UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Detección de disparos en el Parque Natural de la Albufera mediante inteligencia artificial

Trabajo Fin de Grado

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

AUTOR/A: Moreno Lara, Sergio

Tutor/a: Piñero Sipán, María Gemma

Cotutor/a externo: Pérez García de la Puente, Natalia Lourdes

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

NO PONER PORTADA

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO. ES-CV- 000030

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la ingeniería de telecomunicación

CONCEPT (ABET)	**CONCEPTO (traducción)**	**¿Cumple? (S/N)**	**¿Dónde? (páginas)**
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	Si	Página 1
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	Si	Página 1
1.3. Setting of goals	1.3. Establecimiento de objetivos	Si	Página 2
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	Si	Páginas 31-35
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)	Si	Páginas 35-41
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	Si	Página 41
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	Si	Página 42-43

Resumen

La detección de disparos en entornos naturales es muy importante, no sólo en entornos urbanos sino también en espacios naturales protegidos y sus especies en peligro de extinción. En este trabajo, presentamos un conjunto de datos construido a partir de extractos de grabaciones de paisajes sonoros en cinco lugares diferentes del Parque Nacional de la Albufera española. Luego realizamos un estudio experimental para detectar disparos desde diferentes distancias, etiquetados como “background”, “cerca”, “medio”, “lejos”. Para ello, desarrollamos un extractor de funciones de audio y dos Deep Convolutional Neural Networks (DCNN) eficientes, luego mediremos su rendimiento, además los compararemos con anteriores trabajos.

Resum

La detecció de trets en entorns naturals és molt important, no sols en entorns urbans sinó també en espais naturals protegits i les seues espècies en perill d'extinció. En este treball, presentem un conjunt de dades construït a partir d'extractes de gravacions de paisatges sonors en cinc llocs diferents del Parc Nacional de l'Albufera espanyola. Després realitzem un estudi experimental per a detectar trets des de diferents distàncies, etiquetats com a “background”, “cerca”, “medio”, “lejos”. Per a això, desenrotllem un extractor de funcions d'àudio i Deep Convolutional Neural Networks (DCNN) eficients, després mesurem el seu rendiment, ademas els comparem amb treballs anteriors.

Abstract

Gunshot detection in natural environments is very important, not only in urban environments but also in natural protected areas and their endangered species. In this work, we present a dataset built from extracts of the soundscape recordings at five different locations of the Spanish Albufera National Park. We then carry out an experimental study to detect gunshots from different distances, labeled as “background” “cerca” “medio” “lejos”. For this purpose, we develop an audio feature extractor and two efficient Deep Convolutional Neural Networks (DCNNs), then we measure their performance, in addition we compare them to previous works.

Índice general

Índice de figuras

Glosario

IA Inteligencia Artificial.

ML Machine Learning.

DL Deep Learning.

DCNN Deep Convolutional Neural Network.

SED Sound Event Detection.

SVM Support Vector Machine.

GMM Gaussian Mixture Model.

LSTM Long Short Term Memory.

TF Transformada de Fourier.

TIF Transformada Inversa de Fourier.

STFT Short Time Fourier Transform.

FFT Fast Fourier Transform.

DFT Discrete Fourier Transform.

KNN K-Nearest Neighbours.

ZEPA Zona de especial protección de las Aves.

LIC Lugar de Interés Comunitario.

CPU Central Processing Unit.

GPU Graphic Processing Unit.

FAIR Facebook's AI Research Lab.

NLP Natural Language Processing.

I Memoria

1 Introducción

1.1 Contexto y motivación del estudio

La monitorización ambiental para el análisis de eventos sonoros ha dependido históricamente de una considerable intervención humana, tanto en la recopilación de los datos como en la supervisión del proceso. Sin embargo, las nuevas tecnologías de captura de audio e imágenes que han surgido en los últimos años, junto con su procesamiento avanzado, permiten su aplicación en la monitorización ambiental con dos objetivos principales: por un lado, automatizar la recolección de datos, disminuyendo el tiempo que los investigadores dedican a esta tarea, y por otro, extraer información valiosa que pueda ser usada en futuras investigaciones. Todo esto mediante el uso de técnicas de Inteligencia Artificial (IA).

1.2 Descripción del problema

El problema que se plantea en este trabajo es cómo desarrollar un sistema de detección de disparos en entornos naturales protegidos que sea capaz de discriminar con precisión entre los sonidos de disparos a diferentes distancias, garantizando una alta tasa de detección de disparos reales. Este problema se aborda mediante el uso de técnicas Deep Learning, en concreto con las redes basadas en Deep Convolutional Neural Networks (DCNN), para modelar y clasificar los diferentes tipos de sonidos de disparos presentes en el entorno del Parque Nacional de Albufera. Además, la recopilación de datos debe ser lo menos intrusiva con el entorno debido a la importancia de la biodiversidad que hay en este espacio natural protegido.

Planteamiento del problema y oportunidad

Debido al alto grado de participación humana en la monitorización medioambiental, tanto en la recopilación de datos como en la supervisión del procedimiento, surge la necesidad de desarrollar un sistema de detección de disparos automático, capaz de detectar con precisión y diferenciar estos a distintas distancias. La recopilación de datos debe ser además lo menos intrusiva para la biodiversidad de esta zona protegida. Esto permitirá reducir la implicación humana en las tareas de monitorización y nos permitirá extraer datos útiles para nuevas líneas de trabajo.

Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)

Debido a que este trabajo se centra en la investigación en el campo de la Inteligencia Artificial, no existe ninguna regulación ni normas técnicas que debamos seguir. El trabajo se centra en desarrollar nuevos sistemas de detección de sonidos, un campo relativamente nuevo y que puede ser desarrollado de diferentes formas dependiendo de los propios investigadores.

1.3 Objetivo del trabajo

Este TFG tiene como objetivo desarrollar un sistema para la detección de disparos en el entorno del Parque de la Albufera mediante Inteligencia Artificial, en concreto usando modelos de Deep Learning (DL). La detección de disparos es crucial para la seguridad pública y la conservación del medio ambiente en áreas naturales. Utilizando estos modelos de DL, se busca automatizar el proceso y mejorar la eficiencia y precisión de la detección. En concreto, los objetivos del TFG son:

- Crear una base de datos a partir de sonidos grabados en el entorno del Parc de l'Albufera etiquetando los disparos en tres categorías: lejano, medio y cercano. Existen grabaciones tomadas en las matas de la laguna, en los alrededores de la laguna y en el Vedat de Sueca en distintos días de las temporadas de caza 2022-2023 y 2023-2024.
- Desarrollar un modelo de DL para la detección automática de disparos y su categorización según la distancia.
- Evaluar el rendimiento del modelo en entornos reales, en términos de precisión y eficacia en comparación con modelos existentes.
- Revisar el estado de arte en la materia.

Establecimiento de objetivos

Como se ha mencionado en el apartado anterior, los objetivos principales de este trabajo son la creación de una base de datos nueva a partir de las grabaciones obtenidas de los nodos situados en el Parque Natural, el desarrollo de un modelo de DL capaz de detectar disparos a distintas distancias, evaluar nuestro modelo en términos de precisión y eficacia y compararlos con otros modelos, y, por último, revisar el estado del arte en la materia.

1.4 Estado del arte

1.4.1 Revisión de la literatura sobre detección de eventos sonoros y análisis acústico

La detección de eventos y en concreto, la detección de disparos siempre ha tenido un papel importante en el ámbito de la investigación, los primeros sistemas de detección usaban conjuntos de sensores y dispositivos de grabación conectados entre sí para comunicarse [1]. En los últimos años, las investigaciones se han centrado en crear algoritmos de detección automática de disparos en entornos naturales y urbanos [2]. Esta detección es crucial para la seguridad y la conservación de numerosos entornos.

La detección de eventos sonoros (SED) se ha tratado durante los últimos años como una herramienta para detectar eventos en una secuencia de audio [3]. A diferencia de un clasificador de imágenes tradicional, los audios presentan una gran variabilidad como son el ruido, la distancia, la superposición de otros sonidos o incluso como se procesa la ventana temporal.

Los primeros algoritmos usados para la detección de disparos contaban con una fase de poca complejidad de detección de eventos y una más compleja para detectar los disparos. Frecuentemente se usaban *SVM (Support Vector Machine)* o *GMM (Gaussian Mixture Models)* [4], de los que se extraían las características del espectrograma [5] para entrenar la red.

Investigaciones más recientes han demostrado que las redes neuronales profundas tienen mejor desempeño respecto al coste computacional que los vectores de características [6]. Estos modelos resultan útiles para diferentes tareas, como son la clasificación de imágenes o audio.

En nuestro trabajo, usaremos redes neuronales convolucionales, ya que se ha demostrado que son útiles tanto para clasificación de audio como para detección de eventos [7]. Esta versatilidad es la que nos ha hecho decantarnos por estos modelos.

1.4.2 Métodos y tecnologías utilizadas para la detección de sonidos

Existen varios tipos de clasificación de audio, dependiendo de si es necesario o no incluir información temporal en el resultado. Si el resultado depende de las etiquetas generadas a lo largo del tiempo, se trata de un problema de detección de eventos sonoros (Sound Event Detection). Si no se requiere dicha información temporal, podemos distinguir entre dos casos: si la salida corresponde a una única clase asociada al audio introducido, hablamos de clasificación de escenas sonoras (Sound Scene Classification); por otro lado, si la salida contiene múltiples etiquetas que identifican diferentes sonidos, nos encontramos ante un problema de etiquetado de audio (Audio Tagging). Estos tipos de clasificación se ilustran en la Figura 1.

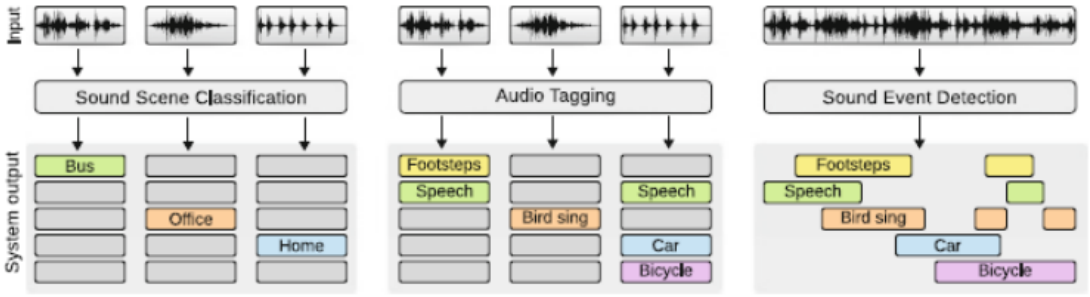


Figura 1 Tipos de análisis de audio según su salida

El problema de detección de audio es el que presenta mayor complejidad a la hora de la clasificación. Se trata de clasificación de audio en tiempo real, el sistema debe ser capaz de detectar cuando empieza y cuando acaba un evento concreto en una secuencia. Los datos de entrada son audios de duración considerable y la salida son las etiquetas que han ido apareciendo en estos audios en el tiempo. Estas etiquetas pueden ser fuertes o débiles. La diferencia es que las etiquetas fuertes son más concretas, ya que muestran el inicio y el final del evento, mientras que las etiquetas débiles solo indican la presencia o la ausencia de ese sonido en el audio. La forma óptima de generar etiquetas es obtener muestras de audio generadas de manera sintética usando el evento y añadiendo ruidos de diferente naturaleza, obteniendo un conjunto de datos diverso. Algunos datasets públicos que pueden ser usados son *UrbanSound8k* o o *FSDnoisy18k*.

Las primeras aproximaciones a este tipo de clasificación usaban una mezcla de características armónicas y espaciales apoyándose en los LSTM (*Long Short Term Memory*), este aprovechamiento espacial multicanal permite detectar mejor los eventos solapados que en canales mono [8].

Trabajos posteriores proponen sistemas basados en DNN que permiten la clasificación en entornos complejos, donde el número de eventos pueden ser indeterminados [9]. Otras investigaciones plantean la implementación de espectrogramas de múltiples resoluciones utilizando múltiples DNN en paralelo y añadiendo las salidas independientes en una única salida, con este método la precisión de trabajos anteriores mejora notablemente [10].

2 Metodología

2.1 Técnicas de procesamiento de audio

Las señales de audio poseen una amplia variedad de características tanto temporales como frecuenciales, resulta evidente que algunas no aportan tanta información que otras en determinados escenarios. Para poder crear nuestro modelo con éxito, es importante extraer y procesar aquellas características que resulten más relevantes para nuestra tarea.

En este trabajo se utilizará el espectrograma como representación temporal y frecuencial de las señales de audio debido a que nos permite usar modelos de clasificación de imágenes muy potentes.

2.1.1 Short Time Fourier Transform

Una señal de sonido es el resultado de la superposición de varias ondas con frecuencias diferentes. Las amplitudes de esta señal pueden representarse a lo largo del tiempo. La transformada de Fourier (TF) es una herramienta matemática que permite convertir una señal del dominio temporal al dominio frecuencial para obtener más información sobre ella. La Transformada Inversa de Fourier (TIF) es el método utilizado para realizar el proceso inverso, transformando la señal del dominio frecuencial al temporal. Estas herramientas permiten analizar el contenido espectral y temporal de una señal y comprender su comportamiento en términos de frecuencia y tiempo,para nuestro caso, lo que nos permite identificar las frecuencias asociadas a cada sonido. La Figura 2 explica este proceso.

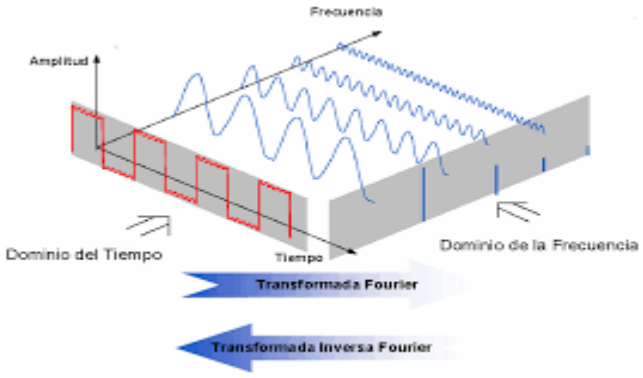


Figura 2 Transformada de Fourier en el dominio frecuencial y temporal.

Para procesamiento de señales, la transformada más útil es la *Discrete Fourier Transform* (DFT) que no es más que una transformada de Fourier aplicada a un intervalo concreto de tiempo [11]. La fórmula es la que aparece en (1):

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi k}{N} n} \quad k = 0,1,2 \dots N-1 \qquad (1)$$

Donde x[n] es la señal discreta y $e^{\frac{-j2\pi k}{N} n}$ es la exponencial compleja.

Para procesar señales digitales de audio se utiliza la *Short Time Fourier Transform* (STFT) que utiliza una versión optimizada de las DFT (2), llamadas *Fast Fourier Transform* (FFT). La señal

de audio es dividida en ventanas temporales de duración variable y se aplica esta FFT, obteniendo la información frecuencial de cada ventana.

$$X[n,k] = \sum_{n=0}^{N-1} w[m]x[n+m]e^{\frac{-j2\pi k}{N}n} \quad (2)$$

Donde $w[m]$ es la ventana temporal, $x[n+m]$ es la señal desplazada m muestras y $e^{\frac{-j2\pi k}{N}n}$ es la exponencial compleja.

2.1.3 El Espectrograma

El espectrograma representa el módulo de la STFT, nos permite obtener información sobre la energía de la señal. En el eje vertical se representa la frecuencia y su energía, esta puede estar representada en decibelios o unidades logarítmicas. En el eje horizontal tenemos el tiempo de la señal.

