollaron un modelo *Wake Word* [370] para el sonido "mmmmmm" debido a su universalidad. Entrenaron un ResNet50 [82] con forma de entrada (300, 200) de MFCC para discriminar la palabra "Ellos" de otras usando LibriSpeech, un conjunto de datos de audiolibros con ≈1,000 horas de habla [371]. El modelo se entrenó mediante la creación de un conjunto de muestras equilibrado de 11 000 fragmentos de audio de dos segundos, la mitad con la palabra y la otra mitad sin ella, y logró una precisión de validación del 89 %. Encontramos que las características aprendidas de este biomarcador permiten la detección de variaciones en las cuerdas vocales que existen entre los sujetos de control y COVID-19, discriminando el 54 % del conjunto de prueba. Para el 19 % de los sujetos, este es el único biomarcador para discriminarlos correctamente.

Los estudios muestran un deterioro cognitivo en pacientes con COVID-19 y la evidencia clínica respalda la importancia de los sentimientos en el diagnóstico temprano del deterioro neurodegenerativo por lo que también aplican el biomarcador 3 (Sentimiento). Diferentes escenarios clínicos enfatizan diferentes sentimientos, como la duda o la frustración como posibles indicadores neurodegenerativos. Para obtener un biomarcador que detecte esta disminución, entrenamos un modelo clasificador *Sentiment Speech* para aprender las características de los sentimientos en el conjunto de datos del habla RAVDESS [372], que incluye actores que entonan en 8 estados emocionales: neutral, tranquilo, feliz, triste, enojado, temeroso, repugnancia y sorpresa. ResNet50 [82] fue entrenado en muestras de 3 segundos para la clasificación categórica de las 8 entonaciones con forma de entrada (300, 200) de MFCC que logró una precisión de validación del 71 %.

Por último aplicaron el biomarcador 4 (Pulmones y Tracto Respiratorio) ya que se ha demostrado que la tos humana es útil en el diagnóstico de varias enfermedades mediante el reconocimiento automático de audio. La estructura física de los pulmones y las vías respiratorias se altera con las infecciones respiratorias, y en los primeros días del COVID-19, los epidemiólogos escuchaban los pulmones mientras los pacientes forzaban la tos como parte de sus métodos de diagnóstico. Existe evidencia de que muchas otras enfermedades pueden diagnosticarse usando IA en la tos forzada. Por lo tanto, se seleccionó un modelo de biomarcador capaz de capturar características en los pulmones y el tracto respiratorio.

Los modelos anteriores que crearon con un superconjunto del conjunto de datos de tos recopilados a través de MIT Open Voice para la detección de COVID-19 [366] predijeron con precisión el género y la lengua materna de una persona en función de una tos. Presumimos que tales modelos capaces de aprender características y variaciones acústicas en toses forzadas entrenadas para diferenciar la lengua materna podrían mejorar la detección de COVID-19 mediante el aprendizaje por transferencia. Eliminaron del conjunto de datos todos los metadatos excepto el idioma hablado de la persona que tosía (inglés, español) y dividimos los audios en fragmentos de 6 segundos. ResNet50 [82] fue entrenado en clasificación binaria de inglés vs español con forma de entrada (600, 200) de MFCC y 86 % de precisión. Encontramos que el biomarcador de tos es el que proporciona las características más relevantes con un 23 % de detección única y un 58 % de detección general.

Capítulo 6
METODOLOGÍA

Este trabajo tiene como objetivo crear un sistema de detección de tos robusto que sepa diferenciar entre audios con tos y audios sin tos. Además, queremos hacer una primera aproximación al diagnóstico de enfermedades usando las toses publicadas en la base de datos COUGHVID [373]. Para eso, hemos trabajado con señales de audio de varios pacientes con distintas enfermedades respiratorias, primero haciendo un preprocesado y luego pasándolas a un clasificador que usa algoritmos de *Deep Learning*, en particular mediante redes C-BiLSTM. En este capítulo explicaremos el procedimiento que hemos seguido para desarrollar ambos sistemas y los parámetros de entrenamiento utilizados.

6.1. Diseño de redes neuronales para la detección de tos

En esta sección describimos las características de la base de datos "Palencia", que constituye el corpus de referencia para nuestro análisis de detección de tos, y las etapas de preprocesamiento necesarias para adaptar las señales de audio a un formato adecuado para el entrenamiento de una red neuronal convolucional. Asimismo, presentamos la arquitectura de las redes neuronales utilizadas en cada escenario de estudio y una descripción de los conjuntos de entrenamiento y test.

6.1.1. Base de datos "Palencia"

La base de datos "Palencia" contiene grabaciones de audio de más de 24 horas de veinte pacientes del Complejo Asistencial Universitario de Palencia con distintas enfermedades respiratorias (ver Tabla 6.1). Usamos un teléfono inteligente *Android Sony Xperia Z2* para recoger los datos, guardando los archivos en formato WAV de 16 bits a 44,1 kHz [296]. Les pedimos a los pacientes que llevaran el dispositivo en sus bolsillos o bolsos como lo harían normalmente, para obtener muestras de entornos reales y ruidosos que son difíciles para la detección de la tos.

La tasa de bits o *bitrate* es la cantidad de bits por segundo que se emplean para codificar el sonido digital, y determina el tamaño final de los archivos. Se calcula multiplicando la frecuencia de muestreo, el número de canales y el número de bits usados en la decodificación. El resultado es la tasa de bits o el tamaño de los archivos por cada segundo. En nuestro caso un segundo de archivo tiene un tamaño de $44,100\ Khz \cdot 1\ canales \cdot 16\ bits = 705,6\ Kbps$. Por ejemplo, un archivo de un segundo sin comprimir con calidad CD-ROM, tiene un tamaño de $44,100\ Khz \cdot 2\ canales \cdot 16\ bits = 1411,2\ Kbps$, algo más de 1 Mb por segundo.

ID	Enfermedad	Edad	Sexo	# toses	Tipo de tos
0	Enfermedad Respiratoria Aguda	36	M	265	Aguda
1	Bronquiectasias & Asma	23	F	0	Crónica que no es EPOC
2	Sarcoidosis	37	M	163	Crónica que no es EPOC
3	Enfermedad Respiratoria Aguda	53	M	352	Aguda
4	Enfermedad Respiratoria Aguda	18	F	602	Aguda

ID	Enfermedad	Edad	Sexo	# toses	Tipo de tos
5	EPOC	79	M	767	EPOC
6	Cáncer de pulmón	84	M	921	Cáncer de pulmón
7	Neumonía bilateral	44	M	2401	Aguda
8	Cáncer de pulmón	62	M	809	Cáncer de pulmón
9	Apnea Obstructiva del Sueño	70	F	1268	
10	Bronquiectasias	63	F	508	Crónica que no es EPOC
11	EPOC & Apnea Obstructiva del Sueño	48	F	586	
12	EPOC	69	F	901	EPOC
13	Neumonía unilobular	31	F	1579	Aguda
14	Neumonía atípica	83	M	504	Aguda
15	Neumonía & EPOC	87	M	1166	
16	Tromboembolismo pulmonar	70	F	917	
17	EPOC	50	F	2033	EPOC
18	Asma	70	F	697	Crónica que no es EPOC
19	EPOC & Hamartoma pulmonar	60	M	615	
20	EPOC & Fibrosis pulmonar	67	M	1379	

Tabla 6.1: Detalles de la base de datos “Palencia”.

Usamos la herramienta de software Praat [43] y creamos un fichero de subtítulos para etiquetar manualmente las toses, que usamos para definir la verdad fundamental o *ground truth*. Si no estaba claro si un segmento de audio era tos o no, lo quitamos de la base de datos [296]. Usamos este archivo de subtítulos y Praat para hacer clips de audio separados para cada evento de sonido como vemos en las Figuras 6.1 y 6.2.

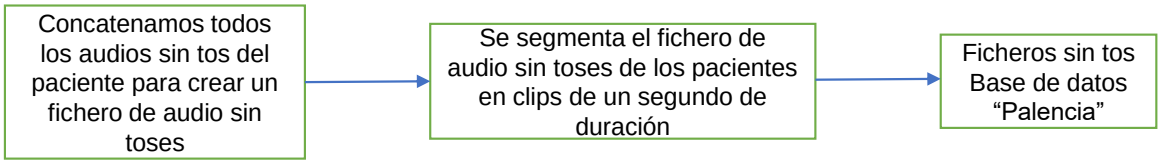


Figura 6.1: Metodología para la segmentación de los clips de audio sin tos.

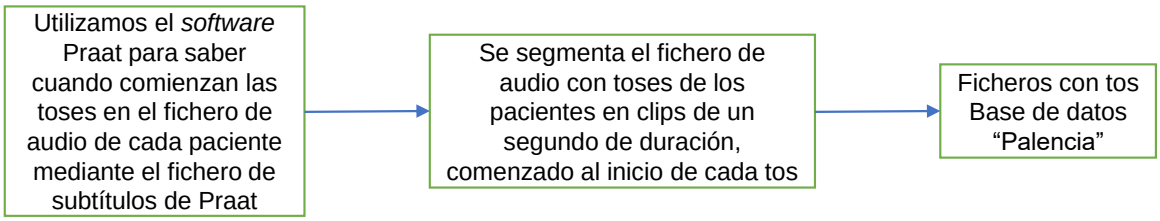


Figura 6.2: Metodología para la segmentación de los clips de audio con tos.

6.1.2. Detección de tos utilizando espectrograma normalizado sin capas LSTM

Como hemos comentado anteriormente, las redes neuronales convolucionales son muy eficientes en la clasificación de imágenes, por lo tanto, crearemos una CNN para cada uno de los estudios que vamos a realizar. Para

ilustrar la forma de las redes neuronales utilizaremos la leyenda de la Figura 6.3.

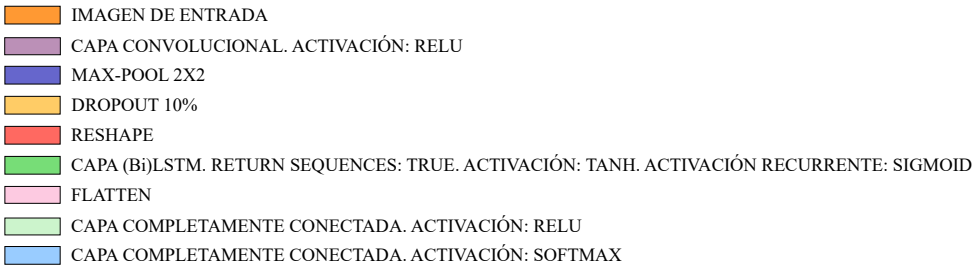


Figura 6.3: Leyenda utilizada para representar las capas de las redes neuronales utilizadas en la investigación.

Presentamos el esquema de la Figura 6.4 que resume el proceso para generar todas las imágenes utilizando el espectrograma normalizado que utilizaremos en detección de tos como entrada de las CNNs en los diferentes entrenamientos.

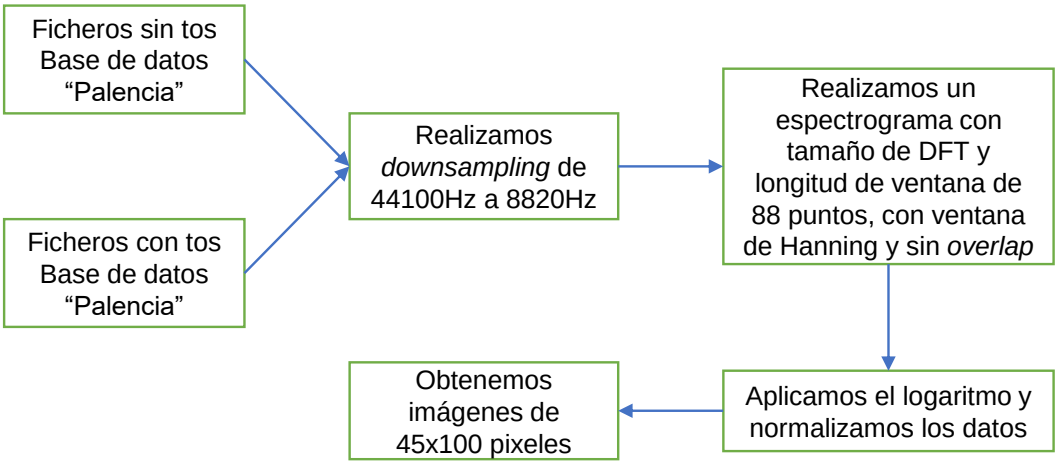


Figura 6.4: Esquema de la generación de las imágenes que nos servirán de entrada para las CNNs de detección de tos mediante espectrograma normalizado.

El espectrograma es una representación visual del espectro de frecuencia de una señal de audio en función del tiempo. La función `log_specgram` toma como entrada una señal de audio y devuelve un espectrograma en escala logarítmica. El tamaño del espectrograma generado por la función depende de la longitud del audio de entrada y de los valores de *window_size* y *step_size*. El número de columnas en el espectrograma es igual al número de segmentos en la señal de audio. El número de segmentos se puede calcular como $(len(audio) - nperseg)/(nperseg - noverlap) + 1$, donde $nperseg$ es la longitud de cada segmento y $noverlap$ es el número de puntos a superponer entre segmentos. El número de filas en el espectrograma es igual a $nperseg/2 + 1$.

```
def log_specgram(audio, sample_rate, window_size=10, step_size=10, eps=1e-10):
nperseg = int(round(window_size * sample_rate / 1e3))
noverlap = int(round(step_size * sample_rate / 1e3))
_, _, spec = signal.spectrogram(audio, fs=sample_rate, window='hann',
nperseg=nperseg, noverlap=noverlap, detrend=False)
return np.log(spec.T.astype(np.float32) + eps)
```

En nuestro caso, con una señal de audio de longitud 8820, una tasa de muestreo de 8820 Hz, un tamaño de ventana de 10 ms y un tamaño de paso de 10 ms. Se calculó que $nperseg = (10 \cdot 8820Hz)/1000 = 88200/1000 = 88,2$ y $noverlap = (0 * 8820Hz)/1000 = 0$. Por tanto el tamaño del espectrograma sería $(88/2 + 1)x((8820 - 88)/(88 - 0) + 1) = 45x100$.

En el estudio de la detección de tos hemos utilizado dos arquitecturas de redes neuronales convolucionales puras, cuyo código se puede ver en el apéndice A.1, cada una de las cuales con características interesantes en nuestro estudio.

Red neuronal convolucional A

Primero ilustramos como es la forma de la primera arquitectura que denominaremos CNN A mediante la Figura 6.5, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

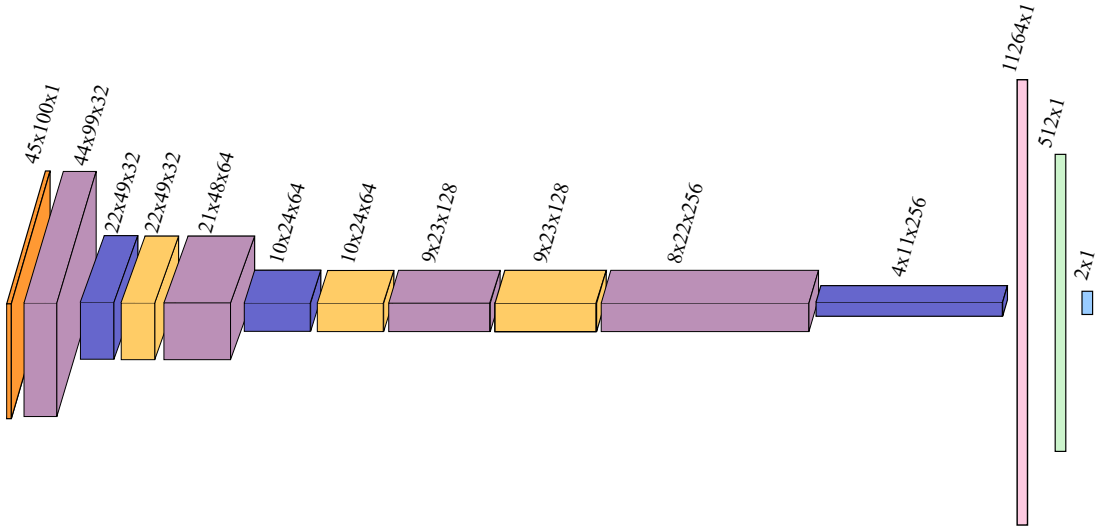


Figura 6.5: Arquitectura de la CNN A utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 4 capas convolucionales con tamaño de filtro 2x2, la leyenda se puede ver en la Figura 6.3

A continuación explicaremos como es esta arquitectura que como veremos posteriormente en 7.1.1 tiene la característica de tener más especificidad que sensibilidad. La primera capa es una capa convolucional de 32 filtros con un kernel (2, 2), la función de activación es una ReLU y admite una entrada con la forma (45, 100, 1), ya que, el tamaño de los espectrogramas logarítmicos normalizados es de (45, 100), pero la capa convolucional necesita una tercera dimensión. Seguidamente tenemos una capa de *MaxPoolling2D*, con un tamaño de *pool* de (2, 2), esta capa reduce la dimensionalidad de los datos para no tener una excesiva carga computacional. A continuación, conectamos una capa de *dropout*, la cual tiene la capacidad de reducir el sobreajuste de las redes neuronales, durante el entrenamiento, cierto número de salidas de capa se ignoran aleatoriamente o se "eliminan", en nuestro caso ignoramos aleatoriamente el 10 % de las salidas.

Todas las capas de *MaxPoolling2D* y *dropout* que vemos posteriormente tienen las mismas características que las descritas anteriormente. El número de capas óptimo, lo hemos determinado experimentalmente, viendo que con el número de capas que vemos en la Figura 6.5, es el que mejor resultados nos proporciona. También hemos comprobado experimentalmente que lo óptimo es ir doblando el número de filtros en las capas convolucionales utilizadas, esta configuración permite realizar entrenamientos de menor duración ofreciendo buenos resultados.

Vemos que tenemos una red neuronal convolucional con la estructura especificada en la Tabla 6.2.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 2	conv2*2 − 64, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 3	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	dropout = 0.1
Capa 4	conv2*2 − 256, padding = 0, stride = 1, activation = relu
	max-pooling 1/2

Tabla 6.2: Estructura de la red neuronal convolucional utilizada para detectar clips de audios con tos específica.

La capa *flatten* es una capa que aplana los datos para las capas posteriores, siendo estas capas densas. Como últimas capas, tenemos una capa densa de 512 neuronas, con la función de activación ReLU y una capa densa de 2 neuronas, ya que es el número de clases que tenemos que clasificar, con una función de activación *softmax*, la salida de la función *softmax* puede ser utilizada para representar una distribución categórica– la distribución de probabilidad sobre las diferentes posibles salidas.

Red neuronal convolucional B

La segunda arquitectura que denominaremos CNN B es la que vemos en la Figura 6.6, donde vemos que desaparecen las capas de *dropout* con respecto a la arquitectura anterior.

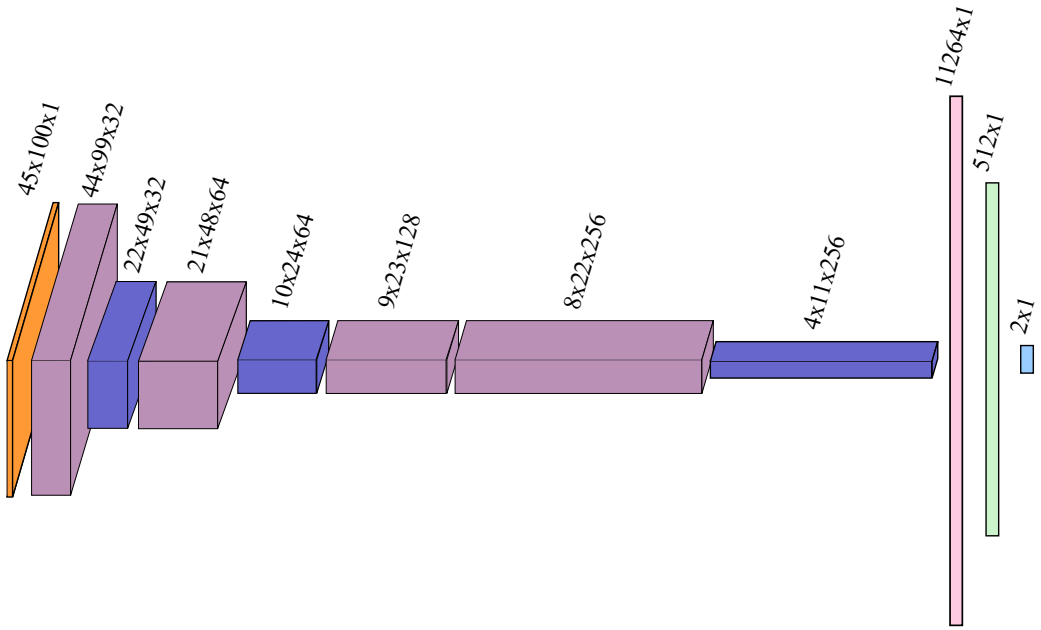


Figura 6.6: Arquitectura de la CNN B utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 4 capas convolucionales con tamaño de filtro 2x2, la leyenda se puede ver en la Figura 6.3

De manera teórica podríamos pensar que al eliminar estas capas tenderíamos a un mayor sobreajuste en la red neuronal, pero como comprobaremos posteriormente el efecto que provoca es conseguir tener más balanceada la

sensibilidad y la especificidad de la red. Esta característica diferenciadora con respecto a nuestra primera red puede ser interesante para nuestro estudio. En la Tabla 6.3 vemos la estructura de la red neuronal convolucional.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv2*2 − 64, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 3	conv2*2 − 128, padding = 0, stride = 1, activation = relu
Capa 4	conv2*2 − 256, padding = 0, stride = 1, activation = relu
	max-pooling 1/2

Tabla 6.3: Estructura de la red neuronal convolucional utilizada para detectar clips de audios con tos balanceada.

6.1.3. Detección de tos utilizando espectrograma normalizado con capas LSTM

Proponemos ahora unas redes C-LSTM profundas, que combinan una CNN y una LSTM, cuyo código podemos ver en el apéndice A.2.1. La CNN analiza el espectro de audio, mientras que LSTM predice el *frame* actual haciendo referencia a los *frames* adyacentes. Las capas convolucionales antes de las capas LSTM mejoran las características de la tos y preservan la información temporal de los audios.

Con el objetivo de realizar la unión entre las capas convolucionales y las capas LSTM, la forma mas optima de realizar esta unión, es con la función *Reshape*, ya que permite eliminar una dimensión dando una nueva forma a los datos.

El esquema de la Figura 6.4 resume el proceso para generar todas las imágenes que nos servirán como entrada de las C-LSTMs que utilizaremos en detección de tos utilizando espectrograma normalizado para los diferentes entrenamientos.

C-LSTM A-1

La primera arquitectura que denominaremos C-LSTM A-1 la ilustramos mediante la Figura 6.7, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

Esta red se basa en la CNN A descrita en la sección 6.1.2 y los datos de entrada son los mismos, ya que alimentamos la red utilizando un espectrograma normalizado. Hemos añadido después de las capas convolucionales dos capas LSTM y en las capas convolucionales también hemos hecho ligeros cambios, las características de las capas convolucionales de esta red neuronal las podemos ver en la Tabla 6.4.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 2	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 3	conv2*2 − 64, padding = 0, stride = 1, activation = relu
	dropout = 0.1
Capa 4	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, reshape = (44, 128)

Tabla 6.4: Estructura de la red neuronal convolucional A utilizando espectrograma normalizado

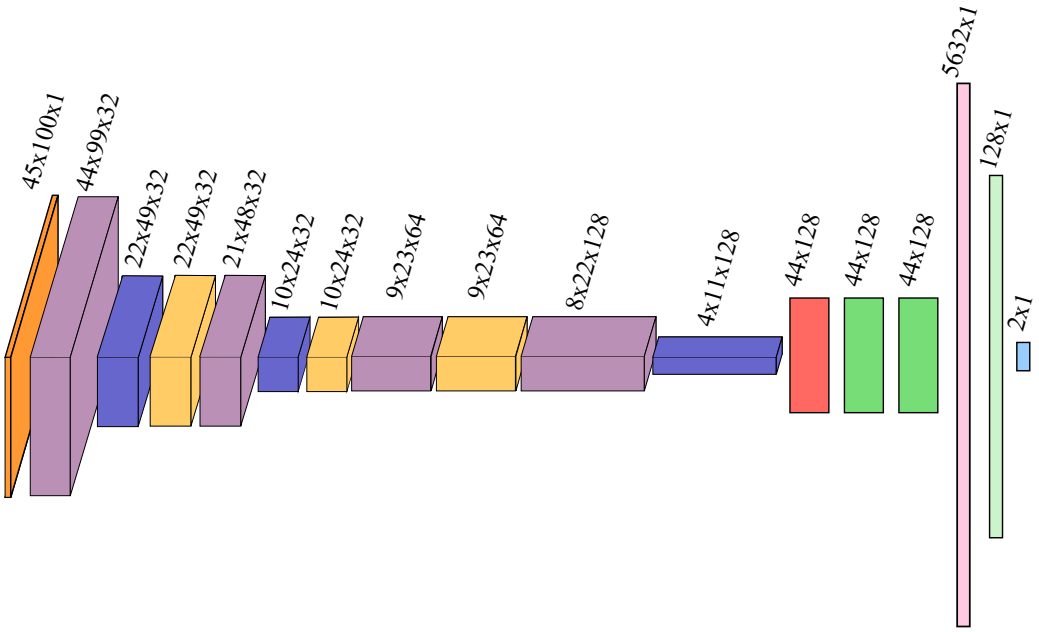


Figura 6.7: Arquitectura de la C-LSTM A-1 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 4 capas convolucionales con tamaño de filtro 2x2 y 2 capas LSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

C-LSTM B-1

A continuación ilustramos las capas de la red en la segunda arquitectura que denominaremos C-LSTM B-1 y el tamaño de los datos al pasar por cada una de estas capas en la Figura 6.8.

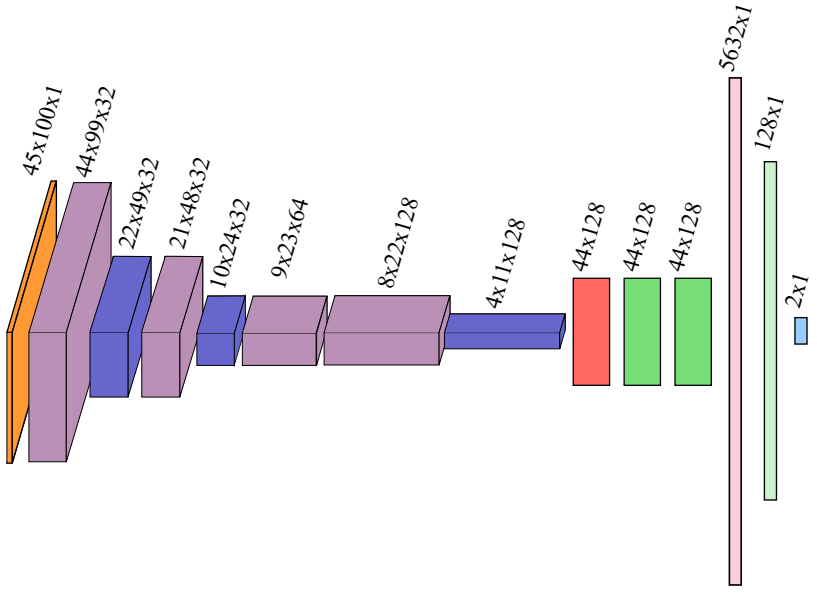


Figura 6.8: Arquitectura de la C-LSTM B-1 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 4 capas convolucionales con tamaño de filtro 2x2 y 2 capas LSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

Esta red neuronal de tipo C-LSTM esta basada en la CNN B descrita en la sección 6.1.2 y es alimentada por un espectrograma normalizado por lo que los datos de entrada son los mismos. Vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.5 y después dos capas LSTM.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 3	conv2*2 − 64, padding = 0, stride = 1, activation = relu
Capa 4	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, reshape = (44, 128)

Tabla 6.5: Estructura de la red neuronal convolucional B utilizando espectrograma normalizado.

C-LSTM C-1

Esta tercera arquitectura es una red neuronal que cambia los *kernel* de forma similar al paper *Automatic Cough Detection from Realistic Audio Recordings using C-BiLSTM with Boundary Regression* [289] buscando que a la hora de aplicar la función *Reshape* una de las dimensiones tenga valor 1 y por lo tanto simplemente elimine esa dimensión. Esta arquitectura que denominaremos C-LSTM C-1 la ilustramos mediante la Figura 6.9, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

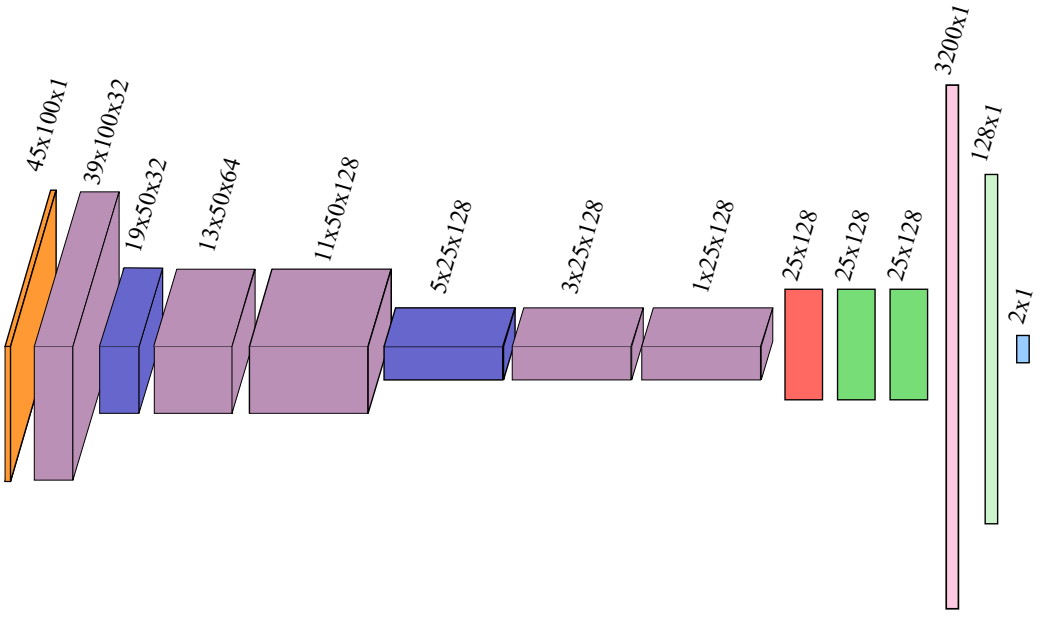


Figura 6.9: Arquitectura de la C-LSTM C-1 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 2 capas convolucionales con tamaño de filtro 7x1, 3 capas convolucionales con tamaño de filtro 3x1 y 2 capas LSTM con 100 neuronas, la leyenda se puede ver en la Figura 6.3.

En esta red neuronal de tipo C-LSTM vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.6 y posteriormente dos capas LSTM.

Capa 1	conv7*1 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv7*1 − 64, padding = 0, stride = 1, activation = relu
Capa 3	conv3*1 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 4	conv3*1 − 128, padding = 0, stride = 1, activation = relu
Capa 5	conv3*1 − 128, padding = 0, stride = 1, activation = relu
	reshape = (25, 128)

Tabla 6.6: Estructura de la red neuronal convolucional C utilizando espectrograma normalizado.

C-LSTM D-1

Esta red neuronal es la cuarta arquitectura que denominaremos C-LSTM D-1 y cambia los *kernel* de forma contraria al paper *Automatic Cough Detection from Realistic Audio Recordings using C-BiLSTM with Boundary Regression* [289]. Ilustramos a continuación las capas de la red y el tamaño de los datos al pasar por cada una de estas capas en la Figura 6.10.

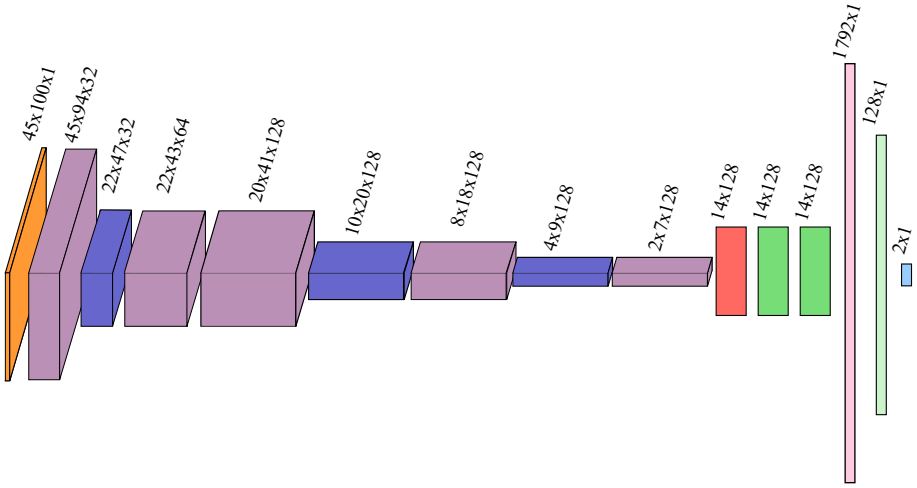


Figura 6.10: Arquitectura de la C-LSTM D-1 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 1 capa convolucional con tamaño de filtro 1x7, 1 capa convolucional con tamaño de filtro 1x5, 3 capas convolucionales con tamaño de filtro 3x3 y 2 capas LSTM con 100 neuronas, la leyenda se puede ver en la Figura 6.3.

En la Tabla 6.7 especificamos la estructura de la red neuronal convolucional con la que comienza esta red neuronal de tipo C-LSTM para seguir con dos capas LSTM.

Capa 1	conv1*7 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv1*5 − 64, padding = 0, stride = 1, activation = relu
Capa 3	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 4	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 5	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	reshape = (14, 128)

Tabla 6.7: Estructura de la red neuronal convolucional D utilizando espectrograma normalizado.

6.1.4. Detección de tos utilizando espectrograma de mel con capas LSTM

Las redes C-LSTM profundas que proponemos a continuación y cuyo código podemos ver en el apéndice A.2.2 no son alimentadas en este caso por el espectrograma normalizado sino que utilizamos el espectrograma de mel [24] debido a que tiene base en la percepción auditiva humana por lo que es útil para la identificación de contenido relevante.

Un espectrograma de mel es una representación bidimensional de la señal de audio en el plano tiempo-frecuencia, donde las frecuencias se mapean a la escala mel. La escala mel es una escala perceptual que aproxima la sensibilidad del oído humano al sonido. Las bandas de mel son segmentos de frecuencia que se emplean para discretizar el espectro de frecuencia en sub-bandas. La dimensión del espectrograma de mel generado por la función `librosa.feature.melspectrogram` dependerá de los argumentos que se especifiquen para computar la transformada de Fourier de corta duración (STFT) y la transformación a la escala mel.

El proceso para generar todas las imágenes que nos servirán como entrada de las C-LSTMs que utilizaremos en detección de tos utilizando espectrograma de mel para los diferentes entrenamientos lo vemos en el esquema de la Figura 6.11.

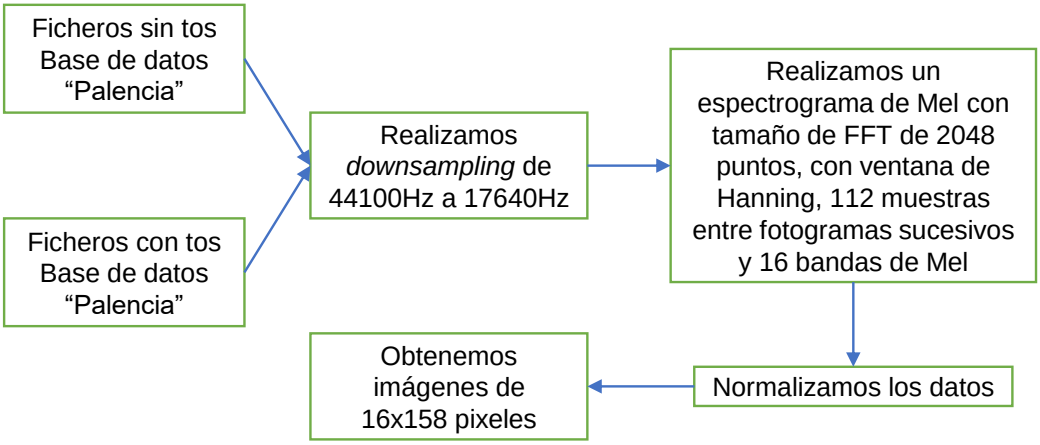


Figura 6.11: Esquema de la generación de las imágenes que nos servirán de entrada para las CNNs de detección de tos mediante espectrograma de mel.