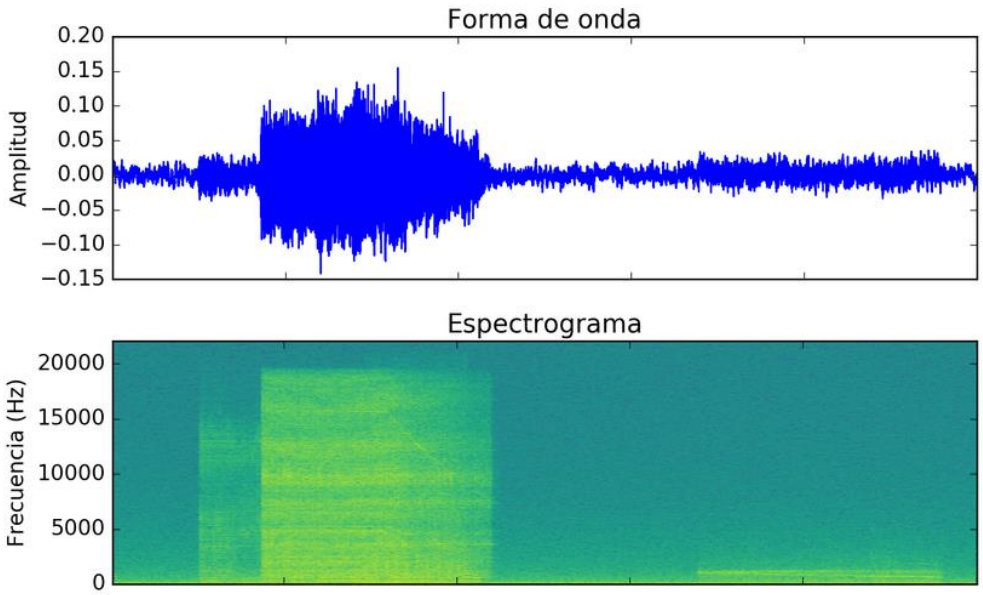


Figura 3 Forma de onda en el tiempo y su espectrograma en frecuencia.

Como se puede apreciar en la Figura 3, la parte donde la amplitud en el dominio temporal es mayor, tiene una representación frecuencial también mayor.

2.1.4 Espectrograma de Mel

El espectrograma de Mel (3) es similar al espectrograma, pero con la diferencia de que usa una escala similar a la que perciben los humanos, el oído humano es más sensible a los cambios de frecuencia baja que a las altas. A medida que la frecuencia aumenta, esta sensibilidad disminuye de manera logarítmica. La escala Mel es la escala que adapta la respuesta frecuencial no lineal humana [12].

$$f_{mel} = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \quad (3)$$

Donde f_{mel} es la frecuencia mel y f la frecuencia normal.

Para crear este espectrograma se usa la misma STFT que en el espectrograma, pero aplicando una serie de filtros llamados filtros de Mel para transformarla a la escala Mel. Estos filtros se pueden ver en la Figura 4.

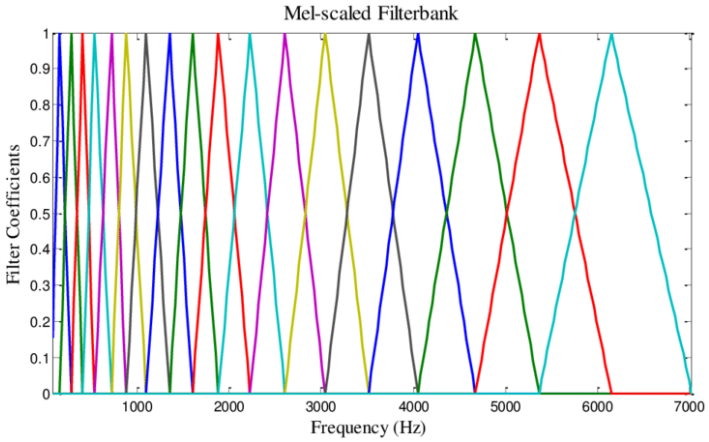


Figura 4 Representación de los filtros de Mel.

El banco de filtros de Mel generalmente está normalizado, de lo contrario, las *MelBands* de las frecuencias más altas serían demasiado grandes y no reflejarían bien la percepción auditiva humana. Debido a su gran utilidad para extraer características en las frecuencias de los sonidos que se analizan, este método de filtrado se utiliza principalmente en tareas como el reconocimiento de voz, la clasificación de sonidos y la detección de eventos sonoros.

2.2 Diseño del modelo de Inteligencia Artificial para la detección de disparos

2.2.1 Inteligencia Artificial

La Inteligencia Artificial es el área de conocimiento que busca la creación de máquinas que imiten comportamientos inteligentes [13]. Esta área de conocimiento engloba una amplia variedad de sistemas. Aunque el campo de la IA parece relativamente nuevo, este lleva entre nosotros desde 1950, con Adam Turing y su trabajo [14]. Es en estos últimos años cuando se han hecho más avances en este campo, Tenemos GPUs más potentes, algoritmos mejor optimizados y redes con las que se puede entrenar cualquier tipo de dato. Gracias a estos avances hemos podido solucionar un problema complejo que algunos años atrás era impensable. Existen dos tipos de IA

La IA débil está diseñada para llevar a cabo tareas muy específicas y no puede realizar ninguna otra tarea diferente. Los sistemas actuales son ejemplos de IA débil. En cambio, una IA fuerte sería capaz de llevar a cabo diversas tareas y adaptarse a diferentes entornos, aunque aún no se ha desarrollado ninguna.

En general, todas las Inteligencias Artificiales están programadas para exhibir comportamientos inteligentes, y existen distintas áreas dentro de la IA que buscan imitar diferentes tipos de comportamiento. Por ejemplo, la robótica se enfoca en que los sistemas realicen movimientos similares a los humanos, mientras que otras áreas se dedican al procesamiento del lenguaje o la visualización de objetos. Aunque estos sistemas realizan sus funciones según su programación, los humanos tienen una capacidad única para ajustar nuestros comportamientos según el problema que enfrentamos, gracias a nuestra habilidad para aprender.

Aprendizaje Supervisado

Se define por el uso de conjuntos de datos etiquetados e introducidos por humanos, de ahí viene lo de “supervisado”. Para entrenar algoritmos que clasifican los datos o predicen los resultados se utilizan diferentes técnicas como el SVM, Random Forest, Naive Bayes o las redes neuronales entre otras. El algoritmo mide su precisión a través de la función de pérdida, ajustando hasta que el error se haya minimizado lo suficiente como para dar un resultado fiable. El aprendizaje supervisado ayuda a resolver una variedad de problemas del mundo real como la clasificación de sonido e imágenes, el análisis predictivo o la detección de spam.

Aprendizaje No Supervisado

El aprendizaje no supervisado o también conocido como *Machine Learning*, introduce al sistema datos de entrada sin etiquetar. El sistema debe de aprender las características similares entre los datos por medio de técnicas estadísticas y obtener patrones en estos. Se trata de un campo en el cual se está avanzando rápidamente, ya que es muy útil, pues reduce la carga de trabajo de los investigadores al no tener que clasificar manualmente los datos. Algunos ejemplos de los sistemas empleados para aprendizaje no supervisado son el Random Forest o las técnicas de clustering como el KNN (*K-Nearest Neighbours*).

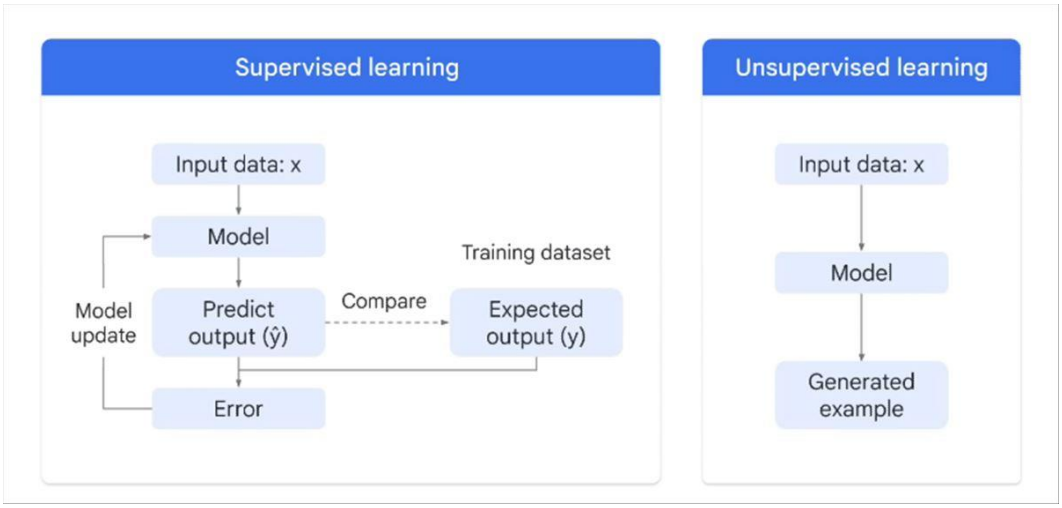


Figura 5 Clasificación de los diferentes tipos de Inteligencia Artificial.

2.2.2 La neurona

Para poder entender cómo funcionan las redes neuronales, debemos entender primero las unidades más básicas que las conforman, las neuronas o también llamadas perceptrones. Estas neuronas tienen una serie de redes de entrada x_i por las cuales reciben estímulos externos, con estos datos de entrada junto a unos pesos w_i asignados a cada entrada, se realiza una regresión lineal y añadiendo un sesgo b tenemos la salida.

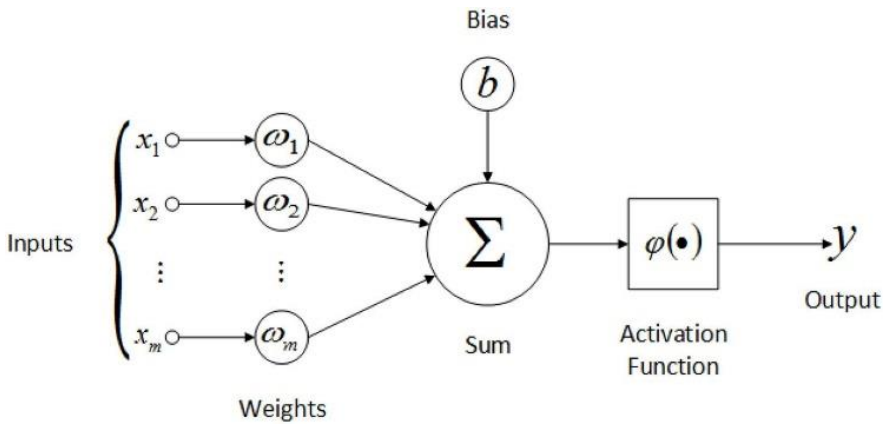


Figura 6 Esquema de una neurona.

.

A esta salida se le aplica una función no lineal llamada función de activación que permite mantener una conexión entre las diferentes capas de neuronas y así aprender patrones complejos. Con estas conexiones se garantiza una red potente capaz de realizar tareas difíciles. Cada función de activación tiene sus propias propiedades y es adecuada para determinados casos de uso (Figura 7).

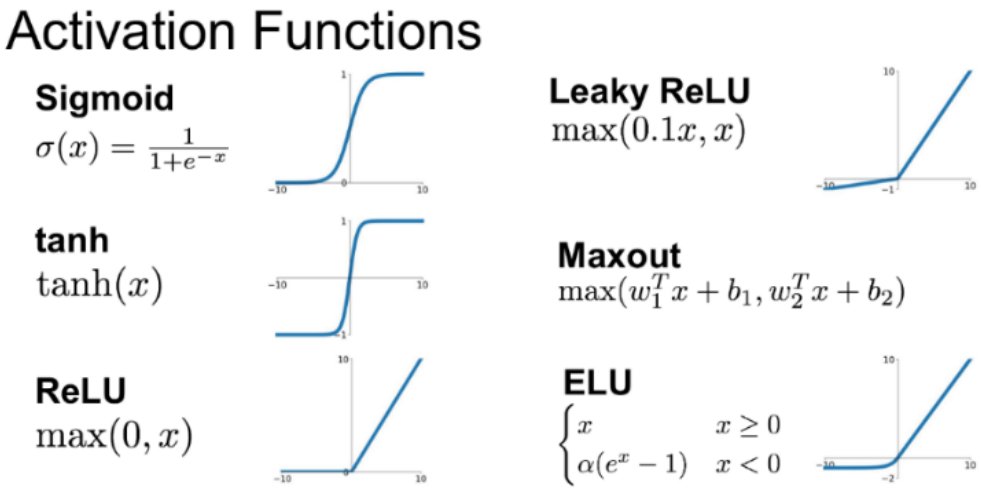


Figura 7 Tipos de funciones de activación.

2.2.4 Redes neuronales

Las redes neuronales proponen una solución al problema de la puerta XOR que tienen las neuronas [15]. El problema de la puerta XOR se refiere a que un perceptrón por sí mismo no es capaz de realizar una operación XOR, para solucionar esto, se añade otra neurona a la capa. Al añadir más neuronas a la red, el modelo es capaz de obtener información más compleja de las entradas. A esta profundidad en la cantidad de capas del modelo se le llama Deep Learning [16].

Cuanto más profunda sea nuestra red, mayor capacidad de abstracción tendrá. Además, estas capas están completamente interconectadas. En otras palabras, cada salida de las neuronas en una capa se conecta a todas las neuronas de la capa siguiente. Este tipo de capas se conoce como capas densamente conectadas.

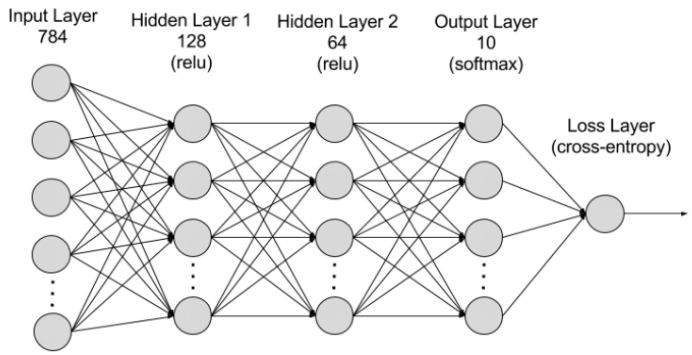


Figura 8 Arquitectura de una red neuronal y sus capas.

Existen diferentes tipos de capas de salida según el tipo de salida que requiera el modelo. Para salidas de una sola clase se usa la capa *Softmax*, es útil para modelos de clasificación de imágenes o sonidos. Para salidas que tienen varias clases se usa la capa *Sigmoid,* esta capa da una probabilidad de que a la salida aparezca una clase u otra. Se usan en etiquetado de eventos o en detección de objetos en video entre otros.

2.2.5 Redes neuronales convolucionales

Las redes neuronales convolucionales son un tipo de red neuronal que tiene por entrada una imagen a la que se le aplican una serie de filtros llamados *kernels,* que pueden tener diferentes tamaños y que extraen diferentes características, primero las más sencillas como pueden ser bordes o formas y a medida que van pasando por las capas, estas características se van haciendo más abstractas, hasta que la red es capaz de combinarlas y dar una predicción fiable.

Las redes neuronales convolucionales han demostrado tener un enorme éxito a la hora de clasificar gran cantidad de imágenes. Numerosos estudios han buscado mejorar la precisión de estas redes. Una de las primeras redes en el ámbito de la visión por computador fue *AlexNet* que logró una precisión del 84,7% en la competición ILSVRC-2012, esta red utiliza el poder de las GPUs y técnicas como el *data augmentation* o el *dropout* para manejar tareas complejas [17].

Convolución

Las capas en cuestión realizan una operación de convolución sobre los datos de entrada, que suelen ser representaciones bidimensionales, como imágenes o espectrogramas. La capa inicial de una red neuronal convolucional tiene dimensiones equivalentes a las de los datos de entrada, lo que exige que todas las imágenes tengan un tamaño uniforme. A continuación, se lleva a cabo la operación de convolución (ver Figura 9). En esta operación, un kernel recorre la imagen de izquierda a derecha y de arriba abajo, avanzando una cantidad específica de píxeles denominada *stride*. Este filtro actúa como la matriz de pesos y sesgos de la capa. La convolución se realiza multiplicando los valores de la imagen con los del kernel y sumándolos. El resultado se transfiere a la siguiente capa de la red, y el proceso se repite.