El número de columnas en el espectrograma de mel es igual al número de segmentos en la señal de audio. El número de segmentos se puede calcular como $(len(audio)//hop\_length) + 1$. El número de filas en el espectrograma de mel es el número de bandas de Mel a generar. En nuestro caso, con una señal de audio de longitud 17640, una frecuencia de muestreo de 17640 Hz, una longitud del salto (*hop_length*) de 112 y un número de bandas de Mel a generar (*n_mels*) de 16. Calculamos que el tamaño del espectrograma de mel sería $16x((17640//112) + 1) = 16x158$.

C-LSTM A-2

La primera arquitectura que alimentamos con espectrogramas de mel la denominaremos C-LSTM A-2 y es ilustrada mediante la Figura 6.12, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

En esta red neuronal de tipo C-LSTM, basada en la CNN A descrita en la sección 6.1.2, vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.8 y después dos capas LSTM.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1

Capa 2	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 3	conv2*2 − 64, padding = 0, stride = 1, activation = relu
	dropout = 0.1
Capa 4	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	reshape = (36, 128)

Tabla 6.8: Estructura de la red neuronal convolucional A utilizando espectrograma de mel

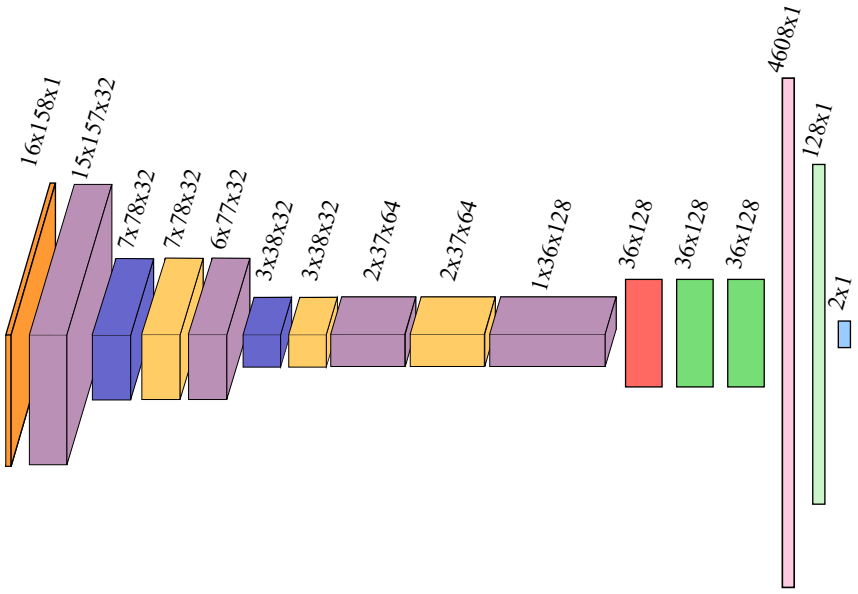


Figura 6.12: Arquitectura de la C-LSTM A-2 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma de mel de un segundo. La red consta de 4 capas convolucionales con tamaño de filtro 2x2 y 2 capas LSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

C-LSTM B-2

Ilustramos a continuación las capas de la red en la segunda arquitectura que alimentamos con espectrogramas de mel denominada C-LSTM B-2, y el tamaño de los datos al pasar por cada una de estas capas en la Figura 6.13.

En esta red neuronal de tipo C-LSTM vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.9 basada en la CNN B descrita en la sección 6.1.2 y después dos capas LSTM.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 3	conv2*2 − 64, padding = 0, stride = 1, activation = relu
Capa 4	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	reshape = (36, 128)

Tabla 6.9: Estructura de la red neuronal convolucional B utilizando espectrograma de mel.

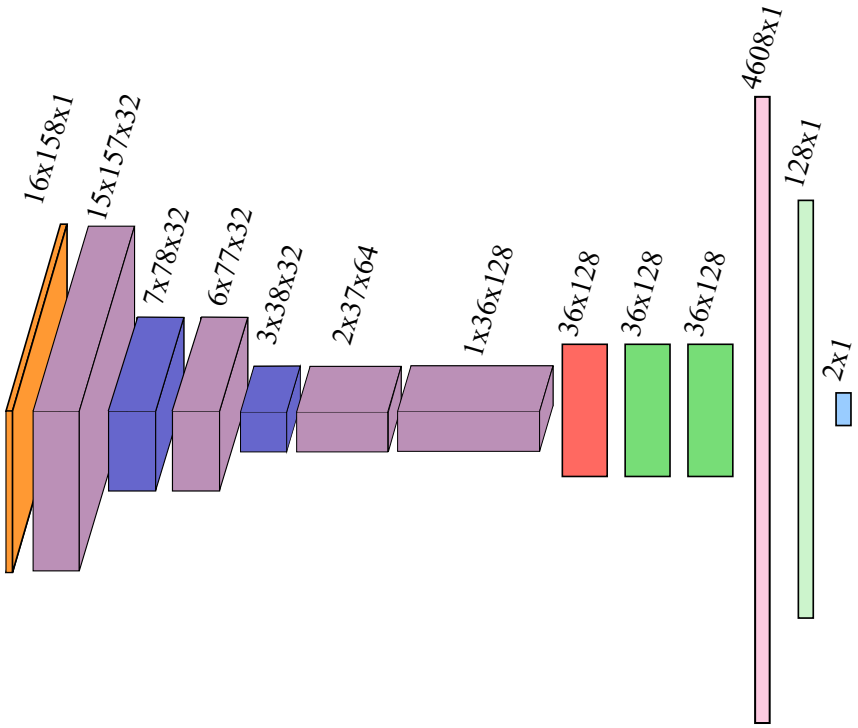


Figura 6.13: Arquitectura de la C-LSTM B-2 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma de mel de un segundo. La red consta de 4 capas convolucionales con tamaño de filtro 2x2 y 2 capas LSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

C-LSTM C-2

Esta red neuronal de tipo C-LSTM cambia los *kernel* de forma similar al paper *Automatic Cough Detection from Realistic Audio Recordings using C-BiLSTM with Boundary Regression* [289] buscando que a la hora de aplicar la función *Reshape* una de las dimensiones tenga valor 1 y por lo tanto simplemente elimine esa dimensión. Denominaremos C-LSTM C-2 a esta tercera arquitectura que ilustramos mediante la Figura 6.14, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

En esta red neuronal de tipo C-LSTM vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.10 y posteriormente dos capas LSTM.

Capa 1	conv7^*1-32, padding = 0, stride = 1, activation = relu
Capa 2	conv3^*1-64, padding = 0, stride = 1, activation = relu
Capa 3	conv3^*1-128, padding = 0, stride = 1, activation = relu
Capa 4	conv3^*1-128, padding = 0, stride = 1, activation = relu
Capa 5	conv3^*1-128, padding = 0, stride = 1, activation = relu
	reshape = (316, 128)

Tabla 6.10: Estructura de la red neuronal convolucional C utilizando espectrograma de mel.

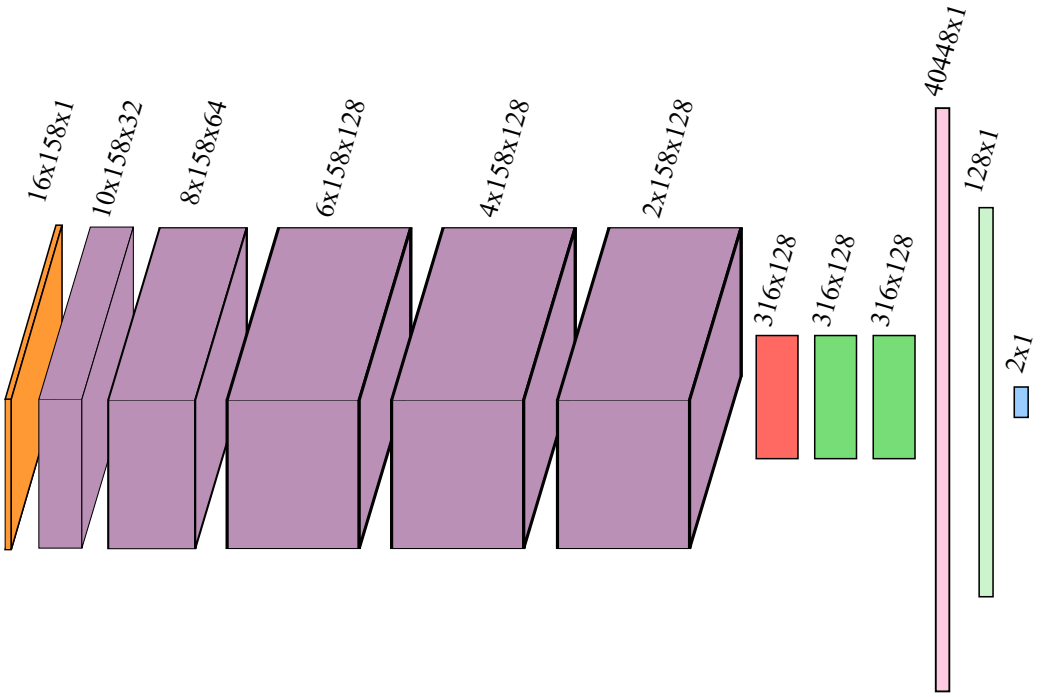


Figura 6.14: Arquitectura de la C-LSTM C-2 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma de mel de un segundo. La red consta de 2 capas convolucionales con tamaño de filtro 7x1, 3 capas convolucionales con tamaño de filtro 3x1 y 2 capas LSTM con 100 neuronas, la leyenda se puede ver en la Figura 6.3.

C-LSTM D-2

Esta red neuronal de tipo C-LSTM cambia los *kernel* de forma contraria al paper *Automatic Cough Detection from Realistic Audio Recordings using C-BiLSTM with Boundary Regression* [289]. En esta cuarta arquitectura que denominaremos C-LSTM D-2 las capas de la red y el tamaño de los datos al pasar por cada una de estas capas las ilustramos en la Figura 6.15.

En la Tabla 6.11 vemos la estructura de la red neuronal convolucional con la que comienza esta arquitectura y que continúa con dos capas LSTM.

Capa 1	conv1*7 − 32, padding = 0, stride = 1, activation = relu
Capa 2	conv1*5 − 64, padding = 0, stride = 1, activation = relu
Capa 3	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 4	conv3*3 − 128, padding = 0, stride = 1, activation = relu
Capa 5	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	reshape = (207, 128)

Tabla 6.11: Estructura de la red neuronal convolucional D utilizando espectrograma de mel.

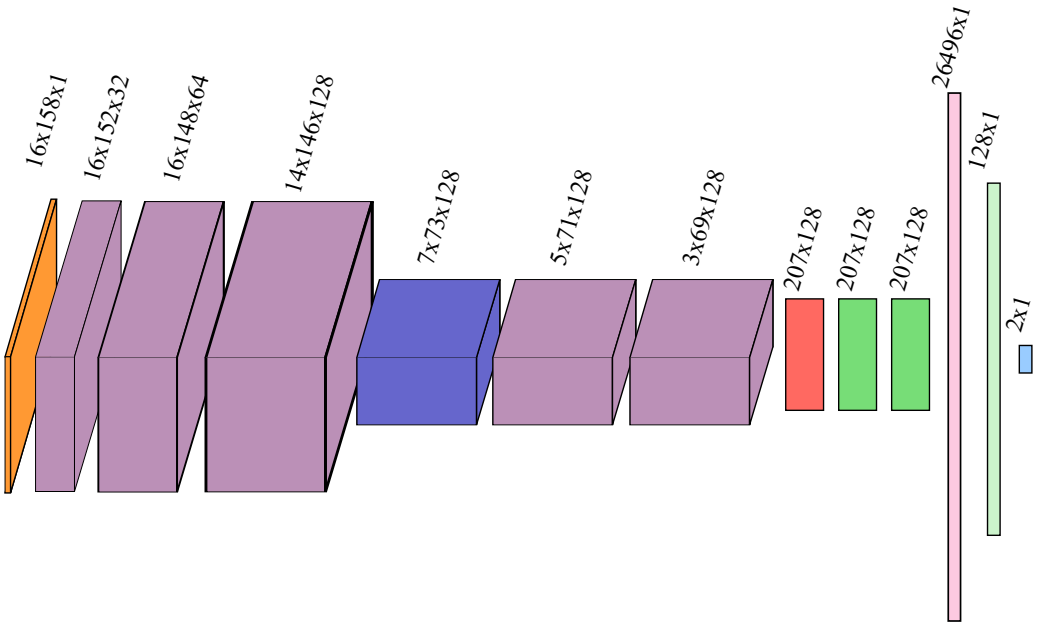


Figura 6.15: Arquitectura de la C-LSTM D-2 utilizada para detectar clips de audios con tos. La entrada a la red es un espectrograma de mel de un segundo. La red consta de 1 capa convolucional con tamaño de filtro 1x7, 1 capa convolucional con tamaño de filtro 1x5, 3 capas convolucionales con tamaño de filtro 3x3 y 2 capas LSTM con 100 neuronas, la leyenda se puede ver en la Figura 6.3.

6.1.5. Detección de tos utilizando ventanas desplazadas con capas LSTM

Las redes C-LSTM profundas que proponemos a continuación y cuyo código podemos ver en el apéndice A.2.3 no son alimentadas en este caso por el espectrograma normalizado sino que introducimos el vector del audio para posteriormente en la red neuronal ponerlo en forma matricial del tal manera que cada columna sea una ventana temporal.

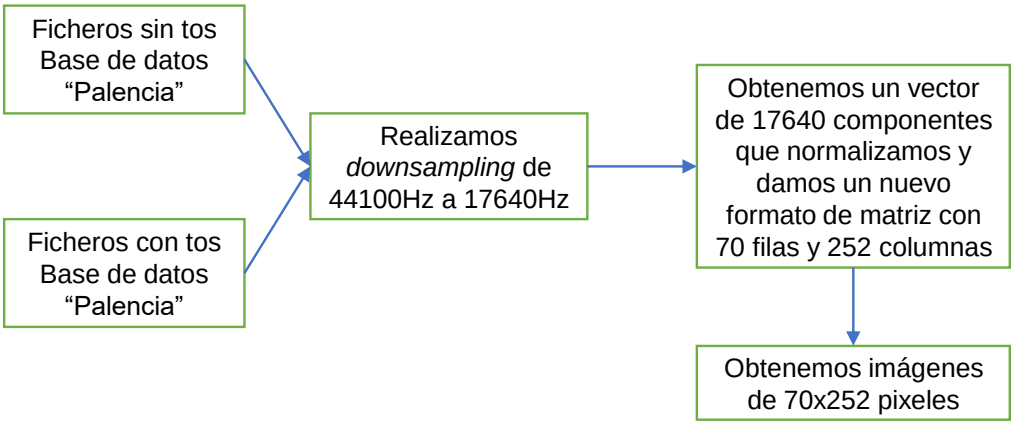


Figura 6.16: Esquema de la generación de las imágenes que nos servirán de entrada para las CNNs de detección de tos mediante ventanas desplazadas.

El esquema de la Figura 6.16 resume el proceso para generar todas las imágenes que nos servirán como en-

trada de las C-LSTMs que utilizaremos en detección de tos utilizando ventanas desplazadas para los diferentes entrenamientos.

C-LSTM A-3

La primera arquitectura que alimentamos con ventanas desplazadas la denominaremos C-LSTM A-3 y es ilustrada mediante la Figura 6.7, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 2	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 3	conv2*2 − 64, padding = 0, stride = 1, activation = relu
	dropout = 0.1
Capa 4	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
	reshape = (210, 128)

Tabla 6.12: Estructura de la red neuronal convolucional A utilizando ventanas desplazadas.

En esta red neuronal de tipo C-LSTM, basada en la CNN A descrita en la sección 6.1.2, vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.12 y después dos capas LSTM.

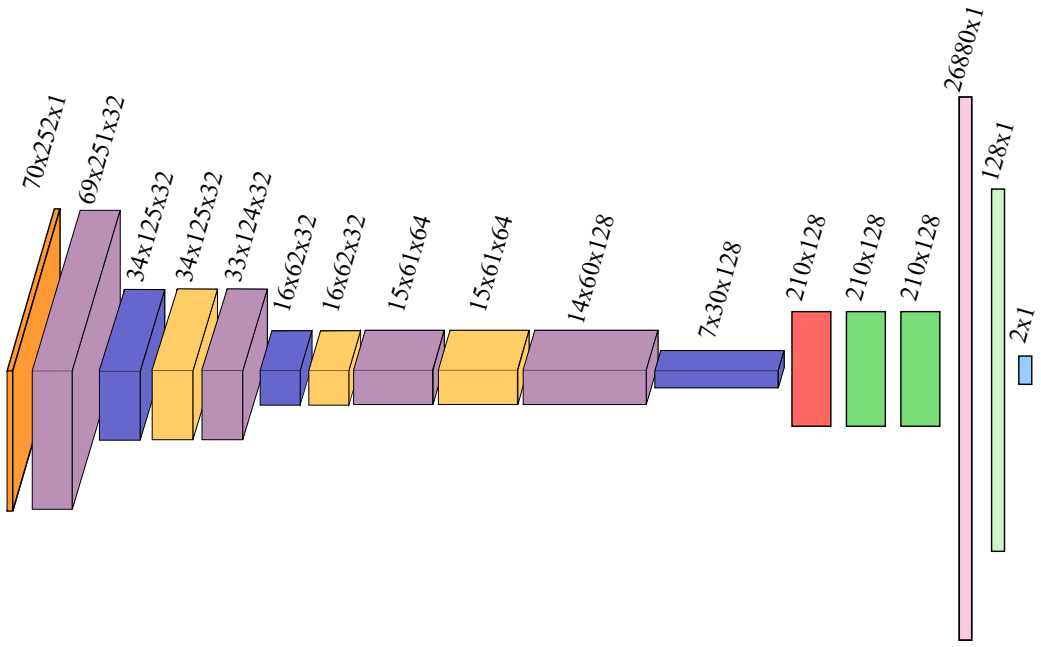


Figura 6.17: Arquitectura de la C-LSTM A-3 utilizada para detectar clips de audios con tos. La entrada a la red son las ventanas desplazadas de un segundo de audio. La red consta de 4 capas convolucionales con tamaño de filtro 2x2 y 2 capas LSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

C-LSTM B-3

Ilustramos a continuación las capas de la red en la segunda arquitectura que alimentamos con ventanas desplazadas denominada C-LSTM B-3, y el tamaño de los datos al pasar por cada una de estas capas en la Figura

6.18.

En esta red neuronal de tipo C-LSTM vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.13 basada en la CNN B descrita en la sección 6.1.2 y después dos capas LSTM.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 3	conv2*2 − 64, padding = 0, stride = 1, activation = relu
Capa 4	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
	reshape = (210, 128)

Tabla 6.13: Estructura de la red neuronal convolucional B utilizando ventanas desplazadas.

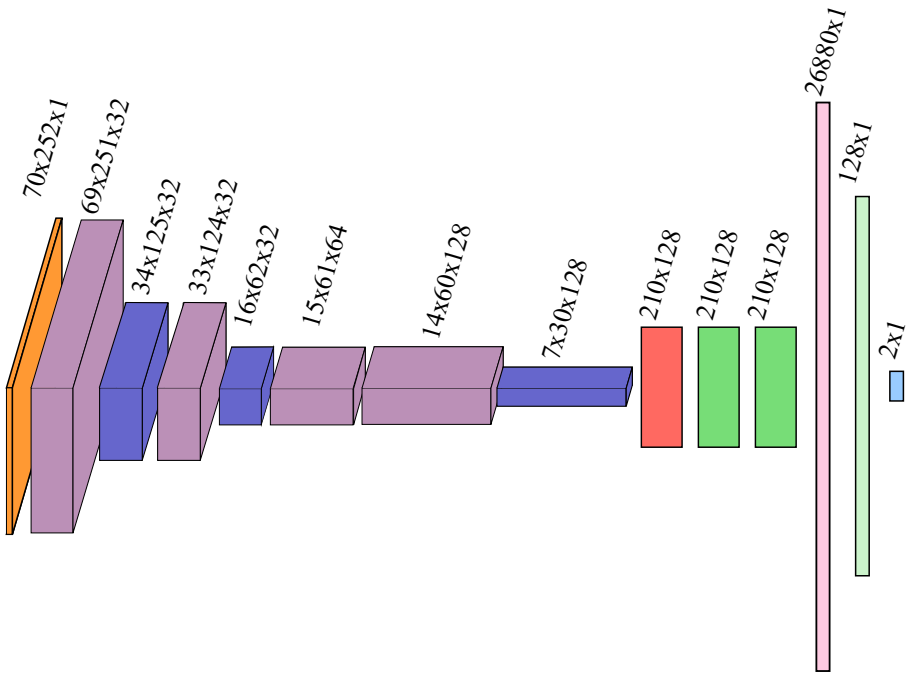


Figura 6.18: Arquitectura de la C-LSTM B-3 utilizada para detectar clips de audios con tos. La entrada a la red son las ventanas desplazadas de un segundo de audio. La red consta de 4 capas convolucionales con tamaño de filtro 2x2 y 2 capas LSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

C-LSTM C-3

Esta red neuronal de tipo C-LSTM cambia los *kernel* de forma similar al paper *Automatic Cough Detection from Realistic Audio Recordings using C-BiLSTM with Boundary Regression* [289] buscando que a la hora de aplicar la función *Reshape* una de las dimensiones tenga valor 1 y por lo tanto simplemente elimine esa dimensión.

Denominaremos C-LSTM C-3 a esta tercera arquitectura que ilustramos mediante la Figura 6.19, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

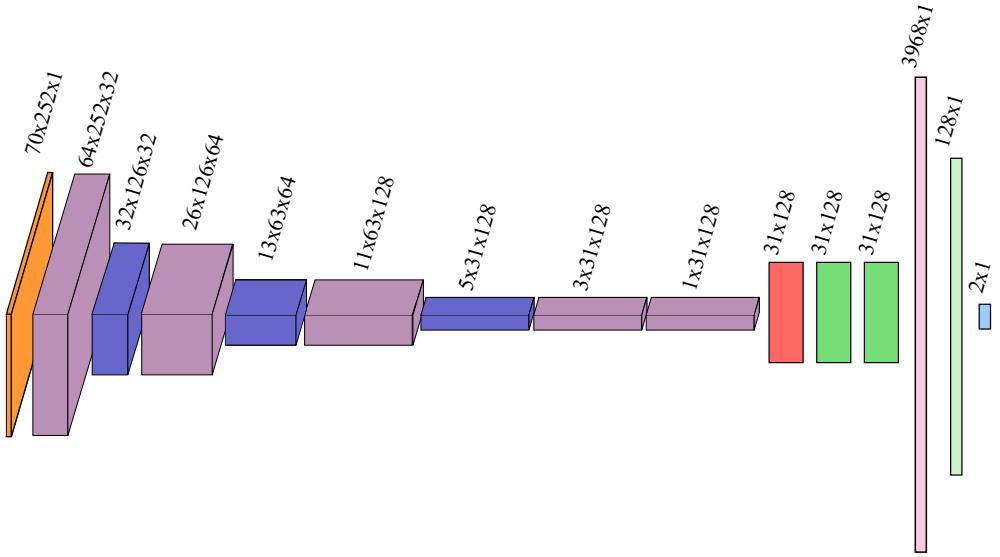


Figura 6.19: Arquitectura de la C-LSTM C-3 utilizada para detectar clips de audios con tos. La entrada a la red son las ventanas desplazadas de un segundo de audio. La red consta de 2 capas convolucionales con tamaño de filtro 7x1, 3 capas convolucionales con tamaño de filtro 3x1 y 2 capas LSTM con 100 neuronas, la leyenda se puede ver en la Figura 6.3.

En esta red neuronal de tipo C-LSTM vemos que tenemos primero una red neuronal convolucional con la estructura especificada en la Tabla 6.14 y posteriormente dos capas LSTM.

Capa 1	conv7^*1-32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv7^*1-64, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 3	conv3^*1-128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 4	conv3^*1-128, padding = 0, stride = 1, activation = relu
Capa 5	conv3^*1-128, padding = 0, stride = 1, activation = relu
	reshape = (31, 128)

Tabla 6.14: Estructura de la red neuronal convolucional C utilizando ventanas desplazadas.

C-LSTM D-3

Esta cuarta arquitectura que denominaremos C-LSTM D-3 cambia los *kernel* de forma contraria al paper *Automatic Cough Detection from Realistic Audio Recordings using C-BiLSTM with Boundary Regression* [289]. Ilustramos a continuación las capas de esta red neuronal y el tamaño de los datos al pasar por cada una de estas capas en la Figura 6.20.

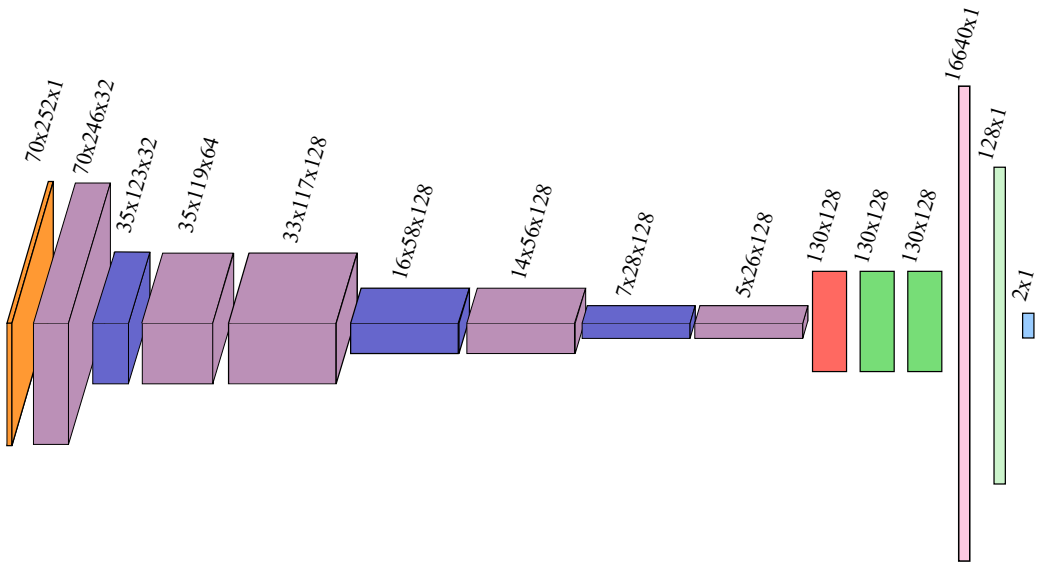


Figura 6.20: Arquitectura de la C-LSTM D-3 utilizada para detectar clips de audios con tos. La entrada a la red son las ventanas desplazadas de un segundo de audio. La red consta de 1 capa convolucional con tamaño de filtro 1x7, 1 capa convolucional con tamaño de filtro 1x5, 3 capas convolucionales con tamaño de filtro 3x3 y 2 capas LSTM con 100 neuronas, la leyenda se puede ver en la Figura 6.3.

En la Tabla 6.15 vemos la estructura de la red neuronal convolucional con la que comienza esta arquitectura y que continúa con dos capas LSTM.

Capa 1	conv1*7 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 2	conv1*5 − 64, padding = 0, stride = 1, activation = relu
Capa 3	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 4	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2
Capa 5	conv3*3 − 128, padding = 0, stride = 1, activation = relu
	reshape = (130, 128)

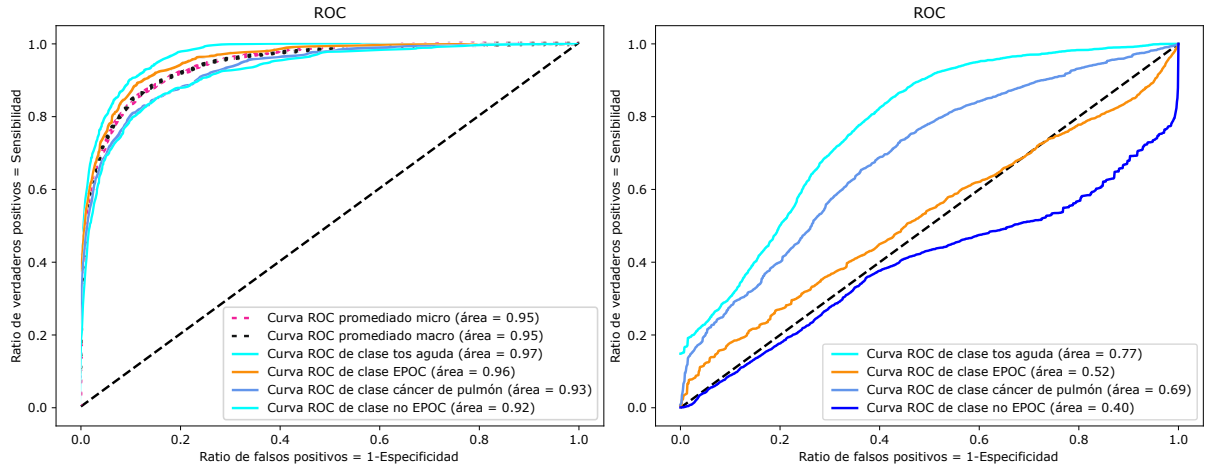
Tabla 6.15: Estructura de la red neuronal convolucional D utilizando ventanas desplazadas.

6.1.6. Características de los entrenamientos para la detección de tos

En esta sección explicamos cómo son los conjuntos de entrenamiento y test. Para determinar los conjuntos de entrenamiento y test, consideramos el objetivo de entrenar una red neuronal que clasifique los clips de audio en dos clases: con tos y sin tos. Previamente, balanceamos las clases para evitar el sobreentrenamiento de la red para alguna de ellas. También, eliminamos el sesgo potencial de tener pacientes repetidos en ambos conjuntos, asegurándonos de que los pacientes del conjunto de test no estén en el conjunto de entrenamiento. Un ejemplo de lo que ocurre si no hacemos esta separación se muestra a continuación.

Si entrenamos un clasificador de enfermedades con pacientes en ambos conjuntos, los resultados que se observan en la Figura 6.21a son aparentemente buenos, pero no son fiables, porque la red neuronal aprende la “firma

vocal" de los pacientes como una ayuda para el diagnóstico a partir de cada clip de tos. En cambio, si hacemos la separación adecuada de pacientes entre los conjuntos, como se ve en la Figura 6.21b, los resultados empeoran notablemente.



(a) Curvas ROC sin *k-folds*.

(b) Curvas ROC con *5-fold cross-validation*.

Figura 6.21: Curvas ROC en la clasificación de enfermedades con las bases de datos "Palencia", "Edimburgo" y "Glasgow" [198].

Para estimar el rendimiento del modelo, aplicaremos las siguientes técnicas de validación cruzada: *10-fold cross-validation*, *5-fold cross-validation* y *3-fold cross-validation*. Además nos referiremos a los pacientes de la base de datos "Palencia" con los que trabajamos para realizar este estudio por su ID, indicado en la Tabla 6.1.

Ficheros	4806	4066	3158	3766	3552	3504	3748	3594	3196	3474
***Fold* 0**	**7**	17	13	14, 20	9, 10	11, 15	3, 4, 6	0, 16, 19	12, 18	2, 5, 8
***Fold* 1**	7	**17**	13	14, 20	9, 10	11, 15	3, 4, 6	0, 16, 19	12, 18	2, 5, 8
***Fold* 2**	7	17	**13**	14, 20	9, 10	11, 15	3, 4, 6	0, 16, 19	12, 18	2, 5, 8
***Fold* 3**	7	17	13	**14, 20**	9, 10	11, 15	3, 4, 6	0, 16, 19	12, 18	2, 5, 8
***Fold* 4**	7	17	13	14, 20	**9, 10**	11, 15	3, 4, 6	0, 16, 19	12, 18	2, 5, 8
***Fold* 5**	7	17	13	14, 20	9, 10	**11, 15**	3, 4, 6	0, 16, 19	12, 18	2, 5, 8
***Fold* 6**	7	17	13	14, 20	9, 10	11, 15	**3, 4, 6**	0, 16, 19	12, 18	2, 5, 8
***Fold* 7**	7	17	13	14, 20	9, 10	11, 15	3, 4, 6	**0, 16, 19**	12, 18	2, 5, 8
***Fold* 8**	7	17	13	14, 20	9, 10	11, 15	3, 4, 6	0, 16, 19	**12, 18**	2, 5, 8
***Fold* 9**	7	17	13	14, 20	9, 10	11, 15	3, 4, 6	0, 16, 19	12, 18	**2, 5, 8**

Tabla 6.16: Descripción de los *folds* en el *10-fold cross-validation* utilizado para entrenar la red en la detección de tos.

La Tabla 6.16 muestra los *folds* del *10-fold cross-validation*. Los pacientes del conjunto de test se resaltan con un color de fondo marrón arena y el número de clips de audio de tos del conjunto de test se indica en la fila superior. Los demás pacientes forman el conjunto de entrenamiento de cada *fold*.

Ficheros	7572	7394	7238	7390	7270
***Fold* 0**	**0, 3, 5, 7**	2, 4, 12, 17	10, 13, 16, 19	6, 8, 11, 20	9, 14, 15, 18
***Fold* 1**	0, 3, 5, 7	**2, 4, 12, 17**	10, 13, 16, 19	6, 8, 11, 20	9, 14, 15, 18
***Fold* 2**	0, 3, 5, 7	2, 4, 12, 17	**10, 13, 16, 19**	6, 8, 11, 20	9, 14, 15, 18
***Fold* 3**	0, 3, 5, 7	2, 4, 12, 17	10, 13, 16, 19	**6, 8, 11, 20**	9, 14, 15, 18
***Fold* 4**	0, 3, 5, 7	2, 4, 12, 17	10, 13, 16, 19	6, 8, 11, 20	**9, 14, 15, 18**

Tabla 6.17: Descripción de los *folds* en el *5-fold cross-validation* utilizado para entrenar la red en la detección de tos.

Las Tablas 6.17 y 6.18 muestran los *folds* del *5-fold cross-validation* y del *3-fold cross-validation*, respectivamente. Se presentan de forma similar a la Tabla 6.16, pero con un color de fondo azul turquesa para los pacientes del conjunto de test.

Ficheros	12210	12178	12476
***Fold* 0**	**3, 7, 11, 12, 15, 18**	4, 5, 9, 14, 16, 17	0, 2, 6, 8, 10, 13, 19, 20
***Fold* 1**	3, 7, 11, 12, 15, 18	**4, 5, 9, 14, 16, 17**	0, 2, 6, 8, 10, 13, 19, 20
***Fold* 2**	3, 7, 11, 12, 15, 18	4, 5, 9, 14, 16, 17	**0, 2, 6, 8, 10, 13, 19, 20**

Tabla 6.18: Descripción de los *folds* en el *3-fold cross-validation* utilizado para entrenar la red en la detección de tos.

6.2. Diseño de redes neuronales para la detección de enfermedades

En esta sección describimos las características de la base de datos COUGHVID [373], que contiene muestras de audio de tos que sirven como base de datos para nuestro análisis de detección de enfermedades mediante técnicas de aprendizaje profundo. Explicamos cómo preprocesamos las señales de audio para obtener un formato adecuado para el entrenamiento de una red neuronal convolucional y describimos la arquitectura de las redes neuronales empleadas en cada caso de estudio.

6.2.1. Base de datos COUGHVID

La base de datos COUGHVID [373] puede estar en los formatos WebM [374] u Ogg [375], respectivamente con la extensión `.webm` y `.ogg`. En todos los casos, el códec de audio es Opus [367], por lo que usa un códec con perdidas lo cual tiene ventajas y desventajas que podemos ver en la Tabla 6.19.