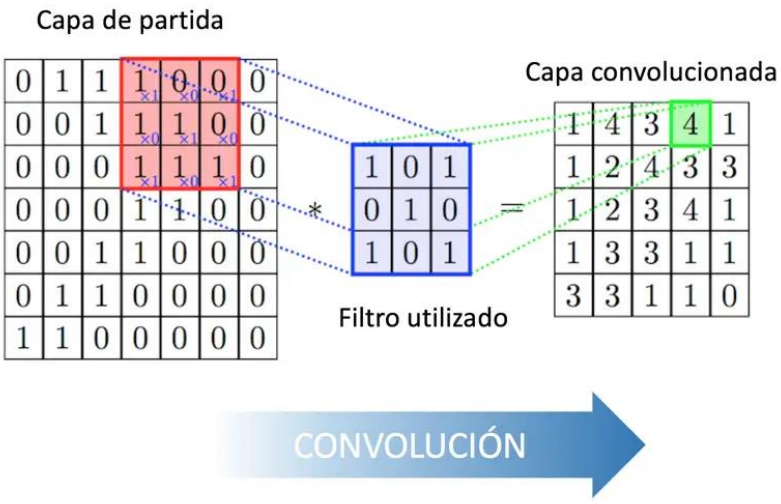


Figura 9 Operación de convolución.

La operación descrita corresponde a una única convolución en una capa, lo que equivale a la actividad de una sola neurona. Por esta razón, las capas suelen tener múltiples filtros con diferentes pesos y sesgos, lo que les permite realizar varias convoluciones de manera simultánea. Esto aumenta la capacidad de la red para capturar diferentes características de los datos de entrada.

2.2.6 Entrenamiento y validación

Pooling

Otra capa utilizada en las redes neuronales convolucionales es la capa de pooling. Para identificar patrones más complejos en las imágenes, se necesitan varias capas de convolución consecutivas, lo que implica un alto costo computacional y un tiempo considerable de entrenamiento. Para disminuir la cantidad de parámetros, se emplean las capas de pooling, que funcionan de manera similar a las capas de convolución. Estas capas aplican un filtro sobre la imagen y producen un resultado. En lugar de operar sobre todos los píxeles, estas capas seleccionan los píxeles en una región y devuelven un solo valor. La operación de pooling más común es el *max pooling* (Figura 10), en la que se toma un grupo de píxeles, como en una región de 2x2, y se pasa a la siguiente capa solo el valor del píxel más alto.

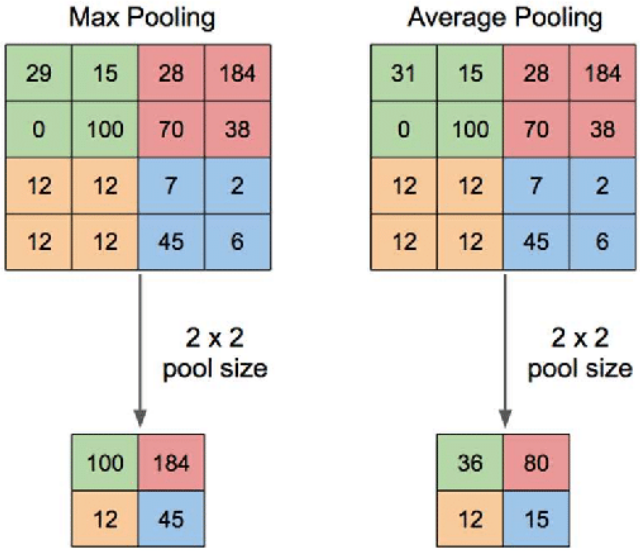


Figura 10 Representación de Max Pooling.

Flatten

Para finalizar, el final de una red neuronal incluye una capa de salida densamente conectada. Para conectar adecuadamente los datos bidimensionales a las siguientes capas densas o a la capa de salida, es necesario transformarlos en una estructura que permita dicha conexión. Aquí es donde entra en juego la capa *flatten*, que convierte la capa anterior de convolución o pooling en una capa de neuronas, permitiendo así su conexión densa con la siguiente capa.

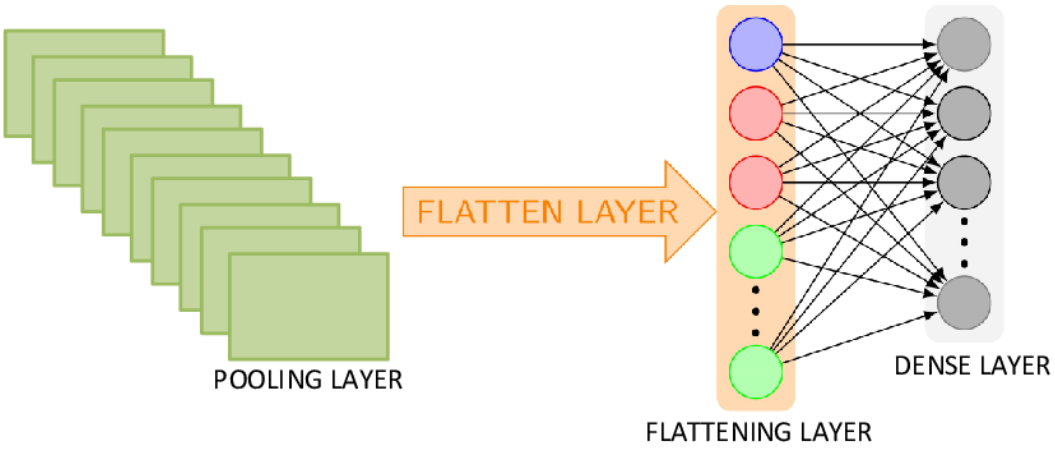


Figura 11 Operación de Flattening.

Dropout

Existen varios métodos para prevenir el sobreajuste en una red neuronal, uno de los cuales implica una capa específica que se inserta en el modelo con funciones diversas. Se trata del algoritmo de *Dropout*. Este algoritmo funciona de manera que, en cada iteración del modelo, un porcentaje aleatorio de neuronas se desactiva durante el entrenamiento, estableciendo sus entradas a cero. Un ejemplo de cómo opera se muestra en la Figura 12. Un valor que suele ser utilizado para *Dropout* es 0.5, esto quiere decir que el 50% de las neuronas seleccionadas aleatoriamente no formarán parte del entrenamiento. En nuestro caso, probaremos diferentes valores para encontrar el resultado óptimo.

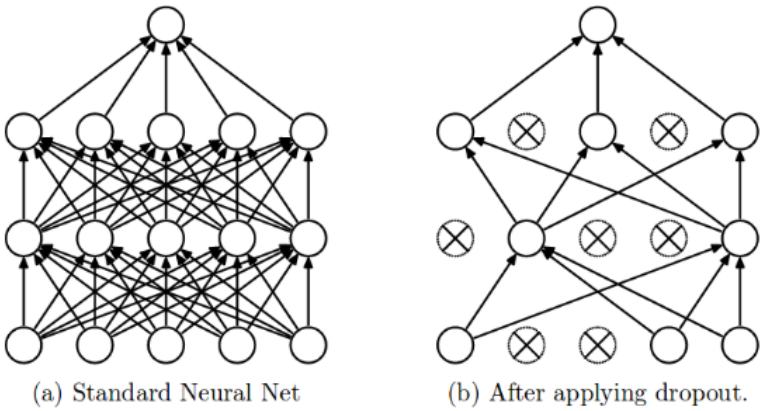


Figura 12 Aplicación de Dropout a las neuronas.

2.2.7 Función de coste

El entrenamiento de la red se basa en un proceso de prueba y error. Al iniciar la red, los pesos y sesgos se establecen con valores aleatorios. Cuando un dato de entrenamiento pasa por la red y llega a la capa de salida, es probable que el resultado inicial sea incorrecto. Para medir este error, se utilizan diferentes funciones de pérdida, como el error cuadrático medio. El objetivo es mejorar la red para reducir el error, y aquí es donde entra en juego el algoritmo de *Backpropagation*.

2.2.8 Backpropagation

El algoritmo de *Backpropagation* [19] es fundamental en la estructura de una red neuronal, ya que permite ajustar automáticamente los parámetros de la red y aprender una representación interna de los datos procesados. Este algoritmo calcula todas las derivadas parciales de la función de pérdida con respecto a cada uno de los pesos y bias del modelo. Aunque a priori pueda parecer una tarea de alta complejidad debido a la gran cantidad de parámetros en una red, la potencia del algoritmo yace en su capacidad para calcular las derivadas parciales de manera retroactiva. Es decir, empieza calculando las derivadas en la última capa oculta y, utilizando esos resultados, pasa a calcular las derivadas en la capa anterior, repitiendo este proceso hacia atrás a lo largo de la red.

Antes de que el algoritmo de Backpropagation se crease el cálculo de las derivadas se hacía por perturbación aleatoria o lo que es lo mismo, fuerza bruta, calculando todos los posibles caminos desde la capa de entrada hasta la capa de salida y viendo los posibles cambios al alterar diferentes pesos, siendo una tarea con un coste computacional muy elevado.

2.2.9 Función de reducción del coste

Una vez que se han calculado todas las derivadas parciales, es necesario utilizar un algoritmo de optimización para reducir el coste. Entre estos, el más conocido es el algoritmo de descenso por gradiente [20]. Este tipo de algoritmo utiliza las derivadas parciales obtenidas en el paso anterior, conocidas como gradientes, para ajustar los valores de los pesos y sesgos. El objetivo es que, en la siguiente iteración, el error disminuya, guiando así el modelo hacia una mejor solución.

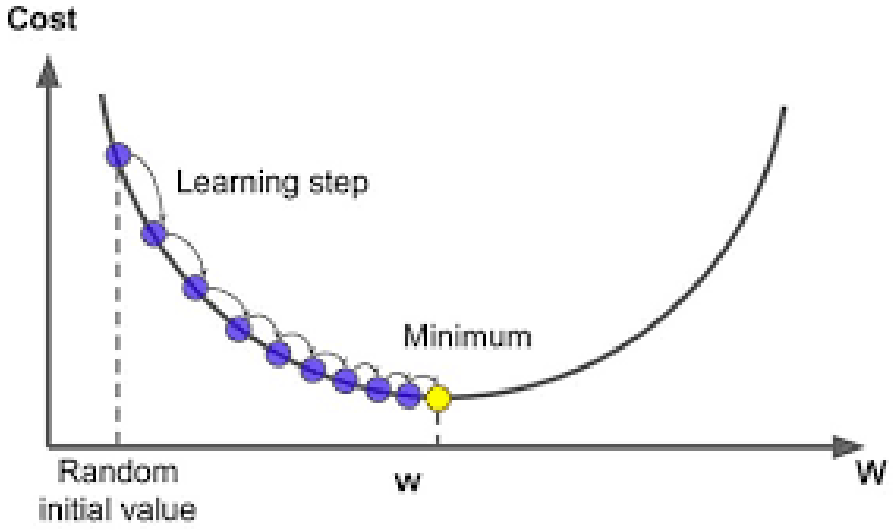


Figura 13 Grafica de reducción de coste.

Los datos de entrenamiento generalmente constan de miles de muestras, que se dividen en grupos más pequeños llamados *batches* y se entrenan durante múltiples iteraciones conocidas como *epochs*. A lo largo de este proceso, el algoritmo de reducción del error va mejorando hasta acercarse al mínimo global, que es el conjunto de valores de pesos y sesgos que minimiza el error. Para lograr esto, es crucial ajustar un parámetro llamado *learning rate*. Si este valor es alto, los pasos del algoritmo serán más grandes y la iteración será más rápida, pero existe el riesgo de no alcanzar el mínimo global. Por otro lado, si el *learning rate* es bajo, los pasos serán más pequeños, lo que aumenta la probabilidad de llegar al mínimo global, aunque el proceso será más lento. Por lo tanto, es importante encontrar un equilibrio entre la reducción del error y el tiempo necesario para alcanzarlo.

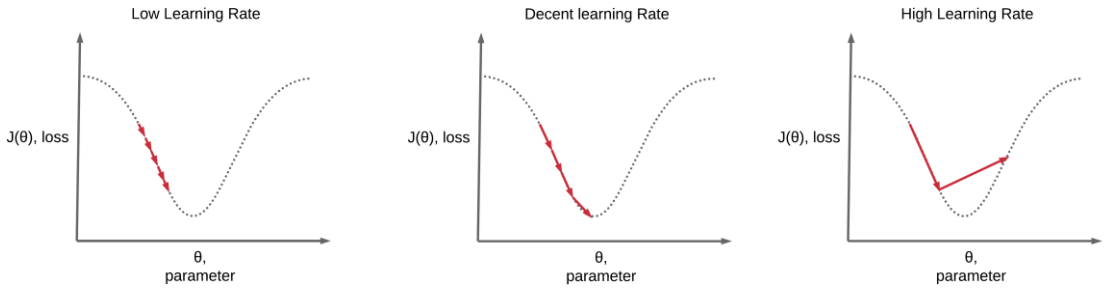


Figura 14 Diferentes learning rates y su resultado.

2.3 Métricas de evaluación y validación del modelo

2.3.1 Accuracy

Es una métrica de evaluación que se utiliza para medir el rendimiento de un modelo de aprendizaje automático, incluidas las redes neuronales convolucionales. En el contexto de una red neuronal, la accuracy representa la proporción de predicciones correctas realizadas por el modelo con respecto al número total de predicciones.

$$Accuracy = \frac{numero\ de\ predicciones\ correctas}{numero\ de\ predicciones\ totales} \quad (4)$$

Según el resultado de esta métrica podremos conocer cómo se comporta nuestro modelo. Podemos dividir estos comportamientos en tres (Figura 15):

Overfitting

El objetivo principal al entrenar una red neuronal es lograr un buen ajuste que le permita generalizar bien a nuevos datos. Esto se consigue mediante múltiples iteraciones. Sin embargo, si se entrena la red en exceso, esta puede llegar a memorizar los datos de entrada, lo que reduce su capacidad de generalización. Como resultado, cuando se le presenta un dato nuevo que no ha visto antes, la red podría clasificarlo incorrectamente. Este problema se conoce como *overfitting* y es un fenómeno común en el entrenamiento de redes neuronales.

Underfitting

El underfitting ocurre cuando un modelo no se ajusta adecuadamente a los datos de entrenamiento, resultando en clasificaciones incorrectas. Este problema suele surgir cuando el modelo es demasiado simple para el tipo de datos que se desea clasificar. Una posible solución es aumentar la complejidad del modelo, por ejemplo, añadiendo más capas de neuronas, lo que puede mejorar su capacidad para abstraer.

Buen ajuste

Un buen ajuste se da cuando el modelo logra adaptarse de manera correcta a los datos de entrenamiento y proporciona clasificaciones precisas. Asimismo, un modelo bien ajustado es capaz de generalizar, lo que significa que puede clasificar con alta probabilidad nuevos datos que no haya visto previamente.

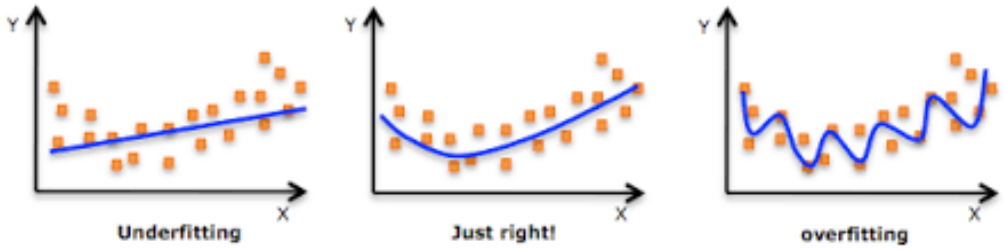


Figura 15 Diferentes resultados de un entrenamiento.

2.3.2 Precisión

La precisión es la métrica que nos indica la capacidad del modelo de predecir verdaderos positivos respecto a todos los positivos (5).

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

2.3.3 Recall

También llamada sensibilidad, es la capacidad del modelo de predecir los verdaderos positivos. El *recall* mide la proporción de ejemplos correctamente clasificados como positivos entre todos los ejemplos que son realmente positivos (6).

$$Recall = \frac{TP}{TP+FN} \quad (6)$$

2.3.4 F1-Score

El *F1-Score* es la media armónica entre la *precision* y el *recall*. Esta métrica se utiliza como un equilibrio entre *precision* y *recall* (7), especialmente cuando hay una necesidad de balancear ambas o cuando el conjunto de datos es desbalanceado.

$$F1 = 2x\frac{precision \times recall}{precision+recall} \quad (7)$$

El *F1-Score* es útil si queremos tener en cuenta tanto los falsos positivos como los falsos negativos. Un *F1-Score* alto significa que el modelo tiene tanto una alta *precision* como un alto *recall*, lo que indica que el modelo es capaz de identificar correctamente la mayoría de los ejemplos positivos y que no comete muchos errores al identificar ejemplos negativos y positivos.

2.3.5 Matriz de confusión

Una matriz de confusión es una tabla de doble entrada que compara las predicciones del modelo con los resultados reales (Figura16). La tabla muestra cómo de bien el modelo ha clasificado los ejemplos de cada clase, proporcionando una visión clara de los errores cometidos y aciertos logrados por el modelo [21].

La matriz de confusión es una herramienta de evaluación fundamental en el aprendizaje automático y, más específicamente, en problemas de clasificación. Es una tabla que se utiliza para describir el rendimiento de un modelo de clasificación en un conjunto de datos de prueba para el cual se conocen los valores verdaderos.

Figura 16 Matriz de confusión.

Los valores que se muestran en la matriz de confusión son:

- **VP**: el modelo predice un valor positivo y el verdadero valor es positivo.
- **VN**: el modelo predice un valor negativo y el verdadero valor es negativo
- **FP**: el modelo predice un valor positivo y el verdadero valor es negativo
- **FN**: el modelo predice un valor negativo y el verdadero valor es positivo
-

3 Implementación

3.1 Dataset

3.1.1 Área de estudio

El Parque Natural de la Albufera está situado en la costa del Golfo de Valencia, en el este de España (39◦17′ N, 00◦20′ E), y tiene una superficie de 20.956 hectáreas. El parque cuenta con una gran laguna de aguas someras de 23,94 km2 llamada "L'Albufera", que es alimentada por arroyos, ríos y acequias, y es uno de los humedales costeros más representativos y valiosos de la Comunidad Valenciana y de la cuenca mediterránea

L'Albufera fue declarada Parc Natural en 1986, y desde 1989 está reconocida como "Humedal de importancia Internacional", figura derivada de la "Convención Relativa a los Humedales de Importancia Internacional, especialmente como Hábitat de Aves Acuáticas", celebrada en Ramsar (Irán) el 2 de febrero de 1971 [22].

Además, es parte integrante de la Red Natura 2000, al haber sido declarada como "Zona de especial protección de las Aves" (ZEPA) en 1990 y seleccionado como "Lugar de Importancia Comunitaria" (LIC) desde 2006. Además, algunas partes de su ámbito han sido también declaradas como "Microrreserva de Flora" y como "Reserva de Fauna".

A pesar de que hay 45 especies animales en peligro de extinción en la zona, muchas de ellas aves acuáticas, la caza está permitida en áreas restringidas dentro del parque natural. La temporada de caza del año de las grabaciones empezó el 13 de noviembre de 2022 y finalizó el 12 de febrero de 2023, y solo se permitió la caza los sábados, domingos y festivos, excepto durante la tercera semana de enero, cuando también se permitió la caza entre semana. En este sentido, la detección automática de disparos es muy importante para el personal técnico del parque para:

• Detección de cazadores furtivos en días prohibidos.

• Localización de la caza en las proximidades de los parques naturales de aves de la Albufera.

3.1.2 Técnicas de grabación

Las grabaciones utilizadas para crear la base de datos fueron obtenidas mediante diez nodos acústicos distribuidos en la zona de la laguna, cuyos emplazamientos exactos se muestran en la Figura 17. Cinco de estos nodos se colocaron en islas dentro de la laguna: Mata de l'Antina, Mata de Sac Roc (donde se instalaron dos nodos), La Maseguerota y Mata del Fang. Los otros cinco nodos se situaron en los terrenos que rodean la laguna: dos en el Tancat de Milia, otros dos en La Tancadeta y uno en la oficina técnica del parque en El Palmar. Estos nodos acústicos, fabricados por la empresa Wildlife Acoustics, son dispositivos comerciales accesibles únicamente al personal designado. En las ubicaciones con dos nodos, hay una distancia de 80 metros entre ellos.

Un parámetro clave para el análisis de los audios recopilados es la frecuencia de muestreo con la que operan los nodos, que en este caso es de Fs=24000 Hz [23].

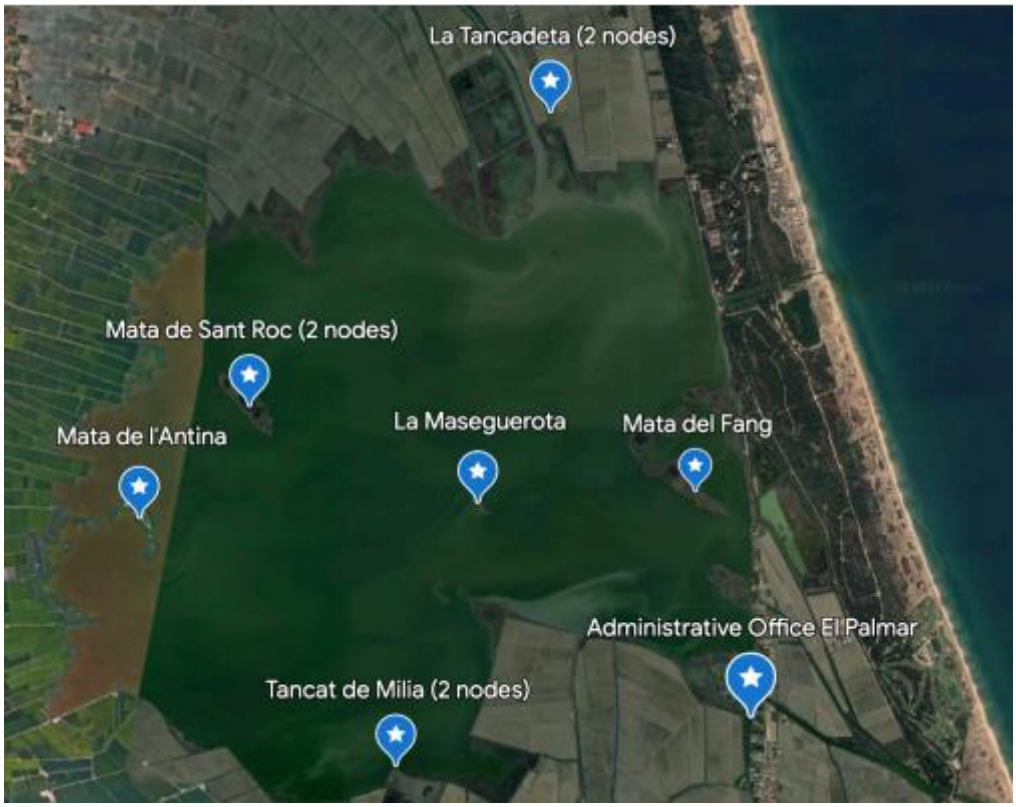


Figura 17 Mapa de L'Albufera y posición de los nodos.

El objetivo de instalar estos dispositivos es monitorizar y estudiar el comportamiento de las aves en el entorno de la Albufera. Por este motivo, los nodos fueron programados para grabar antes y después del amanecer y el atardecer, momentos en los que las aves suelen presentar mayor actividad. No obstante, los nodos no solo capturan sonidos de aves, sino también otros eventos sonoros del entorno, como ruidos de vehículos, personas hablando o disparos. Actualmente, contamos con grabaciones desde junio de 2022 hasta junio de 2023, almacenadas en archivos de audio en formato .wav, con una duración de 30 o 60 minutos cada uno (Tabla 2).

3.2.3 Dataset inicial

El dataset inicial está formado por 6 audios de 30 minutos en diferentes periodos a las 8:30 am y otros 6 audios de una hora que se graban a las 7:30 am. Estos audios contienen numerosos sonidos de diferente naturaleza. Para este trabajo se han elegido grabaciones en periodo de caza de modo que podamos extraer fragmentos que contengan disparos a diferentes distancias.

Para la extracción de datos se ha usado el programa *Audacity,* con el que se han extraído manualmente fragmentos de tres segundos que contienen disparos y otros que no contienen. Para ello se ha usado la herramienta de *Espectrograma* que tiene el programa para detectar estos

disparos. Estos fragmentos se han usado para entrenar ambos modelos. La Tabla 1 muestra el conjunto de muestras de nuestro dataset y las muestras que componen cada clase.

Clase	Muestras
background	635
lejos	933
medio	697
cerca	241

Tabla 1 Set de muestras de cada clase.

En las Figuras 18-21 se muestran ejemplos de los espectrogramas de cada clase, podemos ver como la diferencia entre las Figuras 20-21 son pocas, es por ello que el modelo podría confundirlas como veremos más adelante.

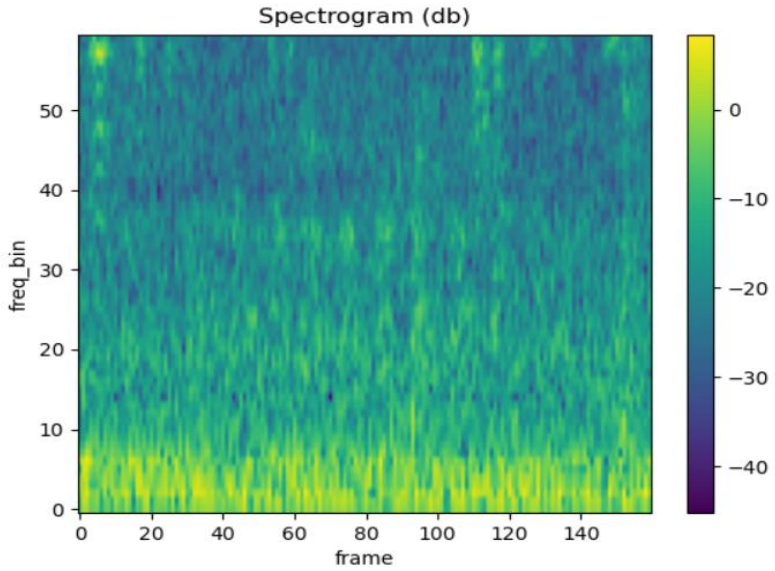


Figura 18 Espectrograma de la clase background

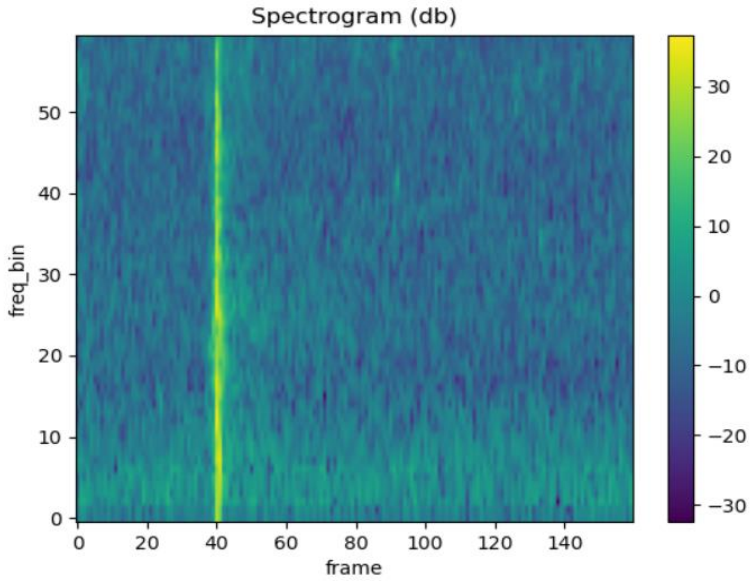
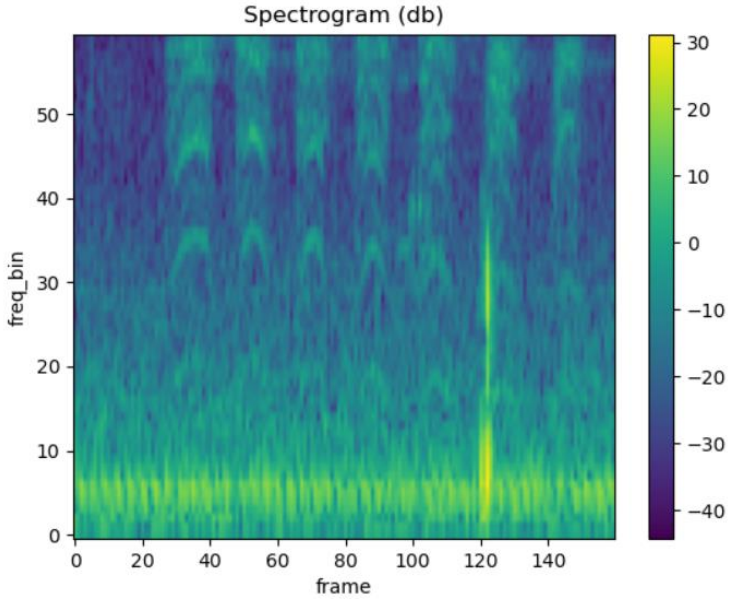


Figura 19 Espectrograma de la clase cerca.

Figura 20 Espectrograma de la clase medio.



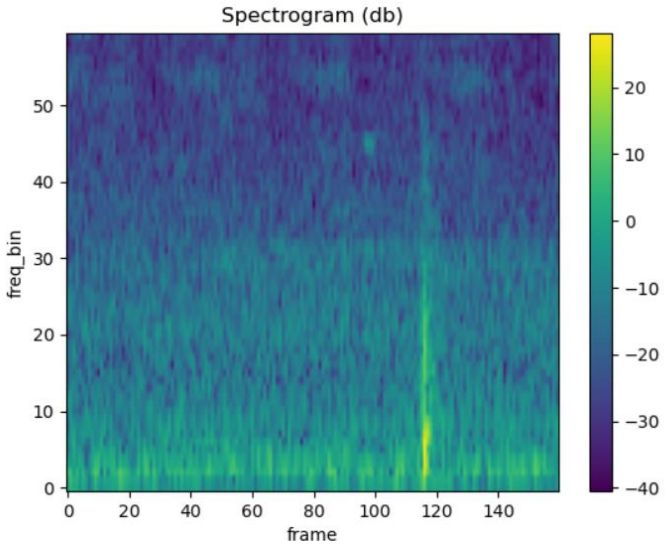


Figura 21 Espectrograma de la clase lejos.

Una segunda fase de la obtención de datos ha sido usar etiquetas fuertes en los audios completos, anotando los instantes en los que se producen disparos para posteriormente usarlos para las pruebas de test [23]. Con este proceso lo que buscamos es hacer un modelo end to end que permita introducir un audio completo y sea capaz de detectar los disparos en cada instante a tiempo real. Originalmente contamos con las etiquetas de los audios de 30 minutos, nuestro trabajo es etiquetar los nuevos audios y aumentar el dataset (Tabla 2). Los audios para esta segunda fase son los siguientes:

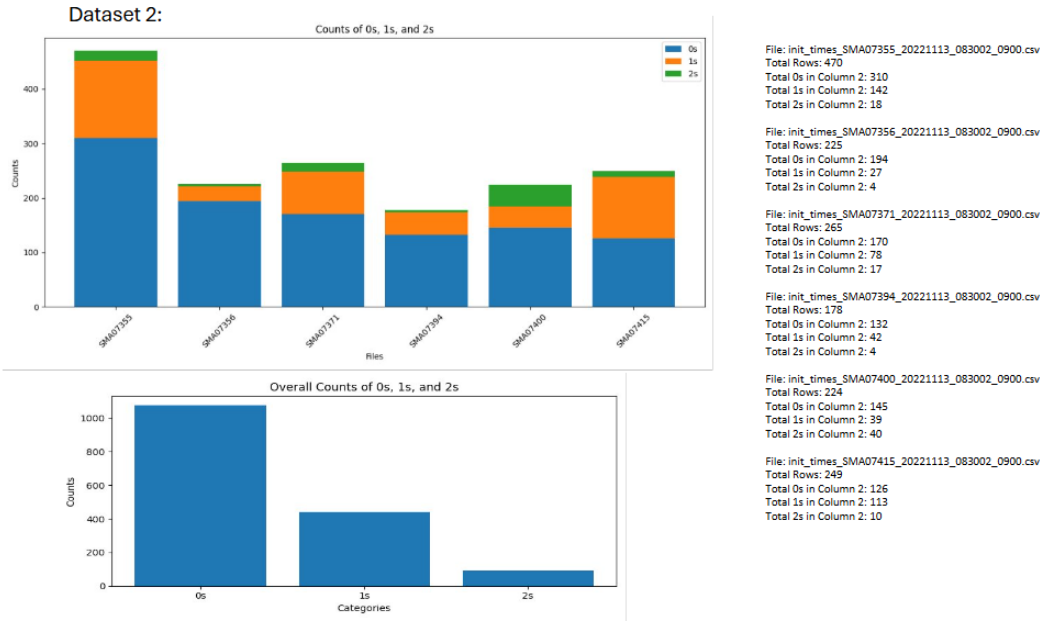
Nombre del archivo	Duración
SMA07355_20221113_083002_0900.wav	30 minutos
SMA07355_20221113_073002.wav	60 minutos
SMA07356_20221113_083002_0900.wav	30 minutos
SMA07356_20221113_073002.wav	60 minutos
SMA07400_20221113_083002_0900.wav	30 minutos
SMA07400_20221113_073002.wav	60 minutos
SMA07415_20221113_083002_0900.wav	30 minutos
SMA07415_20221113_073002.wav	60 minutos
SMA07371_20221113_083002_0900.wav	30 minutos
SMA07371_20221113_073002.wav	60 minutos
SMA07394_20221113_083002_0900.wav	30 minutos
SMA07394_20221113_073002.wav	60 minutos
SMA07370_20221120_083002.wav	60 minutos

Tabla 2 Dataset formado por las grabaciones.