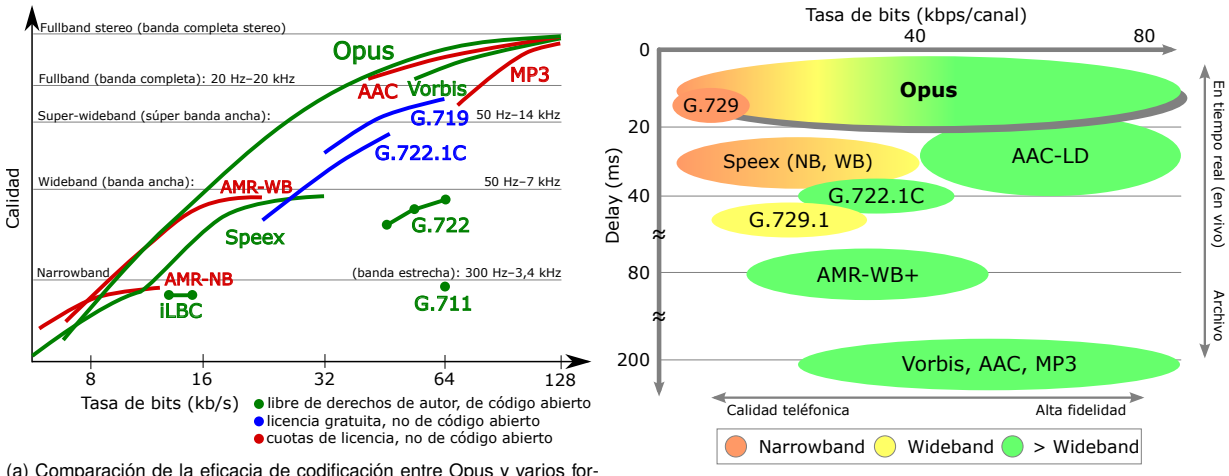
Entre los códecs con pérdida, el mejor rendimiento lo tiene el códec propuesto, según se muestra en la Figura 6.22, salvo por los códecs de última generación basados en IA que vimos en la sección 2.1.2. Utiliza una frecuencia de muestreo de 48 kHz y opera en modo de tasa de bits variable (VBR). En más del 40 % de las grabaciones, la tasa de bits efectiva es de 48 kbit/s, que corresponde al rango de ancho de banda *fullband* (48-64 kbit/s), que es el más alto para grabaciones mono. La razón de tener un *bitrate* efectivo más bajo para el resto de las grabaciones es la presencia de largos periodos de silencio, lo que de hecho es deseable dada la aplicación.

Esta base de datos tiene 859 ficheros de extensión `.ogg` y 19213 ficheros `.webm` que convertimos utilizando `ffmpeg` [377] a la extensión `.wav` por lo que tenemos un total de 20.072 ficheros de audios de tos. Estos audios nos serán útiles para buscar clasificarlos por la etiqueta COVID-19, *healthy* y *symptomatic* de la variable *status* o por la etiqueta *dry* y *wet* de la variable *cough_type* que nos ofrece la tabla `metadata_compiled.csv`.

Tipos de formato	Ventajas	Contras	Formatos
Sin comprimir	Sin pérdida de datos. Compatible con otro software.	Requiere de mucho espacio de almacenamiento.	WAV y AIFF LPCM BWF
Comprimido sin pérdidas	Reduce el tiempo de procesamiento. Conserva los datos y logra una buena tasa de compresión de alrededor de la mitad del tamaño original.	Los ficheros siguen siendo grandes comparados con el formato con pérdida.	FLAC ALAC Monkey's Audio WavPack
Comprimido con pérdida	Tamaño de fichero pequeño con una tasa de compresión de una décima o una quinta parte del tamaño original. Formatos de audio más usados por los consumidores.	Elimina información de audio "imperceptible" usando psicoacústica.	MP3 AAC Ogg Vorbis Opus EVS

Tabla 6.19: Ventajas y desventajas de los tipos de formatos de audio.

La variable *status* nos da la información de si el paciente autoinforma que ha sido diagnosticado con COVID-19, que tiene síntomas pero sin diagnóstico (sintomático), o que está sano (saludable). Después de analizar los datos, vemos que con la variable *status* obtenemos que 1010 grabaciones de tos con COVID-19, 1742 sintomáticas, 8562 saludables y 8758 sin etiquetar. Por lo que para balancear los entrenamientos utilizaremos 1010 grabaciones de tos de cada clase.



(a) Comparación de la eficacia de codificación entre Opus y varios formatos populares.

(b) Comparación de latencia entre varios codecs a distintas tasas de bits.

Figura 6.22: Calidad de varios códecs en función de la tasa de bits [376].

De las variables proporcionadas por los expertos *cough_type* y *diagnosis* son las que más nos interesan. La variable *diagnosis* es la impresión del experto sobre el estado del paciente, puede ser una infección del tracto respiratorio superior o inferior, una enfermedad obstructiva (asma, EPOC, etc), COVID-19, o una tos saludable pero como vemos en la Tabla 6.20 hay menos datos etiquetados que con la variable *status* y por este motivo no la utilizamos.

Tipo de tos	Experto 1	Experto 2	Experto 3
COVID-19	279	285	1
Tos saludable	259	67	199
Tracto respiratorio superior	23	183	364
Tracto respiratorio inferior	244	173	194
Enfermedad obstructiva	15	112	35
Sin etiquetar	19252	19252	19279

Tabla 6.20: Impresión de los expertos sobre el tipo de tos según la variable *diagnosis* del fichero *metadata_compiled.csv*

La variable *cough_type* nos indica la impresión del experto sobre el tipo de tos pudiendo ser tos húmeda (productiva), tos seca, desconocida o también puede estar sin etiquetar como se muestra en la Tabla 6.21. Por lo que para balancear los entrenamientos utilizaremos 473 grabaciones de tos de cada clase, ya que, tenemos 473 grabaciones donde al menos un experto dice que es tos húmeda (productiva), sin embargo de tos seca tenemos muchas más grabaciones.

Tipo de tos	Experto 1	Experto 2	Experto 3
Húmeda	72	133	291
Seca	425	600	358
Desconocida	323	86	146
Sin etiquetar	19252	19253	19277

Tabla 6.21: Impresión de los expertos sobre el tipo de tos según la variable *cough_type* del fichero *metadata_compiled.csv*

6.2.2. Detección de tos utilizando espectrograma normalizado con capas BiLSTM

Presentamos el esquema de la Figura 6.23 que resume el proceso para generar todas las imágenes utilizando el espectrograma normalizado que utilizaremos en detección de enfermedades como entrada de las C-BiLSTMs en los diferentes entrenamientos.

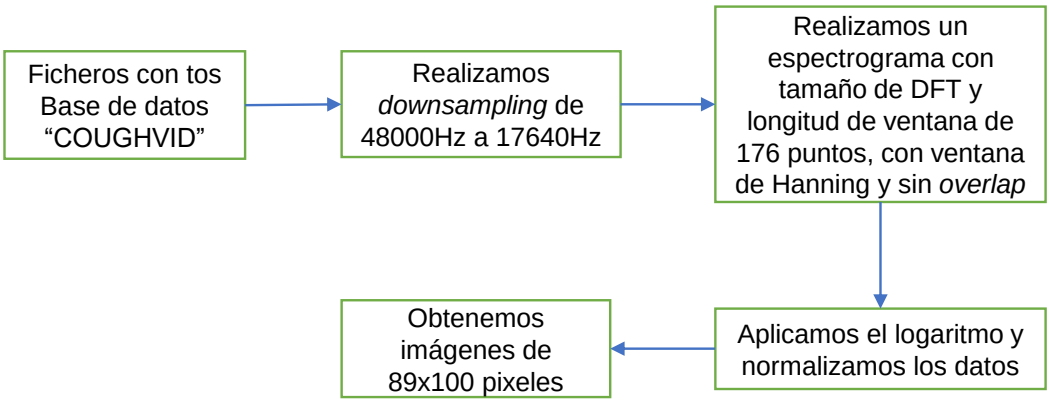


Figura 6.23: Esquema de la generación de las imágenes que nos servirán de entrada para las C-BiLSTMs utilizada para detectar enfermedades en clips de audios con tos mediante espectrograma normalizado.

Las redes C-BiLSTM profundas que proponemos a continuación y cuyo código podemos ver en el apéndice

A.3.1 buscan una capacidad más fuerte usando la estructura bidireccional de BiLSTM para juzgar respectivamente el marco actual en las direcciones hacia adelante y hacia atrás [289].

Utilizando la base de datos COUGHVID sin *transfer learning*

Primero ilustramos como es la forma de la primera arquitectura que denominaremos C-BiLSTM A mediante la Figura 6.24, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

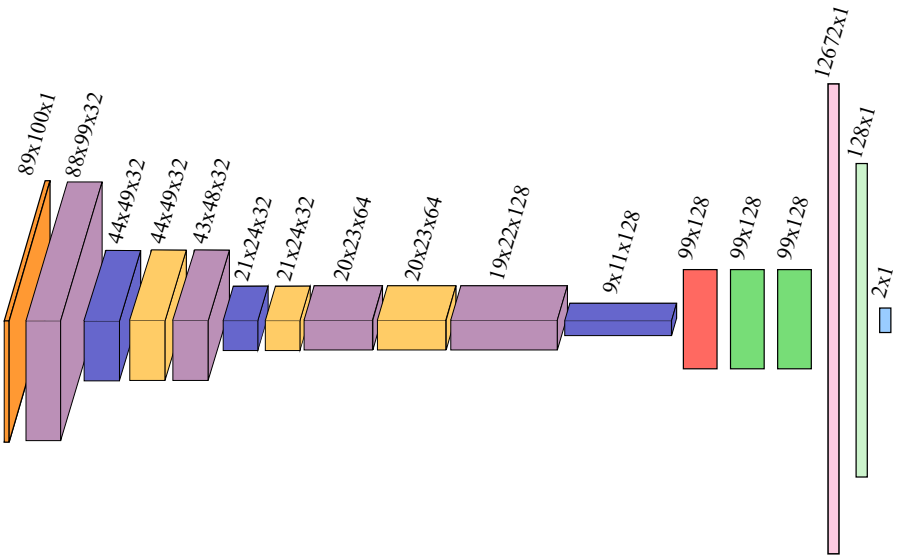


Figura 6.24: Arquitectura de la C-BiLSTM A utilizada para detectar enfermedades en clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 4 capas convolucionales con tamaño de filtro 2x2 y 2 capas BiLSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

Esta red se basa en la C-LSTM A-1 descrita en la sección 6.1.3 aunque los datos de entrada no son los mismos, ya que también alimentamos la red utilizando un espectrograma normalizado pero no es del mismo tamaño. Hemos modificado las dos capas LSTM cambiándolas por capas BiLSTM. En esta red neuronal de tipo C-BiLSTM vemos que tenemos primero una red neuronal convolucional en la que las características de las capas convolucionales de esta red neuronal las podemos ver en la Tabla 6.22.

Capa 1	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 2	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 3	conv2*2 − 64, padding = 0, stride = 1, activation = relu
	dropout = 0.1
Capa 4	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, reshape = (99, 128)

Tabla 6.22: Estructura de la red neuronal convolucional

Utilizando la base de datos COUGHVID con *transfer learning*

A continuación ilustramos las capas de la red en la segunda arquitectura que denominaremos C-BiLSTM B y el tamaño de los datos al pasar por cada una de estas capas en la Figura 6.25.

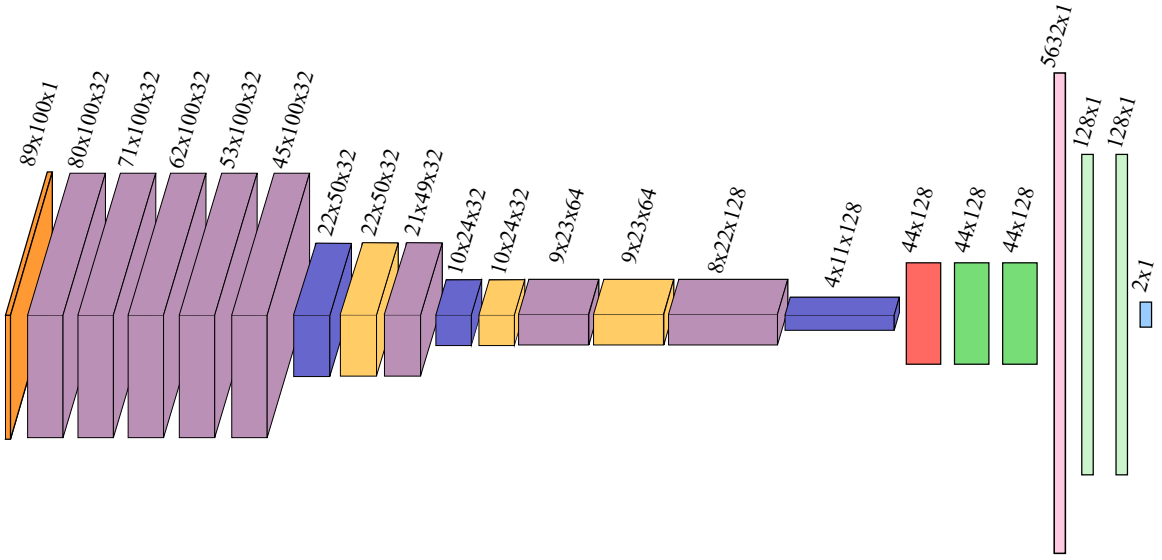


Figura 6.25: Arquitectura de la C-BiLSTM B utilizada para detectar enfermedades en clips de audios con tos. La entrada a la red es un espectrograma STFT de un segundo. La red consta de 8 capas convolucionales las 4 primeras con tamaño de filtro 10x1, después una capa con tamaño de filtro 9x1 y por último 3 capas convolucionales con tamaño de filtro 2x2, y 2 capas BiLSTM con 128 neuronas, la leyenda se puede ver en la Figura 6.3.

Esta red neuronal de tipo C-BiLSTM esta basada en la C-LSTM A-1 descrita en la sección 6.1.3 pero en su versión BiLSTM reutilizando la red pre-entrenada sin capa inicial y final. Esta nueva red funciona como un extractor de *features* fijas para realizar la tarea de detectar enfermedades. Fijamos el peso de ciertas capas durante el entrenamiento y afinamos el resto para responder al problema. Esta red también es alimentada por un espectrograma normalizado aunque los datos de entrada no son del mismo tamaño que en la red C-LSTM A-1 en la que esta basada. En esta red neuronal de tipo C-BiLSTM vemos que tenemos primero una red neuronal convolucional donde las características de las capas convolucionales de esta red neuronal las podemos ver en la Tabla 6.23.

Capa 1	conv10*1 − 32, padding = 0, stride = 1, activation = relu
Capa 2	conv10*1 − 32, padding = 0, stride = 1, activation = relu
Capa 3	conv10*1 − 32, padding = 0, stride = 1, activation = relu
Capa 4	conv10*1 − 32, padding = 0, stride = 1, activation = relu
Capa 5	conv9*1 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 6	conv2*2 − 32, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, dropout = 0.1
Capa 7	conv2*2 − 64, padding = 0, stride = 1, activation = relu
	dropout = 0.1
Capa 8	conv2*2 − 128, padding = 0, stride = 1, activation = relu
	max-pooling 1/2, reshape = (44, 128)

Tabla 6.23: Estructura de la red neuronal convolucional

6.2.3. Detección de tos utilizando espectrograma de mel con capas BiLSTM

La red C-LSTM profunda que proponemos a continuación y cuyo código podemos ver en el apéndice A.3.2 no es alimentada en este caso por el espectrograma normalizado sino que utilizamos el espectrograma de mel [24] debido a que tiene base en la percepción auditiva humana por lo que es útil para la identificación de contenido relevante.

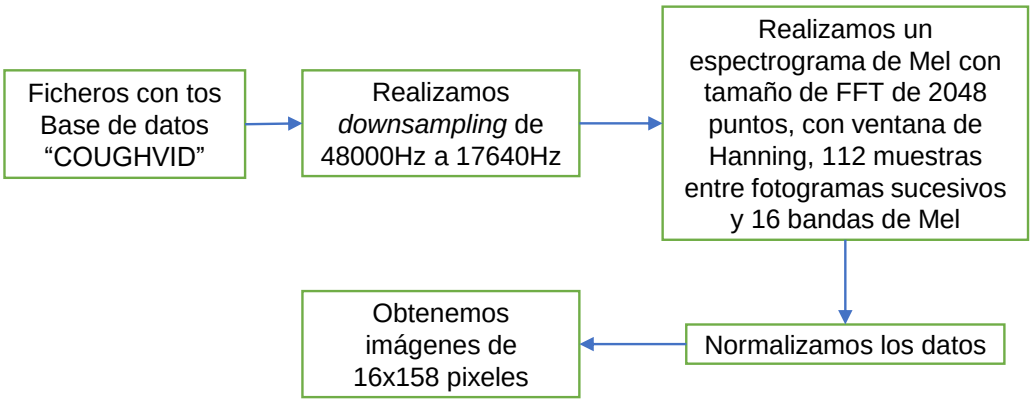


Figura 6.26: Esquema de la generación de las imágenes que nos servirán de entrada para las C-BiLSTMs utilizada para detectar enfermedades en clips de audios con tos mediante espectrograma de mel.

El proceso para generar todas las imágenes que nos servirán como entrada de la C-BiLSTMs que utilizaremos en detección de enfermedades utilizando espectrograma de mel como entrada de la C-BiLSTM para los diferentes entrenamientos lo vemos en el esquema de la Figura 6.26.

Esta red se basa en la C-LSTM A-2 descrita en la sección 6.1.4 donde hemos modificado las dos capas LSTM cambiándolas por capas BiLSTM. Aunque mantenemos las características de las capas convolucionales de esta red neuronal por lo que las podemos ver en la Tabla 6.22.

Debido a que ni el tamaño de los datos de entrada, ni como se modifica el tamaño de los datos dentro de la red cambia, esta arquitectura que alimentamos con espectrogramas de mel es ilustrada mediante la Figura 6.12, donde podemos observar las capas de la red y el tamaño de los datos al pasar por cada una de ellas.

6.3. Parámetros de entrenamiento

La redes neuronales que hemos definido en este capítulo, deben completarse definiendo una función de pérdida, un optimizador y las métricas de rendimiento. Esto se llama modelo de “compilación”.

La función de pérdida que usamos es *categorical crossentropy* 4.2.1, que es la más adecuada para problemas de clasificación multiclase con objetivos codificados en *one-hot*. Esta función es el primer parámetro requerido para compilar el modelo.

El optimizador que elegimos es AdaMax, que se explica en la sección 4.2.2. Este optimizador tiene tres parámetros configurables: α, β_1 y β_2. Los valores que asignamos a estos parámetros son $\alpha = 0,002$, $\beta_1 = 0,9$ y $\beta_2 = 0,999$ [249], [378].

La métrica de rendimiento que nos interesa es la exactitud (*accuracy*) de la validación, que nos indica el porcentaje de aciertos de la clasificación en cada época del entrenamiento. También observamos la pérdida (*loss*) de la validación en cada época por defecto. Esta métrica es especialmente importante, ya que guardamos el modelo en la época de entrenamiento que tenga el mínimo de perdida en la validación, aplicando así la técnica de regularización de parada temprana 4.2.3.

Hemos determinado experimentalmente que 50 épocas son suficientes para obtener resultados óptimos. El tamaño de *batch* que usamos es 128 para la clasificación de clips de audio con tos y sin tos, y 32 para la clasificación de clips de audio con tos según la enfermedad subyacente. Además, reservamos el 20 % del conjunto de entrenamiento para realizar la validación.

Capítulo 7

RESULTADOS Y DISCUSIÓN

En este capitulo discutiremos los resultados que hemos obtenido realizando los diferentes experimentos, en los que primeramente buscamos detectar clips de audios con tos entrenando redes neuronales convolucionales, C-LSTM y C-BiLSTM, y posteriormente buscamos clasificar clips de tos en distintos tipos de tos.

Primero recordaremos los resultados expuestos en el TFG [198], para a continuación exponer los nuevos experimentos.

7.1. Detección de tos utilizando validación cruzada

En esta sección observaremos los resultados obtenidos buscando detectar clips de audios con tos utilizando distintos modelos de redes neuronales evaluados mediante la técnica de validación cruzada para detectar el sobreajuste. Los modelos de redes neuronales utilizados son CNN, C-LSTM y C-BiLSTM, y los datos de entrada a estos modelos de redes neuronales son espectrogramas normalizados, espectrogramas de mel y ventanas desplazadas.

7.1.1. Rendimiento de las redes neuronales convolucionales utilizando espectrograma normalizado

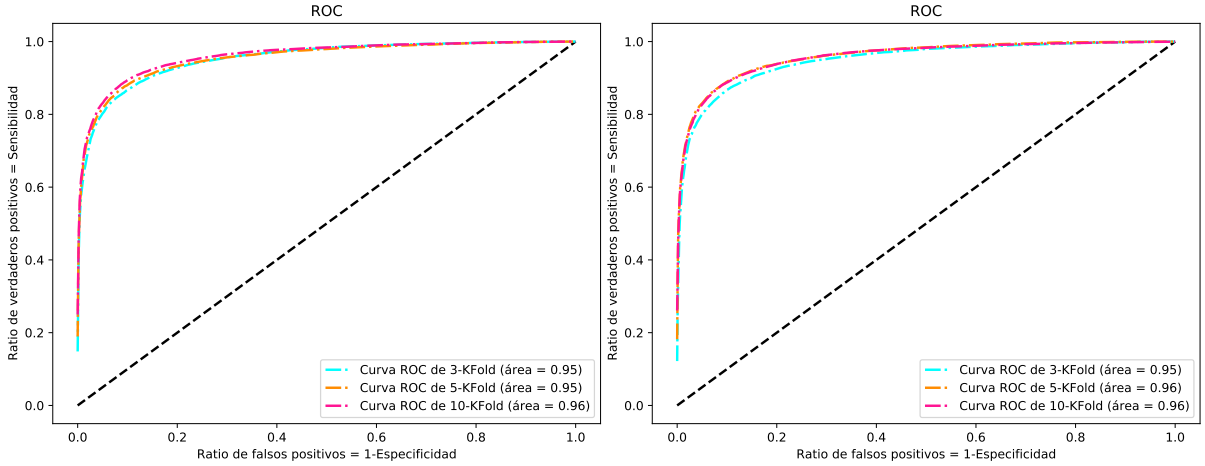
Para la detección de clips de audio con tos frente a clips de audio sin tos hemos realizado experimentos con dos arquitecturas diferentes de redes neuronales convolucionales alimentadas con espectrogramas normalizados, descritas en la sección 6.1.2. Los resultados obtenidos en estas cuatro redes neuronales se han agrupado en la Tabla 7.1, además en la Figura 7.1 se presentan las curvas ROC asociadas a las dos CNN.

Vamos a comenzar viendo los resultados que obtenemos al entrenar la primera red neuronal convolucional, denominada CNN A, que hemos visto en la Figura 6.5. Lo primero que observamos es que siempre vemos en la Tabla 7.1 que la especificidad es superior al 90 % llegando incluso al 93 % y la sensibilidad es en torno al 6 % inferior. También se observa en dicha tabla que la AUC es siempre superior al 95 %, lo que indica que es un test muy bueno, además en la subfigura 7.1a se presentan las curvas ROC de esta red neuronal.

A continuación vemos los resultados que obtenemos al entrenar la segunda red neuronal convolucional, denominada CNN B, que hemos visto en la Figura 6.6. En esta ocasión observamos en la Tabla 7.1 que esta red es mas sensible y menos específica, consiguiendo que en el caso de la validación cruzada *10-Fold* la sensibilidad y especificidad estén prácticamente en el mismo porcentaje, y en el caso de la validación cruzada *5-Fold* la mayor sensibilidad de todo el documento llegando al 91 %.También se observa en dicha tabla que la AUC es siempre superior al 95 % como vimos con la anterior red, lo que indica que también es un test muy bueno, además en la Figura 7.1b se presentan las curvas ROC de esta red neuronal.

	SEN	SPE	AUC	PPV/Precision	NPV	ACC
CNN A *3-Fold*	0,86±0,09	0,90±0,02	0,95±0,02	0,90±0,02	0,87±0,07	0,88±0,04
CNN A *5-Fold*	0,85±0,05	0,93±0,02	0,95±0,01	0,92±0,02	0,86±0,03	0,89±0,02
CNN A *10-Fold*	0,86±0,01	0,93±0,04	0,96±0,03	0,92±0,03	0,87±0,08	0,89±0,05
CNN B *3-Fold*	0,88±0,07	0,86±0,06	0,95±0,02	0,86±0,04	0,89±0,06	0,87±0,00
CNN B *5-Fold*	0,91±0,03	0,87±0,05	0,96±0,01	0,88±0,04	0,90±0,02	0,89±0,02
CNN B *10-Fold*	0,89±0,08	0,90±0,04	0,96±0,03	0,90±0,04	0,89±0,07	0,89±0,04

Tabla 7.1: Media±Desviación estándar de las métricas de los *k-folds* de Palencia con CNN utilizando espectrograma normalizado en la detección de tos.



(a) CNN A (b) CNN B

Figura 7.1: Curvas ROC de los *k-folds* de Palencia con CNN utilizando espectrograma normalizado en la detección de tos.

7.1.2. Rendimiento de las redes neuronales C-LSTM utilizando espectrograma normalizado

En esta ocasión hemos realizado experimentos con cuatro arquitecturas diferentes de redes neuronales C-LSTM alimentadas con espectrogramas normalizados, descritas en la sección 6.1.3. Los resultados obtenidos en estas cuatro redes neuronales se han agrupado en la Tabla 7.2, además en la Figura 7.2 se presentan las curvas ROC asociadas.

Vamos a comenzar viendo los resultados que obtenemos al entrenar la primera red neuronal C-LSTM, denominada C-LSTM A-1, que hemos visto en la Figura 6.7. En todos los casos la especificidad es superior al 90 % y la sensibilidad es en torno al 4 % inferior por lo que es ligeramente superior a cuando exclusivamente se utilizan CNN, como en la sección 7.1.1. También se observa en dicha tabla que la AUC es siempre superior al 95 %, lo que indica que es un test muy bueno, además en la Figura 7.2a se presentan las curvas ROC asociadas.

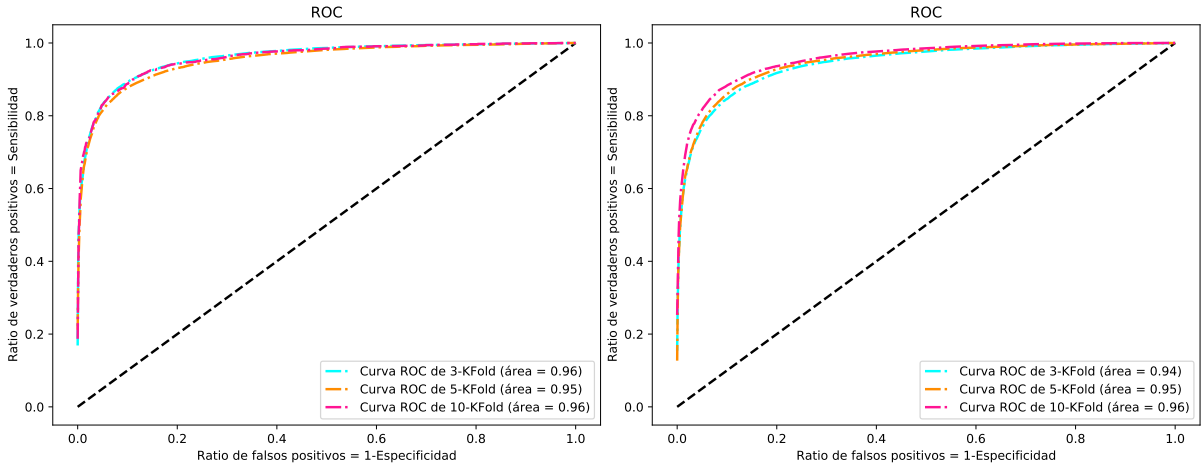
A continuación vemos los resultados que obtenemos al entrenar la segunda red neuronal C-LSTM, denominada C-LSTM B-1, que hemos visto en la Figura 6.8. En esta ocasión observamos que la especificidad es inferior a la que se obtiene en la C-BiLSTM A como ocurría en la sección 7.1.1, ya que en ningún caso supera el 90 %, pero en cambio la sensibilidad aun siendo siempre igual o superior a la red C-BiLSTM A las diferencias son mas estrechas. También se observa en dicha tabla que la AUC es siempre superior al 94 %, lo que indica como vimos con la anterior red, que también es un test muy bueno, además en la Figura 7.2b se presentan las curvas ROC asociadas.

Posteriormente vemos los resultados que obtenemos al entrenar la tercera red neuronal C-LSTM, denominada C-LSTM C-1, que hemos visto en la Figura 6.9. Podemos ver que esta red neuronal tiene la sensibilidad mas baja de las cuatro y una especificidad media. Además podemos saber que es un test muy bueno, ya que, la AUC es siempre superior al 93 %, presentamos las curvas ROC asociadas en la Figura 7.2c.

Por último vemos los resultados que obtenemos al entrenar la cuarta red neuronal C-LSTM, denominada C-LSTM D-1, que hemos visto en la Figura 6.10. Con claridad se observa que esta red neuronal tiene la especificidad mas baja de las cuatro, aunque la sensibilidad es bastante buena siempre superior al 88 %. Es un test muy bueno como se puede comprobar con las curvas ROC de la Figura 7.2d, ya que siempre es superior al 93 %.

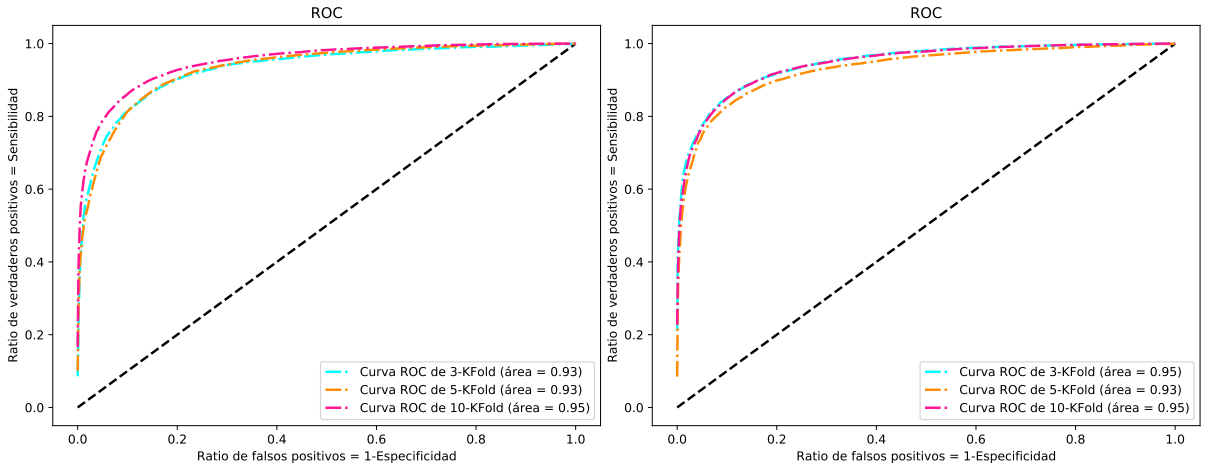
	SEN	**SPE**	**AUC**	**PPV/Precision**	**NPV**	**ACC**
C-LSTM A-1 *3-Fold*	0,88±0,04	0,92±0,03	0,96±0,02	0,91±0,03	0,89±0,04	0,90±0,03
C-LSTM A-1 *5-Fold*	0,87±0,04	0,90±0,04	0,95±0,01	0,90±0,04	0,87±0,03	0,89±0,02
C-LSTM A-1 *10-Fold*	0,87±0,08	0,93±0,02	0,96±0,02	0,93±0,02	0,88±0,06	0,90±0,04
C-LSTM B-1 *3-Fold*	0,88±0,05	0,88±0,05	0,94±0,03	0,88±0,04	0,88±0,05	0,88±0,04
C-LSTM B-1 *5-Fold*	0,90±0,04	0,84±0,10	0,95±0,01	0,85±0,07	0,90±0,03	0,87±0,03
C-LSTM B-1 *10-Fold*	0,88±0,08	0,89±0,04	0,96±0,02	0,89±0,04	0,89±0,07	0,89±0,04
C-LSTM C-1 *3-Fold*	0,89±0,05	0,82±0,08	0,93±0,05	0,83±0,07	0,88±0,06	0,85±0,06
C-LSTM C-1 *5-Fold*	0,87±0,05	0,85±0,06	0,93±0,02	0,85±0,05	0,87±0,04	0,86±0,03
C-LSTM C-1 *10-Fold*	0,84±0,14	0,90±0,05	0,95±0,03	0,90±0,04	0,86±0,09	0,87±0,06
C-LSTM D-1 *3-Fold*	0,89±0,08	0,83±0,06	0,95±0,03	0,84±0,04	0,89±0,07	0,86±0,01
C-LSTM D-1 *5-Fold*	0,88±0,07	0,82±0,08	0,93±0,02	0,83±0,05	0,88±0,05	0,85±0,02
C-LSTM D-1 *10-Fold*	0,88±0,11	0,82±0,15	0,95±0,03	0,85±0,09	0,89±0,08	0,85±0,06

Tabla 7.2: Media±Desviación estándar de las métricas de los *k-folds* de Palencia con C-LSTM utilizando espectrograma normalizado en la detección de tos.



(a) C-LSTM A-1

(b) C-LSTM B-1



(c) C-LSTM C-1 (d) C-LSTM D-1

Figura 7.2: Curvas ROC de los *k-folds* de Palencia con C-LSTM utilizando espectrograma normalizado en la detección de tos.

En esta ocasión vemos en la Tabla 7.3 los resultados que obtenemos al entrenar la redes neuronales C-LSTM que hemos descrito en la sección 6.1.3 pero con capas LSTM bidireccionales por lo que en este caso son redes neuronales C-BiLSTM. En este caso todos los entrenamientos se han hecho con 10 *folds*. La red neuronal C-BiLSTM A-1 es la que tiene mayor especificidad de todas las que se presentan en el documento aunque es bastante similar a la CNN A y C-LSTM A-1. La red neuronal C-BiLSTM B-1 es la más equilibrada siendo esta ligeramente inferior a la CNN B en especificidad y ligeramente superior a la C-LSTM B-1 en sensibilidad. La red neuronal C-BiLSTM C-1 tiene peor especificidad teniendo la misma sensibilidad que la C-BiLSTM B-1 y la red neuronal C-BiLSTM D-1 tiene peor sensibilidad y especificidad que la C-BiLSTM C-1. Son test muy buenos como se puede comprobar con las curvas ROC de la Figura 7.3, ya que siempre es superior al 94 %.

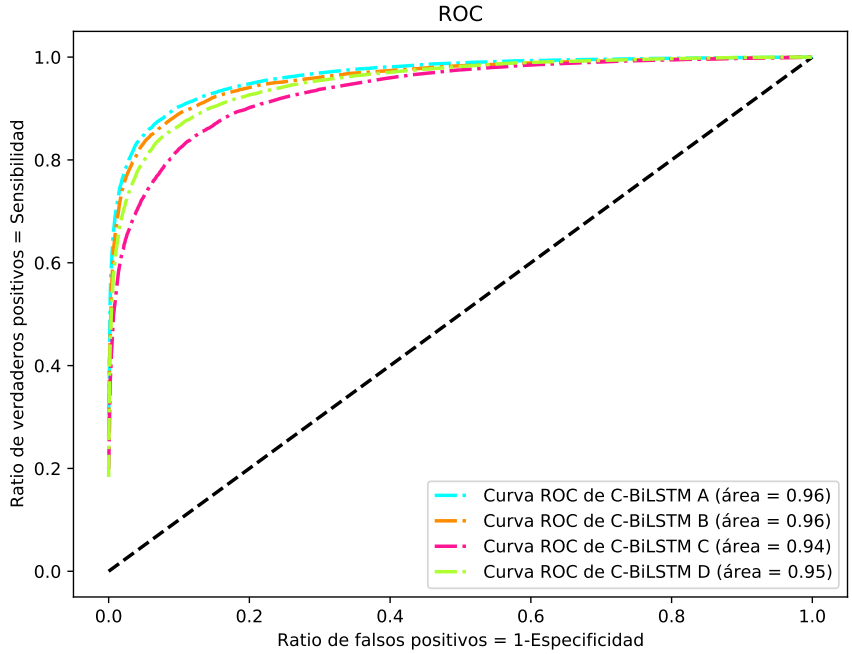


Figura 7.3: Curvas ROC de los 10-Folds de Palencia con C-BiLSTM utilizando espectrograma normalizado en la detección de tos.

	SEN	SPE	AUC	PPV/Precision	NPV	ACC
C-BiLSTM A-1	0,86±0,08	0,94±0,02	0,96±0,02	0,93±0,02	0,88±0,06	0,90±0,04
C-BiLSTM B-1	0,89±0,07	0,89±0,03	0,96±0,02	0,90±0,03	0,89±0,06	0,89±0,03
C-BiLSTM C-1	0,87±0,10	0,85±0,10	0,94±0,05	0,85±0,07	0,87±0,08	0,86±0,07
C-BiLSTM D-1	0,89±0,09	0,87±0,06	0,95±0,03	0,87±0,04	0,89±0,07	0,88±0,05

Tabla 7.3: Media±Desviación estándar de las métricas de los *10-folds* de Palencia con C-BiLSTM utilizando espectrograma normalizado en la detección de tos.

7.1.3. Rendimiento de las redes neuronales C-LSTM utilizando espectrograma de mel

En esta ocasión hemos realizado experimentos con cuatro arquitecturas diferentes de redes neuronales C-LSTM alimentadas con espectrogramas de mel, descritas en la sección 6.1.4. Los resultados obtenidos en estas cuatro redes neuronales se han agrupado en la Tabla 7.4, además en la Figura 7.4 se presentan las curvas ROC asociadas.

	SEN	SPE	AUC	PPV/Precision	NPV	ACC
C-LSTM A-2 *3-Fold*	0,75±0,09	0,86±0,04	0,89±0,04	0,84±0,03	0,78±0,06	0,81±0,05
C-LSTM A-2 *5-Fold*	0,76±0,11	0,85±0,06	0,89±0,04	0,84±0,04	0,79±0,07	0,81±0,04
C-LSTM A-2 *10-Fold*	0,79±0,11	0,87±0,05	0,91±0,05	0,86±0,05	0,81±0,08	0,83±0,06
C-LSTM B-2 *3-Fold*	0,77±0,12	0,86±0,02	0,90±0,05	0,85±0,04	0,79±0,09	0,82±0,06
C-LSTM B-2 *5-Fold*	0,77±0,13	0,88±0,05	0,91±0,04	0,87±0,04	0,80±0,08	0,82±0,05
C-LSTM B-2 *10-Fold*	0,80±0,11	0,87±0,06	0,92±0,04	0,87±0,04	0,82±0,08	0,84±0,05
C-LSTM C-2 *3-Fold*	0,81±0,10	0,87±0,02	0,91±0,04	0,86±0,03	0,82±0,08	0,84±0,05
C-LSTM C-2 *5-Fold*	0,82±0,11	0,82±0,12	0,90±0,02	0,83±0,08	0,83±0,07	0,82±0,03
C-LSTM C-2 *10-Fold*	0,78±0,13	0,87±0,07	0,91±0,04	0,86±0,05	0,81±0,09	0,83±0,05
C-LSTM D-2 *3-Fold*	0,74±0,07	0,87±0,01	0,89±0,03	0,85±0,01	0,77±0,05	0,81±0,04
C-LSTM D-2 *5-Fold*	0,70±0,12	0,88±0,02	0,88±0,04	0,86±0,02	0,75±0,08	0,79±0,05
C-LSTM D-2 *10-Fold*	0,73±0,20	0,91±0,03	0,90±0,07	0,88±0,03	0,79±0,11	0,82±0,10

Tabla 7.4: Media±Desviación estándar de las métricas de los *k-folds* de Palencia con C-LSTM utilizando espectrograma de mel en la detección de tos.

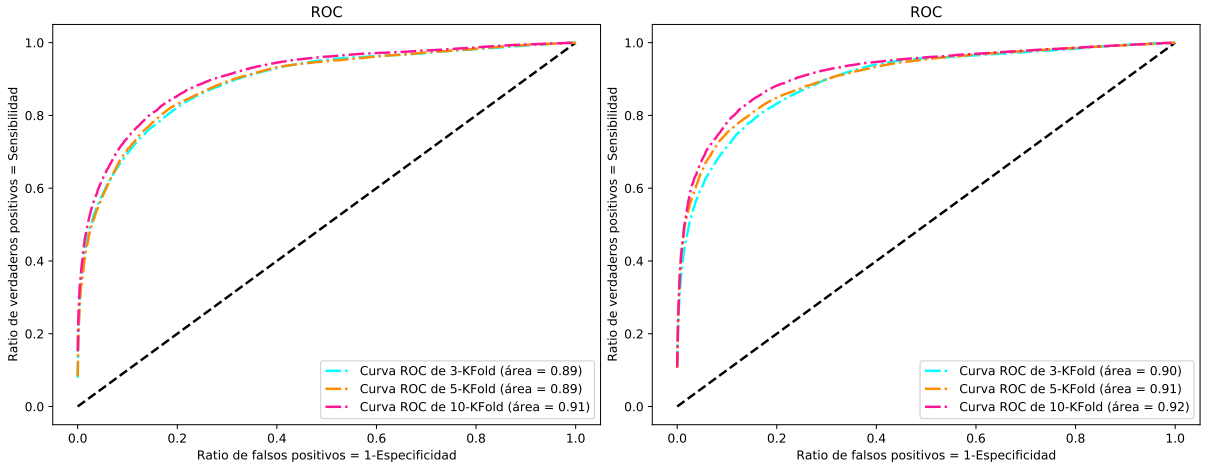
Vamos a comenzar viendo los resultados que obtenemos al entrenar la primera red neuronal C-LSTM, denominada C-LSTM A-2, que hemos visto en la Figura 6.12. En todos los casos la especificidad es superior al 85 % y la sensibilidad es en torno al 8 % inferior por lo que el rendimiento de esta red notablemente más bajo a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2. También se observa en dicha tabla que la AUC es siempre superior al 89 %, lo que indica que es un test bueno, además en la Figura 7.4a se presentan las curvas ROC asociadas.

A continuación vemos los resultados que obtenemos al entrenar la segunda red neuronal C-LSTM, denominada C-LSTM B-2, que hemos visto en la Figura 6.13. En todos los casos la especificidad es superior al 86 % y la sensibilidad es en torno al 9 % inferior, por lo que la especificidad que se obtiene con esta red es similar a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2, pero la sensibilidad es notablemente más

baja. También se observa en dicha tabla que la AUC es siempre superior al 90 %, lo que indica que es un test muy bueno, además en la Figura 7.4b se presentan las curvas ROC asociadas.

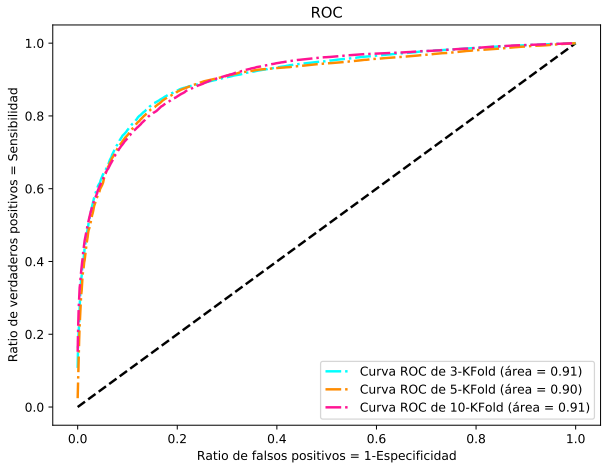
Posteriormente vemos los resultados que obtenemos al entrenar la tercera red neuronal C-LSTM, denominada C-LSTM C-2, que hemos visto en la Figura 6.14. En todos los casos la especificidad es superior al 82 % y la sensibilidad es en torno al 5 % inferior, por lo que la especificidad que se obtiene con esta red es similar a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2, pero la sensibilidad es sensiblemente más baja. Podemos ver que esta red neuronal tiene la especificidad mas baja de las cuatro y la sensibilidad más alta. Además como vimos con la anterior red podemos saber que es un test muy bueno, ya que, la AUC también es siempre superior al 90 %, presentamos las curvas ROC asociadas en la Figura 7.4c.

Por último vemos los resultados que obtenemos al entrenar la cuarta red neuronal C-LSTM, denominada C-LSTM D-2, que hemos visto en la Figura 6.15. En todos los casos la especificidad es superior al 88 % y la sensibilidad es en torno al 16 % inferior, por lo que la especificidad que se obtiene con esta red es superior en un 6 % a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2, pero la sensibilidad es sensiblemente más baja, alrededor de 16 %. Con claridad se observa que esta red neuronal tiene la sensibilidad mas baja de las cuatro y la mejor especificidad siempre superior al 87 %. Es un test bueno como se puede comprobar con las curvas ROC de la Figura 7.4d, ya que siempre es superior al 88 %.

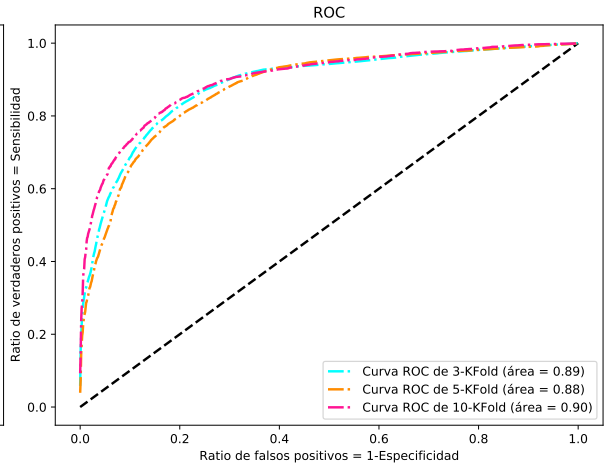


(a) C-LSTM A-2

(b) C-LSTM B-2



(c) C-LSTM C-2



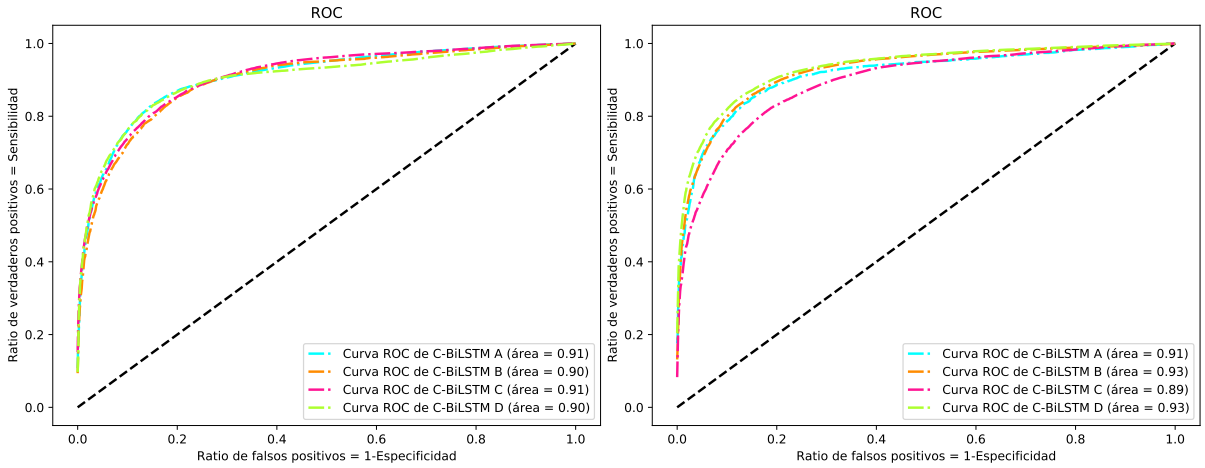
(d) C-LSTM D-2

Figura 7.4: Curvas ROC de los *k-folds* de Palencia con C-LSTM utilizando espectrograma de mel en la detección de tos.

En esta ocasión vemos en la Tabla 7.5 los resultados que obtenemos al entrenar la redes neuronales C-LSTM que hemos descrito en la sección 6.1.4 pero con capas LSTM bidireccionales por lo que en este caso son redes neuronales C-BiLSTM. En este caso todos los entrenamientos se han hecho con 10 *folds*. En general la especificidad es similar a cuando estas redes se alimentan con espectrograma normalizado, como en la sección 7.1.2, pero la sensibilidad es notablemente inferior, en torno al 11 %. Podemos ver que la sensibilidad y especificidad en todas las redes es similar, siendo A y C las mejores en sensibilidad, y en especificidad las redes C y D. Son test muy buenos como se puede comprobar con las curvas ROC de la Figura 7.5a, ya que siempre es superior al 90 %.

	SEN	**SPE**	**AUC**	**PPV/Precision**	**NPV**	**ACC**
C-BiLSTM A-2	0,78±0,16	0,87±0,05	0,91±0,06	0,86±0,05	0,81±0,11	0,82±0,07
C-BiLSTM B-2	0,75±0,16	0,88±0,04	0,90±0,05	0,86±0,03	0,80±0,10	0,82±0,07
C-BiLSTM C-2	0,79±0,14	0,89±0,03	0,91±0,05	0,88±0,02	0,82±0,09	0,84±0,06
C-BiLSTM D-2	0,73±0,19	0,90±0,04	0,90±0,06	0,87±0,03	0,79±0,11	0,81±0,09

Tabla 7.5: Media±Desviación estándar de las métricas de los *10-folds* de Palencia con C-BiLSTM utilizando espectrograma de mel en la detección de tos.



(a) Espectrograma de mel

(b) Ventanas desplazadas

Figura 7.5: Curvas ROC de los 10-Folds de Palencia con C-BiLSTM utilizando espectrograma de mel y ventanas desplazadas en la detección de tos.

7.1.4. Rendimiento de las redes neuronales C-LSTM utilizando ventanas desplazadas

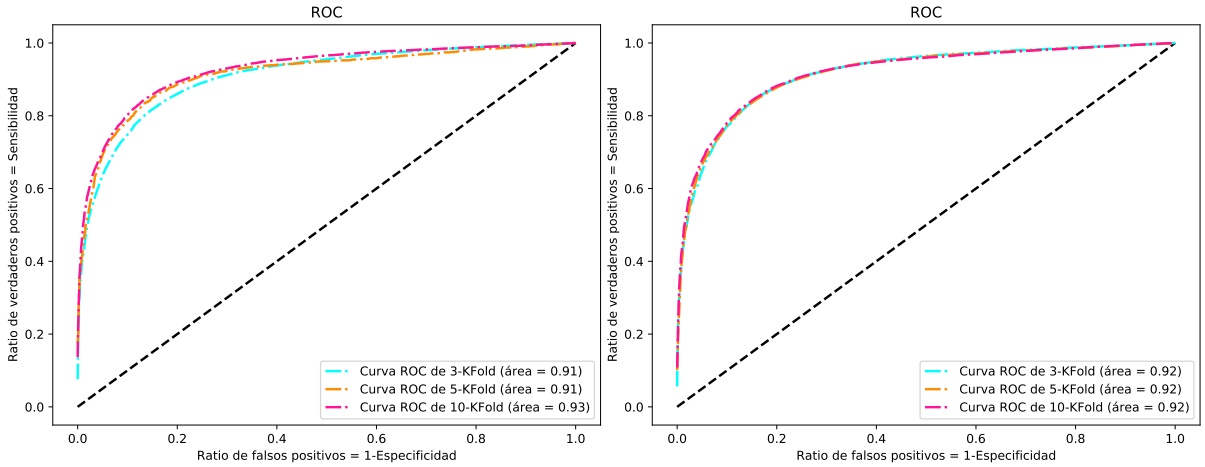
En esta ocasión hemos realizado experimentos con cuatro arquitecturas diferentes de redes neuronales C-LSTM alimentadas con ventanas desplazadas, descritas en la sección 6.1.5. Los resultados obtenidos en estas cuatro redes neuronales se han agrupado en la Tabla 7.6, además en la Figura 7.6 se presentan las curvas ROC asociadas.

Vamos a comenzar viendo los resultados que obtenemos al entrenar la primera red neuronal C-LSTM, denominada C-LSTM A-3, que hemos visto en la Figura 6.17. En todos los casos la especificidad es superior al 82 % y la sensibilidad al 81 % inferior por lo que el rendimiento de esta red es un 8 % inferior en especificidad a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2 y un 3 % inferior en sensibilidad. También se observa en dicha tabla que la AUC es siempre superior al 91 %, lo que indica que es un test bueno, además en la Figura 7.6a se presentan las curvas ROC asociadas.

	SEN	SPE	AUC	PPV/Precision	NPV	ACC
C-LSTM A-3 *3-Fold*	0,81±0,11	0,86±0,01	0,91±0,04	0,85±0,01	0,82±0,08	0,83±0,05
C-LSTM A-3 *5-Fold*	0,84±0,09	0,82±0,04	0,91±0,02	0,83±0,02	0,85±0,07	0,83±0,03
C-LSTM A-3 *10-Fold*	0,86±0,12	0,83±0,06	0,93±0,04	0,84±0,04	0,87±0,09	0,85±0,05
C-LSTM B-3 *3-Fold*	0,82±0,09	0,87±0,01	0,92±0,04	0,86±0,01	0,83±0,07	0,84±0,05
C-LSTM B-3 *5-Fold*	0,85±0,08	0,81±0,08	0,92±0,02	0,82±0,05	0,85±0,06	0,83±0,02
C-LSTM B-3 *10-Fold*	0,84±0,12	0,85±0,05	0,92±0,04	0,85±0,04	0,85±0,08	0,85±0,05
C-LSTM C-3 *3-Fold*	0,79±0,18	0,75±0,13	0,87±0,05	0,77±0,06	0,81±0,13	0,77±0,05
C-LSTM C-3 *5-Fold*	0,82±0,08	0,83±0,03	0,90±0,03	0,83±0,01	0,83±0,06	0,83±0,03
C-LSTM C-3 *10-Fold*	0,80±0,14	0,82±0,06	0,90±0,04	0,82±0,04	0,82±0,09	0,81±0,06
C-LSTM D-3 *3-Fold*	0,85±0,09	0,85±0,04	0,93±0,03	0,85±0,02	0,85±0,08	0,85±0,03
C-LSTM D-3 *5-Fold*	0,83±0,09	0,86±0,06	0,92±0,03	0,86±0,04	0,84±0,06	0,85±0,03
C-LSTM D-3 *10-Fold*	0,83±0,14	0,87±0,04	0,93±0,04	0,87±0,03	0,85±0,09	0,85±0,06

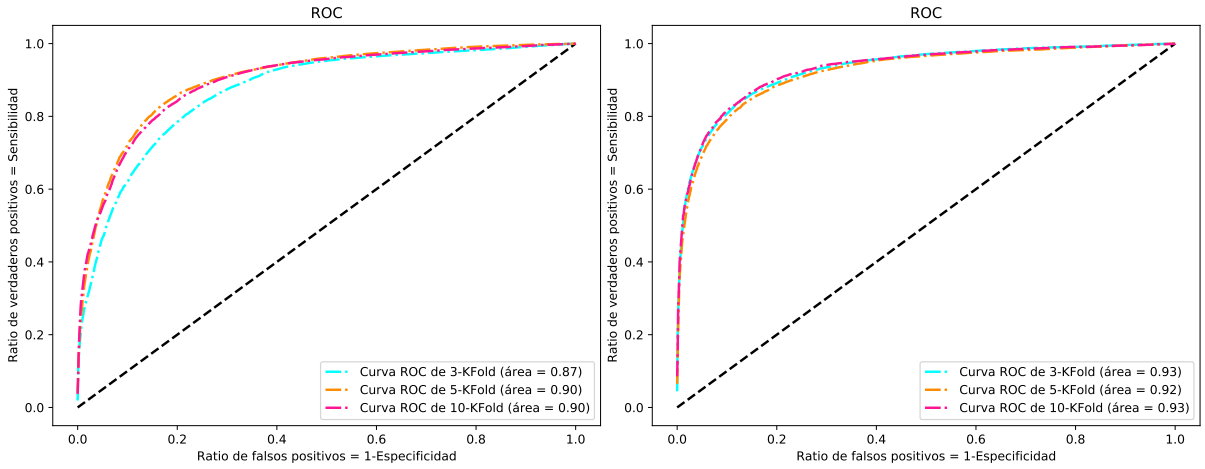
Tabla 7.6: Media±Desviación estándar de las métricas de los *k-folds* de Palencia con C-LSTM utilizando ventanas desplazadas en la detección de tos.

A continuación vemos los resultados que obtenemos al entrenar la segunda red neuronal C-LSTM, denominada C-LSTM B-3, que hemos visto en la Figura 6.18. En todos los casos la especificidad es superior al 81 % y la sensibilidad es bastante similar, por lo que la especificidad que se obtiene con esta red es alrededor de un 3 % inferior a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2, y la sensibilidad es en torno a un 5 % más baja. También se observa en dicha tabla que la AUC es siempre superior al 92 % como vimos con la anterior red, lo que indica que también es un test muy bueno, además en la Figura 7.6b se presentan las curvas ROC asociadas.



(a) C-LSTM A-3

(b) C-LSTM B-3



(c) C-LSTM C-3 (d) C-LSTM D-3

Figura 7.6: Curvas ROC de los *k-folds* de Palencia con C-LSTM utilizando ventanas desplazadas en la detección de tos.

Posteriormente vemos los resultados que obtenemos al entrenar la tercera red neuronal C-LSTM, denominada C-LSTM C-3, que hemos visto en la Figura 6.19. En todos los casos la especificidad es superior al 75 % y la sensibilidad es bastante similar, por lo que la especificidad y sensibilidad que se obtienen con esta red son alrededor de un 6 % inferior a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2. Con claridad se observa que de las cuatro redes neuronales esta es la que tiene la sensibilidad y especificidad mas bajas. Además podemos saber que es un test bueno, ya que, la AUC es siempre superior al 87 %, presentamos las curvas ROC asociadas en la Figura 7.6c.

Por último vemos los resultados que obtenemos al entrenar la cuarta red neuronal C-LSTM, denominada C-LSTM D-3, que hemos visto en la Figura 6.20. En todos los casos la especificidad es superior al 85 % y la sensibilidad es bastante similar, por lo que la especificidad que se obtiene con esta red es superior en un 4 % a cuando se alimenta con espectrograma normalizado, como en la sección 7.1.2, pero la sensibilidad es ligeramente más baja, alrededor de 5 %. Es un test muy bueno como se puede comprobar con las curvas ROC de la Figura 7.6d, ya que siempre es superior al 92 %.

En esta ocasión vemos en la Tabla 7.7 los resultados que obtenemos al entrenar la redes neuronales C-LSTM que hemos descrito en la sección 6.1.5 pero con capas LSTM bidireccionales por lo que en este caso son redes neuronales C-BiLSTM. En este caso todos los entrenamientos se han hecho con 10 *folds*. Salvo la red C-BiLSTM D-3, en el resto de redes la especificidad es inferior a cuando estas redes se alimentan con espectrograma normalizado, como en la sección 7.1.2, siendo alrededor de un 4 % inferior. La sensibilidad sin embargo es más baja en todos los casos en torno a un 6 % inferior. Podemos ver que la sensibilidad y especificidad en todas las redes es similar, siendo C la peor en sensibilidad y especificidad. Son test buenos como se puede comprobar con las curvas ROC de la Figura 7.5b, ya que siempre es superior al 89 %.

	SEN	**SPE**	**AUC**	**PPV/Precision**	**NPV**	**ACC**
C-BiLSTM A-3	0,82±0,14	0,84±0,07	0,91±0,05	0,84±0,04	0,84±0,10	0,83±0,06
C-BiLSTM B-3	0,82±0,13	0,87±0,04	0,93±0,04	0,86±0,03	0,84±0,09	0,84±0,05
C-BiLSTM C-3	0,79±0,16	0,81±0,07	0,89±0,05	0,81±0,04	0,81±0,10	0,80±0,07
C-BiLSTM D-3	0,83±0,10	0,89±0,05	0,93±0,04	0,88±0,04	0,85±0,08	0,86±0,05

Tabla 7.7: Media±Desviación estándar de las métricas de los *10-folds* de Palencia con C-BiLSTM utilizando ventanas desplazadas en la detección de tos.

7.2. Clasificación de toses utilizando matrices de confusión

En esta sección buscamos clasificar clips de tos de la base de datos COUGHVID [373] en distintos tipos de toses. En el primer caso clasificamos las toses en COVID-19, que tiene síntomas pero sin diagnóstico (sintomático), o que está sano (saludable) y en el segundo caso en tos húmeda (productiva) o tos seca.

A continuación vamos a visualizar el desempeño de tres redes neuronales de tipo C-BiLSTM, que hemos descrito en la sección 6.2, mediante la herramienta de la matriz de confusión permitiéndonos ver qué tipos de aciertos y errores está teniendo nuestro modelo a la hora de pasar por el proceso de aprendizaje con los datos. Los datos de entrada a estos modelos de redes neuronales son espectrogramas normalizados y espectrogramas de mel.

7.2.1. Clasificar tos de pacientes positivos para COVID-19 de tos de pacientes negativos para COVID-19

En este caso buscamos clasificar toses de tres tipos pacientes, positivos para COVID-19, sintomáticos para COVID-19 y sanos. Tenemos 1010 clips de tos para cada uno de los tipos de pacientes por lo que tenemos 3030 clips de tos en total, de los cuales se usará el 75 % (2272) para el entrenamiento de las C-BiLSTM y el 25 % (758) para el test.

Mostramos a continuación la matriz de confusión de la Tabla 7.8 que nos ofrece la red neuronal ilustrada en la Figura 6.24, la matriz de confusión de la Tabla 7.9 que nos ofrece la red neuronal ilustrada en la Figura 6.25 y la matriz de confusión de la Tabla 7.10 que nos ofrece la red neuronal ilustrada en la Figura 6.12.

Etiqueta\Predicción	**COVID-19**	**Saludable**	**Sintomático**
COVID-19	65	54	142
Saludable	60	41	156
Sintomático	55	32	153

Tabla 7.8: Matriz de confusión utilizando C-BiLSTM sin *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos COVID-19.

Etiqueta\Predicción	**COVID-19**	**Saludable**	**Sintomático**
COVID-19	101	104	41
Saludable	102	119	30
Sintomático	101	125	35

Tabla 7.9: Matriz de confusión utilizando C-BiLSTM con *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos COVID-19.

Etiqueta\Predicción	**COVID-19**	**Saludable**	**Sintomático**
COVID-19	0	0	256
Saludable	0	0	265
Sintomático	0	0	237

Tabla 7.10: Matriz de confusión utilizando C-BiLSTM alimentada con espectrogramas de mel para la clasificación de tos COVID-19.

7.2.2. Clasificar tos de pacientes positivos para COVID-19 de tos de pacientes sanos

En este caso buscamos clasificar toses de dos tipos pacientes, positivos para COVID-19 y sanos. Tenemos 1010 clips de tos para cada uno de los tipos de pacientes por lo que tenemos 2020 clips de tos en total, de los cuales se usará el 75 % (1515) para el entrenamiento de las C-BiLSTM y el 25 % (505) para el test.

Mostramos a continuación la matriz de confusión de la Tabla 7.11 que nos ofrece la red neuronal ilustrada en la Figura 6.24, la matriz de confusión de la Tabla 7.12 que nos ofrece la red neuronal ilustrada en la Figura 6.25 y la matriz de confusión de la Tabla 7.13 que nos ofrece la red neuronal ilustrada en la Figura 6.12.

Etiqueta\Predicción	**COVID-19**	**Saludable**
COVID-19	108	150
Saludable	90	157

Tabla 7.11: Matriz de confusión utilizando C-BiLSTM sin *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos COVID-19 frente a pacientes sanos, alcanza un 52,48 % de exactitud.

Etiqueta\Predicción	**COVID-19**	**Saludable**
COVID-19	230	25
Saludable	230	20

Tabla 7.12: Matriz de confusión utilizando C-BiLSTM con *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos COVID-19 frente a pacientes sanos.

Etiqueta\Predicción	**COVID-19**	**Saludable**
COVID-19	0	261
Saludable	1	243

Tabla 7.13: Matriz de confusión utilizando C-BiLSTM alimentada con espectrogramas de mel para la clasificación de tos COVID-19 frente a pacientes sanos.

7.2.3. Clasificar tos de pacientes positivos para COVID-19 de tos de pacientes sintomáticos para COVID-19

En este caso buscamos clasificar toses de dos tipos pacientes, positivos para COVID-19 y sintomáticos para COVID-19. Tenemos 1010 clips de tos para cada uno de los tipos de pacientes por lo que tenemos 2020 clips de tos en total, de los cuales se usará el 75 % (1515) para el entrenamiento de las C-BiLSTM y el 25 % (505) para el test.

Mostramos a continuación la matriz de confusión de la Tabla 7.14 que nos ofrece la red neuronal ilustrada en la Figura 6.24, la matriz de confusión de la Tabla 7.15 que nos ofrece la red neuronal ilustrada en la Figura 6.25 y la matriz de confusión de la Tabla 7.16 que nos ofrece la red neuronal ilustrada en la Figura 6.12.

Etiqueta\Predicción	COVID-19	Sintomático
COVID-19	163	89
Sintomático	122	131

Tabla 7.14: Matriz de confusión utilizando C-BiLSTM sin *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos COVID-19 frente a pacientes sintomáticos, alcanza un 58,21 % de exactitud.

Etiqueta\Predicción	COVID-19	Sintomático
COVID-19	46	208
Sintomático	52	199

Tabla 7.15: Matriz de confusión utilizando C-BiLSTM con *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos COVID-19 frente a pacientes sintomáticos.

Etiqueta\Predicción	COVID-19	Sintomático
COVID-19	0	253
Sintomático	0	252

Tabla 7.16: Matriz de confusión utilizando C-BiLSTM alimentada con espectrogramas de mel para la clasificación de tos COVID-19 frente a pacientes sintomáticos.

7.2.4. Clasificar tos húmeda (productiva) y tos seca

En este caso buscamos clasificar toses de dos tipos, seca y húmeda (productiva). Tenemos 473 clips de tos para cada uno de los tipos por lo que tenemos 946 clips de tos en total, de los cuales se usará el 75 % (709) para el entrenamiento de las C-BiLSTM y el 25 % (237) para el test.

Mostramos a continuación la matriz de confusión de la Tabla 7.17 que nos ofrece la red neuronal ilustrada en la Figura 6.24, la matriz de confusión de la Tabla 7.18 que nos ofrece la red neuronal ilustrada en la Figura 6.25 y la matriz de confusión de la Tabla 7.19 que nos ofrece la red neuronal ilustrada en la Figura 6.12.

Etiqueta\Predicción	Seca	Húmeda (productiva)
Seca	110	0
Húmeda (productiva)	127	0

Tabla 7.17: Matriz de confusión utilizando C-BiLSTM sin *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos húmeda (productiva) frente a tos seca.

Etiqueta\Predicción	Seca	Húmeda (productiva)
Seca	92	21
Húmeda (productiva)	92	32

Tabla 7.18: Matriz de confusión utilizando C-BiLSTM con *transfer learning* alimentada con espectrogramas normalizados para la clasificación de tos húmeda (productiva) frente a tos seca, alcanza un 52,32 % de exactitud.

Etiqueta\Predicción	Seca	Húmeda (productiva)
Seca	124	0
Húmeda (productiva)	113	0

Tabla 7.19: Matriz de confusión utilizando C-BiLSTM alimentada con espectrogramas de mel para la clasificación de tos húmeda (productiva) frente a tos seca.

7.3. Discusión

En este apartado presentamos el rendimiento de los modelos propuestos por los artículos revisados en el capítulo 5, junto con un breve resumen para poder compararlos con nuestros resultados.

7.3.1. Detección de tos

Primero mostramos en la tabla 7.20 el rendimiento de la aplicación evaluada en la sección 5.1.1.

	Referencia		**Entrenado**	
	Sensibilidad (%)	**FP/h**	**Sensibilidad (%)**	**FP/h**
App con clasificador K-NN [296]	61,56±18,36	19±12	80,63±15,26	16±5

Tabla 7.20: Rendimiento de la aplicación desarrollada en el grupo de investigación LPI.

Podemos observar que, al incorporar el módulo de auto-formación, el rendimiento se incrementa significativamente, pero no llega a los métodos examinados posteriormente, los cuales no están integrados en una aplicación móvil.

A continuación, mostramos al comienzo de la tabla 7.21 el rendimiento de los mejores modelos presentados en la sección 7.1 y posteriormente el rendimiento de los modelos evaluados en la sección 5.2.

	SEN	**SPE**	**AUC**	**PPV**	**NPV**	**ACC**
CNN A *5-Fold*	0,85±0,05	0,93±0,02	0,95±0,01	0,92±0,02	0,86±0,03	0,89±0,02
CNN A *10-Fold*	0,86±0,01	0,93±0,04	0,96±0,03	0,92±0,03	0,87±0,08	0,89±0,05
CNN B *5-Fold*	**0,91**±0,03	0,87±0,05	0,96±0,01	0,88±0,04	**0,90**±0,02	0,89±0,02
CNN B *10-Fold*	0,89±0,08	0,90±0,04	0,96±0,03	0,90±0,04	0,89±0,07	0,89±0,04
C-LSTM A-1 *3-Fold*	0,88±0,04	0,92±0,03	0,96±0,02	0,91±0,03	0,89±0,04	**0,90**±0,03
C-LSTM A-1 *5-Fold*	0,87±0,04	0,90±0,04	0,95±0,01	0,90±0,04	0,87±0,03	0,89±0,02
C-LSTM A-1 *10-Fold*	0,87±0,08	0,93±0,02	0,96±0,02	**0,93**±0,02	0,88±0,06	**0,90**±0,04
C-LSTM B-1 *10-Fold*	0,88±0,08	0,89±0,04	0,96±0,02	0,89±0,04	0,89±0,07	0,89±0,04
C-BiLSTM A-1 *10-Fold*	0,86±0,08	**0,94**±0,02	0,96±0,02	**0,93**±0,02	0,88±0,06	**0,90**±0,04
C-BiLSTM B-1 *10-Fold*	0,89±0,07	0,89±0,03	0,96±0,02	0,90±0,03	0,89±0,06	0,89±0,03
XGBoost [2]	0,78±0,10	**0,95**±0,08	0,96±0,03	**0,95**±0,07	-	-
Deep BiLSTM [289]	0,748	0,998	-	0,788	0,997	0,995
C-BiLSTM [289]	0,828	0,998	-	0,807	0,998	0,996
C-BiLSTM con regresión de límites [289]	0,841	**0,998**	-	0,841	0,998	0,996
CNNs con espectrograma de Mel (Escenario 1) [24]	0,92±0,03	0,87±0,03	0,95	0,89±0,03	0,90±0,04	0,89±0,02
CNNs con bagging (Escenario 1) [24]	**0,92**±0,03	0,90±0,04	0,96	0,92±0,03	0,90±0,04	0,91±0,02

Continuación de la Tabla 7.21

	SEN	SPE	AUC	PPV	NPV	ACC
CNNs con espectrograma de Mel (Escenario 2) [24]	0,87±0,06	0,06±0,07	-	0,89±0,07	0,82±0,11	0,86±0,02
CNNs con bagging (Escenario 2) [24]	0,87±0,07	0,91±0,05	-	0,93±0,05	0,81±0,10	0,88±0,03

Tabla 7.21: Comparación de los modelos de detección de audios con tos propuestos en este trabajo con los analizados en la literatura científica.

El criterio para seleccionar los mejores modelos de nuestro estudio fue que la exactitud obtenida fuera superior al 89 %. Debido a esta restricción, solo se muestran redes que se han entrenado utilizando el espectrograma normalizado expuestas en la sección 7.1.2.

En el caso de las redes expuestas en la sección 7.1.3, que se han alimentado utilizando el espectrograma de mel, la exactitud más alta es de 84 % y en el caso de las expuestas en la sección 7.1.4, donde se han utilizado ventanas desplazadas, la exactitud más alta alcanza el 86 %.

Respecto a los modelos evaluados en la sección 5.2 vemos que el detector de tos a partir de características de audio usando un clasificador XGBoost [2] tiene una especificidad un 1 % superior a nuestro modelo C-BiLSTM A *10-Fold*, pero la sensibilidad es un 8 % inferior, por lo que se puede apreciar que nuestro modelo tiene un rendimiento general superior. Este estudio empleó 215 grabaciones seleccionadas al azar de la base de datos COUGHVID, que contiene más de 25000 grabaciones de tos, con un procedimiento deCV *k-fold* anidado para evaluar el rendimiento.

En el caso de la red C-BiLSTM con regresión de límites [289] alimentada con características extraídas con MFCC de 13 dimensiones y la primera y segunda derivadas temporales, tiene una especificidad un 6 % superior a nuestro modelo C-BiLSTM A *10-Fold* pero la sensibilidad es un 2 % inferior, por lo que es un modelo más desequilibrado que el nuestro, con una especificidad cercana al 100 %. Lo que resulta en que si tiene un resultado positivo podemos confirmar que el audio contiene una tos. Este estudio empleó 9969 muestras de tos realista de 42 pacientes distintos y el rendimiento fue evaluado mediante un procedimiento *3-fold* con datos de prueba no superpuestos de 10 pacientes.