Tras etiquetar todos los disparos y guardarlos en archivos .csv, hemos generado 3 datasets que se muestran en las Figuras 22-24 y que contienen todos los disparos de las diferentes grabaciones:

- **Dataset 2**: Dataset formado por 6 audios de 30 minutos cada uno. Tiene un total de 1611 disparos. En este dataset predomina la clase lejos, con más de 1000 muestras.
- **Dataset 3**: Dataset formado por 6 audios de 60 minutos cada uno. Tiene 1366 y presenta un porcentaje más equilibrado entre disparos lejanos y a media distancia
- **Dataset total**: formado por los 2 datasets anteriores. Ambos datasets presentan un total de 2980 muestras, donde más de la mitad de los disparos son lejanos. El desbalanceo es evidente y limitará nuestro trabajo a la hora de entrenar.

Figura 22 Dataset inicial y el número de muestras.



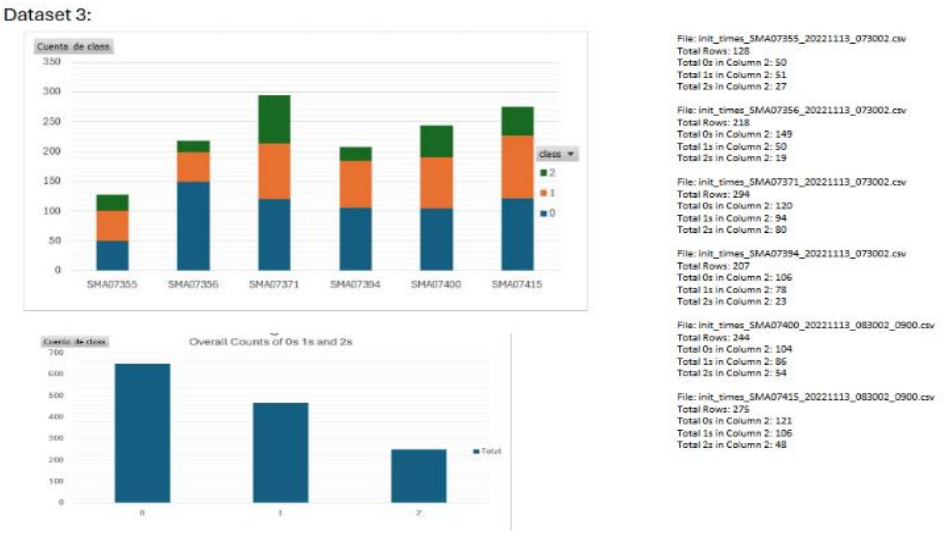


Figura 23 Dataset nuevo y el número de muestras.

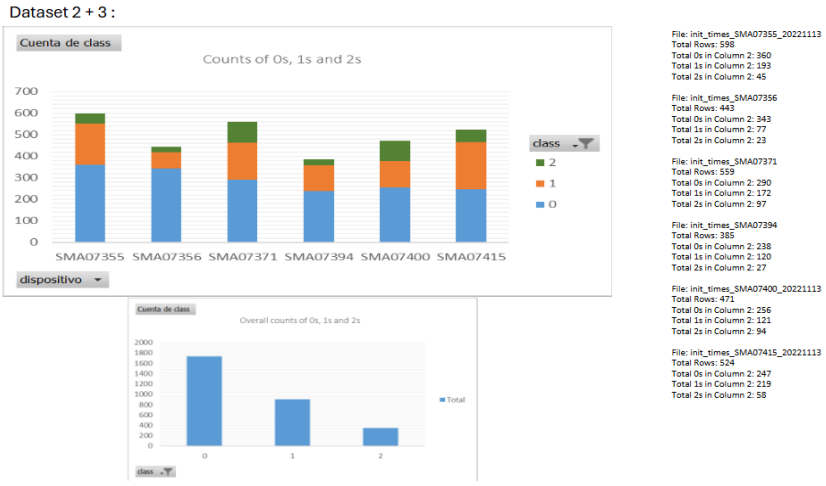


Figura 24 Dataset total y muestras totales.

3.2 Hardware

Las tareas de *Deep Learning* requieren una considerable potencia computacional. Como se ha mencionado, un modelo puede tener millones de parámetros, y cada ajuste puede cambiar significativamente el resultado final. Por esta razón, los requisitos de hardware son bastante elevados. Tanto la Central Proccesing Unit (CPU) como la Graphic Proccesing Unit (GPU) son los encargados de realizar los cálculos matemáticos necesarios. Tener una GPU potente en el equipo puede reducir drásticamente el tiempo de entrenamiento de la red y permitir la gestión de redes más grandes y complejas. Además, muchos frameworks modernos están optimizados para aprovechar la GPU, lo que mejora la eficiencia en el diseño de modelos. Para nuestro trabajo, utilizaremos un portátil personal con las características que se detallan en la Tabla 3. Dado que se trata de un portátil de uso académico, no cuenta con una tarjeta gráfica dedicada. Por lo tanto, emplearemos la CPU para nuestras tareas. Esto puede tener un impacto negativo en los tiempos de procesamiento y entrenamiento, haciéndolos más largos en comparación con el uso de una GPU. Sin embargo, una ventaja de nuestro portátil es que está equipado con un disco SSD, lo que puede mejorar el rendimiento general.

Las memorias SSD son más eficaces que las memorias HDD puesto que acceden más rápido a los datos. Esta memoria nos permitirá ejecutar tareas más rápidamente, lo que nos puede ahorrar tiempo y ser más eficientes a la hora de realizar el modelo y redactar la memoria.

Componente	Modelo
CPU	Intel Core i3 1115G4 3GHz
GPU	Integrada
RAM	8 GB
SDD	500 GB
SO	Microsoft Windows 11

Tabla 3 Especificaciones del ordenador usado para el trabajo.

3.3 Software

Se ha optado por Python como lenguaje de programación para el desarrollo del trabajo debido a su amplia popularidad y versatilidad. En los campos de Inteligencia Artificial, Machine Learning y Deep Learning Python es el lenguaje predominante, gracias a su facilidad de uso y comprensión. Su sintaxis intuitiva y la posibilidad de aprenderlo de manera autodidacta, especialmente para quienes ya tienen conocimientos en otros lenguajes como Java, lo convierten en la opción preferida.

Entorno de desarrollo y librerías

Existen numerosas opciones para comenzar a programar en Python. Al estar familiarizado con ella, utilizamos *Anaconda*, desarrollada por *Continuum Analytics*. Elegimos esta opción y la configuramos para trabajar con la versión 3.11.7 de Python, la más actual. Para diseñar nuestro trabajo utilizamos el entorno de programación *Visual Studio Code*. Anaconda incluye una amplia variedad de librerías preinstaladas, de las cuales utilizamos las siguientes para nuestro trabajo:

Seaborn

Se basa en Matplotlib para la visualización de datos que proporciona una interfaz de alto nivel para crear gráficos estadísticos visualmente atractivos y fáciles de interpretar. Seaborn está diseñada para trabajar bien con estructuras de datos como las de Pandas y facilita la creación de gráficos complejos con pocas líneas de código.

Torchaudio

Esta librería del ecosistema de PyTorch está diseñada para el procesamiento de audio. Esta permite a los desarrolladores manejar tareas de procesamiento de señales de audio, como la carga, transformación y manipulación de datos de audio. Es útil para tareas de Deep Learning relacionadas con audio, como el reconocimiento de voz o la clasificación de sonidos.

Matplotlib

Es una de las librerías de visualización de datos más popular en Python. Es muy potente y flexible, permitiendo a los desarrolladores crear gráficos de calidad en diferentes formatos y estilos. Matplotlib es la base sobre la cual se construyen muchas otras librerías como ya hemos visto anteriormente.

Sklearn

Se trata de una de las librerías más populares y usadas para el aprendizaje automático en Python. Proporciona herramientas para el análisis de datos, y es ampliamente utilizada en la comunidad científica y de investigación.

Pandas

Es una librería fundamental para el análisis de datos en Python. Proporciona estructuras de datos de alto rendimiento (DataFrames y Series) y herramientas de análisis de datos que facilitan la manipulación y el análisis de grandes conjuntos de datos tabulares.

Librosa

Es una librería de Python especializada en análisis y procesamiento de señales de audio. Es ampliamente utilizada en aplicaciones de análisis de música, donde es necesario extraer características del audio para tareas de Machine Learning y otras aplicaciones.

Numpy

Proporciona soporte para arrays y matrices de gran tamaño, junto con una colección de funciones matemáticas para operaciones rápidas y eficientes en estos arrays.

Aunque parezca que algunas de estas librerías son muy similares, todas contribuyen a realizar una función específica en nuestro código. Cada herramienta de las librerías tiene un propósito y todas ellas se complementan para obtener los resultados que queremos.

3.4 Framework

3.4.1 Pytorch

PyTorch es una biblioteca *open source* desarrollada por *Facebook's AI Research lab* (FAIR). Es muy utilizada en investigación y desarrollo por su fácil uso y flexibilidad, especialmente en tareas de Deep Learning.

PyTorch es una biblioteca muy eficaz en el campo del aprendizaje automático, destacada por su capacidad de procesamiento en GPU y su enfoque en grafos computacionales dinámicos. Con características avanzadas como Autograd y DataLoaders, PyTorch permite a los desarrolladores construir, entrenar y experimentar con modelos complejos de manera eficiente. Su versatilidad y ecosistema lo convierten en una elección preferida tanto en la investigación académica como en aplicaciones más comerciales. Las principales características de Pytorch son:

Tensores:

PyTorch maneja los datos mediante estructuras llamadas tensores, que son similares a los arrays de NumPy pero que permiten más funcionalidades, como la posibilidad de ejecutarse en GPUs. Esto permite realizar cálculos matematicos más rápido y de manera más eficiente.

Computación en GPU:

PyTorch facilita el uso de GPUs para acelerar el entrenamiento de modelos. Con un simple comando, los tensores y operaciones pueden ser movidos a la GPU, lo que mejora significativamente la velocidad de los cálculos, especialmente en redes neuronales profundas.

Dataloaders:

Los **DataLoaders** en PyTorch son una herramienta fundamental para manejar y procesar datos. Permiten cargar conjuntos de datos de manera eficiente y flexible, gestionando tareas como el preprocesamiento, el manejo de lotes (batching), y el barajado (shuffling). PyTorch proporciona una clase de Dataset, que se puede personalizar para definir cómo se acceden y se transforman los datos. Luego, DataLoader se encarga de iterar sobre estos datos, suministrándolos al modelo en lotes, lo que es crucial para optimizar el rendimiento durante el entrenamiento. Los DataLoaders soportan el uso de múltiples procesos paralelos para cargar los datos, lo que es especialmente útil para grandes conjuntos de datos y entrenamientos en GPU, donde mantener un flujo constante de datos puede ser un desafío.

Ecosistema:

PyTorch cuenta con bibliotecas como TorchVision (para tareas de visión por computadora), TorchText (para procesamiento de lenguaje natural), y TorchAudio (para procesamiento de audio). Estas bibliotecas proporcionan datasets, modelos preentrenados y funciones especializadas que aceleran el desarrollo de modelos.

Aplicaciones de PyTorch

PyTorch se utiliza en una amplia gama de aplicaciones de aprendizaje automático, que incluyen:

- **Visión por Computadora**
  Desde clasificación de imágenes hasta detección de objetos y segmentación, usando CNNs.

- **Procesamiento de Lenguaje Natural (NLP)**
  Implementación de modelos como GPT para tareas como traducción automática, análisis de sentimientos y generación de texto.

- **Aprendizaje por Refuerzo**
  Desarrollo de agentes que interactúan con su entorno para aprender y mejorar su rendimiento a través de la retroalimentación.

- **Investigación en Inteligencia Artificial**
  PyTorch es una herramienta preferida para la creación de nuevas arquitecturas y algoritmos en IA debido a su flexibilidad y capacidad de experimentación rápida.

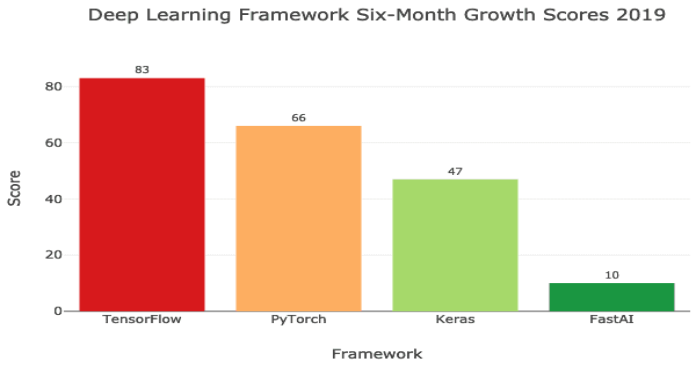


Figura 25 Evolución del uso de los diferentes Frameworks.

PyTorch ha sido adoptado rápidamente en la investigación académica debido a su flexibilidad, y está ganando popularidad en la industria, donde se valora por su facilidad de uso y su integración con herramientas de despliegue. Es por ello que hemos optado por usarlo en este trabajo.

3.5 Preprocesamiento de datos

Esta es una parte fundamental de nuestro trabajo, ya que, sin unos datos bien procesados, el modelo no podría entrenar de manera correcta. Nuestro objetivo es calcular el espectrograma de los extractos de audio y que estos sean los datos de entrada de nuestro modelo. Para ello todos los extractos de audio deben tener la misma longitud, ya que la longitud es un parámetro del espectrograma y podría afectar al tamaño, por eso, a la hora de obtener las muestras usamos extractos de 3 segundos de duración.

Finalmente, debemos definir la función de procesamiento y crear las matrices con el tamaño y formato que la red neuronal requiere. Los datos de entrada tendrán una dimensión de 1x60x160, y cada dato estará asociado con una etiqueta que puede ser un valor entre 0 y 4. Tanto el espectrograma como su etiqueta se introducen como entrada al modelo. Un paso crucial antes de comenzar el entrenamiento de la red neuronal es la normalización de los datos. Este proceso es esencial porque no solo estandariza los valores de los datos, sino que también hace que sean más manejables desde el punto de vista computacional.

Para el entrenamiento y validación utilizaremos la herramienta de Pytorch llamada DataLoader para dividir nuestros datos en un porcentaje de 85% para entrenamiento y 15% para validación, además el *shuffle* en True.

```
batch_size=64
train_size = int(0.85 * len(MDA))
val_size = len(MDA) - train_size
train_dataset, val_dataset = random_split(MDA, [train_size, val_size])

train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(dataset=val_dataset, batch_size=batch_size, shuffle=True)

```

Figura 26 Dataloader para la división del dataset.

Nuestra primera clase *MyAudioData* (Figura 27) se usa para extraer los espectrogramas, funciona ajustando los parámetros de la ventana del espectrograma y añadiendo un archivo csv con la ruta de los extractos de audio junto a su clase.

```python
if __name__=="__main__":
    FILE_CSV = './DATA/DATASET_FOLD2.csv'
    win_length = 900
    n_fft = 1024
    crop = 60
    ref = 1.0
    norm = True

    MDA = MyDataAudio(annots_file=FILE_CSV,
                      crop=crop, ref=ref, norm=norm, win_length=win_length, n_fft=n_fft)

    meldB, lab = MDA[1402]
```

Figura 27 Clase MyDataAudio para obtener los espectrogramas.