El artículo de [24] presenta una investigación sobre la detección de tos con 6737 muestras de tos, 3985 de risa, 3695 de carraspeo, 731 de habla y 443 de expiración forzada de audio, obtenidas de 43 participantes sanos. De ellos, 27 se usaron para entrenar una red CNN que recibía espectrogramas de mel como entrada y se evaluó con datos no superpuestos de prueba de 11 participantes. Como estas grabaciones se hicieron simultáneamente en un conjunto de cinco dispositivos, utilizaron el método de conjunto de embolsado (bagging). En el Escenario 1, el entrenamiento se hizo en todos los dispositivos y se probó en participantes no conocidos, mientras que en el Escenario 2, el entrenamiento se hizo en todos los dispositivos menos en el que se indicó como nombre de columna y luego se probó en las grabaciones del dispositivo desconocido con participantes no vistos.

Podemos ver que el escenario 1 ofrece mejores resultados que el escenario 2 y también mejora los resultados cuando se utiliza la técnica de bagging, por lo que nos centraremos en el rendimiento de las CNNs con bagging. En este caso vemos que la sensibilidad es un 1 % superior a la de nuestro modelo CNN B *5-Fold* y también tiene una especificidad un 3 % superior, por lo que es claramente mejor que este modelo. Con respecto a nuestro modelo con mayor especificidad, vemos que el modelo C-BiLSTM A *10-Fold* tiene una sensibilidad que es un 6 % inferior y una especificidad un 4 % superior. También se puede observar que este estudio tiene la limitación evidente de que solo se emplearon toses voluntarias para evaluar el desempeño de los distintos enfoques.

Otra limitación de los estudios analizados es el entorno en el que se grabaron las toses, que restringe la generalización de las métricas de rendimiento. En el primer estudio se usaron toses donadas por los participantes que quisieron colaborar, por lo que el entorno sería con poco ruido. En el segundo caso se usaron toses reflejas grabadas en el hospital de Tongji (China), por lo que sí que es un entorno realista pero también de bajo ruido. Finalmente, en el tercer estudio se usaron toses voluntarias grabadas en un entorno de laboratorio, por lo que no se sabe si se obtendrían los mismos patrones de rendimiento en entornos que se parecen a la vida diaria de los usuarios, a diferencia de nuestra base de datos.

7.3.2. Clasificación de tos

A continuación mostramos las Tablas 7.22 y 7.23, que resumen el rendimiento de los modelos evaluados en la sección 5.3 sobre la identificación de COVID-19 a partir de audios de tos. Podemos ver que estos resultados son ampliamente mejores que el rendimiento que obtenemos en los modelos presentados en la sección 7.2.

El *paper* de [290] utilizó cinco conjuntos de datos con un total de 813 grabaciones de audio de tos. De estas, 346 toses eran de COVID-19 (C), 346 toses eran negativas de COVID-19 sin tos (N), 101 toses eran no COVID-19 con tos inespecífica como síntoma (NC), 20 toses eran sin COVID-19 con tos ferina (PT) y el último conjunto era la suma de los sujetos que no eran COVID-19 (NNC). De estos audios de tos se extrajeron características cuyo número posteriormente se redujo a través de los métodos de eliminación recursiva de características (RFE) y utilizando un *autoencoder*. Finalmente, se implementaron los modelos supervisados más populares en la clasificación de la tos, pero con diferencia el de mejor rendimiento fue RF con RFE, cuyos resultados vemos en la Tabla 7.22.

	SEN	**SPE**	**AUC**	**PPV/Precision**	**ACC**	**F-score**
C vs. N	**93,81 %**	81,54 %	**96,04 %**	90,97 %	**89,79 %**	**92,10 %**
C vs. NC	95,49 %	**76,09 %**	**92,53 %**	**88,42 %**	**88,79 %**	**91,79 %**
C vs. NNC	**85,96 %**	**85,09 %**	**89,65 %**	**85,14 %**	**85,53 %**	**85,58 %**
C vs. PT	**98,91 %**	72,00 %	95,67 %	95,20 %	**94,81 %**	**97,00 %**

Tabla 7.22: Métricas obtenidas con bosque aleatorio (RF) con eliminación recursiva de características (RFE) en la clasificación de tos con COVID-19 en el estudio de Alberto Tena y colaboradores [290]

Mientras que en el artículo de [339] utilizaron un conjunto de datos que consta de 496 toses, de las cuales 249 corresponden a 82 participantes infectados y 247 a participantes no infectados. El número de toses por participante oscila entre 2 y 5. Se aplicaron cuatro métodos de extracción de características a los audios: GeMaps, eGeMaps, ComParE y *wavelet scaterring*. Los dos primeros métodos no fueron efectivos y no se realizó ninguna reducción de características debido al bajo número de características. En los dos últimos casos, se seleccionaron el 1 % y el 10 % de las características más relevantes según el criterio de información mutua para los métodos ComParE y *wavelet scaterring*, respectivamente. Se entrenaron cuatro modelos de clasificación alimentados con las características extraídas: un bosque aleatorio (RF) con 250 árboles, dos clasificadores basados en árboles de decisión con técnicas de *boosting* y *bagging* con 250 árboles de profundidad máxima cuatro y un MLP que obtuvo malos resultados. Se utilizó una validación cruzada estratificada de 5 veces para evaluar los modelos, asegurando que las toses del mismo hablante no se mezclaran entre el conjunto de entrenamiento y el de prueba. Además, se intentaron usar modelos VGGish [362] y OpenL3 [363] para obtener incrustaciones de audio profundas, pero estos modelos también fueron ineficaces.

	RF [339]		**bagging [339]**		**Boosting [339]**		**CNN con biomarcadores [291]**
	ComParE	**Wavelet**	**ComParE**	**Wavelet**	**ComParE**	**Wavelet**	**MFCC**
ACC	87,31 %	87,50 %	87,12 %	84,92 %	82,70 %	**88,52 %**	**97,1 %**
SEN	87,37 %	86,94 %	83,71 %	**88,75 %**	78,79 %	87,19 %	**98,5 %**
SPE	87,72 %	87,74 %	**90,87 %**	81,06 %	87,33 %	89,82 %	**94,2 %**

Tabla 7.23: Métricas obtenidas en la clasificación de tos con COVID-19 en los estudios de Vladimir Despotovic [339] y Jordi Laguarta [291]

Finalmente, en la investigación de [291], el conjunto de datos usado incluye grabaciones de audio de tos de duración variable de 2660 sujetos positivos de COVID-19 y una proporción de 1-10 de sujetos positivos a control (con un promedio de 3 toses por sujeto), acompañadas de un conjunto de 10 preguntas. Se usaron los audios de tos forzada de 4256 sujetos (80 %) para entrenamiento y 1064 (20 %) para validación. Cada grabación de tos se procesa con el paquete MFCC y se pasa a través del biomarcador 1 (Degradación Muscular). Después,

alimenta la arquitectura de CNNs que se compone de tres ResNet-50 en paralelo, entrenadas previamente para capturar características acústicas en los biomarcadores 2 (Cuerdas vocales), 3 (Sentimiento) y 4 (Pulmones y vías respiratorias). A la salida de estos tres tensores concatenados se agrupa utilizando una capa 2D de agrupación promedio global, seguida de una MLP. El conjunto de sujetos diagnosticados con una prueba oficial obtuvo un AUC de 0,97.

Capítulo 8

CONCLUSIONES Y LÍNEAS FUTURAS

En este capítulo se presentan las conclusiones principales que se han obtenido a partir del desarrollo y la evaluación del trabajo realizado, así como las líneas futuras de investigación que se plantean para continuar y mejorar el proyecto. Se trata de un resumen crítico y reflexivo de los objetivos, la metodología, los resultados y las aportaciones del trabajo, así como de las limitaciones y dificultades encontradas. También se proponen algunas ideas y sugerencias para ampliar el alcance y la aplicabilidad del trabajo en el futuro.

8.1. Conclusiones

El objetivo principal de este trabajo fin de máster fue proponer y desarrollar un método de audición de máquina basado en el aprendizaje profundo para la segmentación robusta de eventos de audio de tos.

Hemos usado los mismos datos preprocesados de varias maneras para entrenar distintas redes neuronales y hemos visto que las capas LSTM o BiLSTM pueden mejorar un poco el rendimiento pero no son determinantes, sin embargo se nota una diferencia significativa al usar los espectrogramas normalizados como método de preprocesado, con los que se logra una precisión máxima del 90 % superando a las ventanas desplazadas con una precisión máxima del 86 % y a los espectrogramas de mel con una precisión máxima del 84 %.

El método propuesto se ha probado en distintos escenarios y condiciones acústicas, usando una base de datos con toses reflejas de veinte pacientes que seguían su rutina normal, con un *smartphone* en sus bolsillos o bolsos, para obtener toses reflejas registradas de ambientes reales y ruidosos que complican la detección de la tos. Esto significa que el método es robusto al ruido, la reverberación y debido a que otras personas del entorno entraban en las grabaciones también es robusto a la variabilidad inter e intra hablante.

Nuestro sistema de detección de tos propuesto es capaz de detectar los eventos de tos con una alta precisión y una baja tasa de falsos positivos, alcanzando un rendimiento comparable a los métodos del estado del arte analizados en este estudio. También evaluamos su capacidad de generalización con una estrategia de validación cruzada.

Nuestro sistema facilita una monitorización del paciente más cómoda y menos invasiva, que puede ayudar a los pacientes a controlar sus síntomas de tos. Además, nuestro sistema puede apoyar la evaluación de tratamientos y el conocimiento clínico de los patrones de tos. Para este fin, se pueden detectar y analizar los patrones de audio de la tos. Finalmente, los sistemas nacionales de salud y las economías también se verían favorecidas con una disminución de las hospitalizaciones y un aumento de la productividad. Los resultados obtenidos muestran que las redes neuronales pueden ser una herramienta útil para encontrar patrones en espectrogramas logarítmicos normalizados y distinguir así si el audio tiene una tos o no.

También tenemos otro objetivo que es el sistema de clasificación de audios de tos buscando detectar por tipo de tos (productiva y no productiva) y por enfermedad subyacente (pacientes sanos, pacientes sintomáticos sin COVID-19 y positivos para COVID-19). En este caso usamos la base de datos COUGHVID donde encontramos 946 clips de tos para alimentar la red que busca distinguir los tipo de tos y 3030 clips de tos para alimentar la red que busca distinguir estas toses según la enfermedad subyacente. En este caso los resultados de clasificación de

audios con tos no fueron buenos.

En el contexto de estos resultados, este trabajo representa un paso más hacia algoritmos de detección de tos escalables, de bajo costo, ubicuos y precisos.

8.2. Líneas futuras

Con el objetivo de continuar y mejorar el trabajo realizado en este estudio se podría ampliar el conjunto de datos utilizado para entrenar y evaluar el método propuesto, incluyendo más muestras de tos de diferentes hablantes, géneros, edades y patologías, así como más tipos y niveles de ruido y reverberación.

Esto es posible gracias a las bases de datos publicas como es el caso de COUGHVID. Aunque estas bases de datos no aseguran que todos los audios contengan una tos, en algunos casos no hay tos y en otros hay varias por lo que para utilizar este tipo de bases de datos publicas para la deteccion de tos se debería hacer un cribado manual en el que se eliminen los audios sin tos y se dividan los audios con varias toses.

Dado que la voz contiene información tanto lingüística como paralingüística (e.g., tono vocal, entonación, velocidad del habla, volumen), existe un interés creciente en la anonimización de la voz para preservar la privacidad y la identidad del hablante. Esto abre una nueva posibilidad para aplicar *data augmentation* mediante el uso de la anonimización de voz [379] que puede ser más efectiva que el *data augmentation* tradicional basado en transformaciones de la representación temporal o de frecuencia de la señal; los métodos representativos pueden incluir cambios de tono [380] o enmascaramiento de frecuencia [381] con las herramientas de TensorFlow I/O [37].

Otra posibilidad de *data augmentation* es aplicar la reducción de ruido mediante métodos propios o utilizando Audacity v2.4.2 [342] para eliminar el ruido de fondo [339]. Sin embargo, se debe tener en cuenta que la reducción de ruido puede afectar a la calidad y a la información paralingüística de la voz, por lo que se recomienda usarla con precaución.

El aprendizaje por transferencia también es otra línea de exploración con redes preentrenadas como VGGish [362] y OpenL3 [363] o con codificadores de código abierto como Lyra [182] y EnCodec [183].

También puede resultar interesante explorar otras arquitecturas y técnicas de aprendizaje profundo para mejorar la capacidad de generalización y adaptación del método propuesto, como por ejemplo utilizando *embeddings* para alimentar *transformers* con arquitecturas similares a ViT o Swin-T.

Además se podría desarrollar una interfaz gráfica de usuario que facilite el uso del método propuesto por parte de los usuarios finales, como médicos, enfermeros, pacientes o investigadores, como en el caso de *SmartCough* [296].

Con respecto al estudio sobre detección de enfermedades subyacentes mediante el análisis de los sonidos de tos. Podía aumentar la base de datos capturando diferentes sonidos que no son solo tos y que pueden ayudar a entrenar mejores clasificadores. Esto lo hace el estudio CDCVA de [339] donde se usan cinco tareas vocales: decir /aaaa/ de forma sostenida, toser (3 veces), respirar hondo por la boca (3 veces), contar del 1 al 20 y leer un texto específico.

Además de los sonidos vocales, también se recopila información sobre las características personales, los hábitos y los síntomas de los participantes, así como las comorbilidades relacionadas con COVID-19. De esta forma, se añade información contextual y semántica al método propuesto, como el tipo y la duración de la tos, el estado de salud del hablante, el ambiente acústico y la presencia de otros sonidos o voces.

ACRÓNIMOS

AAC Advanced Audio Coding 95
AAE Adversarial Autoencoder 56
ACC Accuracy 24, 102, 103, 105, 107–109, 113–115
Adaboost Adaptive **Boost**ing 21
AdaDelta Adaptive Delta 40
AdaGrad Adaptive **Grad**ient 40
ADALINE Adaptive **Lin**ear **E**lements 11
Adam Adaptive Moment Estimation 40, 41
AE Autoencoder 55, 56
AGI Artificial General Intelligence 15
AIBO Artificial Intelligence [Ro]Bot 12
AIFF Audio Interchange File Format 95
ALAC Apple Lossless Audio Codec 95
AlexNet Alex Krizhevsky Network 13
ANI Artificial Narrow Intelligence 15
ANN Artificial Neural Networks 46, 48
ASI Artificial Super Intelligence 15
ASIMO Advanced Step in Innovative Mobilit 12
ASR Automatic Speech Recognition 14, 16, 17
AUC Area Under the Curve 28, 64, 68, 101–103, 105–109, 113–116

BA Balanced Accuracy 25
bagging bootstrap **agg**regat**ing** 21, 46, 69, 70, 113–115
BERT Bidirectional Encoder Representations from Transformers 13
BGD Batch Gradient Descent 39
BiLSTM Bidirectional Long Short-Term Memory 53, 62, 65, 97–99, 113, 117
BLOOM BigScience Language Open-science Open-access Multilingual 14
BP Back-Propagation 48
BPTS Back-Propagation Through Structure 48
BPTT Back-Propagation Through Time 49
BWF Broadcast Wave Format 95

C-BiLSTM Convolutional - Bidirectional Long Short-Term Memory 53, 62, 65, 75, 82, 83, 86, 87, 90, 91, 96–99, 101, 102, 104, 105, 107, 109–114, 161, 162
C-GRU Convolutional - Gated Recurrent Unit 53
C-LSTM Convolutional - Long Short-Term Memory 5, 53, 80–91, 97–99, 101–109, 113, 154–158, 160–162
CART Classification And Regression Trees 19, 21
CASA Computational auditory scene analysis 16, 17
CD-ROM Compact Disc Read-Only Memory 75
CDCVA COVID-19 Detection by Cough and Voice Analysis 68, 118
CEIm Comité de Ética de la Investigación con medicamentos 66
CEST Central European Summer Time 2
CLIP Contrastive Language-Image Pre-training 13
CM Classical Momentum 39
CNN Convolutional Neural Network 13, 50–53, 62, 65, 66, 73, 76–80, 82, 84, 85, 89, 90, 101, 102, 104, 113–116, 153
CoCa Contrastive Captioners 13
ComParE Computational **Par**alinguistics Challeng**E** 69, 71, 72, 115
COPD Chronic Obstructive Pulmonary Disease 2
COVID-19 Corona**vi**rus **D**isease 20**19** 2, 3, 7, 23, 24, 62, 64, 66–70, 72–74, 94, 95, 110–112, 115, 117, 118
CPU Central Processing Unit 5
CV Cross-Validation 64, 114
CWD Choi–Williams Distribution 67

DARPA Defense Advanced Research Projects Agency 11
DBM Deep Boltzmann Machine 12
DBN Deep Belief Network 12
DBSCAN Density-Based Spatial Clustering of Applications with Noise 20
DFT Discrete Fourier Transform 67
DL Deep Learning 1, 7, 16, 31, 46, 75
DOR Diagnostic Odds Ratio 25

e.g. exempli gratia/por ejemplo 3, 19, 28, 45, 46, 68, 118
EDOs Ecuaciones Diferenciales Ordinarias 6

EEPD Energy Envelope Peak Detection 64
eGeMaps extended Geneva Minimalistic Acoustic Parameter Set 69, 71, 115
ELU Exponencial Linear Unit 34
EMA Exponential Moving Average 40
EPFL École Polytechnique Fédérale de Lausanne 64
EPI Enfermedad Pulmonar Intersticial 2
EPO Enfermedad Pulmonar Ocupacional 2
EPOC Enfermedad Pulmonar Obstructiva Crónica 2, 3, 75, 76, 95
ESM Evolutionary Scale Modeling 14
ETSIT Escuela Técnica Superior de Ingenieros de Telecomunicación 6
EVS Enhanced Voice Services 95

FCN Fully Convolutional Networks 53
FDR False Discovery Rate 25
FFT Fast Fourier Transform 6, 66
FLAC Free Lossless Audio Codec 95
FN Falso Negativo/False Negative 23, 26, 29
FNN Feedforward Neural Network 48
FNR False Negative Rate 25
FOR False Omission Rate 25
FP Falso Positivo/False Positive 24, 26, 29, 113
FPI Fibrosis Pulmonar Idiopática 2
FPR False Positive Rate 25, 26, 63

GAN Generative Adversarial Net 56
GBDT Gradient Boosted Decision Trees 21
GD Gradient Descent 38, 39
GELU Gaussian Error Linear Unit 33, 35, 52
GeMaps Geneva Minimalistic Acoustic Parameter Set 69, 71, 115
GLIDE Guided Language-to-Image Diffusion for Generation and Editing 13
GPT Generative Pre-Training 14
GPU Graphics Pprocessing Unit 5, 6, 62
GRU Gated Recurrent Unit 49, 50, 53
GS Gold Standard 24, 25
GT Ground Truth 26
GTCC GammaTone Cepstral Coefficients 61
GTX Giga Texel Shader eXtreme 5

HTC High Tech Computer 65, 66
HUAV Hospital Universitario Arnau de Vilanova 66

IA Inteligencia Artificial 7, 10–16, 23, 31, 72, 74, 94
IBM International Business Machines Corporation 11, 12
ICA Independent Component Analysis 20
IDE Integrated Development Environment 6
ILSVRC ImageNet Large Scale Visual Recognition Challenge 13
IMU Inertial Measurement Unit 15
IoU Intersection Over Union 26, 65

K-NN K-Nearest Neighbours 19, 22, 63, 113
KDE Kool Desktop Environment 5
KL Kullback-Leibler 56

LaMDA Language Model for Dialogue Applications 13, 14
LASSO Least Absolute Shrinkage and Selection Operator 43
LDA Linear Discriminant Analysis 20, 22, 68
LLD Low-Level Descriptor 69
LLM Large Language Model 14
LN Layer Norm 54
LOO Leave-One-Out 29
LPCM Linear Pulse Code Modulation 95
LPI Laboratorio de Procesado de Imagen 5, 6
LR Logistic Regression 68
LReLU Leaky Rectified Linear Unit 34
LSTM Long Short-Term Memory 12, 49, 50, 53, 80, 82–87, 89–92, 97, 99, 104, 107, 109, 117

M-health mobile-health 62
MBGD Mini-Batch Gradient Descent 39
MFCC Mel Frequency Cepstral Coefficients 61, 62, 64, 65, 73, 74, 114, 115
MIT Massachusetts Institute of Technology 11, 12, 72, 73
ML Machine Learning 7, 16, 18, 19, 22, 23, 31, 37, 44, 47, 57
MLP MultiLayer Perceptrons 48, 70, 71, 115, 116
MP3 Moving Picture Experts Group Layer-3 Audio 95
MSA Multi-head Self-Attention 54
MSE Mean Squared Error 37, 66
MSRA MicroSoft Research Asia 13
MT-NLG Megatron-Turing Natural Language Generation 14

Nadam Nesterov-accelerated Adaptive Moment Estimation 41
NAG Nesterov's Accelerated Gradient 39, 41
NB Naïve Bayes 68
NeRF Neural Radiance Fields 13
NLG Natural Language Generation 16
NLLB No Language Left Behind 14
NLP Natural Language Processing 16, 46
NLR Negativeeg Likelihood Ratio 25
NLU Natural Language Understanding 16
NMC Neumoconiosis de los Mineros del Carbón 2
NPV Negative Predictive Value 25, 102, 103, 105, 107–109, 113, 114

OMS Organización Mundial de la Salud 2, 3
OPT Open Pretrained Transformer 14

p.m. Post Meridiem 2
PaLM Pathways Language Model 14
PCA Principal Component Analysis 20
PLN Procesamiento del Lenguaje Natural 44
PLR Positive Likelihood Ratio 25
POS Part-Of-Speech 57
PPV Positive Predictive Value 25, 102, 103, 105, 107–109, 113–115
PR Precision-Recall 27
PReLU Parametric Rectified Linear Unit 34
PSD Power Spectral Density 64, 66
PT Prevalence Threshold 25

R2D2 Recurrent Replay Distributed DQN 13
RAM Random Access Memory 5
RASTA Relative Spectral Transform 70
RAT Rapid Antigen Test 68
RBM Restricted Boltzmann Machine 12
ReLU Rectified Linear Unit 33–35, 52, 70, 71, 73, 78, 79
ResNet Residual Network 13, 59, 116
RF Random Forest 68, 115
RFE Recursive Feature Elimination 67, 68, 115
RLHF Reinforcement Learning from Human Feedback 14, 20
RMSProp Root Mean Square Propagation 40
RNN Recurrent Neural Networks 48–50, 53, 54, 62
ROC Receiver Operating Characteristic 5, 7, 26–28, 101–109
RT-qPCR Reverse Transcription Quantitative Polymerase Chain Reaction 68
RTX Ray Tracing 5

SAM Segment Anything Model 13
SAOS Síndrome de Apnea Obstructiva del Sueño 3
SARS-CoV-2 Severe Acute Respiratory Syndrome CoronaVirus 2 2
SD Standard Deviation 65
SE Speech Enhancement 17
SELU Scaled Exponencial Linear Unit 34
SEN Sensitivity 25, 102, 103, 105, 107–109, 113–115
SENet Squeeze-and-Excitation Networks 13
Seq2Seq Sequence-to-Sequence 50
SER Speech Emotion Recognition 17
SFX Sound Effects 63
SGD Stochastic Gradient Descent 39
SiLU Sigmoid-Weighted Linear Unit 35
SM Segmentation Mask 26
SPE Specificity 25, 102, 103, 105, 107–109, 113–115
SR Speaker Recognition 17
SRN Simple Recurrent Networks 12
STFT Short-Time Fourier Transform 67, 84
STT Speech To Text 17
SVM Support Vector Machine 19, 21, 47, 48, 58, 68
SW-MSA Shifted Windowing - Multi-head Self-Attention 54
Swin-T Swin Transformer 54, 118

t-SNE t-distributed Stochastic Neighbor Embedding 20, 69
TFG Trabajo de Fin de Grado 101
TFM Trabajo de Fin de Master 1, 4, 7
TL Transfer Learning 57
TN True Negative 29
TNR True Negative Rate 25
TP True Positive 29
TPE Tree structured Parzen Estimator 64
TPR True Positive Rate 25, 26
TTS Text-To-Speech 14, 16, 17

UC University of Cambridge 66
UdL Universitat de Lleida 66
UE Unión Europea 4
USA United States of America 11
USM Universal Speech Model 14
UVa Universidad de Valladolid 5, 6

VAE Variational Autoencoder 56
VBR Variable Bit Rate 94
VGG Visual Geometry Group 59
ViT Vision Transformer 13, 54, 118
VN Verdadero Negativo 24, 29
VP Verdadero Positivo 23, 26, 29

W-MSA Windowing - Multi-head Self-Attention 54
WAV Waveform Audio File Format 63, 68, 72, 75, 95
WD Wigner distribution 67

XGBoost **E**xtreme **G**radient **Boost**ing 21, 64, 113, 114

YAMNet Yet Another Mobile Network 62
YOLO You Only Look Once 13

ZFNet Zeiler and Fergus Network 13

BIBLIOGRAFÍA

[1] A. B. Chang, «The physiology of cough», *Paediatric Respiratory Reviews*, vol. 7, n.º 1, págs. 2-8, 2006, ISSN: 1526-0542. DOI: 10.1016/j.prrv.2005.11.009. dirección: https://www.sciencedirect.com/science/article/pii/S1526054205001107.

[2] L. Orlandic, T. Teijeiro y D. Atienza, «The COUGHVID crowdsourcing dataset, a corpus for the study of large-scale cough analysis algorithms», *Scientific Data*, vol. 8, n.º 1, pág. 156, mayo de 2021, ISSN: 2052-4463. DOI: 10.1038/s41597-021-00937-4. dirección: https://www.nature.com/articles/s41597-021-00937-4.

[3] A. H. Morice, G. A. Fontana, M. G. Belvisi, S. S. Birring, K. F. Chung, P. V. Dicpinigaitis, J. A. Kastelik, L. P. McGarvey, J. A. Smith, M. Tatar y J. Widdicombe, «ERS guidelines on the assessment of cough», *European Respiratory Journal*, vol. 29, n.º 6, págs. 1256-1276, 2007, ISSN: 0903-1936. DOI: 10.1183/09031936.00101006. eprint: https://erj.ersjournals.com/content/29/6/1256.full.pdf. dirección: https://erj.ersjournals.com/content/29/6/1256.

[4] K. K. Lee y S. S. Birring, «Cough», *Medicine*, vol. 40, n.º 4, págs. 173-176, 2012, Respiratory Disorders: Part 1 of 3, ISSN: 1357-3039. DOI: 10.1016/j.mpmed.2012.01.002. dirección: https://www.sciencedirect.com/science/article/pii/S1357303912000035.

[5] G. A. Fontana, «Before We Get Started: What Is a Cough?», *Lung*, vol. 186, n.º 1, págs. 3-6, feb. de 2008, ISSN: 1432-1750. DOI: 10.1007/s00408-007-9036-8. dirección: https://link.springer.com/article/10.1007/s00408-007-9036-8.

[6] G. A. Fontana y J. Widdicombe, «What is cough and what should be measured?», *Pulmonary Pharmacology & Therapeutics*, vol. 20, n.º 4, págs. 307-312, 2007, Special Issue: Fourth International Symposium on Cough, ISSN: 1094-5539. DOI: 10.1016/j.pupt.2006.11.009. dirección: https://www.sciencedirect.com/science/article/pii/S1094553906001441.

[7] N. Zhu, D. Zhang, W. Wang, X. Li, B. Yang, J. Song, X. Zhao, B. Huang, W. Shi, R. Lu, P. Niu, F. Zhan, X. Ma, D. Wang, W. Xu, G. Wu, G. F. Gao y W. Tan, «A Novel Coronavirus from Patients with Pneumonia in China, 2019», en, *New England Journal of Medicine*, vol. 382, n.º 8, págs. 727-733, feb. de 2020, PMID: 31978945. DOI: 10.1056/NEJMoa2001017.

[8] World Health Organization (WHO), *Coronavirus (COVID-19) Cases and Deaths*, The Humanitarian Data Exchange (HDX), ed., https://data.humdata.org/dataset/coronavirus-covid-19-cases-and-deaths. (visitado 22-05-2023).

[9] N. I. of Health (NIH) National Library of Medicine (NLM), *Enfermedad pulmonar intersticial*, https://medlineplus.gov/spanish/ency/article/000128.htm. (visitado 22-05-2023).

[10] F. of International Respiratory Societies, *The Global Impact of Respiratory Disease*. European Respiratory Society, sep. de 2021. eprint: https://www.firsnet.org/images/publications/FIRS_Master_09202021.pdf. (visitado 22-05-2023).

[11] E. R. Society, *European White Lung Book. Chapter 2. The Economic Burden of Lung Disease*, 2021. dirección: https://www.erswhitebook.org/chapters/the-economic-burden-of-lung-disease/the-cost-of-respiratory-disease/ (visitado 22-05-2023).

[12] J. Smith y A. Woodcock, «New Developments in the Objective Assessment of Cough», *Lung*, vol. 186, n.º 1, págs. 48-54, feb. de 2008, ISSN: 1432-1750. DOI: 10.1007/s00408-007-9059-1.

[13] S. S. Birring, T. Fleming, S. Matos, A. A. Raj, D. H. Evans e I. D. Pavord, «The Leicester Cough Monitor: preliminary validation of an automated cough detection system in chronic cough», *European Respiratory Journal*, vol. 31, n.º 5, págs. 1013-1018, 2008, ISSN: 0903-1936. DOI: 10.1183/09031936.00057407. eprint: https://erj.ersjournals.com/content/31/5/1013.full.pdf. dirección: https://erj.ersjournals.com/content/31/5/1013.

[14] C. J. Ryerson, D. Donesky, S. Z. Pantilat y H. R. Collard, «Dyspnea in Idiopathic Pulmonary Fibrosis: A Systematic Review», *Journal of Pain and Symptom Management*, vol. 43, n.º 4, págs. 771-782, 2012, ISSN: 0885-3924. DOI: https://doi.org/10.1016/j.jpainsymman.2011.04.026. dirección: https://www.sciencedirect.com/science/article/pii/S0885392411007640.

[15] S. S. Birring, B. Prudon, A. J. Carr, S. J. Singh, M. D. L. Morgan e I. D. Pavord, «Development of a symptom specific health status measure for patients with chronic cough: Leicester Cough Questionnaire (LCQ)», *Thorax*, vol. 58, n.º 4, págs. 339-343, 2003, ISSN: 0040-6376. DOI: 10.1136/thorax.58.4.339. eprint: https://thorax.bmj.com/content/58/4/339.full.pdf. dirección: https://thorax.bmj.com/content/58/4/339.

[16] K. Chung, «Measurement of cough», *Respiratory Physiology & Neurobiology*, vol. 152, n.º 3, págs. 329-339, 2006, Cough and its Regulation, ISSN: 1569-9048. DOI: https://doi.org/10.1016/j.resp.2006.02.001. dirección: https://www.sciencedirect.com/science/article/pii/S1569904806000668.

[17] K. Kosasih, U. Abeyratne y V. Swarnkar, «High frequency analysis of cough sounds in pediatric patients with respiratory diseases», en *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, ago. de 2012, págs. 5654-5657. DOI: 10.1109/EMBC.2012.6347277.

[18] J. A. Smith, H. L. Ashurst, S. Jack y J. E. Woodcock Ashley A. and Earis, «The description of cough sounds by healthcare professionals», *Cough*, vol. 2, n.º 1, pág. 1, 2006. DOI: 10.1186/1745-9974-2-1. dirección: https://coughjournal.biomedcentral.com/articles/10.1186/1745-9974-2-1.

[19] J. Smith, «Ambulatory methods for recording cough», *Pulmonary Pharmacology & Therapeutics*, vol. 20, n.º 4, págs. 313-318, 2007, Special Issue: Fourth International Symposium on Cough, ISSN: 1094-5539. DOI: https://doi.org/10.1016/j.pupt.2006.10.016. dirección: https://www.sciencedirect.com/science/article/pii/S1094553906001325.

[20] C. of the European Communities, «Market study on telemedicine», European Union, inf. téc., oct. de 2018. eprint: https://health.ec.europa.eu/system/files/2019-08/2018_provision_marketstudy_telemedicine_en_0.pdf.

[21] S. J. Barry, A. D. Dane, A. H. Morice y A. D. Walmsley, «The automatic recognition and counting of cough», *Cough*, vol. 2, n.º 1, pág. 8, sep. de 2006, ISSN: 1745-9974. DOI: 10.1186/1745-9974-2-8. dirección: https://coughjournal.biomedcentral.com/articles/10.1186/1745-9974-2-8.

[22] E. C. Larson, T. Lee, S. Liu, M. Rosenfeld y S. N. Patel, «Accurate and Privacy Preserving Cough Sensing Using a Low-Cost Microphone», en *Proceedings of the 13th International Conference on Ubiquitous Computing*, ép. UbiComp '11, Beijing, China: Association for Computing Machinery, 2011, págs. 375-384, ISBN: 9781450306300. DOI: 10.1145/2030112.2030163. dirección: https://dl.acm.org/doi/10.1145/2030112.2030163.

[23] A. Windmon, M. Minakshi, P. Bharti, S. Chellappan, M. Johansson, B. A. Jenkins y P. R. Athilingam, «TussisWatch: A Smart-Phone System to Identify Cough Episodes as Early Symptoms of Chronic Obstructive Pulmonary Disease and Congestive Heart Failure», *IEEE Journal of Biomedical and Health Informatics*, vol. 23, n.º 4, págs. 1566-1573, jul. de 2019, ISSN: 2168-2208. DOI: 10.1109/JBHI.2018.2872038.

[24] F. Barata, K. Kipfer, M. Weber, P. Tinschert, E. Fleisch y T. Kowatsch, «Towards Device-Agnostic Mobile Cough Detection with Convolutional Neural Networks», en *2019 IEEE International Conference on Healthcare Informatics (ICHI)*, jun. de 2019, págs. 1-11. DOI: 10.1109/ICHI.2019.8904554.

[25] N. Simou, N. Stefanakis y P. Zervas, «A Universal System for Cough Detection in Domestic Acoustic Environments», en *2020 28th European Signal Processing Conference (EUSIPCO)*, ene. de 2021, págs. 111-115. DOI: 10.23919/Eusipco47968.2020.9287659.

[26] C. Parkinson y J. Woodall, «Automatic Speech Recognition (ASR) Feedback for Head Mounted Displays (HMD)», 20 150 142 440, US Patent App. 14/540,943, mayo de 2015. dirección: https://patents.google.com/patent/US20150142440A1/en (visitado 22-05-2023).

[27] O. Ghahabi, W. Zhou y V. Fischer, «A Robust Voice Activity Detection for Real-Time Automatic Speech Recognition», en *Studientexte zur Sprachkommunikation: Elektronische Sprachsignalverarbeitung 2018*, A. Berton, U. Haiber y W. Minker, eds., TUDpress, Dresden, 2018, págs. 85-91, ISBN: 978-3-959081-28-3. eprint: https://www.essv.de/pdf/2018_85_91.pdf?id=393. dirección: https://www.essv.de/paper.php?id=393.

[28] M. R. Price, J. R. Glass y A. P. Chandrakasan, «Low-Power Automatic Speech Recognition Device», 20 190 147 856, US Patent App. 16/099,589, mayo de 2019. dirección: https://patents.google.com/patent/US20190147856A1/en (visitado 22-05-2023).

[29] J. Monge-Álvarez, C. Hoyos-Barceló, P. Lesso y P. Casaseca-de-la-Higuera, «Robust Detection of Audio-Cough Events Using Local Hu Moments», *IEEE Journal of Biomedical and Health Informatics*, vol. 23, n.º 1, págs. 184-196, ene. de 2019, ISSN: 2168-2194. DOI: 10.1109/JBHI.2018.2800741.

[30] G. J. Gibson, R. Loddenkemper, B. Lundbäck e Y. Sibille, «Respiratory health and disease in Europe: the new European Lung White Book», *European Respiratory Journal*, vol. 42, n.º 3, págs. 559-563, 2013, ISSN: 0903-1936. DOI: 10.1183/09031936.00105513. eprint: https://erj.ersjournals.com/content/42/3/559.full.pdf. dirección: https://erj.ersjournals.com/content/42/3/559.

[31] J. Monge-Álvarez, C. Hoyos-Barceló, L. M. San-José-Revuelta y P. 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*, vol. 66, n.º 8, págs. 2319-2330, ago. de 2019, ISSN: 0018-9294. DOI: 10.1109/TBME.2018.2888998.

[32] *Documentación oficial de TensorFlow*, https://www.tensorflow.org/api_docs/python. (visitado 22-05-2023).

[33] *Documentación oficial de Keras*, https://keras.io/. (visitado 22-05-2023).

[34] *Documentación oficial de Numpy*, https://docs.scipy.org/doc/numpy/. (visitado 22-05-2023).

[35] *Documentación oficial de Scipy*, https://docs.scipy.org/doc/scipy/reference/. (visitado 22-05-2023).

[36] B. McFee, C. Raffel, D. Liang, D. Ellis, M. Mcvicar, E. Battenberg y O. Nieto, «librosa: Audio and Music Signal Analysis in Python», en *Proceedings of the 14th Python in Science Conference*, K. Huff y J. Bergstra, eds., ene. de 2015, págs. 18-24. DOI: 10.25080/Majora-7b98e3ed-003.

[37] *Audio Data Preparation and Augmentation*. dirección: https://www.tensorflow.org/io/tutorials/audio.

[38] *FFmpeg 6.0 "Von Neumann"*, feb. de 2023. dirección: https://ffmpeg.org/.

[39] *Documentación oficial de Matplotlib*, https://matplotlib.org/api/index.html#the-pyplot-api. (visitado 22-05-2023).

[40] *Documentación oficial de Pandas*, https://pandas-docs.github.io/pandas-docs-travis/reference/index.html. (visitado 22-05-2023).

[41] *Documentación oficial de Scikit-learn*, https://scikit-learn.org/stable/modules/classes.html. (visitado 22-05-2023).

[42] H. Buschmeier y M. Wlodarczak, «TextGridTools: A TextGrid processing and analysis toolkit for Python», en *Tagungsband der 24. Konferenz zur Elektronischen Sprachsignalverarbeitung (ESSV 2013)*, 2013, págs. 152-157. DOI: 10.5281/zenodo.2582132. dirección: https://github.com/hbuschme/TextGridTools.

[43] P. Boersma y D. Weenink, «PRAAT: Doing phonetics by computer (Version 6.0.52)», mayo de 2019. dirección: http://www.fon.hum.uva.nl/praat/.

[44] W. Mora-Flores y A. Borbón-Alpízar, *Edición de Textos Científicos con LATEX. Composición, Diseño Editorial, Gráficos, Inkscape, Tikz y Presentaciones Beamer*, Segunda Edición. Escuela de Matemática, Instituto Tecnológico de Costa Rica., 2012, ISBN: 978-9977-66-227-5.

[45] *TeXstudio*. dirección: https://www.texstudio.org/ (visitado 22-05-2023).

[46] N. Oliver, *La Historia de la Inteligencia Artificial: Un paseo personal por la historia de la IA*. dirección: https://ellisalicante.org/book/historia-de-la-inteligencia-artificial (visitado 22-05-2023).

[47] A. Hernández Silva, «Diseño de redes neuronales antagónicas para estegoanálisis», jul. de 2020, dirección: https://oa.upm.es/63719/.

[48] R. Descartes, «Discours de la Méthode Pour bien conduire sa raison, et chercher la vérité dans les sciences», 1637.

[49] ——, «Meditationes de Prima Philosophia, in qua Dei existentia et animæ inmortalitas demonstratur (en francés: Méditations Métaphysiques)», 1641.

[50] J. Cottingham, R. Stoothoff, D. Murdoch, A. Kenny y col., *The philosophical writings of Descartes*. Cambridge University Press Cambridge, 1985.

[51] A. Lovelace, «Translator's Notes to an Article on Babbage's Analytical Engine», *Scientific Memoirs*, vol. 3, págs. 691-731, 1842. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-72449165714&partnerID=40&md5=e6156caaba5b892992f68208e3a13a58.

[52] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *The bulletin of mathematical biophysics*, vol. 5, n.º 4, págs. 115-133, dic. de 1943, ISSN: 1522-9602. DOI: 10.1007/BF02478259. dirección: https://link.springer.com/article/10.1007/BF02478259.

[53] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. Psychology Press, 2005, ISBN: 9781135631901. dirección: https://books.google.es/books?id=ddB4AgAAQBAJ.

[54] A. M. Turing, «I.—COMPUTING MACHINERY AND INTELLIGENCE», *Mind*, vol. LIX, n.º 236, págs. 433-460, oct. de 1950, ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433. eprint: https://academic.oup.com/mind/article-pdf/LIX/236/433/30123314/lix-236-433.pdf. dirección: https://academic.oup.com/mind/article/LIX/236/433/986238.

[55] B. G. Buchanan, «A (Very) Brief History of Artificial Intelligence», *AI Magazine*, vol. 26, n.º 4, pág. 53, dic. de 2005. DOI: 10.1609/aimag.v26i4.1848. dirección: https://ojs.aaai.org/index.php/aimagazine/article/view/1848.

[56] F. Rosenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain.», *Psychological Review*, vol. 65, págs. 386-408, 1958. DOI: 10.1037/h0042519. dirección: https://psycnet.apa.org/record/1959-09865-001.

[57] B. Widrow y M. E. Hoff, «Adaptive switching circuits», Stanford University California. Stanford Electronics Laboratories et. al., inf. téc., jun. de 1960. eprint: https://apps.dtic.mil/dtic/tr/fulltext/u2/241531.pdf.

[58] M. L. Minsky, S. A. Papert y L. Bottou, *Perceptrons: An Introduction to Computational Geometry*, ép. The MIT Press. MIT Press, 2017, ISBN: 9780262534772. dirección: https://books.google.es/books?id=PLQ5DwAAQBAJ.

[59] J. A. Anderson, J. W. Silverstein, S. A. Ritz y R. S. Jones, «Distinctive features, categorical perception, and probability learning: Some applications of a neural model.», *Psychological Review*, vol. 84, n.º 5, págs. 413-451, sep. de 1977, ISSN: 1939-1471(Electronic), 0033-295X(Print). DOI: 10.1037/0033-295X.84.5.413.

[60] D. Marr, «Vision: A computational investigation into the human representation and processing of visual information», *WH San Francisco: Freeman and Company, San Francisco*, 1982.

[61] J. J. Hopfield, «Neural networks and physical systems with emergent collective computational abilities», *Proceedings of the National Academy of Sciences*, vol. 79, n.º 8, págs. 2554-2558, 1982, ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554. eprint: https://www.pnas.org/content/79/8/2554.full.pdf. dirección: https://www.pnas.org/content/79/8/2554.

[62] J. Pearl, «Bayesian Networks: a Model of Self-Activated Memory for Evidential Reasoning», en *Proceedings of the 7th Conference of the Cognitive Science Society, 1985*, 1985, págs. 329-334. eprint: http://ftp.cs.ucla.edu/tech-report/198_-reports/850017.pdf.

[63] D. H. Ackley, G. E. Hinton y T. J. Sejnowski, «A learning algorithm for boltzmann machines», *Cognitive Science*, vol. 9, n.º 1, págs. 147-169, 1985, ISSN: 0364-0213. DOI: 10.1016/S0364-0213(85)80012-4. dirección: https://www.sciencedirect.com/science/article/pii/S0364021385800124.

[64] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975. dirección: https://books.google.es/books?id=z81XmgEACAAJ.

[65] D. B. Parker, *Learning-logic: Casting the Cortex of the Human Brain in Silicon*, ép. Technical report: Center for Computational Research in Economics and Management Science. Massachusetts Institute of Technology, Center for Computational Research in Economics y Management Science, 1985. dirección: https://books.google.es/books?id=2kS9GwAACAAJ.

[66] Y. Le Cun, «Learning Process in an Asymmetric Threshold Network», en *Disordered Systems and Biological Organization*, E. Bienenstock, F. F. Soulié y G. Weisbuch, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, págs. 233-240, ISBN: 978-3-642-82657-3. DOI: 10.1007/978-3-642-82657-3_24.

[67] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning internal representations by error propagation», Institute for Cognitive Science, University of California San Diego, La Jolla, inf. téc., 1985. eprint: https://apps.dtic.mil/dtic/tr/fulltext/u2/a164453.pdf.

[68] H. Bourlard e Y. Kamp, «Auto-association by multilayer perceptrons and singular value decomposition», *Biological Cybernetics*, vol. 59, n.º 4, págs. 291-294, sep. de 1988, ISSN: 1432-0770. DOI: 10.1007/BF00332918. dirección: https://link.springer.com/article/10.1007/BF00332918.

[69] M. I. Jordan, «Serial order: a parallel distributed processing approach. Technical report, June 1985-March 1986», mayo de 1986. DOI: 10.1016/S0166-4115(97)80111-2. dirección: https://www.osti.gov/biblio/6910294.

[70] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard y L. D. Jackel, «Backpropagation Applied to Handwritten Zip Code Recognition», *Neural Computation*, vol. 1, n.º 4, págs. 541-551, dic. de 1989, ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. eprint: https://direct.mit.edu/neco/article-pdf/1/4/541/811941/neco.1989.1.4.541.pdf. dirección: https://direct.mit.edu/neco/article-abstract/1/4/541/5515/Backpropagation-Applied-to-Handwritten-Zip-Code.

[71] R. A. Brooks, «Elephants don't play chess», *Robotics and Autonomous Systems*, vol. 6, n.º 1, págs. 3-15, 1990, Designing Autonomous Agents, ISSN: 0921-8890. DOI: https://doi.org/10.1016/S0921-8890(05)80025-9. dirección: https://www.sciencedirect.com/science/article/pii/S0921889005800259.

[72] M. A. Turk y A. P. Pentland, «Face recognition using eigenfaces», en *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*, IEEE Computer Society, 1991, págs. 586-587.

[73] S. Hochreiter y J. Schmidhuber, «Long Short-Term Memory», *Neural Computation*, vol. 9, n.º 8, págs. 1735-1780, nov. de 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. dirección: https://direct.mit.edu/neco/article-abstract/9/8/1735/6109/Long-Short-Term-Memory.

[74] M. Campbell, A. J. Hoane Jr y F.-h. Hsu, «Deep blue», *Artificial intelligence*, vol. 134, n.º 1-2, págs. 57-83, 2002, ISSN: 0004-3702. DOI: 10.1016/S0004-3702(01)00129-1. dirección: http://www.sciencedirect.com/science/article/pii/S0004370201001291.

[75] G. E. Hinton, S. Osindero e Y.-W. Teh, «A Fast Learning Algorithm for Deep Belief Nets», *Neural Computation*, vol. 18, n.º 7, págs. 1527-1554, jul. de 2006, ISSN: 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. eprint: https://direct.mit.edu/neco/article-pdf/18/7/1527/816558/neco.2006.18.7.1527.pdf. dirección: https://direct.mit.edu/neco/article-abstract/18/7/1527/7065/A-Fast-Learning-Algorithm-for-Deep-Belief-Nets.

[76] R. Salakhutdinov y G. Hinton, «Deep Boltzmann Machines», en *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, D. van Dyk y M. Welling, eds., ép. Proceedings of Machine Learning Research, vol. 5, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA: PMLR, abr. de 2009, págs. 448-455. dirección: https://proceedings.mlr.press/v5/salakhutdinov09a.html.

[77] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh y A. Hung Byers, *Big data: the next frontier for innovation, competition, and productivity*, 2011. dirección: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/big-data-the-next-frontier-for-innovation (visitado 22-05-2023).

[78] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei, «ImageNet: A large-scale hierarchical image database», en *2009 IEEE Conference on Computer Vision and Pattern Recognition*, jun. de 2009, págs. 248-255. DOI: 10.1109/CVPR.2009.5206848.

[79] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg y L. Fei-Fei, «ImageNet Large Scale Visual Recognition Challenge», *International Journal of Computer Vision*, vol. 115, n.º 3, págs. 211-252, dic. de 2015, ISSN: 1573-1405. DOI: 10.1007/s11263-015-0816-y. dirección: https://link.springer.com/article/10.1007/s11263-015-0816-y?sa_campaign=email/event/articleAuthor/onlineFirst#.

[80] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», *Commun. ACM*, vol. 60, n.º 6, págs. 84-90, mayo de 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. dirección: https://dl.acm.org/doi/abs/10.1145/3065386.

[81] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, «Going deeper with convolutions», en *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2015, págs. 1-9. DOI: 10.1109/CVPR.2015.7298594. dirección: https://ieeexplore.ieee.org/document/7298594.

[82] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition», en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2016, págs. 770-778. DOI: 10.1109/CVPR.2016.90.

[83] S. Xie, R. Girshick, P. Dollár, Z. Tu y K. He, «Aggregated Residual Transformations for Deep Neural Networks», en *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jul. de 2017, págs. 5987-5995. DOI: 10.1109/CVPR.2017.634.

[84] J. Hu, L. Shen, S. Albanie, G. Sun y E. Wu, «Squeeze-and-Excitation Networks», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, n.º 8, págs. 2011-2023, ago. de 2020, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2019.2913372.

[85] X. Zhai, A. Kolesnikov, N. Houlsby y L. Beyer, «Scaling Vision Transformers», en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2022, págs. 12 104-12 113. DOI: 10.48550/arXiv.2106.04560. eprint: https://openaccess.thecvf.com/content/CVPR2022/papers/Zhai_Scaling_Vision_Transformers_CVPR_2022_paper.pdf. dirección: https://openaccess.thecvf.com/content/CVPR2022/html/Zhai_Scaling_Vision_Transformers_CVPR_2022_paper.html.

[86] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger e I. Sutskever, «Learning Transferable Visual Models From Natural Language Supervision», en *Proceedings of the 38th International Conference on Machine Learning*, M. Meila y T. Zhang, eds., ép. Proceedings of Machine Learning Research, vol. 139, PMLR, jul. de 2021, págs. 8748-8763. dirección: https://proceedings.mlr.press/v139/radford21a.html.

[87] OpenAI, *CLIP: Connecting Text and Images*, https://openai.com/blog/clip/, 2021. (visitado 22-05-2023).

[88] J. Yu, Z. Wang, V. Vasudevan, L. Yeung, M. Seyedhosseini e Y. Wu, *CoCa: Contrastive Captioners are Image-Text Foundation Models*, 2022. DOI: 10.48550/ARXIV.2205.01917. dirección: https://arxiv.org/abs/2205.01917.

[89] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection», en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2016, págs. 779-788. DOI: 10.1109/CVPR.2016.91.

[90] J. Redmon y A. Farhadi, «YOLO9000: Better, Faster, Stronger», en *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jul. de 2017, págs. 6517-6525. DOI: 10.1109/CVPR.2017.690.

[91] M. J. Shafiee, B. Chywl, F. Li y A. Wong, «Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video», *CoRR*, vol. abs/1709.05943, 2017. arXiv: 1709.05943. dirección: http://arxiv.org/abs/1709.05943.

[92] H. Nakahara, M. Shimoda y S. Sato, «A Demonstration of FPGA-Based You Only Look Once Version2 (YOLOv2)», en *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, ago. de 2018, págs. 457-4571. DOI: 10.1109/FPL.2018.00088.

[93] J. Redmon y A. Farhadi, «YOLOv3: An Incremental Improvement», *CoRR*, vol. abs/1804.02767, 2018. arXiv: 1804.02767. dirección: http://arxiv.org/abs/1804.02767.

[94] A. Bochkovskiy, C.-Y. Wang y H.-Y. M. Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection», *CoRR*, vol. abs/2004.10934, 2020. arXiv: 2004.10934. dirección: https://arxiv.org/abs/2004.10934.

[95] C.-Y. Wang, A. Bochkovskiy y H.-Y. M. Liao, «Scaled-YOLOv4: Scaling Cross Stage Partial Network», en *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2021, págs. 13 024-13 033. DOI: 10.1109/CVPR46437.2021.01283.

[96] C.-Y. Wang, I.-H. Yeh y H.-Y. M. Liao, «You Only Learn One Representation: Unified Network for Multiple Tasks», *CoRR*, vol. abs/2105.04206, 2021. arXiv: 2105.04206. dirección: https://arxiv.org/abs/2105.04206.

[97] Z. Ge, S. Liu, F. Wang, Z. Li y J. Sun, «YOLOX: Exceeding YOLO Series in 2021», *CoRR*, vol. abs/2107.08430, 2021. arXiv: 2107.08430. dirección: https://arxiv.org/abs/2107.08430.

[98] S. Xu, X. Wang, W. Lv, Q. Chang, C. Cui, K. Deng, G. Wang, Q. Dang, S. Wei, Y. Du y B. Lai, *PP-YOLOE: An evolved version of YOLO*, 2022. DOI: 10.48550/ARXIV.2203.16250. dirección: https://arxiv.org/abs/2203.16250.

[99] C.-Y. Wang, A. Bochkovskiy y H.-Y. M. Liao, *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*, 2022. DOI: 10.48550/ARXIV.2207.02696. dirección: https://arxiv.org/abs/2207.02696.

[100] Amazon, *Amazon Go: Frequently Asked Questions*, https://amazon.com/go, 2023. (visitado 22-05-2023).

[101] D. Ardila, A. P. Kiraly, S. Bharadwaj, B. Choi, J. J. Reicher, L. Peng, D. Tse, M. Etemadi, W. Ye, G. Corrado, D. P. Naidich y S. Shetty, «End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography», *Nature Medicine*, vol. 25, n.º 6, págs. 954-961, jun. de 2019, ISSN: 1546-170X. DOI: 10.1038/s41591-019-0447-x. dirección: https://www.nature.com/articles/s41591-019-0447-x.

[102] S. M. McKinney, M. Sieniek, V. Godbole, J. Godwin, N. Antropova, H. Ashrafian, T. Back, M. Chesus, G. S. Corrado, A. Darzi, M. Etemadi, F. Garcia-Vicente, F. J. Gilbert, M. Halling-Brown, D. Hassabis, S. Jansen, A. Karthikesalingam, C. J. Kelly, D. King, J. R. Ledsam, D. Melnick, H. Mostofi, L. Peng, J. J. Reicher, B. Romera-Paredes, R. Sidebottom, M. Suleyman, D. Tse, K. C. Young, J. De Fauw y S. Shetty, «International evaluation of an AI system for breast cancer screening», *Nature*, vol. 577, n.º 7788, págs. 89-94, ene. de 2020, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1799-6. dirección: https://www.nature.com/articles/s41586-019-1799-6.

[103] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár y R. Girshick, *Segment Anything*, 2023. arXiv: 2304.02643 [cs.CV].

[104] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu y M. Chen, *Hierarchical Text-Conditional Image Generation with CLIP Latents*, 2022. DOI: 10.48550/ARXIV.2204.06125. dirección: https://arxiv.org/abs/2204.06125.

[105] OpenAI, *DALL-E 2*, https://openai.com/dall-e-2/, 2022. (visitado 22-05-2023).

[106] A. Q. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. Mcgrew, I. Sutskever y M. Chen, «GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models», en *Proceedings of the 39th International Conference on Machine Learning*, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu y S. Sabato, eds., ép. Proceedings of Machine Learning Research, vol. 162, PMLR, jul. de 2022, págs. 16 784-16 804. dirección: https://proceedings.mlr.press/v162/nichol22a.html.

[107] Midjourney, *Midjourney*, https://www.midjourney.com/, 2022. (visitado 22-05-2023).

[108] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet y M. Norouzi, *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*, 2022. DOI: 10.48550/ARXIV.2205.11487. dirección: https://arxiv.org/abs/2205.11487.

[109] R. Rombach, A. Blattmann, D. Lorenz, P. Esser y B. Ommer, «High-Resolution Image Synthesis With Latent Diffusion Models», en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2022, págs. 10 684-10 695.

[110] S. Forsgren y H. Martiros, *Riffusion - Stable diffusion for real-time music generation*, 2022. dirección: https://riffusion.com/about.

[111] Y. Mehdi, *Create images with your words – Bing Image Creator comes to the new Bing*, https://blogs.microsoft.com/blog/2023/03/21/create-images-with-your-words-bing-image-creator-comes-to-the-new-bing/, mar. de 2023. (visitado 22-05-2023).

[112] S. Wang, C. Saharia, C. Montgomery, J. Pont-Tuset, S. Noy, S. Pellegrini, Y. Onoe, S. Laszlo, D. J. Fleet, R. Soricut, J. Baldridge, M. Norouzi, P. Anderson y W. Chan, *Imagen Editor and EditBench: Advancing and Evaluating Text-Guided Image Inpainting*, 2022. DOI: 10.48550/ARXIV.2212.06909. dirección: https://arxiv.org/abs/2212.06909.

[113] J. Ho, W. Chan, C. Saharia, J. Whang, R. Gao, A. Gritsenko, D. P. Kingma, B. Poole, M. Norouzi, D. J. Fleet y T. Salimans, *Imagen Video: High Definition Video Generation with Diffusion Models*, 2022. DOI: 10.48550/ARXIV.2210.02303. dirección: https://arxiv.org/abs/2210.02303.

[114] T. Müller, A. Evans, C. Schied y A. Keller, «Instant Neural Graphics Primitives with a Multiresolution Hash Encoding», *ACM Trans. Graph.*, vol. 41, n.º 4, jul. de 2022, ISSN: 0730-0301. DOI: 10.1145/3528223.3530127. dirección: https://dl.acm.org/doi/10.1145/3528223.3530127.

[115] B. Poole, A. Jain, J. T. Barron y B. Mildenhall, *DreamFusion: Text-to-3D using 2D Diffusion*, 2022. DOI: 10.48550/ARXIV.2209.14988. dirección: https://arxiv.org/abs/2209.14988.

[116] Y. Li, Z.-H. Lin, D. Forsyth, J.-B. Huang y S. Wang, *ClimateNeRF: Physically-based Neural Rendering for Extreme Climate Synthesis*, 2022. DOI: 10.48550/ARXIV.2211.13226. dirección: https://arxiv.org/abs/2211.13226.

[117] D. Silver, A. Huang, A. Maddison Chris J. and Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, D. Dieleman Sander and Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, M. Lillicrap Timothy and Leach, T. Kavukcuoglu Koray and Graepel y D. Hassabis, «Mastering the game of Go with deep neural networks and tree search», *Nature*, vol. 529, n.º 7587, págs. 484-489, ene. de 2016, ISSN: 1476-4687. DOI: 10.1038/nature16961. dirección: https://www.nature.com/articles/nature16961.

[118] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel y D. Hassabis, «Mastering the game of Go without human knowledge», *Nature*, vol. 550, n.º 7676, págs. 354-359, oct. de 2017, ISSN: 1476-4687. DOI: 10.1038/nature24270. dirección: https://www.nature.com/articles/nature24270.

[119] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan y D. Hassabis, «A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play», *Science*, vol. 362, n.º 6419, págs. 1140-1144, 2018, ISSN: 0036-8075. DOI: 10.1126/science.aar6404. eprint: https://science.sciencemag.org/content/362/6419/1140.full.pdf. dirección: https://www.science.org/doi/10.1126/science.aar6404.

[120] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, E. Guez Arthur and Lockhart, D. Hassabis, T. Graepel Thore and Lillicrap y D. Silver, «Mastering Atari, Go, chess and shogi by planning with a learned model», *Nature*, vol. 588, n.º 7839, págs. 604-609, dic. de 2020, ISSN: 1476-4687. DOI: 10.1038/s41586-020-03051-4. dirección: https://www.nature.com/articles/s41586-020-03051-4.

[121] S. Kapturowski, G. Ostrovski, W. Dabney, J. Quan y R. Munos, «Recurrent Experience Replay in Distributed Reinforcement Learning», en *International Conference on Learning Representations*, 2019. eprint: https://openreview.net/pdf?id=r1lyTjAqYX. dirección: https://openreview.net/forum?id=r1lyTjAqYX.

[122] W. Ye, S. Liu, T. Kurutach, P. Abbeel e Y. Gao, «Mastering Atari Games with Limited Data», en *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang y J. W. Vaughan, eds., 2021. eprint: https://openreview.net/pdf?id=OKrNPg3xR3T. dirección: https://openreview.net/forum?id=OKrNPg3xR3T.

[123] O. Vinyals, I. Babuschkin, J. Chung, M. Mathieu, M. Jaderberg, W. M. Czarnecki, A. Dudzik, A. Huang, P. Georgiev, R. Powell, T. Ewalds, D. Horgan, M. Kroiss, I. Danihelka, J. Agapiou, J. Oh, V. Dalibard, D. Choi, L. Sifre, Y. Sulsky, S. Vezhnevets, J. Molloy, T. Cai, D. Budden, T. Paine, C. Gulcehre, Z. Wang, T. Pfaff, T. Pohlen, Y. Wu, D. Yogatama, J. Cohen, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, C. Apps, K. Kavukcuoglu, D. Hassabis y D. Silver, *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*, https://www.deepmind.com/blog/alphastar-mastering-the-real-time-strategy-game-starcraft-ii, 2019. (visitado 22-05-2023).

[124] V. F. Zambaldi, D. Raposo, A. S. and Victor Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. P. R. and Timothy P. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. P. and Matthew Botvinick y O. V. and Peter W. Battaglia, «Relational Deep Reinforcement Learning», *CoRR*, vol. abs/1806.01830, 2018. arXiv: 1806.01830. dirección: http://arxiv.org/abs/1806.01830.

[125] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. Botvinick, O. Vinyals y P. Battaglia, «Deep reinforcement learning with relational inductive biases», en *International Conference on Learning Representations*, 2019. eprint: https://openreview.net/pdf?id=HkxaFoC9KQ. dirección: https://openreview.net/forum?id=HkxaFoC9KQ.

[126] K. Arulkumaran, A. Cully y J. Togelius, «AlphaStar: An Evolutionary Computation Perspective», en *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ép. GECCO '19, Prague, Czech Republic: Association for Computing Machinery, 2019, págs. 314-315, ISBN: 9781450367486. DOI: 10.1145/3319619.3321894. dirección: https://dl.acm.org/doi/10.1145/3319619.3321894.

[127] M. Čertický, D. Churchill, K.-J. Kim, M. Čertický y R. Kelly, «StarCraft AI Competitions, Bots, and Tournament Manager Software», *IEEE Transactions on Games*, vol. 11, n.º 3, págs. 227-237, sep. de 2019, ISSN: 2475-1510. DOI: 10.1109/TG.2018.2883499.

[128] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps y D. Silver, «Grandmaster level in StarCraft II using multi-agent reinforcement learning», *Nature*, vol. 575, n.º 7782, págs. 350-354, nov. de 2019, ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z. dirección: https://www.nature.com/articles/s41586-019-1724-z.

[129] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu y T. Graepel, «Human-level performance in 3D multiplayer games with population-based reinforcement learning», *Science*, vol. 364, n.º 6443, págs. 859-865, 2019. DOI: 10.1126/science.aau6249. eprint: https://www.science.org/doi/pdf/10.1126/science.aau6249. dirección: https://www.science.org/doi/abs/10.1126/science.aau6249.

[130] OpenAI, *OpenAI Five*, https://blog.openai.com/openai-five/, 2018. (visitado 22-05-2023).

[131] N. Brown y T. Sandholm, «Superhuman AI for heads-up no-limit poker: Libratus beats top professionals», *Science*, vol. 359, n.º 6374, págs. 418-424, 2018. DOI: 10.1126/science.aao1733. eprint: https://www.science.org/doi/pdf/10.1126/science.aao1733. dirección: https://www.science.org/doi/abs/10.1126/science.aao1733.

[132] OpenAI, *Learning to Play Minecraft with Video PreTraining (VPT)*, https://openai.com/blog/vpt/, 2022. (visitado 22-05-2023).

[133] B. Baker, I. Akkaya, P. Zhokhov, J. Huizinga, J. Tang, A. Ecoffet, B. Houghton, R. Sampedro y J. Clune, *Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos*, 2022. DOI: 10.48550/ARXIV.2206.11795. dirección: https://arxiv.org/abs/2206.11795.

[134] D. Hafner, J. Pasukonis, J. Ba y T. Lillicrap, *Mastering Diverse Domains through World Models*, 2023. DOI: 10.48550/ARXIV.2301.04104. dirección: https://arxiv.org/abs/2301.04104.

[135] DeepMind, *Discovering novel algorithms with AlphaTensor*, https://www.deepmind.com/blog/discovering-novel-algorithms-with-alphatensor, 2022. (visitado 22-05-2023).

[136] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis y P. Kohli, «Discovering faster matrix multiplication algorithms with reinforcement learning», *Nature*, vol. 610, n.º 7930, págs. 47-53, oct. de 2022, ISSN: 1476-4687. DOI: 10.1038/s41586-022-05172-4. dirección: https://www.nature.com/articles/s41586-022-05172-4.

[137] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, A. Kay Jackie and Merle, J.-M. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis y M. Riedmiller, «Magnetic control of tokamak plasmas through deep reinforcement learning», *Nature*, vol. 602, n.º 7897, págs. 414-419, feb. de 2022, ISSN: 1476-4687. DOI: 10.1038/s41586-021-04301-9. dirección: https://www.nature.com/articles/s41586-021-04301-9.

[138] DeepMind, *Timeline of a breakthrough*, https://www.deepmind.com/research/highlighted-research/alphafold/timeline-of-a-breakthrough, 2022. (visitado 22-05-2023).

[139] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, A. Bridgland, C. Meyer, S. A. A. Kohl, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, E. Clancy, M. Zielinski, M. Steinegger, M. Pacholska, T. Berghammer, S. Bodenstein, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli y D. Hassabis, «Highly accurate protein structure prediction with AlphaFold», *Nature*, vol. 596, n.º 7873, págs. 583-589, ago. de 2021, ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2. dirección: https://www.nature.com/articles/s41586-021-03819-2.

[140] R. Evans, M. O'Neill, A. Pritzel, N. Antropova, A. Senior, T. Green, A. Žídekdek, R. Bates, S. Blackwell, J. Yim, O. Ronneberger, S. Bodenstein, M. Zielinski, A. Bridgland, A. Potapenko, A. Cowie, K. Tunyasuvunakool, R. Jain, E. Clancy, P. Kohli, J. Jumper y D. Hassabis, «Protein complex prediction with AlphaFold-Multimer», *bioRxiv*, 2022. DOI: 10.1101/2021.10.04.463034. eprint: https://www.biorxiv.org/content/early/2022/03/10/2021.10.04.463034.full.pdf. dirección: https://www.biorxiv.org/content/early/2022/03/10/2021.10.04.463034.

[141] J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding», *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. dirección: http://arxiv.org/abs/1810.04805.

[142] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser e I. Polosukhin, «Attention is All You Need», en *International Conference on Advances in Neural Information Processing Systems 30 (NIPS 2017)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan y R. Garnett, eds., ép. NIPS'17, vol. 30, Long Beach, California, USA: Curran Associates Inc., 2017, págs. 6000-6010, ISBN: 9781510860964. dirección: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[143] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, Y. Li, H. Lee, H. S. Zheng, A. Ghafouri, M. Menegali, Y. Huang, M. Krikun, D. Lepikhin, J. Qin, D. Chen, Y. Xu, Z. Chen, A. Roberts, M. Bosma, V. Zhao, Y. Zhou, C.-C. Chang, I. Krivokon, W. Rusch, M. Pickett, P. Srinivasan, L. Man, K. Meier-Hellstern, M. R. Morris, T. Doshi, R. D. Santos, T. Duke, J. Soraker, B. Zevenbergen, V. Prabhakaran, M. Diaz, B. Hutchinson, K. Olson, A. Molina, E. Hoffman-John, J. Lee, L. Aroyo, R. Rajakumar, A. Butryna, M. Lamm, V. Kuzmina, J. Fenton, A. Cohen, R. Bernstein, R. Kurzweil, B. Aguera-Arcas, C. Cui, M. Croak, E. Chi y Q. Le, «LaMDA: Language Models for Dialog Applications», *CoRR*, vol. abs/2201.08239, 2022. DOI: 10.48550/ARXIV.2201.08239. arXiv: 2201.08239. dirección: https://arxiv.org/abs/2201.08239.

[144] B. Lemoine, *Is LaMDA Sentient? — an Interview*, https://cajundiscordian.medium.com/is-lamda-sentient-an-interview-ea64d916d917, 2022. (visitado 22-05-2023).

[145] N. Tiku, *The Google engineer who thinks the company's AI has come to life*, https://www.washingtonpost.com/technology/2022/06/11/google-ai-lamda-blake-lemoine/, 2022. (visitado 22-05-2023).

[146] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. v. d. Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals y L. Sifre, *Training Compute-Optimal Large Language Models*, 2022. DOI: 10.48550/ARXIV.2203.15556. dirección: https://arxiv.org/abs/2203.15556.

[147] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, E. Rutherford, T. Hennigan, J. Menick, A. Cassirer, R. Powell, G. v. d. Driessche, L. A. Hendricks, M. Rauh, P.-S. Huang, A. Glaese, J. Welbl, S. Dathathri, S. Huang, J. Uesato, J. Mellor, I. Higgins, A. Creswell, N. McAleese, A. Wu, E. Elsen, S. Jayakumar, E. Buchatskaya, D. Budden, E. Sutherland, K. Simonyan, M. Paganini, L. Sifre, L. Martens, X. L. Li, A. Kuncoro, A. Nematzadeh, E. Gribovskaya, D. Donato, A. Lazaridou, A. Mensch, J.-B. Lespiau, M. Tsimpoukelli, N. Grigorev, D. Fritz, T. Sottiaux, M. Pajarskas, T. Pohlen, Z. Gong, D. Toyama, C. d. M. dÁutume, Y. Li, T. Terzi, V. Mikulik, I. Babuschkin, A. Clark, D. d. L. Casas, A. Guy, C. Jones, J. Bradbury, M. Johnson, B. Hechtman, L. Weidinger, I. Gabriel, W. Isaac, E. Lockhart, S. Osindero, L. Rimell, C. Dyer, O. Vinyals, K. Ayoub, J. Stanway, L. Bennett, D. Hassabis, K. Kavukcuoglu y G. Irving, «Scaling Language Models: Methods, Analysis & Insights from Training Gopher», *CoRR*, vol. abs/2112.11446, 2021. DOI: 10.48550/ARXIV.2112.11446. arXiv: 2112.11446. dirección: https://arxiv.org/abs/2112.11446.

[148] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov y N. Fiedel, *PaLM: Scaling Language Modeling with Pathways*, 2022. DOI: 10.48550/ARXIV.2204.02311. dirección: https://arxiv.org/abs/2204.02311.

[149] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet, N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan y A. Zeng, *Do As I Can, Not As I Say: Grounding Language in Robotic Affordances*, 2022. DOI: 10.48550/ARXIV.2204.01691. dirección: https://arxiv.org/abs/2204.01691.

[150] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur, G. Gur-Ari y V. Misra, *Solving Quantitative Reasoning Problems with Language Models*, 2022. DOI: 10.48550/ARXIV.2206.14858. dirección: https://arxiv.org/abs/2206.14858.

[151] K. Singhal, S. Azizi, T. Tu, S. S. Mahdavi, J. Wei, H. W. Chung, N. Scales, A. Tanwani, H. Cole-Lewis, S. Pfohl, P. Payne, M. Seneviratne, P. Gamble, C. Kelly, N. Scharli, A. Chowdhery, P. Mansfield, B. A. y. Arcas, D. Webster, G. S. Corrado, Y. Matias, K. Chou, J. Gottweis, N. Tomasev, Y. Liu, A. Rajkomar, J. Barral, C. Semturs, A. Karthikesalingam y V. Natarajan, *Large Language Models Encode Clinical Knowledge*, 2022. DOI: 10.48550/ARXIV.2212.13138. dirección: https://arxiv.org/abs/2212.13138.

[152] J. V. Wright, *IA en Google: PaLM 2*, https://ai.google/discover/palm2, mayo de 2023. (visitado 22-05-2023).

[153] S. Pichai, *An important next step on our AI journey*, https://blog.google/technology/ai/bard-google-ai-search-updates/, feb. de 2023. (visitado 22-05-2023).

[154] J. V. Wright, *An important next step on our AI journey*, https://workspace.google.com/blog/product-announcements/generative-ai, mar. de 2023. (visitado 22-05-2023).