Para el test tenemos un audio de 60 minutos totalmente nueva que la red no ha visto. Es por ello por lo que necesitamos utilizar una nueva clase para preprocesar el audio entero. La nueva función *MyAudioDataFile* utiliza una ventana deslizante que calcula el espectrograma de cada segundo del audio, asignándole una etiqueta previamente anotada en un archivo csv con el instante temporal que corresponde a esa etiqueta (ver Anexo).

A la clase se le añaden los parámetros como en la de la Figura 27, pero la diferencia está en que ahora debemos pasarle el archivo de audio entero y nuestro csv con las etiquetas temporales (Figura 28).

```python
if __name__=="__main__":
    FILE_CSV2 = 'DATA/init_times_SMA07370_221120_073002.csv'
    FILE_AUDIO2 = 'DATA/SMA07370_20221120_073002.wav'
    win_length = 900
    n_fft = 1024
    crop = 60
    ref = 1.0
    convertir_stereo_a_mono('DATA/SMA07370_20221120_073002.wav')
    norm = True
    MDA = MyDataAudioFile(annots_file=FILE_CSV2, audio_file=FILE_AUDIO2,
                          crop=crop, ref=ref, norm=norm, win_length=win_length, n_fft=n_fft)
```

Figura 28 Clase MyAudioDataFile para el test.

3.6 Descripción de la arquitectura del modelo y sus componentes

3.6.1 VGG16

Las redes VGG son un tipo de redes neuronales convolucionales, que marcaron un hito en el campo del reconocimiento de imágenes. Estas redes se destacan por su simplicidad en el diseño, aprovechando la profundidad como un factor clave para mejorar la precisión en tareas de clasificación de imágenes a gran escala. [24]. La característica principal de las redes VGG es su profundidad. La versión VGG-16 tiene 16 capas de profundidad. Estas capas están compuestas principalmente por capas convolucionales con filtros muy pequeños de 3x3 píxeles, que se aplican de manera repetitiva a lo largo de la red. Las capas convolucionales son seguidas por capas de *max pooling* (de 2x2 píxeles), que reducen la dimensionalidad espacial de las representaciones intermedias, lo que ayuda a controlar el número de parámetros y a reducir el sobreajuste.

Las redes VGG lograron resultados sobresalientes en el desafío ImageNet Large Scale Visual Recognition Challenge (ILSVRC) de 2014, alcanzando una precisión top-5 superior al 92%. Este rendimiento, combinado con la simplicidad de la arquitectura, hizo que VGG se convirtiera rápidamente en un modelo de referencia en el ámbito de la visión por computadora.

Más allá de su rendimiento en tareas de clasificación, las redes VGG demostraron ser extremadamente valiosas como modelos preentrenados para otras tareas de visión por computadora, como la detección de objetos y la segmentación de imágenes. Esto se debe a su capacidad para capturar representaciones robustas y transferibles de las imágenes, que pueden ser reutilizadas en otros contextos con una cantidad mínima de ajuste

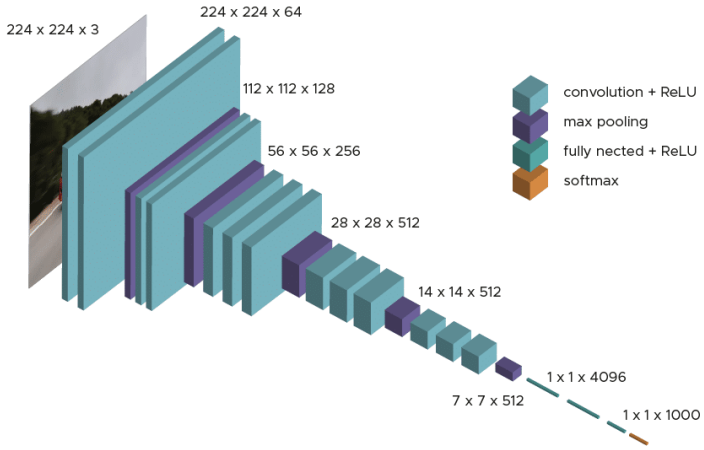


Figura 29 Arquitectura de VGG-16.

3.6.2 RESNET18

ResNet-18 es una arquitectura CNN que forma parte de la familia de ResNet (Redes Residuales), [25]. ResNet-18 es una versión con 18 capas de profundidad, diseñada para abordar el problema de degradación del rendimiento en redes muy profundas. A medida que las redes neuronales se hacen más profundas, se espera que puedan aprender características más complejas y, por lo tanto, mejorar su precisión. Sin embargo, en la práctica, las redes muy profundas a veces experimentan un problema llamado "degradación". Esto significa que, en lugar de mejorar, el rendimiento comienza a disminuir o se estanca, incluso cuando se agregan más capas. Este problema no se debe a un sobreajuste, sino a la dificultad de entrenar redes tan profundas. Para superar este desafío, ResNet introduce la idea de bloques residuales. Los bloques residuales permiten que la red aprenda una función residual. Esto significa que la red intenta aprender la diferencia (residuo) entre la entrada y la salida esperada. Cada bloque residual tiene una conexión de "atajo" o "skip connection". Esto facilita el entrenamiento porque, si las capas adicionales no mejoran el rendimiento, la red puede fácilmente omitirlas, simplemente aprendiendo una función residual cercana a cero. ResNet-18 y sus variantes más profundas, demostraron un rendimiento excepcional en conjuntos de datos como ImageNet, superando a modelos anteriores.

Otra ventaja significativa de ResNet-18 es su eficiencia. Al ser menos profundo que sus contrapartes (como ResNet-50), es más rápido de entrenar y requiere menos recursos computacionales, lo que lo hace adecuado para aplicaciones que necesitan un buen rendimiento con limitaciones de hardware

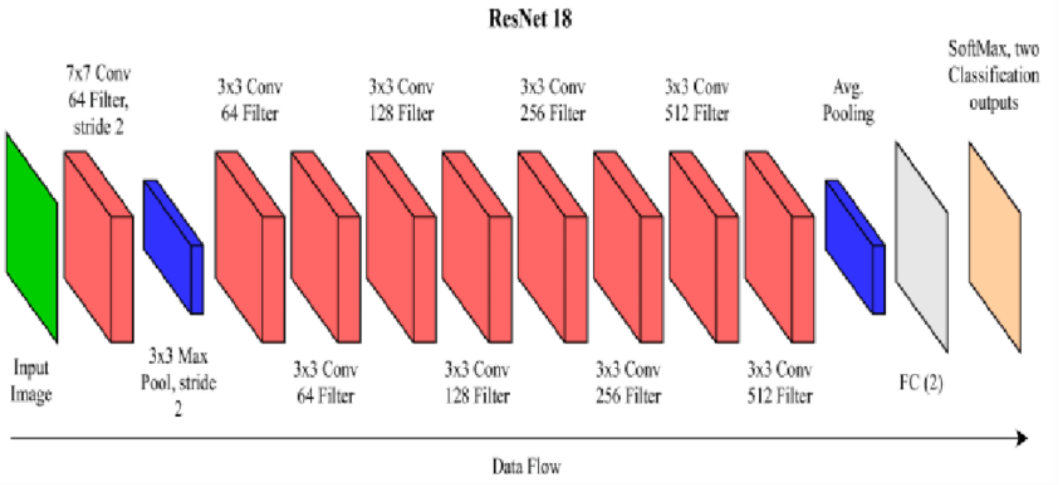


Figura 30 Arquitectura de ResNte18

Generación de soluciones creativas

Para nuestro trabajo, hemos optado por dividir nuestro sistema de clasificación de disparos en dos partes. La primera parte consiste en crear una base de datos con extractos de tres segundos que contienen las diferentes clases. Se ha diseñado una función de extracción de características para preprocesar los datos y poder introducirlos a la red de entrenamiento en forma de espectrograma.

Una segunda parte es la parte de test, se ha creado otra función similar a la primera que extrae las características de una grabación completa mediante una ventana deslizante, obteniendo el espectrograma en cada segundo. Esta función genera una gráfica con las predicciones de la red entrenada y los disparos reales, anotados previamente en un archivo csv, de modo que obtenemos una prueba en un entorno real.

Para las redes hemos elegido dos DCNN para comparar sus rendimientos, ambas han demostrado ser redes potentes y eficaces para tareas de detección de sonidos y clasificación de audio.
.

4 Resultados

4.1 Resultados del modelo VGG16

4.1.1 Entrenamiento y validación

Nuestra primera red es una VGG-16 con 13 capas convolucionales. Decidimos entrenar el modelo durante 40 épocas, debido a que el proceso de entrenamiento es relativamente lento por iteración. Si hubiéramos incrementado el número de épocas, el error de validación habría empezado a aumentar, lo que implicaría una disminución en la precisión y un posible sobreajuste. Usamos un batch size de 64, lo que significa que el modelo procesa los datos en lotes de 64 muestras. Además, aplicamos un dropout de 0.6 para mitigar el sobreajuste, y configuramos una tasa de aprendizaje (learning rate) de 0.001 para acelerar las iteraciones. Con estos parámetros definidos, iniciamos el entrenamiento, pasando tanto los datos de entrenamiento como los de validación. Después de tres horas de entrenamiento, analizamos las gráficas que muestran la evolución del modelo, visibles en la Figura 31.

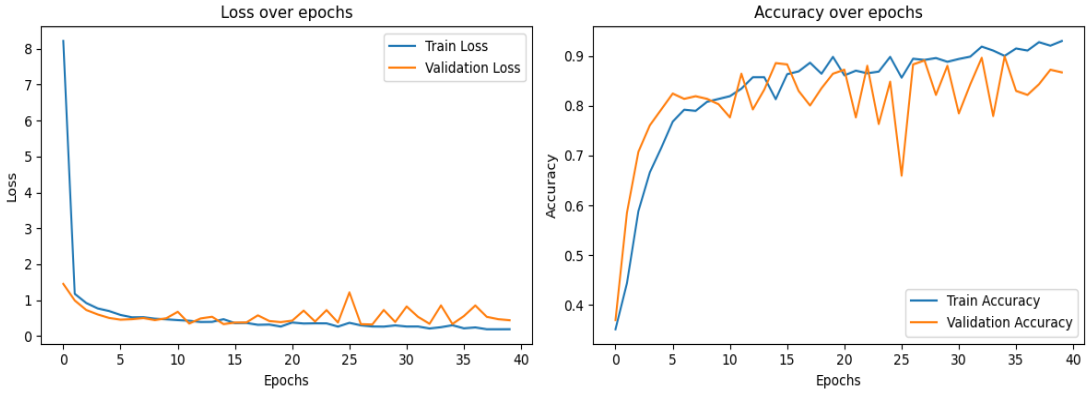


Figura 31 Métricas de Accuray y loss de VGG-16.

Los resultados del entrenamiento y validación obtenidos son bastante buenos a primera vista (Figura 31), con una precisión de validación del 86,74% y una pérdida del 44%. En cuanto al *recall*, que es la métrica más importante en este problema de clasificación, se obtienen resultados bastante buenos (86,7%), sin embargo, la Tabla 4 muestra que el *recall* de la clase *cerca* es algo menor a la del resto, esto puede explicarse debido a que es la clase menos representativa de todas, con tan solo 38 muestras en la validación. No obstante, el resultado general es bastante aceptable.

	precision	recall	F1-score	support
background	**0.94**	**0.99**	**0.96**	**93**
lejos	**0.85**	**0.87**	**0.86**	**134**
medio	**0.80**	**0.80**	**0.80**	**111**
cerca	**0.94**	**0.76**	**0.84**	**38**

Tabla 4 Metricas de validación de VGG-16.

Matriz de confusión

El resultado de la matriz de confusión nos permite ver como nuestra red ha clasificado los datos de validación (Figura 32). Esta red clasifica bien en general todas las muestras, pero hay un número mayor de error detectando disparos lejanos como medios y viceversa. Esto se debe a que hay más muestras de estas clases que de la de disparos cercanos, sin embargo, en porcentaje el error es menor. En conclusión, podemos decir que la red clasifica bien todas las clases.

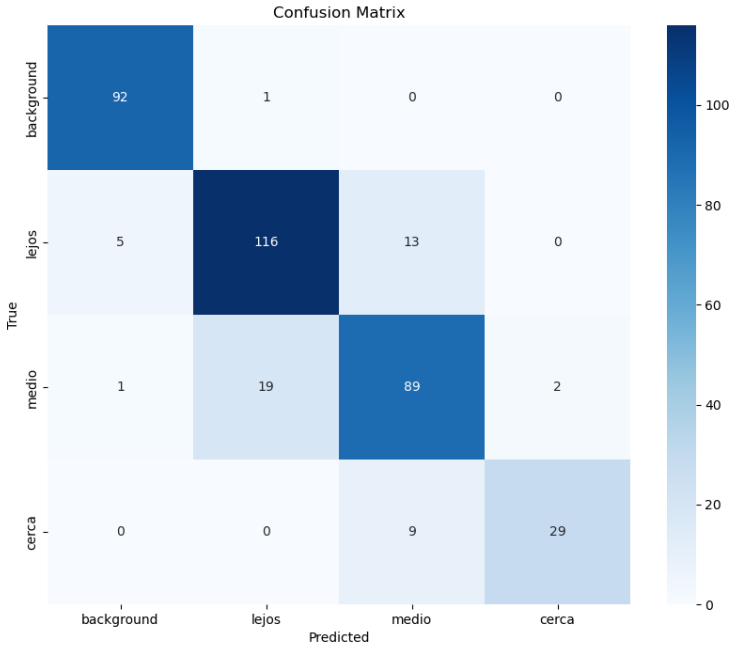


Figura 32 Matriz de confusión de VGG-16.

4.1.2 Métricas de test

Para hacer el test se ha usado un audio de una hora de duración totalmente nuevo para la red. Que cuenta con 102 disparos etiquetados. Se ha hecho una prueba en condiciones de entorno real (Figura 33) y se han obtenido los resultados que aparecen en la Tabla 5. La precisión y las pérdidas de test son de 92% y 70% respectivamente. Al observar las métricas de recall de la Tabla 5, vemos que los peores resultados son del 43% para lejos y del 57% para medio.

	precision	recall	F1-score	support
background	**0.99**	**0.93**	**0.96**	**3493**
lejos	**0.15**	**0.43**	**0.22**	**69**
medio	**0.20**	**0.57**	**0.25**	**23**
cerca	**0.29**	**0.80**	**0.42**	**10**

Tabla 5 Métricas de test de VGG-16.

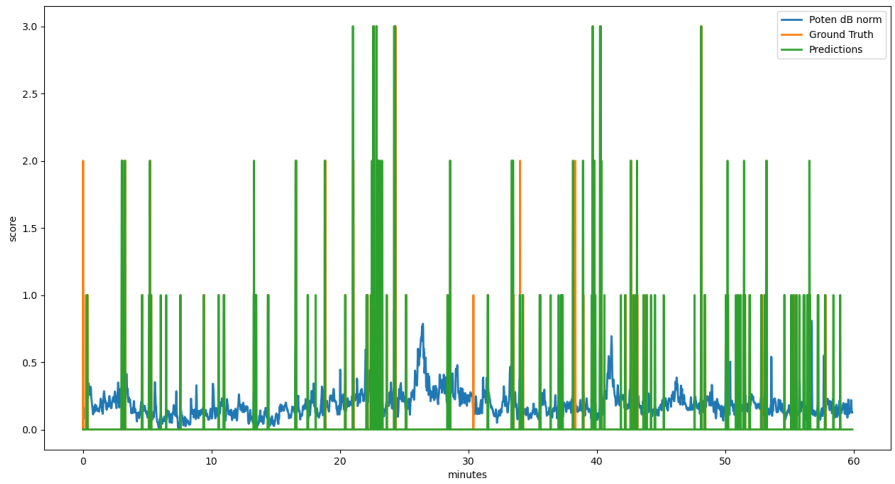


Figura 33 Grafica con los resultados de test de VGG-16.