[155] K. Singhal, T. Tu, J. Gottweis, R. Sayres, E. Wulczyn, L. Hou, K. Clark, S. Pfohl, H. Cole-Lewis, D. Neal, M. Schaekermann, A. Wang, M. Amin, S. Lachgar, P. Mansfield, S. Prakash, B. Green, E. Dominowska, B. A. y Arcas, N. Tomasev, Y. Liu, R. Wong, C. Semturs, S. S. Mahdavi, J. Barral, D. Webster, G. S. Corrado, Y. Matias, S. Azizi, A. Karthikesalingam y V. Natarajan, *Towards Expert-Level Medical Question Answering with Large Language Models*, 2023. arXiv: 2305.09617 [cs.CL].

[156] Z. Ghahramani, *Presentación de PaLM 2*, https://blog.google/technology/ai/google-palm-2-ai-large-language-model/, mayo de 2023. (visitado 22-05-2023).

[157] OpenAI, *Improving Language Understanding with Unsupervised Learning*, https://openai.com/blog/language-unsupervised/, 2018. (visitado 22-05-2023).

[158] ——, *Better Language Models and Their Implications*, https://openai.com/blog/better-language-models/, 2019. (visitado 22-05-2023).

[159] A. G.-F. y Jordi Armengol-Estapé y Marc Pàmies y Joan Llop-Palao y Joaquin Silveira-Ocampo y Casimiro Pio Carrino y Carme Armentano-Oller y Carlos Rodriguez-Penagos y Aitor Gonzalez-Agirre y Marta Villegas, «MarIA: Spanish Language Models», *Procesamiento del Lenguaje Natural*, vol. 68, n.º 0, págs. 39-60, 2022, ISSN: 1989-7553. dirección: http://journal.sepln.org/sepln/ojs/ojs/index.php/pln/article/view/6405.

[160] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever y D. Amodei, «Language Models Are Few-Shot Learners», en *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ép. NIPS'20, Vancouver, BC, Canada: Curran Associates Inc., 2020, ISBN: 9781713829546.

[161] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever y W. Zaremba, «Evaluating Large Language Models Trained on Code», *CoRR*, vol. abs/2107.03374, 2021. DOI: 10.48550/ARXIV.2107.03374. arXiv: 2107.03374. dirección: https://arxiv.org/abs/2107.03374.

[162] GitHub, *Your AI pair programmer*, https://github.com/features/copilot/, 2021. (visitado 22-05-2023).

[163] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. de Masson d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Gowal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu y O. Vinyals, «Competition-level code generation with AlphaCode», *Science*, vol. 378, n.º 6624, págs. 1092-1097, 2022. DOI: 10.1126/science.abq1158. eprint: https://www.science.org/doi/pdf/10.1126/science.abq1158. dirección: https://www.science.org/doi/abs/10.1126/science.abq1158.

[164] OpenAI, *ChatGPT: Optimizing Language Models for Dialogue*, https://openai.com/blog/chatgpt/, 2022. (visitado 22-05-2023).

[165] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike y R. Lowe, *Training language models to follow instructions with human feedback*, 2022. DOI: 10.48550/ARXIV.2203.02155. dirección: https://arxiv.org/abs/2203.02155.

[166] OpenAI, *GPT-4*, https://openai.com/research/gpt-4, mar. de 2023. (visitado 22-05-2023).

[167] J. Spataro, *Introducing Microsoft 365 Copilot – your copilot for work*, https://blogs.microsoft.com/blog/2023/03/16/introducing-microsoft-365-copilot-your-copilot-for-work/, mar. de 2023. (visitado 22-05-2023).

[168] Y. Mehdi, *Reinventing search with a new AI-powered Microsoft Bing and Edge, your copilot for the web*, https://blogs.microsoft.com/blog/2023/02/07/reinventing-search-with-a-new-ai-powered-microsoft-bing-and-edge-your-copilot-for-the-web/, 2023. (visitado 22-05-2023).

[169] K. Roose, *Una conversación con el chatbot de Bing me dejó profundamente perturbado*, https://www.nytimes.com/es/2023/02/17/espanol/chatbot-bing-ia.html, 2023. (visitado 22-05-2023).

[170] P. Panay, *Bringing the power of AI to Windows 11 – unlocking a new era of productivity for customers and developers with Windows Copilot and Dev Home*, https://blogs.windows.com/windowsdeveloper/2023/05/23/bringing-the-power-of-ai-to-windows-11-unlocking-a-new-era-of-productivity-for-customers-and-developers-with-windows-copilot-and-dev-home/, mayo de 2023. (visitado 22-05-2023).

[171] O. Lieber, O. Sharir, B. Lenz e Y. Shoham, «Jurassic-1: Technical Details And Evaluation», AI21 Labs, inf. téc., ago. de 2021. eprint: https://uploads-ssl.webflow.com/60fd4503684b466578c0d307/61138924626a6981ee09caf6_jurassic_tech_paper.pdf. dirección: https://github.com/ai21labs/lm-evaluation.

[172] S. Smith, M. Patwary, B. Norick, P. LeGresley, S. Rajbhandari, J. Casper, Z. Liu, S. Prabhumoye, G. Zerveas, V. Korthikanti, E. Zhang, R. Child, R. Y. Aminabadi, J. Bernauer, X. Song, M. Shoeybi, Y. He, M. Houston, S. Tiwary y B. Catanzaro, «Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model», *CoRR*, vol. abs/2201.11990, 2022. DOI: 10.48550/ARXIV.2201.11990. arXiv: 2201.11990. dirección: https://arxiv.org/abs/2201.11990.

[173] B. Workshop, : T. L. Scao y col., *BLOOM: A 176B-Parameter Open-Access Multilingual Language Model*, 2022. DOI: 10.48550/ARXIV.2211.05100. dirección: https://arxiv.org/abs/2211.05100.

[174] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang y L. Zettlemoyer, *OPT: Open Pre-trained Transformer Language Models*, 2022. DOI: 10.48550/ARXIV.2205.01068. dirección: https://arxiv.org/abs/2205.01068.

[175] A. Rives, J. Meier, T. Sercu, S. Goyal, Z. Lin, J. Liu, D. Guo, M. Ott, C. L. Zitnick, J. Ma y R. Fergus, «Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences», *Proceedings of the National Academy of Sciences*, vol. 118, n.º 15, e2016239118, 2021. DOI: 10.1073/pnas.2016239118. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2016239118. dirección: https://www.pnas.org/doi/abs/10.1073/pnas.2016239118.

[176] Z. Lin, H. Akin, R. Rao, B. Hie, Z. Zhu, W. Lu, N. Smetanin, R. Verkuil, O. Kabeli, Y. Shmueli, A. dos Santos Costa, M. Fazel-Zarandi, T. Sercu, S. Candido y A. Rives, «Evolutionary-scale prediction of atomic level protein structure with a language model», *bioRxiv*, 2022. DOI: 10.1101/2022.07.20.500902. eprint: https://www.biorxiv.org/content/early/2022/12/21/2022.07.20.500902.full.pdf. dirección: https://www.biorxiv.org/content/early/2022/12/21/2022.07.20.500902.

[177] NLLB Team, M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard, A. Sun, S. Wang, G. Wenzek, A. Youngblood, B. Akula, L. Barrault, G. M. Gonzalez, P. Hansanti, J. Hoffman, S. Jarrett, K. R. Sadagopan, D. Rowe, S. Spruit, C. Tran, P. Andrews, N. F. Ayan, S. Bhosale, S. Edunov, A. Fan, C. Gao, V. Goswami, F. Guzmán, P. Koehn, A. Mourachko, C. Ropers, S. Saleem, H. Schwenk y J. Wang, *No Language Left Behind: Scaling Human-Centered Machine*

Translation, 2022. DOI: 10.48550/ARXIV.2207.04672. dirección: https://arxiv.org/abs/2207.04672.

[178] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey e I. Sutskever, *Robust Speech Recognition via Large-Scale Weak Supervision*, 2022. DOI: 10.48550/ARXIV.2212.04356. dirección: https://arxiv.org/abs/2212.04356.

[179] Google Cloud, *Chirp: Universal speech model*, https://cloud.google.com/speech-to-text/v2/docs/chirp-model. (visitado 22-05-2023).

[180] Y. Zhang y J. Qin, *Universal Speech Model (USM): State-of-the-art speech AI for 100+ languages*, https://ai.googleblog.com/2023/03/universal-speech-model-usm-state-of-art.html, mar. de 2023. (visitado 22-05-2023).

[181] J. Dani y S. Srinivasan, *Satin: Microsoft's latest AI-powered audio codec for real-time communications*, https://techcommunity.microsoft.com/t5/microsoft-teams-blog/satin-microsoft-s-latest-ai-powered-audio-codec-for-real-time/ba-p/2141382, feb. de 2021. (visitado 22-05-2023).

[182] H. Yeh, *Lyra V2 - a better, faster, and more versatile speech codec*, https://opensource.googleblog.com/2022/09/lyra-v2-a-better-faster-and-more-versatile-speech-codec.html, sep. de 2022. (visitado 22-05-2023).

[183] A. Défossez, J. Copet, G. Synnaeve e Y. Adi, *High Fidelity Neural Audio Compression*, 2022. arXiv: 2210.13438 [eess.AS].

[184] C. Wang, S. Chen, Y. Wu, Z. Zhang, L. Zhou, S. Liu, Z. Chen, Y. Liu, H. Wang, J. Li, L. He, S. Zhao y F. Wei, *Neural Codec Language Models are Zero-Shot Text to Speech Synthesizers*, 2023. DOI: 10.48550/ARXIV.2301.02111. dirección: https://arxiv.org/abs/2301.02111.

[185] M. Le, A. Vyas, B. Shi, B. Karrer, L. Sari, R. Moritz, M. Williamson, V. Manohar, Y. Adi, J. Mahadeokar y W.-N. Hsu, *Voicebox: Text-Guided Multilingual Universal Speech Generation at Scale*, jun. de 2023. dirección: https://research.facebook.com/publications/voicebox-text-guided-multilingual-universal-speech-generation-at-scale/ (visitado 22-05-2023).

[186] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior y K. Kavukcuoglu, *WaveNet: A Generative Model for Raw Audio*, 2016. DOI: 10.48550/ARXIV.1609.03499. dirección: https://arxiv.org/abs/1609.03499.

[187] A. Agostinelli, T. I. Denk, Z. Borsos, J. Engel, M. Verzetti, A. Caillon, Q. Huang, A. Jansen, A. Roberts, M. Tagliasacchi, M. Sharifi, N. Zeghidour y C. Frank, *MusicLM: Generating Music From Text*, 2023. DOI: 10.48550/ARXIV.2301.11325. dirección: https://arxiv.org/abs/2301.11325.

[188] Z. Borsos, R. Marinier, D. Vincent, E. Kharitonov, O. Pietquin, M. Sharifi, O. Teboul, D. Grangier, M. Tagliasacchi y N. Zeghidour, *AudioLM: a Language Modeling Approach to Audio Generation*, 2022. DOI: 10.48550/ARXIV.2209.03143. dirección: https://arxiv.org/abs/2209.03143.

[189] Q. Huang, A. Jansen, J. Lee, R. Ganti, J. Y. Li y D. P. W. Ellis, *MuLan: A Joint Embedding of Music Audio and Natural Language*, 2022. DOI: 10.48550/ARXIV.2208.12415. dirección: https://arxiv.org/abs/2208.12415.

[190] J. Copet, F. Kreuk, I. Gat, T. Remez, D. Kant, G. Synnaeve, Y. Adi y A. Défossez, *Simple and Controllable Music Generation*, 2023. arXiv: 2306.05284 [cs.SD].

[191] M. Team, *Mubert launches Text-to-Music interface – a completely new way to generate music from a single text prompt*, https://mubert.com/blog/mubert-launches-text-to-music-interface-a-completely-new-way-to-generate-music-from-a-single-text-prompt, 2022. (visitado 22-05-2023).

[192] Y. Shen, K. Song, X. Tan, D. Li, W. Lu e Y. Zhuang, *HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in HuggingFace*, 2023. arXiv: 2303.17580 [cs.CL].

[193] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, T. Eccles, J. Bruce, A. Razavi, A. Edwards, N. Heess, Y. Chen, R. Hadsell, O. Vinyals, M. Bordbar y N. de Freitas, «A Generalist Agent», 2022. DOI: 10.48550/ARXIV.2205.06175. dirección: https://arxiv.org/abs/2205.06175.

[194] R. Girdhar, A. El-Nouby, Z. Liu, M. Singh, K. V. Alwala, A. Joulin e I. Misra, *ImageBind: One Embedding Space To Bind Them All*, 2023. arXiv: 2305.05665 [cs.CV].

[195] S. J. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, ép. Always learning. Pearson, 2016, ISBN: 9781292153964. dirección: https://books.google.es/books?id=XS9CjwEACAAJ.

[196] Iberdrola, *¿Qué es la inteligencia artificial?*, https://www.iberdrola.com/innovacion/que-es-inteligencia-artificial, 2023. (visitado 22-05-2023).

[197] J. Mollo Suni, «Análisis de propiedades acústicas relacionadas a propiedades mecánicas de textura de galletas», Tesis doct., 2021. dirección: https://renati.sunedu.gob.pe/handle/sunedu/2823339.

[198] D. A. Pérez-Alonso, «Análisis de señales de tos para detección temprana de enfermedades respiratorias», spa, info:eu-repo/semantics/bachelorThesis, Tesis doct., sep. de 2019. dirección: http://uvadoc.uva.es/handle/10324/38797.

[199] G. de Acústica, *Características del sonido: intensidad, tono, timbre y duración*, https://www.ehu.eus/acustica/bachillerato/casoes/casoes.html, 2003. (visitado 22-05-2023).

[200] U. N. de Quilmes, *Sonido*, http://libros.uvq.edu.ar/spm/3_sonido.html, 2023. (visitado 22-05-2023).

[201] A. Alija, *Auto ML. Una aproximación diferente en la ciencia de datos*, https://datos.gob.es/es/blog/auto-ml-una-aproximacion-diferente-en-la-ciencia-de-datos, 2021. (visitado 22-05-2023).

[202] A. Balaji y A. Allen, *Choosing the best AutoML Framework*, https://medium.com/georgian-impact-blog/choosing-the-best-automl-framework-4f2a90cb1826, 2018. (visitado 22-05-2023).

[203] A. Allen y A. Balaji, *Automatic Machine Learning (AutoML) Landscape Survey*, https://medium.com/georgian-impact-blog/automatic-machine-learning-aml-landscape-survey-f75c3ae3bbf2, 2018. (visitado 22-05-2023).

[204] J. A. Sanchez, *¿Cómo aprenden las máquinas? Machine Learning y sus diferentes tipos*, https://datos.gob.es/es/blog/como-aprenden-las-maquinas-machine-learning-y-sus-diferentes-tipos, 2020. (visitado 22-05-2023).

[205] Jeevan, *Introduction to Machine Learning*, https://technovert.com/blog/introduction-to-machine-learning/, 2019. (visitado 22-05-2023).

[206] J. Barranquero, *Aprendizaje por refuerzo: soluciones IA que aprenden sin datos históricos*, https://datos.gob.es/es/blog/aprendizaje-por-refuerzo-soluciones-ia-que-aprenden-sin-datos-historicos, 2022. (visitado 22-05-2023).

[207] R. S. Sutton y A. G. Barto, *Reinforcement Learning, second edition: An Introduction*, ép. Adaptive Computation and Machine Learning series. MIT Press, 2018, ISBN: 9780262352703. dirección: https://books.google.es/books?id=uWV0DwAAQBAJ.

[208] J. F. Díaz Sepúlveda y J. C. Correa Morales, «Comparación entre árboles de regresión CART y regresión lineal.», *Comunicaciones En Estadística*, vol. 6, n.º 2, págs. 175-195, 2013. DOI: 10.15332/s2027-3355.2013.0002.05. dirección: https://revistas.usantotomas.edu.co/index.php/estadistica/article/view/1101.

[209] Microsoft, *Evaluación de los resultados del experimento de aprendizaje automático automatizado*, https://learn.microsoft.com/es-es/azure/machine-learning/how-to-understand-automated-ml, 2022. (visitado 22-05-2023).

[210] S. Abraham, C. Huynh y H. Vu, «Classification of Soils into Hydrologic Groups Using Machine Learning», *Data*, vol. 5, n.º 1, 2020, ISSN: 2306-5729. DOI: 10.3390/data5010002. dirección: https://www.mdpi.com/2306-5729/5/1/2.

[211] Lavender888000, *Confusion Matrix, Table of Confusion, Preventive Medicine*, https://commons.wikimedia.org/wiki/File:Preventive_Medicine_-_Statistics_Sensitivity_TPR,_Specificity_TNR,_PPV,_NPV,_FDR,_FOR,_ACCuracy,_Likelihood_Ratio,_Diagnostic_Odds_Ratio_2_Final_wiki.png?uselang=es, ago. de 2015. (visitado 22-05-2023).

[212] F. Salech, V. Mery, F. Larrondo y G. Rada, «Estudios que evalúan un test diagnóstico: interpretando sus resultados», es, *Revista médica de Chile*, vol. 136, págs. 1208-1208, sep. de 2008, ISSN: 0034-9887. DOI: 10.4067/S0034-98872008000900018. dirección: https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0034-98872008000900018&nrm=iso.

[213] J. Balayla, «On the formalism of the screening paradox», *PLOS ONE*, vol. 16, n.º 9, págs. 1-13, sep. de 2021. DOI: 10.1371/journal.pone.0256645. dirección: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0256645.

[214] ——, «Prevalence threshold (Φ_e) and the geometry of screening curves», *PLOS ONE*, vol. 15, n.º 10, págs. 1-12, oct. de 2020. DOI: 10.1371/journal.pone.0240215. dirección: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0240215.

[215] J. Cerda y L. Cifuentes, «Uso de curvas ROC en investigación clínica: Aspectos teórico-prácticos», es, *Revista chilena de infectología*, vol. 29, n.º 2, págs. 138-141, abr. de 2012, ISSN: 0716-1018. DOI: 10.4067/S0716-10182012000200003. dirección: https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0716-10182012000200003&nrm=iso.

[216] J. Ramírez, *Curvas PR y ROC*, https://medium.com/bluekiri/curvas-pr-y-roc-1489fbd9a527, 2018. (visitado 22-05-2023).

[217] J. G. Abad, «Calibración local de predicciones numéricas de viento con tecnicas estadísticas no lineales.(Downscaling Estadístico)», Tesis de mtría., oct. de 2012. dirección: http://hdl.handle.net/10902/1006.

[218] W. J. Youden, «Index for rating diagnostic tests», *Cancer*, vol. 3, n.º 1, págs. 32-35, 1950. DOI: 10.1002/1097-0142(1950)3:1<32::AID-CNCR2820030106>3.0.CO;2-3. eprint: https://acsjournals.onlinelibrary.wiley.com/doi/pdf/10.1002/1097-0142%281950%293%3A1%3C32%3A%3AAID-CNCR2820030106%3E3.0.CO%3B2-3. dirección: https://acsjournals.onlinelibrary.wiley.com/doi/abs/10.1002/1097-0142%281950%293%3A1%3C32%3A%3AAID-CNCR2820030106%3E3.0.CO%3B2-3.

[219] L. Pérez Planells, J. Delegido, J. P. Rivera-Caicedo y J. Verrelst, «Análisis de métodos de validación cruzada para la obtención robusta de parámetros biofísicos», *Revista Española de Teledetección*, vol. 44, págs. 55-65, 2015. eprint: https://core.ac.uk/download/pdf/71051261.pdf.

[220] V. Van Asch, «Macro-and micro-averaged evaluation measures [[basic draft]]», *Belgium: CLiPS*, págs. 1-27, sep. de 2013. eprint: https://pdfs.semanticscholar.org/1d10/6a2730801b6210a67f7622e4d192bb1pdf.

[221] scikit-learn developers, *Cross-validation: evaluating estimator performance*, https://scikit-learn.org/stable/modules/cross_validation.html, 2023. (visitado 22-05-2023).

[222] F. Herrera, C. Hervás, J. Otero y L. Sánchez, «Un estudio empírico preliminar sobre los tests estadísticos más habituales en el aprendizaje automático», *Tendencias de la Minería de Datos en España, Red Española de Minería de Datos y Aprendizaje (TIC2002-11124-E)*, págs. 403-412, 2004. eprint: https://www.lsi.us.es/redmidas/Capitulos/LMD35.pdf.

[223] J. Calvo Martin, *La importancia de las funciones de activación en una red neuronal*, https://www.linkedin.com/pulse/la-importancia-de-las-funciones-activaci%C3%B3n-en-una-red-calvo-martin/, 2022. (visitado 22-05-2023).

[224] J. Ponce, *Conoce qué son las funciones de activación y cómo puedes crear tu función de activación usando Python, R y Tensorflow*, https://jahazielponce.com/funciones-de-activacion-y-como-puedes-crear-la-tuya-usando-python-r-y-tensorflow/, 2021. (visitado 22-05-2023).

[225] J. Turian, J. Bergstra e Y. Bengio, «Quadratic Features and Deep Architectures for Chunking», en *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, ép. NAACL-Short '09, Boulder, Colorado: Association for Computational Linguistics, 2009, págs. 245-248. dirección: https://dl.acm.org/doi/abs/10.5555/1620853.1620921.

[226] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas y H. S. Seung, «Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit», *Nature*, vol. 405, n.º 6789, págs. 947-951, jun. de 2000, ISSN: 1476-4687. DOI: 10.1038/35016072. dirección: https://www.nature.com/articles/35016072.

[227] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto y H. Adam, *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*, 2017. DOI: 10.48550/ARXIV.1704.04861. dirección: https://arxiv.org/abs/1704.04861.

[228] A. L. Maas, A. Y. Hannun y A. Y. Ng, «Rectifier nonlinearities improve neural network acoustic models», en *In International Conference on Machine Learning*, vol. 30, 2013. eprint: http://robotics.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[229] K. He, X. Zhang, S. Ren y J. Sun, «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification», en *2015 IEEE International Conference on Computer Vision (ICCV)*, dic. de 2015, págs. 1026-1034. DOI: 10.1109/ICCV.2015.123.

[230] M. Angelini, *¡Google lo ha vuelto a conseguir!*, https://www.linkedin.com/pulse/google-lo-ha-vuelto-conseguir-massimo-angelini/?originalSubdomain=es, 2018. (visitado 22-05-2023).

[231] D.-A. Clevert, T. Unterthiner y S. Hochreiter, *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*, 2015. DOI: 10.48550/ARXIV.1511.07289. dirección: https://arxiv.org/abs/1511.07289.

[232] G. Klambauer, T. Unterthiner, A. Mayr y S. Hochreiter, «Self-Normalizing Neural Networks», en *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ép. NIPS'17, Long Beach, California, USA: Curran Associates Inc., 2017, págs. 972-981, ISBN: 9781510860964. dirección: https://dl.acm.org/doi/abs/10.5555/3294771.3294864.

[233] V. Nair y G. E. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines», en *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ép. ICML'10, Haifa, Israel: Omnipress, 2010, págs. 807-814, ISBN: 9781605589077. eprint: https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf.

[234] Hemingway, *Aprendiendo Inteligencia Artificial: ¿Qué es la función de activación?*, https://forum.huawei.com/enterprise/es/aprendiendo-inteligencia-artificial-%C2%BFqu%C3%A9-es-la-funci%C3%B3n-de-activaci%C3%B3n/thread/745533-100757, 2021. (visitado 22-05-2023).

[235] P. Ramachandran, B. Zoph y Q. V. Le, *Searching for Activation Functions*, 2017. DOI: 10.48550/ARXIV.1710.05941. dirección: https://arxiv.org/abs/1710.05941.

[236] S. Elfwing, E. Uchibe y K. Doya, «Sigmoid-weighted linear units for neural network function approximation in reinforcement learning», *Neural Networks*, vol. 107, págs. 3-11, 2018, Special issue on deep reinforcement learning, ISSN: 0893-6080. DOI: 10.1016/j.neunet.2017.12.012. dirección: https://www.sciencedirect.com/science/article/pii/S0893608017302976.

[237] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam y Q. Le, «Searching for MobileNetV3», en *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, oct. de 2019, págs. 1314-1324. DOI: 10.1109/ICCV.2019.00140.

[238] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting», *Journal of Machine Learning Research*, vol. 15, n.º 56, págs. 1929-1958, 2014. dirección: http://jmlr.org/papers/v15/srivastava14a.html.

[239] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. Courville y C. Pal, *Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations*, 2016. DOI: 10.48550/ARXIV.1606.01305. dirección: https://arxiv.org/abs/1606.01305.

[240] D. Hendrycks y K. Gimpel, *Gaussian Error Linear Units (GELUs)*, 2016. DOI: 10.48550/ARXIV.1606.08415. dirección: https://arxiv.org/abs/1606.08415.

[241] D. Misra, *Mish: A Self Regularized Non-Monotonic Activation Function*, 2019. DOI: 10.48550/ARXIV.1908.08681. dirección: https://arxiv.org/abs/1908.08681.

[242] Y. LeCun, Y. Bengio y G. Hinton, «Deep learning», *Nature*, vol. 521, n.º 7553, págs. 436-444, mayo de 2015, ISSN: 1476-4687. DOI: 10.1038/nature14539. dirección: https://www.nature.com/articles/nature14539.

[243] I. Sutskever, J. Martens, G. Dahl y G. Hinton, «On the importance of initialization and momentum in deep learning», en *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta y D. McAllester, eds., ép. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, jun. de 2013, págs. 1139-1147. dirección: https://proceedings.mlr.press/v28/sutskever13.html.

[244] B. Polyak, «Some methods of speeding up the convergence of iteration methods», *USSR Computational Mathematics and Mathematical Physics*, vol. 4, n.º 5, págs. 1-17, 1964, ISSN: 0041-5553. DOI: 10.1016/0041-5553(64)90137-5. dirección: https://www.sciencedirect.com/science/article/pii/0041555364901375.

[245] Y. E. Nesterov, «A method of solving a convex programming problem with convergence rate $O\left(\frac{1}{k^2}\right)$», *Dokl. Akad. Nauk SSSR*, vol. 269, págs. 543-547, 3 1983. dirección: https://www.mathnet.ru/eng/dan46009.

[246] J. Duchi, E. Hazan e Y. Singer, «Adaptive Subgradient Methods for Online Learning and Stochastic Optimization», *Journal of Machine Learning Research*, vol. 12, n.º 61, págs. 2121-2159, 2011. dirección: http://jmlr.org/papers/v12/duchi11a.html.

[247] M. D. Zeiler, *ADADELTA: An Adaptive Learning Rate Method*, 2012. DOI: 10.48550/ARXIV.1212.5701. dirección: https://arxiv.org/abs/1212.5701.

[248] T. Tieleman y G. Hinton, «Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude.», *COURSERA: Neural networks for machine learning*, págs. 26-31, 2012. eprint: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[249] D. P. Kingma y J. Ba, «Adam: A method for stochastic optimization», *arXiv preprint arXiv:1412.6980*, 2014. DOI: 10.48550/ARXIV.1412.6980. eprint: https://arxiv.org/pdf/1412.6980.pdf. dirección: https://arxiv.org/abs/1412.6980.

[250] T. Dozat, «Incorporating nesterov momentum into adam», 2016. eprint: https://cs229.stanford.edu/proj2015/054_report.pdf. dirección: https://openreview.net/forum?id=OM0jvwB8jIp57ZJjtNEZ.

[251] A. Anwar, *Cheat Sheets for Machine Learning Interview Topics*, https://medium.com/swlh/cheat-sheets-for-machine-learning-interview-topics-51c2bc2bab4f, oct. de 2020. (visitado 22-05-2023).

[252] J. Ahmad, K. Muhammad y S. W. Baik, «Data augmentation-assisted deep learning of hand-drawn partially colored sketches for visual search», *PLOS ONE*, vol. 12, n.º 8, págs. 1-19, ago. de 2017. DOI: 10.1371/journal.pone.0183838. dirección: https://doi.org/10.1371/journal.pone.0183838.

[253] S. Ioffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift», en *Proceedings of the 32nd International Conference on Machine Learning*, F. Bach y D. Blei, eds., ép. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, jul. de 2015, págs. 448-456. dirección: https://proceedings.mlr.press/v37/ioffe15.html.

[254] A. Tang, R. Tam, A. Cadrin-Chênevert, W. Guest, J. Chong, J. Barfett, L. Chepelev, R. Cairns, J. R. Mitchell, M. D. Cicero, M. G. Poudrette, J. L. Jaremko, C. Reinhold, B. Gallix, B. Gray, R. Geis, for the Canadian Association of Radiologists (CAR) Artificial Intelligence Working Group, T. O'Connell, P. Babyn, D. Koff, D. Ferguson, S. Derkatch, A. Bilbily y W. Shabana, «Canadian Association of Radiologists White Paper on Artificial Intelligence in Radiology», *Canadian Association of Radiologists Journal*, vol. 69, n.º 2, págs. 120-135, 2018, PMID: 29655580. DOI: 10.1016/j.carj.2018.02.002. eprint: https://journals.sagepub.com/doi/epdf/10.1016/j.carj.2018.02.002. dirección: https://journals.sagepub.com/doi/10.1016/j.carj.2018.02.002.

[255] F. Rosenblatt, *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957. eprint: https://blogs.umass.edu/brain-wars/files/2016/03/rosenblatt-1957.pdf.

[256] W. Krauth y M. Mezard, «Learning algorithms with optimal stability in neural networks», *Journal of Physics A: Mathematical and General*, vol. 20, n.º 11, pág. L745, ago. de 1987. DOI: 10.1088/0305-4470/20/11/013. dirección: https://iopscience.iop.org/article/10.1088/0305-4470/20/11/013.

[257] C. Goller y A. Kuchler, «Learning task-dependent distributed representations by backpropagation through structure», en *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1, jun. de 1996, 347-352 vol.1. DOI: 10.1109/ICNN.1996.548916.

[258] A. online course, *What is Backpropagation Through Structure*, https://www.aionlinecourse.com/ai-basics/backpropagation-through-structure. (visitado 22-05-2023).

[259] M. C. Mozer, «A Focused Backpropagation Algorithm for Temporal Pattern Recognition», *Complex Syst.*, vol. 3, 1989. dirección: https://www.complex-systems.com/abstracts/v03_i04_a04/.

[260] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk e Y. Bengio, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014. arXiv: 1406.1078 [cs.CL].

[261] M. Schuster y K. Paliwal, «Bidirectional recurrent neural networks», *IEEE Transactions on Signal Processing*, vol. 45, n.º 11, págs. 2673-2681, nov. de 1997, ISSN: 1941-0476. DOI: 10.1109/78.650093.

[262] H. Wang y B. Raj, *On the Origin of Deep Learning*, 2017. arXiv: 1702.07800 [cs.LG].

[263] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell y S. Xie, *A ConvNet for the 2020s*, 2022. arXiv: 2201.03545 [cs.CV].

[264] J. P. C. Chiu y E. Nichols, *Named Entity Recognition with Bidirectional LSTM-CNNs*, 2016. arXiv: 1511.08308 [cs.CL].

[265] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit y N. Houlsby, «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale», en *International Conference on Learning Representations*, 2021. dirección: https://openreview.net/forum?id=YicbFdNTTy.

[266] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin y B. Guo, «Swin Transformer: Hierarchical Vision Transformer using Shifted Windows», en *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, oct. de 2021, págs. 9992-10 002. DOI: 10.1109/ICCV48922.2021.00986.

[267] M. Scholz, M. Fraunholz y J. Selbig, «Nonlinear Principal Component Analysis: Neural Network Models and Applications», en *Principal Manifolds for Data Visualization and Dimension Reduction*, A. N. Gorban, B. Kégl, D. C. Wunsch y A. Y. Zinovyev, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, págs. 44-67, ISBN: 978-3-540-73750-6. DOI: 10.1007/978-3-540-73750-6_2.

[268] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville e Y. Bengio, *Generative Adversarial Networks*, 2014. arXiv: 1406.2661 [stat.ML].

[269] A. M. De Luca, M. E. Irigoitia, G. A. Pérez y C. F. Pons, «Uso de la Técnica de Transfer Learning en Machine Learning para la Clasificación de Productos en el Banco Alimentario de La Plata», en *IX Congreso Nacional de Ingeniería Informática/Sistemas de Información (CoNaIISI)(Mendoza, 4 y 5 de noviembre de 2021)*, 2021. dirección: http://sedici.unlp.edu.ar/handle/10915/130474.

[270] *Keras Applications*, https://keras.io/api/applications/. (visitado 22-05-2023).

[271] *VGGish: A VGG-like audio classification model*, https://pypi.org/project/vggish-keras/. (visitado 22-05-2023).

[272] E. Vizel, M. Yigla, Y. Goryachev, E. Dekel, V. Felis, H. Levi, I. Kroin, S. Godfrey y N. Gavriely, «Validation of an ambulatory cough detection and counting application using voluntary cough under different conditions», *Cough*, vol. 6, n.º 1, pág. 3, mayo de 2010, ISSN: 1745-9974. DOI: 10.1186/1745-9974-6-3. eprint: https://coughjournal.biomedcentral.com/track/pdf/10.1186/1745-9974-6-3.pdf. dirección: https://coughjournal.biomedcentral.com/articles/10.1186/1745-9974-6-3#citeas.

[273] T. Drugman, J. Urbain, N. Bauwens, R. Chessini, C. Valderrama, P. Lebecque y T. Dutoit, «Objective Study of Sensor Relevance for Automatic Cough Detection», *IEEE Journal of Biomedical and Health Informatics*, vol. 17, n.º 3, págs. 699-707, mayo de 2013, ISSN: 2168-2208. DOI: 10.1109/JBHI.2013.2239303.

[274] Y. A. Amrulloh, U. R. Abeyratne, V. Swarnkar, R. Triasih y A. Setyati, «Automatic cough segmentation from non-contact sound recordings in pediatric wards», *Biomedical Signal Processing and Control*, vol. 21, págs. 126-136, ago. de 2015, ISSN: 1746-8094. DOI: 0.1016/j.bspc.2015.05.001. dirección: http://www.sciencedirect.com/science/article/pii/S1746809415000804.

[275] K. Tokuda, T. Kobayashi, T. Masuko y S. Imai, «Mel-generalized cepstral analysis - a unified approach to speech spectral estimation», en *Proc. 3rd International Conference on Spoken Language Processing (ICSLP 1994)*, 1994, págs. 1043-1046. DOI: 10.21437/ICSLP.1994-275. eprint: https://www.isca-speech.org/archive/pdfs/icslp_1994/tokuda94_icslp.pdf.

[276] J.-M. Liu, M. You, G.-Z. Li, Z. Wang, X. Xu, Z. Qiu, W. Xie, C. An y S. Chen, «Cough signal recognition with Gammatone Cepstral Coefficients», en *2013 IEEE China Summit and International Conference on Signal and Information Processing*, jul. de 2013, págs. 160-164. DOI: 10.1109/ChinaSIP.2013.6625319.

[277] S. Larson, G. Comina, R. H. Gilman, B. H. Tracey, M. Bravard y J. W. López, «Validation of an Automated Cough Detection Algorithm for Tracking Recovery of Pulmonary Tuberculosis Patients», *PLOS ONE*, vol. 7, n.º 10, págs. 1-10, oct. de 2012. DOI: 10.1371/journal.pone.0046229. dirección: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0046229.

[278] S. Matos, S. S. Birring, I. D. Pavord y H. Evans, «Detection of cough signals in continuous audio recordings using hidden Markov models», *IEEE Transactions on Biomedical Engineering*, vol. 53, n.º 6, págs. 1078-1083, jun. de 2006, ISSN: 0018-9294. DOI: 10.1109/TBME.2006.873548. dirección: https://ieeexplore.ieee.org/document/1634502.

[279] K. Yatani y K. N. Truong, «BodyScope: a wearable acoustic sensor for activity recognition», en *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, ép. UbiComp '12, ACM, Pittsburgh, Pennsylvania: ACM, 2012, págs. 341-350, ISBN: 978-1-4503-1224-0. DOI: 10.1145/2370216.2370269. dirección: http://doi.acm.org/10.1145/2370216.2370269.

[280] T. Drugman, J. Urbain, N. Bauwens, R. Chessini, A.-s. Aubriot, P. Lebecque y T. Dutoit, «Audio and Contact Microphones for Cough Detection», en *Conference of the International Speech Communication Association (INTERSPEECH)*, 2012. dirección: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.500.805.

[281] T. Drugman, «Using mutual information in supervised temporal event detection: Application to cough detection», *Biomedical Signal Processing and Control*, vol. 10, págs. 50-57, 2014, ISSN: 1746-8094. DOI: 10.1016/j.bspc.2014.01.001. dirección: http://www.sciencedirect.com/science/article/pii/S1746809414000020.