4.2 Resultados del modelo ResNet18

4.2.1 Entrenamiento y validación

Nuestro segundo modelo es una ResNet18 preentrenada de 18 capas convolucionales y con unos pesos iniciales preestablecidos. Dado que tiene más parámetros que ajustar que el primer modelo, la entrenaremos con menos épocas. Hemos seleccionado 35 épocas y un learning rate de 0.01, debido a que seleccionar un valor más pequeño, la red convergía muy rápido y producía sobreajuste. Respecto al tamaño de batch, se han seleccionado lotes de 64 puesto que ha sido el tamaño que mejor rendimiento ha dado entre todos los probados. Ajustando un dropout de 0.6 ya tenemos todos los parámetros necesarios para entrenar la red. Al estar optimizada con pesos predeterminados, esta red ha tardado solo 45 minutos en completar el entrenamiento. Los resultados obtenidos se ven en la Figura 34.

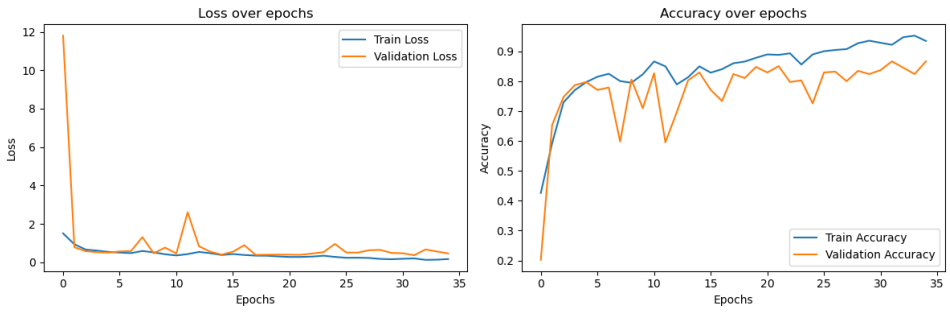


Figura 34 Metricas de accuracy y los de ResNet18.

Como vemos en la Figura anterior (Figura 34), los resultados del entrenamiento y validación son muy prometedores, con una precisión de validación del 86,81% y una pérdida del 45,99%. En cuanto al *recall*, que es la métrica más importante en este problema de clasificación, se obtienen resultados bastante buenos (86,7%), sin embargo, como era de esperar son menores para las clases *medio* y *lejos* como vemos en la Tabla 6.

	precision	recall	F1-score	support
background	**0.98**	**0.99**	**0.98**	**102**
lejos	**0.81**	**0.93**	**0.87**	**136**
medio	**0.83**	**0.68**	**0.75**	**100**
cerca	**0.92**	**0.87**	**0.89**	**38**

Tabla 6 Métricas de validación de ResNet18.

Matriz de confusión

La matriz de confusión (Figura 35) muestra que el modelo clasifica bien prácticamente todas las muestras, sin embargo, podemos ver que la clase medio es confundida en mayor medida. Esto se debe a que el límite en el que se considera un disparo cercano o mediano no está claro, por lo que a la hora de etiquetar los disparos puede haber diferentes criterios de los investigadores.

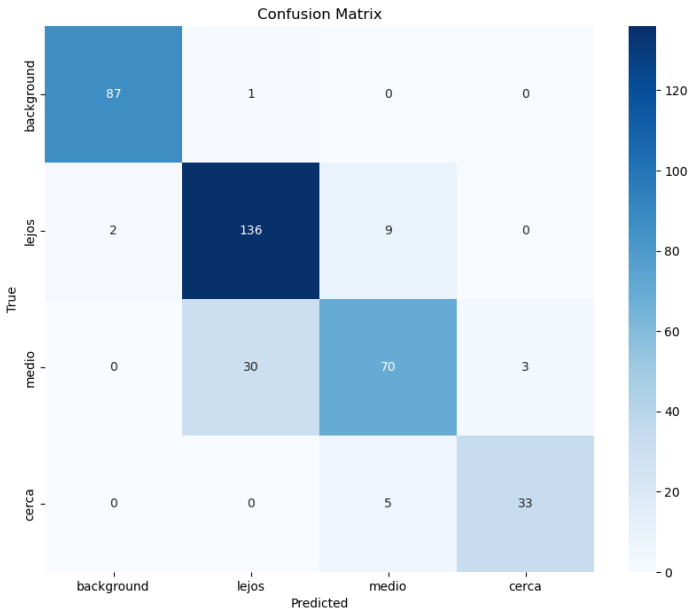


Figura 35 Matriz de confusión de ResNet18.

4.2.2 Métricas de test

4.2.2 Métricas de test

Para realizar el test bajo condiciones reales (Figura 36), se ha seleccionado el mismo audio de una hora de duración que la VGG-16. Esta red tampoco ha visto nuestro audio por lo que será totalmente nueva. El número de eventos es 3595 con 102 disparos y la red ha tenido un resultado de un 83% de precisión de test. Si nos fijamos en el recall, las clases lejos y medio son las que peor resultados presentan, un 45% y un 61% respectivamente.

	precision	recall	F1-score	support
background	**0.99**	**0.78**	**0.87**	**3493**
lejos	**0.04**	**0.45**	**0.08**	**69**
medio	**0.14**	**0.61**	**0.23**	**23**
cerca	**0.29**	**0.80**	**0.42**	**10**

Tabla 7 Métricas de test de ResNet18.

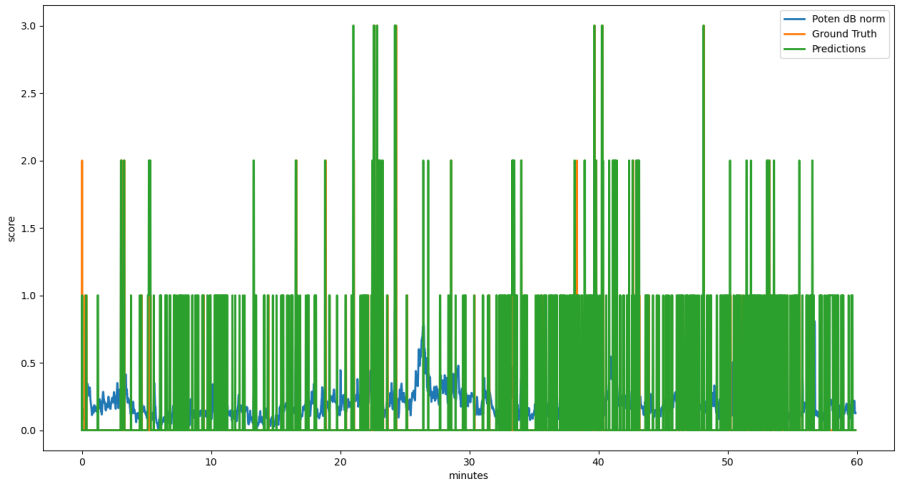


Figura 36 Gráfica con los resultados de test de ResNet18.

Evaluación de múltiples soluciones y toma de decisiones

Para este trabajo hemos utilizado dos DCNN, cada una tiene una arquitectura y unas características distintas, aunque ambas son muy utilizadas en el campo de la detección de sonidos. Las dos han resultado eficaces a la hora de detectar disparos, la VGG-16 ha demostrado ser mejor en el entrenamiento, aunque la diferencia de tiempo de entrenamiento ha sido considerablemente mayor. Debemos mantener un compromiso entre rapidez y precisión.

4.3 Comparación de modelos en el test

Comparando las pruebas de test y sus métricas de recall, que son las que verdaderamente nos dicen como de eficaces son nuestras redes a la hora de detectar disparos, ambas redes nos dan resultados similares. Se ve una ligera mejora respecto a trabajos previos [26] en entornos reales como bosques latinoamericanos o ciudades americanas. Ambos modelos presentan pequeñas variaciones, la más notable es que VGG-16 es mejor detectando los sonidos ambientales, mientras que ResNet18 clasifica mejor los disparos lejanos y medios (Tabla 8).

En general los dos modelos se comportan bien, hay que tener en cuenta que se han probado ambos modelos en pruebas en entornos reales, donde hay muchos factores externos que pueden confundir a la red, como ramas rompiéndose, ladridos de perro o el canto de algún pájaro. Por eso sería conveniente añadir más muestras de disparos al conjunto de datos, además de usar muestras de background con diferentes sonidos similares a disparos.

modelo	VGG16	ResNet18
background	0.93	0.78
lejos	0.43	0.45
medio	0.57	0.61
cerca	0.80	0.80

Tabla 8 Comparación del recall de los dos modelos.

Evaluación del cumplimiento de objetivos

Hemos comparado ambos modelos y el desempeño a la hora de realizar las tareas de clasificación y detección han sido bastante buenos, se ha mejorado el recall de trabajos anteriores y todavía se podrían mejorar más con una serie de ampliaciones en el dataset y técnicas de optimización del entrenamiento.

5 Conclusiones y futuras direcciones

5.1 Conclusiones

A lo largo de este trabajo, hemos aprendido que es la Inteligencia Artificial y como puede ayudarnos a resolver problemas que a priori parecen complejos. Desde la neurona, hasta una red compleja como las DCNN, hemos explicado detalladamente todo lo necesario para desarrollar este trabajo. También hemos visto qué son las señales de audio y como utilizar diferentes herramientas para transformarlas al dominio frecuencial y así poder utilizarlas como datos útiles.

Este trabajo se ha centrado en la clasificación de audios y en el estudio de unas grabaciones de sonido realizadas en el Parc de l'Albufera de Valencia, un área de gran valor ecológico en la que la detección precisa de disparos es esencial tanto para la protección de especies en peligro de extinción como para la regulación de actividades humanas, como la caza. El objetivo era desarrollar un clasificador supervisado capaz de detectar sonidos de disparos a diferentes distancias con dos modelos de DL basados en arquitecturas de DCNN como lo son VGG16 y ResNet18. Para ello se ha creado una base de datos totalmente nueva a partir de las grabaciones de los nodos, se han extraído y analizado una serie de características obtenidas de los espectrogramas de audios y se han clasificado.

Posteriormente se ha comparado su efectividad y se han puesto a prueba mediante una serie de tests en condiciones reales, mejorando ligeramente los trabajos previos en otros entornos como bosques o zonas urbanas [26]. Este avance no solo contribuye a la protección del medio ambiente al mejorar la capacidad de monitoreo y respuesta ante actividades ilegales, sino que también demuestra la capacidad de la IA para automatizar tareas complejas que tradicionalmente requerían una intervención humana intensiva.

En conclusión, este trabajo ha demostrado la viabilidad y eficacia de las DCNN para la detección de disparos en entornos naturales. Además, se ha destacado la necesidad de un conjunto de datos de calidad, así como la necesidad de estandarizar los procesos de anotación, para maximizar la precisión eficacia de los sistemas de detección de. La mejora de estos sistemas nos ayudara en la conservación y protección de la biodiversidad y los entornos naturales.

Evaluación del impacto global y alcance

Como hemos visto en los resultados, la automatización de las tareas de monitorización ambiental, en concreto, la detección de disparos supone para los investigadores una reducción de las horas que deben dedicar a la observación y toma de datos. Además, estos sistemas de detección ayudarán a la conservación de la biodiversidad y a la protección del ecosistema frente a actividades ilegales de caza u otras actividades potencialmente peligrosas para los entornos protegidos.

5.2 Futuras direcciones

Como líneas futuras para mejorar este trabajo se podrían plantear varios aspectos principales. En primer lugar, sería interesante aumentar la base de datos, debido a que las diferentes clases muestran un desbalanceo evidente en los diferentes tipos de disparo, siendo las muestras de disparos cercanos muy inferiores a las demás. Al aumentar las muestras y reducir el desbalanceo, se puede obtener una mejora considerable en el rendimiento del modelo y su precisión. Otro factor importante es la subjetividad a la hora de anotar los disparos, ya que diferentes investigadores podrían clasificar de diferente manera un disparo, es por ello por lo que sería optimo llegar a una estandarización sobre las anotaciones.

En segundo lugar, se podría profundizar en la búsqueda de un modelo que combine las capacidades de ambos modelos utilizados para mejorar el rendimiento actual de la clasificación. También se podrían buscar otras formas de obtener las características de nuestros datos de entrada de modo que nuestra red pueda abstraer estas características mejor.

Por último, se podría considerar la posibilidad de crear un algoritmo capaz de predecir la localización de un disparo en base a distintos nodos. Esto permitiría obtener con precisión los lugares donde más se concentran los disparos y llevar una mayor vigilancia de estas zonas, con el objetivo de evitar la caza furtiva y asegurar la conservación del espacio natural.

Referencias

[1] Akyildiz, I. F., Melodia, T., & Chowdhury, K. R. (2007). A survey on wireless multimedia sensor networks. *Computer networks*, *51*(4), 921-960.

[2] Hill, A. P., Prince, P., Piña Covarrubias, E., Doncaster, C. P., Snaddon, J. L., & Rogers, A. (2018). AudioMoth: Evaluation of a smart open acoustic device for monitoring biodiversity and the environment. *Methods in Ecology and Evolution*, *9*(5), 1199-1211.

[3] Mesaros, A., Heittola, T., Virtanen, T., & Plumbley, M. D. (2021). Sound event detection: A tutorial. *IEEE Signal Processing Magazine*, *38*(5), 67-83.

[4] Arslan, Y. (2017). Impulsive sound detection by a novel energy formula and its usage for gunshot recognition. *arXiv preprint arXiv:1706.08759*.

[5] Mulimani, M., & Koolagudi, S. G. (2018, October). Acoustic event classification using spectrogram features. In *TENCON 2018-2018 IEEE Region 10 Conference* (pp. 1460-1464). IEEE.

[6] Sigtia, S., Stark, A. M., Krstulović, S., & Plumbley, M. D. (2016). Automatic environmental sound recognition: Performance versus computational cost. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *24*(11), 2096-2107.

[7] Boddapati, V., Petef, A., Rasmusson, J., & Lundberg, L. (2017). Classifying environmental sounds using image recognition networks. *Procedia computer science*, *112*, 2048-2056.

[8] Adavanne, S., Parascandolo, G., Pertilä, P., Heittola, T., & Virtanen, T. (2017). Sound event detection in multichannel audio using spatial and harmonic features. *arXiv preprint arXiv:1706.02293*.

[9] Cakir, E., Heittola, T., Huttunen, H., & Virtanen, T. (2015, July). Polyphonic sound event detection using multi label deep neural networks. In *2015 international joint conference on neural networks (IJCNN)* (pp. 1-7). IEEE.

[10] Espi, M., Fujimoto, M., Kinoshita, K., & Nakatani, T. (2015). Exploiting spectro-temporal locality in deep learning based acoustic event detection. *EURASIP Journal on Audio, Speech, and Music Processing*, *2015*, 1-12.

[11] A. Albiol, V. Naranjo, and J. Prades´indice, "Tratamiento Digital de la Señal Teoría y Aplicaciones," Valencia: UPV, pp. 49–51.

[12] Doshi, K. (2021). Audio deep learning made simple-why mel spectrograms perform better. *Towars Data Science*.

[13] Russell, S. J., & Norvig, P. (2016). *Artificial intelligence: a modern approach*. Pearson.

[14] Turing, A. M. (2009). *Computing machinery and intelligence* (pp. 23-65). Springer Netherlands.

[15] Brutzkus, A., & Globerson, A. (2019, May). Why do larger models generalize better? A theoretical perspective via the XOR problem. In *International Conference on Machine Learning* (pp. 822-830). PMLR.

[16] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

[17] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfittin," J. Mach. Learn. Res., no. 15, pp. 1929–1958, 2014, doi: 10.1016/0370-2693(93)90272-J

[19] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533-536.

[20] Cui, N. (2018, April). Applying gradient descent in convolutional neural networks. In *Journal of Physics: Conference Series* (Vol. 1004, p. 012027). IOP Publishing.

[21] Narkhede, S. (2018). Understanding confusion matrix. *Towards Data Science*, *180*(1), 1-12.

[22] Generalitat Valenciana, "L'ALBUFERA," Conselleria de Agricultura, Desarrollo Rural, Emergencia Climática y Transición Ecológica, Valencia. Accessed: Aug. 11, 2023. [Online]. Available: https://parquesnaturales.gva.es/es/web/pn-l-albufera/conocenos

[23] N. P. García-de-la-Puente, F. Fuentes-Hurtado, L. Fuster, V. Naranjo, and G. Pinero, "Deep Learning Models for Gunshot Detection in the Albufera Natural Park," Proc 31st European Signal Proccesing Conference (EUSIPCO) pp 206-210, doi: 10.23919/EUSIPCO58844.2023.10289971.

[24] Simonyan, K. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[25] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[26] Katsis, L. K., Hill, A. P., Piña-Covarrubias, E., Prince, P., Rogers, A., Doncaster, C. P., & Snaddon, J. L. (2022). Automated detection of gunshots in tropical forests using convolutional neural networks. *Ecological Indicators*, *141*, 109128.

II Anexo

1.1 Implementación del código en Python

Este anexo muestra las partes más importantes del código. Se adjunta el proceso de extracción de características y la representación de espectrogramas de las dos partes de nuestro modelo, una parte para entrenar y otra para testear. Además, se adjunta un código que permite realizar el test en un entorno real, mostrando un gráfico con la duración de la grabación, los disparos reales y las predicciones que hace el modelo. El código inicial pertenece al trabajo "*Deep Learning Models for Gunshot Detection in the Albufera Natural Park*" [23]. En este trabajo hemos ajustado las clases para que sea una clasificación multiclase en vez de una clasificación binaria. Además, se ha creado la clase que realiza la prueba de test en un entorno real y creando una gráfica que representa los valores verdaderos de todas las clases y la representación de las predicciones. Adicionalmente se creó una CNN con 5 capas convolucionales, pero fue descartada, ya que los dos modelos del trabajo pueden ser utilizados para futuras ampliaciones de esta investigación.

```python
import torch
import torchaudio
import torchaudio.transforms as T
import matplotlib.pyplot as plt
import librosa
import pandas as pd
import numpy as np
from utils import *
print(torch.__version__)
print(torchaudio.__version__)

class MyDataAudio(torch.nn.Module):
    def __init__(
        self,
        annots_file,
        crop,
        ref,
        norm,
        win_length,
        n_fft,

    ):

        super().__init__()
        self.crop = crop
        self.ref = ref
        self.norm = norm
        self.win_length = win_length
        self.hop_length = int(win_length/2)
        self.num_samples = int(72000/self.hop_length) #number samples in 3 secs
        self.annotations = pd.read_csv(annots_file)
        self.n_fft = n_fft

        self.mel_scale = T.MelSpectrogram(n_mels=128, win_length=win_length, hop_length=self.hop_length,
                                          n_fft=n_fft, f_min=20 ,f_max=12000, sample_rate=24000)

    def __len__(self):
        return len(self.annotations)

    def __getitem__(self, index):
        audio_sample_path = self._get_audio_sample_path(index)
        label = self._get_audio_sample_label(index)
        label = torch.tensor(label, dtype = torch.int)
        signal, sr = torchaudio.load(audio_sample_path)
        Melspec = self.mel_scale(signal)
        Melspec = self._vertical_crop_(Melspec)
        Melspec = self._cut_if_necessary_(Melspec)
        MelspecdB = librosa.power_to_db(Melspec, ref=self.ref)
        if self.norm:
            MelspecdB = (MelspecdB-(np.min(MelspecdB))/(np.max(MelspecdB)-np.min(MelspecdB)))

        return torch.from_numpy(MelspecdB), label.type(torch.LongTensor)

    def _get_audio_sample_path(self, index):
        return self.annotations.iloc[index, 0]

    def _get_audio_sample_label(self, index):
        return self.annotations.iloc[index, 1]

    def _cut_if_necessary_(self, signal):
        if signal.shape[2] > self.num_samples:
            signal = signal[:, :, :self.num_samples]
        return signal

    def _vertical_crop_(self, signal):
        if signal.shape[1] > self.crop:
            signal = signal[:, :self.crop, :]
        return signal


def plot_spectrogram(specgram, ylabel="freq_bin"):
    fig, axs = plt.subplots(1, 1)
    axs.set_title("Spectrogram (db)")
    axs.set_ylabel(ylabel)
    axs.set_xlabel("frame")
    im = axs.imshow(specgram, origin="lower", aspect="auto")
    fig.colorbar(im, ax=axs)
    plt.show(block=False)
    plt.savefig('figure.png')

if __name__=="__main__":
    FILE_CSV = './DATA/DATASET_FOLD2.csv'
    win_length = 900
    n_fft = 1024
    crop = 60
    ref = 1.0
    norm = True

    MDA = MyDataAudio(annots_file=FILE_CSV,
                      crop=crop, ref=ref, norm=norm, win_length=win_length, n_fft=n_fft)

    meldB, lab = MDA[1402]
    print(lab)
    plot_spectrogram(meldB[0])
    print(meldB.shape)

    valMax, valmed = calculateref(MDA)

    print(valMax)
    print(valmed)
    import pandas as pd

# Cargar los datos
    data = pd.read_csv('./DATA/DATASET_FOLD2.csv')

# Contar las ocurrencias de cada clase
    class_counts = data['target'].value_counts()

# Imprimir el conteo de cada clase
    print(class_counts)
```

```python
class MyDataAudioFile(torch.nn.Module):
    def __init__(
        self,
        annots_file,
        audio_file,
        crop,
        ref,
        norm,
        win_length,
        n_fft
    ):

        super().__init__()
        self.crop = crop
        self.ref = ref
        self.norm = norm
        self.win_length = win_length
        self.hop_length = int(win_length/2)
        self.num_samples = int(72000/self.hop_length) #number samples in 3 secs
        self.audio_file = audio_file
        self.n_fft = n_fft
        self.mel_scale = T.MelSpectrogram(n_mels=128, win_length=win_length, hop_length=self.hop_length,
                                          n_fft=n_fft, f_min=20 ,f_max=12000, sample_rate=24000)
        # PREPARE AUDIO
        self.wav, self.sr = torchaudio.load(self.audio_file)
        self.lenwav_secs = int((self.wav.shape[1])/self.sr)
        self.signal = self.wav.unsqueeze(0).unsqueeze(0)
        # Define window size and step size
        self.win_size = self.sr*3
        self.hop_size = self.sr
        # Define sliding window parameters
        unfold = torch.nn.Unfold(kernel_size=(1, self.win_size), stride=(1, self.hop_size))
        # Apply sliding window
        self.y_frames = unfold(self.signal)

        # PREPARE LABELS
        self.annotations = pd.read_csv(annots_file, sep=';')
        # To seconds and round
        #self.annotations['seconds'] = pd.to_timedelta(self.annotations['init']).dt.total_seconds().astype(int)
        self.annotations['seconds'] = pd.to_timedelta(self.annotations['init'].str.replace(',', '.')).dt.total_seconds().astype(int)

        # Eliminate duplicated values
        self.res = self.annotations.drop_duplicates(subset='seconds')
        self.res = self.res['seconds']

        self.one_hot = torch.zeros(self.lenwav_secs, dtype=torch.long)

        # Assign labels to one-hot tensor
        for _, row in self.annotations.iterrows():
            if row['seconds'] in self.res.values:
                self.one_hot[int(row['seconds'])] = row['class']




    def __len__(self):
        return self.lenwav_secs-2 # because 3 segs window

    def __getitem__(self, index):

        audio_sample = self._get_audio_sample(index)
        label = self._get_audio_sample_label(index)
        Melspec = self.mel_scale(audio_sample)
        Melspec = self._vertical_crop_(Melspec)
        Melspec = self._cut_if_necessary_(Melspec)
        MelspecdB = librosa.power_to_db(Melspec, ref=self.ref)
        if self.norm:
            MelspecdB = (MelspecdB-(np.min(MelspecdB))/(np.max(MelspecdB)-np.min(MelspecdB)))
        return torch.from_numpy(MelspecdB), label.type(torch.LongTensor)

    def _get_audio_sample(self, index):
        return self.y_frames[:, :, index]

    def _get_audio_sample_label(self, index):
        return self.one_hot[index]

    def _cut_if_necessary_(self, signal):
        if signal.shape[2] > self.num_samples:
            signal = signal[:, :, :self.num_samples]
        return signal

    def _vertical_crop_(self, signal):
        if signal.shape[1] > self.crop:
            signal = signal[:, :self.crop, :]
        return signal


def plot_spectrogram(specgram, ylabel="freq_bin"):
    fig, axs = plt.subplots(1, 1)
    axs.set_title("Spectrogram (db)")
    axs.set_ylabel(ylabel)
    axs.set_xlabel("frame")
    im = axs.imshow(specgram, origin="lower", aspect="auto")
    fig.colorbar(im, ax=axs)
    plt.show(block=False)
    plt.savefig('figure.png')

def convertir_stereo_a_mono(ruta_archivo):
    datos, samplerate = sf.read(ruta_archivo)
    if len(datos.shape) == 2 and datos.shape[1] == 2:
        print(f"El archivo {ruta_archivo} es estéreo. Convirtiendo a mono...")
        datos_mono = np.mean(datos, axis=1)
        ruta_salida = ruta_archivo.replace(".wav", "_mono.wav")
        sf.write(ruta_salida, datos_mono, samplerate)
        print(f"El archivo convertido se ha guardado como {ruta_salida}")
    else:
        print(f"El archivo {ruta_archivo} ya es mono o tiene un formato no soportado.")
```

```python
from torchvision.models import resnet18, ResNet18_Weights
import soundfile as sf
import torch
import torchaudio
import torchaudio.transforms as T
import matplotlib.pyplot as plt
import librosa
import pandas as pd
import numpy as np
from utils import *
from torch.utils.data import DataLoader,Subset
from CNN import CustomCNN
import torch.nn as nn
from model_VGG import VGG
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report,confusion_matrix
import torch.optim as optim
from model_VGG import vgg16
from testing import MyDataAudioFile
import seaborn as sns


if __name__=="__main__":
    FILE_CSV2 = 'DATA/init_times_SMA07370_221120_073002.csv'
    FILE_AUDIO2 = 'DATA/SMA07370_20221120_073002.wav'
    win_length = 900
    n_fft = 1024
    crop = 60
    ref = 1.0

    norm = True
    MDA = MyDataAudioFile(annots_file=FILE_CSV2, audio_file=FILE_AUDIO2,
                          crop=crop, ref=ref, norm=norm, win_length=win_length, n_fft=n_fft)

    batch_size=256
    test_loader = DataLoader(dataset=MDA, batch_size=batch_size, shuffle=False)
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    checkpoint=torch.load('model_VGG16.pth')
    model=vgg16(activation=True).to(device)
    # model=resnet18(weights=ResNet18_Weights.DEFAULT)
    # model.conv1 = nn.Conv2d(1, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    # dropout = nn.Dropout(p=0.8)
    # num_features = model.fc.in_features
    # model.fc = nn.Linear(num_features, 4)
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer = optim.Adam(model.parameters())
    #model.to(device)
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])

# Restaurar el número de épocas y el valor de la pérdida
    epoch = checkpoint['epoch']
    train_loss = checkpoint['loss']

    criterion = nn.CrossEntropyLoss()
    def test(model, test_loader, device):
     model.eval()
     val_loss = 0.0
     correct = 0
     all_preds = []
     all_labels = []
     with torch.no_grad():
         for data, target in test_loader:
             data, target = data.to(device), target.to(device)
             output = model(data)
             val_loss += criterion(output, target).item()
             pred = output.argmax(dim=1, keepdim=True)
             correct += pred.eq(target.view_as(pred)).sum().item()
             all_labels.extend(target.cpu().numpy())
             all_preds.extend(pred.cpu().numpy())

     val_loss /= len(test_loader)
     accuracy = correct / len(test_loader.dataset)
     return val_loss, accuracy, all_preds, all_labels


    def plot_confusion_matrix(labels, preds, class_names):
     conf_matrix = confusion_matrix(labels, preds,normalize=None)
     plt.figure(figsize=(10, 8))
     sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
     plt.xlabel('Predicted')
     plt.ylabel('True')
     plt.title('Confusion Matrix')
     plt.show()


    test_loss, test_acc, all_preds, all_labels = test(model, test_loader, device)
    print(f'Test Loss: {test_loss:.4f}, Test Acc: {test_acc:.2f}')
    print('Classification Report for Test Set:')
    print(classification_report(all_labels, all_preds, target_names=['background','lejos', 'medio', 'cerca']))
    precision = precision_score(all_labels, all_preds, average='weighted')
    recall = recall_score(all_labels, all_preds, average='weighted')
    f1 = f1_score(all_labels, all_preds, average='weighted')
    plot_confusion_matrix(all_labels, all_preds, class_names=['background','lejos', 'medio', 'cerca'])
    print(f'Test Precision: {precision:.4f}')
    print(f'Test Recall: {recall:.4f}')
    print(f'Test F1 Score: {f1:.4f}')


    #plot_confusion_matrix(all_labels, all_preds, class_names=['background','lejos', 'medio', 'cerca'])

    # Plot predictions
    secs = 3600
    np.save('real_labels.npy', np.array(all_labels))
    np.save('predicted_labels.npy', np.array(all_preds))

    # Cargar y mostrar resultados
    real_labels = np.load('real_labels.npy')
    predicted_labels = np.load('predicted_labels.npy')

    # Cargar y mostrar resultados
    data_gt = np.load('real_labels.npy')
    data_y_pred = np.load('predicted_labels.npy')
    signal, sr = torchaudio.load(FILE_AUDIO2)
    signal = signal.unsqueeze(0).unsqueeze(0)
    win_size = sr*3
    hop_size = sr
    unfold = torch.nn.Unfold(kernel_size=(1, win_size), stride=(1, hop_size))
    y_frames = unfold(signal)
    power_frames = torch.mean(torch.square(y_frames), axis=1)
    poten_dB = 10*torch.log(power_frames/torch.min(power_frames))
    poten_dB_norm = poten_dB/torch.max(poten_dB)

    x = np.arange(poten_dB_norm.shape[1])/60
    y_pot = poten_dB_norm.cpu().numpy()
    y = y_pot[0]


    plt.figure(figsize=(36,12))
    plt.plot(x[0:secs], y[0:secs], linewidth=2, label='Poten dB norm')
    plt.plot(x[0:secs], data_gt[0:secs], linewidth=2, label='Ground Truth')
    plt.plot(x[0:secs], data_y_pred[0:secs], linewidth=2, label='Predictions')
    plt.legend()
    plt.ylabel('score')
    plt.xlabel('minutes')
    plt.savefig('./pot_gt_predictions_scrop_VGG_exp6.png')
    plt.show()
```

```python
from utils import *
class CustomCNN(nn.Module):
    def __init__(self):
        super(CustomCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)) #capa 1 conv 2d
        self.pool1 = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))                     #capa 1  pooling
        self.conv2 = nn.Conv2d(32, 64, kernel_size=(3, 3), padding=(1, 1))               #capa 2 conv 2d
        self.pool2 = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))                     #capa 2  pooling
        self.conv3 = nn.Conv2d(64, 128, kernel_size=(3, 3), padding=(1, 1))              #capa 3 conv 2d
        self.pool3 = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))                     #capa 3  pooling
        self.conv4 = nn.Conv2d(128, 256, kernel_size=(3, 3), padding=(1, 1))             #capa 4 conv 2d
        self.pool4 = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))                     #capa 4  pooling
        self.conv5 = nn.Conv2d(256, 512, kernel_size=(3, 3), padding=(1, 1))            #capa 5 conv 2d
        self.pool5 = nn.MaxPool2d(kernel_size=(2, 2), stride=(2, 2))                    #capa 5  pooling
        self.flatten = nn.Flatten()


        self.fc1_input_dim = 512 * 1 * 5
        self.fc1 = nn.Linear(self.fc1_input_dim, 128)
        self.dropout1 = nn.Dropout(0.7)
        self.fc2 = nn.Linear(128, 64)
        self.dropout2 = nn.Dropout(0.7)
        self.fc3 = nn.Linear(64, 4)  # 3 clases de salida
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = self.pool3(F.relu(self.conv3(x)))
        x = self.pool4(F.relu(self.conv4(x)))
        x = self.pool5(F.relu(self.conv5(x)))
        x = self.flatten(x)
        x = self.dropout1(F.relu(self.fc1(x)))
        x = self.dropout2(F.relu(self.fc2(x)))
        x = self.fc3(x)
        x = self.softmax(x)
        return x
model=CustomCNN()
summary(model, input_size=(64, 1, 60, 160))
```