[282] V. Swarnkar, U. R. Abeyratne, A. B. Chang, Y. A. Amrulloh, A. Setyati y R. Triasih, «Automatic Identification of Wet and Dry Cough in Pediatric Patients with Respiratory Diseases», *Annals of Biomedical Engineering*, vol. 41, n.º 5, págs. 1016-1028, mayo de 2013, ISSN: 1573-9686. DOI: 10.1007/s10439-013-0741-6. dirección: https://link.springer.com/article/10.1007/s10439-013-0741-6.

[283] U. R. Abeyratne, A. Swarnkar Vinayak and Setyati y R. Triasih, «Cough Sound Analysis Can Rapidly Diagnose Childhood Pneumonia», *Annals of Biomedical Engineering*, vol. 41, n.º 11, págs. 2448-2462, nov. de 2013, ISSN: 1573-9686. DOI: 10.1007/s10439-013-0836-0.

[284] K. Kosasih, U. R. Abeyratne, V. Swarnkar y R. Triasih, «Wavelet Augmented Cough Analysis for Rapid Childhood Pneumonia Diagnosis», *IEEE Transactions on Biomedical Engineering*, vol. 62, n.º 4, págs. 1185-1194, abr. de 2015, ISSN: 1558-2531. DOI: 10.1109/TBME.2014.2381214.

[285] *ResApp Health*, https://www.resapphealth.com.au/, 2022. (visitado 22-05-2023).

[286] E. Agu, P. Pedersen, D. Strong, B. Tulu, Q. He, L. Wang e Y. Li, «The smartphone as a medical device: Assessing enablers, benefits and challenges», en *2013 IEEE International Conference on Sensing, Communications and Networking (SECON)*, jun. de 2013, págs. 76-80. DOI: 10.1109/SAHCN.2013.6644964.

[287] J.-M. Liu, M. You, Z. Wang, G.-Z. Li, X. Xu y Z. Qiu, «Cough detection using deep neural networks», en *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, nov. de 2014, págs. 560-563. DOI: 10.1109/BIBM.2014.6999220.

[288] J. Amoh y K. Odame, «Deep Neural Networks for Identifying Cough Sounds», *IEEE Transactions on Biomedical Circuits and Systems*, vol. 10, n.º 5, págs. 1003-1011, oct. de 2016, ISSN: 1940-9990. DOI: 10.1109/TBCAS.2016.2598794. dirección: https://ieeexplore.ieee.org/abstract/document/7570164.

[289] M. You, W. Wang, Y. Li, J. Liu, X. Xu y Z. Qiu, «Automatic cough detection from realistic audio recordings using C-BiLSTM with boundary regression», *Biomedical Signal Processing and Control*, vol. 72, pág. 103 304, 2022, ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2021.103304. dirección: https://www.sciencedirect.com/science/article/pii/S1746809421009010.

[290] A. Tena, F. Clarià y F. Solsona, «Automated detection of COVID-19 cough», *Biomedical Signal Processing and Control*, vol. 71, pág. 103 175, ene. de 2022, ISSN: 1746-8094. DOI: 10.1016/j.bspc.2021.103175. dirección: https://www.sciencedirect.com/science/article/pii/S1746809421007722.

[291] J. Laguarta, F. Hueto y B. Subirana, «COVID-19 Artificial Intelligence Diagnosis Using Only Cough Recordings», *IEEE Open Journal of Engineering in Medicine and Biology*, vol. 1, págs. 275-281, sep. de 2020, ISSN: 2644-1276. DOI: 10.1109/OJEMB.2020.3026928. dirección: https://ieeexplore.ieee.org/document/9208795.

[292] U. of Cambridge, *COVID-19 Sounds App*, https://www.covid-19-sounds.org/en/. (visitado 22-05-2023).

[293] I. I. of Science (IISc) Bangalore, *Project Coswara*, https://coswara.iisc.ac.in/. (visitado 22-05-2023).

[294] A. Khanzada, C. Chaurasia, N. Perez y L. Chionis, *Virufy*, https://virufy.org/. (visitado 22-05-2023).

[295] *YAMNet.* https://github.com/tensorflow/models/tree/master/research/audioset/yamnet, 2022. (visitado 22-05-2023).

[296] C. Hoyos-Barceló, J. R. Garmendia-Leiza, M. D. Aguilar-García, J. Monge-Álvarez, D. A. Pérez-Alonso, C. Alberola-López y P. Casaseca-de-la-Higuera, «Evaluation in a Real Environment of a Trainable Cough Monitoring App for Smartphones», en *XV Mediterranean Conference on Medical and Biological Engineering and Computing – MEDICON 2019*, J. Henriques, N. Neves y P. de Carvalho, eds., Cham: Springer International Publishing, 2020, págs. 1175-1180, ISBN: 978-3-030-31635-8. DOI: 10.1007/978-3-030-31635-8_142. dirección: https://link.springer.com/chapter/10.1007/978-3-030-31635-8_142.

[297] *The SmartCough project*, https://smartcough.wordpress.com/, 2017. (visitado 22-05-2023).

[298] *Smartcough*, https://www.dhi-scotland.com/projects/smartcough/, 2020. (visitado 22-05-2023).

[299] *SMARTCOUGH: Continuous intelligent cough detection and identification using smartphones.* http://www.lpi.tel.uva.es/node/578, 2016. (visitado 22-05-2023).

[300] C. Hoyos-Barceló, J. Monge-Álvarez, Z. Pervez, L. M. San-José-Revuelta y P. Casaseca-de-la-Higuera, «Efficient computation of image moments for robust cough detection using smartphones», *Computers in Biology and Medicine*, vol. 100, págs. 176-185, jul. de 2018, ISSN: 0010-4825. DOI: h10.1016/j.compbiomed.2018.07.003. dirección: https://www.sciencedirect.com/science/article/pii/S0010482518301860.

[301] C. Hoyos-Barceló, J. Monge-Álvarez, M. Zeeshan Shakir, J.-M. Alcaraz-Calero y P. Casaseca-de-la-Higuera, «Efficient k-NN Implementation for Real-Time Detection of Cough Events in Smartphones», *IEEE Journal of Biomedical and Health Informatics*, vol. 22, n.º 5, págs. 1662-1671, sep. de 2018, ISSN: 2168-2208. DOI: 10.1109/JBHI.2017.2768162.

[302] J. Monge-Álvarez, C. Hoyos-Barceló, K. Dahal y P. Casaseca-de-la-Higuera, «Audio-cough event detection based on moment theory», *Applied Acoustics*, vol. 135, págs. 124-135, feb. de 2018, ISSN: 0003-682X. DOI: 10.1016/j.apacoust.2018.02.001. dirección: https://www.sciencedirect.com/science/article/pii/S0003682X1730720X.

[303] L. Orlandic, T. Teijeiro y D. Atienza, *The COUGHVID crowdsourcing dataset: A corpus for the study of large-scale cough analysis algorithms*, ver. 3.0, For more information about the data collection, preprocessing, validation, and data structure, please refer to the following publication: https://www.nature.com/articles/s41597-021-00937-4 The cough pre-processing and feature extraction code is available from the following c4science repository: https://c4science.ch/diffusion/10770/, feb. de 2021. DOI: 10.5281/zenodo.7024894. dirección: https://zenodo.org/record/7024894#.Y0AnD3ZBxPZ.

[304] L. Orlandic, T. Teijeiro y D. Atienza, «The COUGHVID crowdsourcing dataset: A corpus for the study of large-scale cough analysis algorithms», *CoRR*, vol. abs/2009.11644, 2020. DOI: 10.48550/arXiv.2009.11644. dirección: https://arxiv.org/abs/2009.11644.

[305] R. X. A. Pramono, S. A. Imtiaz y E. Rodriguez-Villegas, «A Cough-Based Algorithm for Automatic Diagnosis of Pertussis», *PLOS ONE*, vol. 11, n.º 9, págs. 1-20, sep. de 2016. DOI: 10.1371/journal.pone.0162128. dirección: https://doi.org/10.1371/journal.pone.0162128.

[306] H. Chatrzarrin, A. Arcelus, R. Goubran y F. Knoefel, «Feature extraction for the differentiation of dry and wet cough sounds», en *2011 IEEE International Symposium on Medical Measurements and Applications*, mayo de 2011, págs. 162-166. DOI: 10.1109/MeMeA.2011.5966670.

[307] G. Sharma, K. Umapathy y S. Krishnan, «Trends in audio signal feature extraction methods», *Applied Acoustics*, vol. 158, pág. 107 020, 2020, ISSN: 0003-682X. DOI: 10.1016/j.apacoust.2019.107020. dirección: https://www.sciencedirect.com/science/article/pii/S0003682X19308795.

[308] X. developers, *Xgboost documentation*, https://xgboost.readthedocs.io/en/latest, 2020. (visitado 22-05-2023).

[309] G. C. Cawley y N. L. Talbot, «On Over-Fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation», *J. Mach. Learn. Res.*, vol. 11, págs. 2079-2107, oct. de 2010, ISSN: 1532-4435. eprint: https://dl.acm.org/doi/pdf/10.5555/1756006.1859921. dirección: https://dl.acm.org/doi/abs/10.5555/1756006.1859921.

[310] P. Refaeilzadeh, L. Tang y H. Liu, «Cross-Validation», en *Encyclopedia of Database Systems*, L. LIU y M. T. ÖZSU, eds. Boston, MA: Springer US, 2009, págs. 532-538, ISBN: 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565. dirección: https://link.springer.com/referenceworkentry/10.1007/978-0-387-39940-9_565.

[311] J. Bergstra, R. Bardenet, Y. Bengio y B. Kégl, «Algorithms for Hyper-Parameter Optimization», en *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ép. NIPS'11, Granada, Spain: Curran Associates Inc., 2011, págs. 2546-2554, ISBN: 9781618395993. eprint: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf. dirección: https://dl.acm.org/doi/abs/10.5555/2986459.2986743.

[312] *ShuffleSplit scikit-learn 0.24.1 documentation*, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ShuffleSplit.html, 2021. (visitado 22-05-2023).

[313] *The Corp Dataset*, https://mari.tongji.edu.cn/info/1012/1030.htm, 2022. (visitado 22-05-2023).

[314] J. S. Bridle y M. D. Brown, «An experimental automatic word recognition system», *JSRU report*, vol. 1003, n.º 5, pág. 33, 1974.

[315] P. Mermelstein, «Distance Measures for Speech Recognition–Psychological and Instrumental», *PUB DATE 76 NOTE 172p. EDRS PRICE DESCRIPTORS*, vol. 101, n.º 134, pág. 93, 1976. eprint: https://files.eric.ed.gov/fulltext/ED128870.pdf#page=93.

[316] S. Davis y P. Mermelstein, «Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences», *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 28, n.º 4, págs. 357-366, oct. de 1980, ISSN: 0096-3518. DOI: 10.1109/TASSP.1980.1163420.

[317] C.-H. I. Shih, T. Kowatsch, P. Tinschert, F. Barata y M. K. Nißen, «Towards The Design of a Smartphone-Based Biofeedback Breathing Training: Identifying Diaphragmatic Breathing Patterns from a Smartphones'Microphone», en *Mediterranean Conference on Information Systems (MCIS) 2016 Proceedings*, Association for Information Systems (AIS) Electronic Library (AISeL), sep. de 2016. dirección: https://aisel.aisnet.org/mcis2016/47.

[318] P. Stoica y R. L. Moses, *Spectral analysis of signals*. Pearson Prentice Hall, 2005, ISBN: 9780131139565. dirección: https://books.google.es/books?id=h78ZAQAAIAAJ.

[319] J. Korpáš, J. Sadloňová y M. Vrabec, «Analysis of the Cough Sound: an Overview», *Pulmonary Pharmacology*, vol. 9, n.º 5, págs. 261-268, 1996, ISSN: 0952-0600. DOI: 10.1006/pulp.1996.0034. dirección: https://www.sciencedirect.com/science/article/pii/S0952060096900344.

[320] J. W. Cooley y J. W. Tukey, «An Algorithm for the Machine Calculation of Complex Fourier Series», *Mathematics of Computation*, vol. 19, n.º 90, págs. 297-301, 1965, ISSN: 00255718, 10886842. DOI: 10.2307/2003354. dirección: http://www.jstor.org/stable/2003354 (visitado 22-05-2023).

[321] K. Simonyan y A. Zisserman, «Very deep convolutional networks for large-scale image recognition», *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. dirección: https://arxiv.org/abs/1409.1556.

[322] F. Chollet, «Xception: Deep Learning with Depthwise Separable Convolutions», en *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jul. de 2017, págs. 1800-1807. DOI: 10.1109/CVPR.2017.195.

[323] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva y A. Torralba, «Learning Deep Features for Discriminative Localization», en *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, jun. de 2016, págs. 2921-2929. DOI: 10.1109/CVPR.2016.319.

[324] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever y R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, 2012. DOI: 10.48550/ARXIV.1207.0580. dirección: https://arxiv.org/abs/1207.0580.

[325] R. Pascanu, T. Mikolov e Y. Bengio, «On the Difficulty of Training Recurrent Neural Networks», en *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ép. ICML'13, Atlanta, GA, USA: JMLR.org, 2013, III–1310–III-1318. dirección: https://dl.acm.org/doi/abs/10.5555/3042817.3043083.

[326] A. Neelakantan, L. Vilnis, Q. V. Le, I. Sutskever, L. Kaiser, K. Kurach y J. Martens, *Adding Gradient Noise Improves Learning for Very Deep Networks*, 2015. DOI: 10.48550/ARXIV.1511.06807. dirección: https://arxiv.org/abs/1511.06807.

[327] B. Nowok, G. M. Raab y C. Dibben, «synthpop: Bespoke Creation of Synthetic Data in R», *Journal of Statistical Software*, vol. 74, n.º 11, págs. 1-26, 2016. DOI: 10.18637/jss.v074.i11. dirección: https://www.jstatsoft.org/index.php/jss/article/view/v074i11.

[328] *Alberto Tena. COVID-19 Models and Data repository.* https://github.com/atenad/. (visitado 22-05-2023).

[329] K. Gröchenig, «Time-Frequency Analysis and the Uncertainty Principle», en *Foundations of Time-Frequency Analysis*. Boston, MA: Birkhäuser Boston, 2001, págs. 21-36, ISBN: 978-1-4612-0003-1. DOI: 10.1007/978-1-4612-0003-1_3. dirección: https://link.springer.com/chapter/10.1007/978-1-4612-0003-1_3.

[330] J. von Hann y R. Ward, *Handbook of Climatology: Part 1, General Climatology*. Macmillan, 1903. dirección: https://books.google.es/books?id=yL8OAAAAYAAJ.

[331] J. Cooley, P. Lewis y P. Welch, «The finite Fourier transform», *IEEE Transactions on Audio and Electroacoustics*, vol. 17, n.º 2, págs. 77-85, jun. de 1969, ISSN: 1558-2582. DOI: 10.1109/TAU.1969.1162036.

[332] S. S. Stevens, J. Volkmann y E. B. Newman, «A Scale for the Measurement of the Psychological Magnitude Pitch», *The Journal of the Acoustical Society of America*, vol. 8, n.º 3, págs. 185-190, 1937. DOI: 10.1121/1.1915893. eprint: https://asa.scitation.org/doi/pdf/10.1121/1.1915893. dirección: https://asa.scitation.org/doi/10.1121/1.1915893.

[333] S. S. Stevens y J. Volkmann, «The Relation of Pitch to Frequency: A Revised Scale», *The American Journal of Psychology*, vol. 53, n.º 3, págs. 329-353, 1940, ISSN: 00029556. DOI: 10.2307/1417526. eprint: https://www.jstor.org/stable/pdf/1417526.pdf. dirección: http://www.jstor.org/stable/1417526 (visitado 22-05-2023).

[334] *Matlab. Audio Toolbox.* https://es.mathworks.com/help/audio/ref/detectspeech.html. (visitado 22-05-2023).

[335] E. Wigner, «On the Quantum Correction For Thermodynamic Equilibrium», *Phys. Rev.*, vol. 40, págs. 749-759, 5 jun. de 1932. DOI: 10.1103/PhysRev.40.749. dirección: https://link.aps.org/doi/10.1103/PhysRev.40.749.

[336] H.-I. Choi y W. Williams, «Improved time-frequency representation of multicomponent signals using exponential kernels», *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, n.º 6, págs. 862-871, jun. de 1989, ISSN: 0096-3518. DOI: 10.1109/ASSP.1989.28057.

[337] I. Guyon, J. Weston y V. Barnhill Stephen and Vapnik, «Gene Selection for Cancer Classification using Support Vector Machines», *Machine Learning*, vol. 46, n.º 1, págs. 389-422, ene. de 2002, ISSN: 1573-0565. DOI: 10.1023/A:1012487302797. dirección: https://link.springer.com/article/10.1023/A:1012487302797.

[338] P. Baldi y K. Hornik, «Neural networks and principal component analysis: Learning from examples without local minima», *Neural Networks*, vol. 2, n.º 1, págs. 53-58, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90014-2. dirección: https://www.sciencedirect.com/science/article/pii/0893608089900142.

[339] V. Despotovic, M. Ismael, M. Cornil, R. M. Call y G. Fagherazzi, «Detection of COVID-19 from voice, cough and breathing patterns: Dataset and preliminary results», *Computers in Biology and Medicine*, vol. 138, pág. 104 944, nov. de 2021, ISSN: 0010-4825. DOI: 10.1016/j.compbiomed.2021.104944. dirección: https://www.sciencedirect.com/science/article/pii/S0010482521007381.

[340] S. Taylor, M. Wakem, G. Dijkman, M. Alsarraj y M. Nguyen, «A practical approach to RT-qPCR—Publishing data that conform to the MIQE guidelines», *Methods*, vol. 50, n.º 4, S1-S5, 2010, The ongoing Evolution of qPCR, ISSN: 1046-2023. DOI: 10.1016/j.ymeth.2010.01.005. dirección: https://www.sciencedirect.com/science/article/pii/S1046202310000204.

[341] C. for Disease Control, Prevention y col., *Guidance for antigen testing for SARS-CoV-2 for healthcare providers testing individuals in the community*, https://www.cdc.gov/coronavirus/2019-ncov/lab/resources/antigen-tests-guidelines.html, 2022. (visitado 22-05-2023).

[342] *Audacity® software is copyright © 1999-2021 Audacity Team*, https://audacityteam.org/, It is free software distributed under the terms of the GNU General Public License. The name Audacity® is a registered trademark. (visitado 22-05-2023).

[343] F. Eyben, K. R. Scherer, B. W. Schuller, J. Sundberg, E. André, C. Busso, L. Y. Devillers, J. Epps, P. Laukka, S. S. Narayanan y K. P. Truong, «The Geneva Minimalistic Acoustic Parameter Set (GeMAPS) for Voice Research and Affective Computing», *IEEE Transactions on Affective Computing*, vol. 7, n.º 2, págs. 190-202, abr. de 2016, ISSN: 1949-3045. DOI: 10.1109/TAFFC.2015.2457417. dirección: https://ieeexplore.ieee.org/document/7160715.

[344] B. W. Schuller, A. Batliner, C. Bergler, C. Mascolo, J. Han, I. Lefter, H. Kaya, S. Amiriparian, A. Baird, L. Stappen, S. Ottl, M. Gerczuk, P. Tzirakis, C. Brown, J. Chauhan, A. Grammenos, A. Hasthanasombat, D. Spathis, T. Xia, P. Cicuta, L. J. M. Rothkrantz, J. Zwerts, J. Treep y C. Kaandorp, *The INTERSPEECH 2021 Computational Paralinguistics Challenge: COVID-19 Cough, COVID-19 Speech, Escalation & Primates*, 2021. arXiv: 2102.13468 [eess.AS].

[345] F. Eyben, M. Wöllmer y B. Schuller, «Opensmile: The Munich Versatile and Fast Open-Source Audio Feature Extractor», en *Proceedings of the 18th ACM International Conference on Multimedia*, ép. MM '10, Firenze, Italy: Association for Computing Machinery, 2010, págs. 1459-1462, ISBN: 9781605589336. DOI: 10.1145/1873951.1874246. dirección: https://dl.acm.org/doi/pdf/10.1145/1873951.1874246.

[346] J. Bruna y S. Mallat, «Classification with scattering operators», en *CVPR 2011*, jun. de 2011, págs. 1561-1566. DOI: 10.1109/CVPR.2011.5995635.

[347] E. Oyallon, S. Mallat y L. Sifre, *Generic Deep Networks with Wavelet Scattering*, 2013. DOI: 10.48550/ARXIV.1312.5940. dirección: https://arxiv.org/abs/1312.5940.

[348] J. Andén y S. Mallat, «Deep Scattering Spectrum», *IEEE Transactions on Signal Processing*, vol. 62, n.º 16, págs. 4114-4128, ago. de 2014, ISSN: 1941-0476. DOI: 10.1109/TSP.2014.2326991.

[349] A. Kraskov, H. Stögbauer y P. Grassberger, «Estimating mutual information», *Phys. Rev. E*, vol. 69, pág. 066 138, 6 jun. de 2004. DOI: 10.1103/PhysRevE.69.066138. dirección: https://link.aps.org/doi/10.1103/PhysRevE.69.066138.

[350] L. Van Der Maaten y G. Hinton, «Visualizing high-dimensional data using t-sne», *Journal of Machine Learning Research*, vol. 9, n.º 2579-2605, págs. 2579-2605, 2008. dirección: https://www.scopus.com/inward/record.uri?eid=2-s2.0-70349325186&partnerID=40&md5=dbada745efefe67c34445366bf65733b.

[351] H. Hermansky y N. Morgan, «RASTA processing of speech», *IEEE Transactions on Speech and Audio Processing*, vol. 2, n.º 4, págs. 578-589, oct. de 1994, ISSN: 1063-6676. DOI: 10.1109/89.326616.

[352] T. K. Ho, «Random decision forests», en *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, ago. de 1995, 278-282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[353] ——, «The random subspace method for constructing decision forests», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, n.º 8, págs. 832-844, ago. de 1998, ISSN: 1939-3539. DOI: 10.1109/34.709601.

[354] C. Gini, «Concentration and dependency ratios», *Rivista di politica economica*, vol. 87, págs. 769-792, 1997.

[355] ——, *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche.[Fasc. I.]* Tipogr. di P. Cuppini, 1912.

[356] R. E. Schapire, «The Boosting Approach to Machine Learning: An Overview», en *Nonlinear Estimation and Classification*, D. D. Denison, M. H. Hansen, C. C. Holmes, B. Mallick y B. Yu, eds. New York, NY: Springer New York, 2003, págs. 149-171, ISBN: 978-0-387-21579-2. DOI: 10.1007/978-0-387-21579-2_9.

[357] J. Friedman, T. Hastie y R. Tibshirani, «Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors)», *The Annals of Statistics*, vol. 28, n.º 2, págs. 337-407, 2000. DOI: 10.1214/aos/1016218223.

[358] L. Breiman, «Bagging predictors», *Machine Learning*, vol. 24, n.º 2, págs. 123-140, ago. de 1996, ISSN: 1573-0565. DOI: 10.1007/BF00058655.

[359] P. Wasserman y T. Schwartz, «Neural networks. II. What are they and why is everybody so interested in them now?», *IEEE Expert*, vol. 3, n.º 1, págs. 10-15, Spring 1988, ISSN: 2374-9407. DOI: 10.1109/64.2091.

[360] R. Collobert y S. Bengio, «Links between Perceptrons, MLPs and SVMs», en *Proceedings of the Twenty-First International Conference on Machine Learning*, ép. ICML '04, Banff, Alberta, Canada: Association for Computing Machinery, 2004, pág. 23, ISBN: 1581138385. DOI: 10.1145/1015330.1015415. dirección: https://doi-org.ponton.uva.es/10.1145/1015330.1015415.

[361] B. Zhou, A. J. Ruggles, E. Huang y J. H. Frank, «Wavelet-based algorithm for correction of beam-steering artefacts in turbulent flow imaging at elevated pressures», *Experiments in Fluids*, vol. 60, n.º 8, pág. 136, jul. de 2019, ISSN: 1432-1114. DOI: 10.1007/s00348-019-2782-6.

[362] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss y K. Wilson, «CNN architectures for large-scale audio classification», en *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, mar. de 2017, págs. 131-135. DOI: 10.1109/ICASSP.2017.7952132.

[363] J. Cramer, H.-H. Wu, J. Salamon y J. P. Bello, «Look, Listen, and Learn More: Design Choices for Deep Audio Embeddings», en *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, mayo de 2019, págs. 3852-3856. DOI: 10.1109/ICASSP.2019.8682475.

[364] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal y M. Ritter, «Audio Set: An ontology and human-labeled dataset for audio events», en *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, mar. de 2017, págs. 776-780. DOI: 10.1109/ICASSP.2017.7952261.

[365] J. Laguarta y B. Subirana, «Longitudinal Speech Biomarkers for Automated Alzheimer's Detection», *Frontiers in Computer Science*, vol. 3, 2021, ISSN: 2624-9898. DOI: 10.3389/fcomp.2021.624694. dirección: https://www.frontiersin.org/articles/10.3389/fcomp.2021.624694.

[366] B. Subirana, F. Hueto, P. Rajasekaran, J. Laguarta, S. Puig, J. Malvehy, O. Mitja, A. Trilla, C. I. Moreno, J. F. M. Valle, A. E. M. González, B. Vizmanos y S. Sarma, *Hi Sigma, do I have the Coronavirus?: Call for a New Artificial Intelligence Approach to Support Health Care Professionals Dealing With The COVID-19 Pandemic*, 2020. DOI: 10.48550/ARXIV.2004.06510. dirección: https://arxiv.org/abs/2004.06510.

[367] J.-M. Valin, K. Vos y T. B. Terriberry, «Definition of the Opus Audio Codec», RFC Editor, RFC 6716, sep. de 2012. DOI: 10.17487/RFC6716. dirección: https://www.rfc-editor.org/info/rfc6716.

[368] J. Lyons, D. Y.-B. Wang, Gianluca, H. Shteingart, E. Mavrinac, Y. Gaurkar, W. Watcharawisetkul, S. Birch, L. Zhihe, J. Hölzl, J. Lesinskis, H. Almér, C. Lord y A. Stark, *jameslyons/python_speech_features: release v0.6.1*, ver. 0.6.1, ene. de 2020. DOI: 10.5281/zenodo.3607820. dirección: https://zenodo.org/record/3607820#.Y1KgKXZBxPY.

[369] J. Laguarta, F. Hueto, P. Rajasekaran, S. Sarma y B. Subirana, «Longitudinal speech biomarkers for automated alzheimer's detection», ago. de 2020. DOI: 10.21203/rs.3.rs-56078/v1. dirección: https://www.researchsquare.com/article/rs-56078/v1.

[370] B. Subirana, «Call for a Wake Standard for Artificial Intelligence», *Commun. ACM*, vol. 63, n.º 7, págs. 32-35, jun. de 2020, ISSN: 0001-0782. DOI: 10.1145/3402193. dirección: https://doi.org/10.1145/3402193.

[371] V. Panayotov, G. Chen, D. Povey y S. Khudanpur, «Librispeech: An ASR corpus based on public domain audio books», en *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, abr. de 2015, págs. 5206-5210. DOI: 10.1109/ICASSP.2015.7178964.

[372] S. R. Livingstone y F. A. Russo, «The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English», *PLOS ONE*, vol. 13, n.º 5, págs. 1-35, mayo de 2018. DOI: 10.1371/journal.pone.0196391. dirección: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0196391&fbclid=IwAR0pMF9vaxEvCucqjm2DJ1TH6CUv7JpBD79vi8qJcCAdHzJjJ4X2pFGDv_E.

[373] L. Orlandic, T. Teijeiro y D. Atienza, *The COUGHVID crowdsourcing dataset: A corpus for the study of large-scale cough analysis algorithms*, ver. 1.0, For more information about the data collection, preprocessing, validation, and data structure, please refer to the following publication: https://arxiv.org/abs/2009.11644 The cough pre-processing and feature extraction code is available from the following c4science repository: https://c4science.ch/diffusion/10770/, sep. de 2020. DOI: 10.5281/zenodo.4048312. dirección: https://zenodo.org/record/4048312#.Y1p3WHZBxPZ.

[374] J. Bankoski, «Intro to WebM», en *Proceedings of the 21st International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ép. NOSSDAV '11, Vancouver, British Columbia, Canada: Association for Computing Machinery, 2011, págs. 1-2, ISBN: 9781450307772. DOI: 10.1145/1989240.1989242. dirección: https://dl.acm.org/doi/10.1145/1989240.1989242.

[375] S. Pfeiffer, «The Ogg Encapsulation Format Version 0», RFC Editor, RFC 3533, mayo de 2003. DOI: 10.17487/RFC3533. dirección: https://www.rfc-editor.org/info/rfc3533.

[376] J.-M. Valin, *Codec Landscape*, https://opus-codec.org/comparison/, ago. de 2012. (visitado 22-05-2023).

[377] Desarrolladores FFmpeg, *Herramienta ffmpeg*, ver. 5.0, ene. de 2022. dirección: http://ffmpeg.org/.

[378] S. Ruder, «An overview of gradient descent optimization algorithms.», *CoRR*, vol. abs/1609.04747, 2016, cite arxiv:1609.04747Comment: Added derivations of AdaMax and Nadam. arXiv: 1609.04747. dirección: http://arxiv.org/abs/1609.04747.

[379] Y. Zhu, M. Imoussaïne-Aïkous, C. Côté-Lussier y T. H. Falk, *On the Impact of Voice Anonymization on Speech-Based COVID-19 Detection*, 2023. arXiv: 2304.02181 [cs.CL].

[380] J. Schlüter y T. Grill, «Exploring Data Augmentation for Improved Singing Voice Detection with Neural Networks.», en *Proceedings of the 16th International Society for Music Information Retrieval Conference*, (Málaga, Spain), Málaga, Spain: ISMIR, oct. de 2015, págs. 121-126. DOI: 10.5281/zenodo.1417745. dirección: https://zenodo.org/record/1417745.

[381] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk y Q. V. Le, «SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition», en *Proc. Interspeech 2019*, 2019, págs. 2613-2617. DOI: 10.21437/Interspeech.2019-2680. dirección: http://dx.doi.org/10.21437/Interspeech.2019-2680.

Apéndice A
CÓDIGO

A.1. Código de las CNN

Vamos a ver el código de las CNNs cuyos resultados vemos en la sección 7.1.1. Primero comenzamos con el código de la red neuronal convolucional A que describimos en la sección 6.1.2.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 256, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))
model.add((Flatten()))
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A continuación vemos el código de la red neuronal convolucional B que describimos en la sección 6.1.2.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
activation='relu',
```

```
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add((Conv2D(filters = 256, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))
model.add((Flatten()))
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A.2. Código de las C-LSTM

A.2.1. Código de las C-LSTM utilizando espectrograma normalizado

Presentamos el código de las primeras redes neuronales C-LSTM que hemos llevado a estudio y cuyos resultados observamos en la sección 6.1.3. El código de la C-LSTM que vemos a continuación es el de la C-LSTM A, la cual describimos en la sección 6.1.3.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))
model.add(Reshape((4*11, 128)))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(128, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])
```

```
return model
```

El código que presentamos a continuación es de la C-LSTM B. Esta segunda red neuronal que hemos llevado a estudio es descrita en la sección 6.1.3.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))
model.add(Reshape((4*11, 128)))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(128, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

Observamos ahora el código de la C-LSTM C que es la tercera red neuronal que hemos llevado a estudio y se ha descrito en la sección 6.1.3.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (7, 1),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1

model.add((Conv2D(filters = 64, kernel_size = (7, 1), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #2

model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))

model.add(Reshape((1*25, 128)))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100, return_sequences=True))
```

```
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

Se presenta a continuación el código de la cuarta red neuronal que hemos llevado a estudio, que es la C-LSTM D:

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (1, 7),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1

model.add((Conv2D(filters = 64, kernel_size = (1, 5), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #2

model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #3
model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))

model.add(Reshape((2*7, 128)))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A.2.2. Código de las C-LSTM utilizando espectrograma de mel

A continuación se presenta el código de la primera red neuronal que hemos llevado a estudio, que es la C-LSTM A:

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
```

```
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add(Reshape((1*36, 128)))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(128, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A continuación se presenta el código de la segunda red neuronal que hemos llevado a estudio, que es la C-LSTM B:

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add(Reshape((1*36, 128)))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(128, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A continuación se presenta el código de la tercera red neuronal que hemos llevado a estudio, que es la C-LSTM C:

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (7, 1),
activation='relu',
input_shape=(img_rows, img_cols, 1))))

model.add((Conv2D(filters = 64, kernel_size = (3, 1), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))

model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))

model.add(Reshape((2*158, 128)))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A continuación se presenta el código de la cuarta red neuronal que hemos llevado a estudio, que es la C-LSTM D:

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (1, 7),
activation='relu',
input_shape=(img_rows, img_cols, 1))))

model.add((Conv2D(filters = 64, kernel_size = (1, 5), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #2

model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))

model.add(Reshape((3*69, 128)))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
```

```
optimizer='adamax',
metrics=['accuracy'])

return model
```

A.2.3. Código de las C-LSTM utilizando ventanas desplazadas

A continuación se presenta el código de la primera red neuronal que hemos llevado a estudio.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))
model.add(Reshape((7*30, 128)))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(128, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A continuación se presenta el código de la segunda red neuronal que hemos llevado a estudio.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (2, 2),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))
model.add(Reshape((7*30, 128)))
```

```
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(128, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A continuación se presenta el código de la tercera red neuronal que hemos llevado a estudio.

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (7, 1),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1

model.add((Conv2D(filters = 64, kernel_size = (7, 1), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #2
model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #3

model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 1), activation='relu')))

model.add(Reshape((1*31, 128)))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A continuación se presenta el código de la cuarta red neuronal que hemos llevado a estudio, que es la C-LSTM D:

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (1, 7),
activation='relu',
```

```
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1

model.add((Conv2D(filters = 64, kernel_size = (1, 5), activation='relu')))
model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #2

model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2)))) #3
model.add((Conv2D(filters = 128, kernel_size = (3, 3), activation='relu')))

model.add(Reshape((5*26, 128)))
model.add(LSTM(100, return_sequences=True))
model.add(LSTM(100, return_sequences=True))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A.3. Código de las C-BiLSTM

A.3.1. Código de las C-BiLSTM utilizando espectrograma normalizado

Esta red neuronal C-BiLSTM utilizada para la base de datos COUGHVID sin *transfer learning* es similar a la C-LSTM D con entrada de tamaño (89, 100) por lo que la capa Reshape es (9*11, 128):

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (1, 7),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))
model.add(Reshape((9*11, 128)))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
```

```
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

Esta red neuronal C-BiLSTM utilizada para la base de datos COUGHVID con *transfer learning* utilizando como base la C-BiLSTM A:

```
def get_model():
base_model = load_model('./TosVSNoTosPalencia1sKFold_Espectrograma_C-BiLSTM/Prueba A/K1

model = Sequential()
model.add((Conv2D(filters = 32, kernel_size = (10, 1),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((Conv2D(filters = 32, kernel_size = (10, 1), activation='relu')))
model.add((Conv2D(filters = 32, kernel_size = (10, 1), activation='relu')))
model.add((Conv2D(filters = 32, kernel_size = (10, 1), activation='relu')))
model.add((Conv2D(filters = 32, kernel_size = (9, 1), activation='relu')))
for layer in base_model.layers[1:-1]:
model.add(layer)
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

for layer in model.layers[5:-5]:
layer.trainable = False

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```

A.3.2. Código de las C-BiLSTM utilizando espectrograma de mel

Esta red neuronal C-BiLSTM utilizada para la base de datos COUGHVID sin *transfer learning* es similar a la C-LSTM D con entrada de tamaño (16, 158):

```
def get_model():
model = Sequential()

model.add((Conv2D(filters = 32, kernel_size = (1, 7),
activation='relu',
input_shape=(img_rows, img_cols, 1))))
model.add((MaxPooling2D (pool_size = (2,2)))) #1
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 32, kernel_size = (2, 2), activation='relu')))
model.add((MaxPooling2D (pool_size = (2,2))))   #2
```

```
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 64, kernel_size = (2, 2), activation='relu')))
model.add((Dropout(0.1)))
model.add((Conv2D(filters = 128, kernel_size = (2, 2), activation='relu')))
model.add(Reshape((1*36, 128)))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add((Flatten()))
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy,
optimizer='adamax',
metrics=['accuracy'])

return model
```