Redes Neuronales Convolucionales Siamesas y su aplicación a la detección de plagio

UNIVERSIDAD
COMPLUTENSE
MADRID

Carlos Pérez Simón
Facultad de Ciencias Matemáticas

Trabajo de Fin de Grado presentado para optar al
Grado en Ingeniería Matemática

Septiembre de 2023

Tutores:
Antonio López Montes, María Teresa Benavent Merchán,
Antonio Martínez Rata, Ángel González Prieto

Índice general

Índice de figuras

Capítulo 1

Introducción

En las últimas décadas, el campo de la inteligencia artificial ha sido un tema de interés global debido a su amplia variedad de aplicaciones. Las redes neuronales, cuyo sistema se basa en el funcionamiento de las neuronas de un cerebro humano, conforman el pilar fundamental de esta tecnología. Estas se encargan de tareas complejas como son la toma de decisiones, el reconocimiento de patrones y la predicción.

La inteligencia artificial puede parecer ajena a lo cotidiano, sin embargo, es utilizada diariamente por millones de usuarios. Por ejemplo, podemos encontrarla en aplicaciones de navegación para calcular la ruta óptima entre dos ubicaciones, en algoritmos de búsqueda y recomendación, y en funciones tan comunes como la autocorrección del teclado.

El presente estudio está enfocado a comprender el proceso de aprendizaje y el trasfondo algorítmico de las redes neuronales convolucionales siamesas., las cuales resultan útiles a la hora de identificar patrones, segmentar y clasificar señales de imagen y audio.

Conocer las aplicaciones de una herramienta permite una mayor comprensión de su funcionamiento. Por tanto, exploraremos de manera práctica la detección de plagio entre señales de audio, un caso en el que se ven involucradas tanto las señales de imagen como de audio.

Capítulo 2

Justificación del tema y objetivos

2.1. Objetivos

El presente trabajo tiene como propósito examinar los fundamentos matemáticos y algorítmicos de las Redes Neuronales Convolucionales Siamesas (SCNN).
Los objetivos de este trabajo de investigación son los siguientes:

- Contextualizar las aplicaciones y funciones de las Redes Neuronales Convolucionales Siamesas.
- Analizar la estructura general de una Red Neuronal Convolucional Siamesa, explorando la arquitectura básica de estas redes en cuanto a diseño y funcionamiento.
- Explorar diferentes técnicas de optimización para el entrenamiento de la red.
- Profundizar en el concepto del operador convolución.
- Estudiar la mejora del rendimiento y la precisión de la red mediante funciones de activación y técnicas de agrupamiento.
- Explorar la base matemática del procesamiento de señales de audio.
- Aplicar las Redes Neuronales Convolucionales Siamesas a la detección de plagio musical.

Capítulo 3

Redes Neuronales Convolucionales Siamesas

3.1. Introducción histórica

A lo largo de esta sección conoceremos la historia de las redes neuronales, cómo surgieron y cómo han ido evolucionando desde mediados del siglo XX hasta la actualidad.

La exploración del sistema nervioso humano tiene sus raíces a comienzos de siglo a manos del premio Nobel Ramón y Cajal, cuyo trabajo sentó las bases para el estudio del cerebro humano. Ello dio pie al estudio sobre el funcionamiento de las neuronas, donde destacan el neurofisiólogo Warren McCulloch y el matemático Walter Pitts. En su artículo[1] científico más relevante explicaban la estructura de una neurona y la forma en la que esta transmite y procesa la información. Más tarde, en 1948, Shannon planteó su teoría matemática de la comunicación, dando lugar a un sistema de información basado en un valor binario, el bit. Estos dos avances abrieron la puerta al nuevo reto de reproducir, mediante computación, la actividad del cerebro humano.

En 1958, Frank Rosenblatt planteó la primera estructura de redes neuronales, el Perceptrón, conocido actualmente como la unidad básica de una red neuronal. El perceptrón se basa en dos niveles: el nivel de entrada, por el cual se introduce información del exterior en código binario, esto es, respuestas de sí (1) o no (0); y el nivel de salida, el cual elabora una única decisión o respuesta binaria mediante la ponderación de los datos de entrada. Su estructura esquemática puede apreciarse en **Fig.3.1:**

[1]

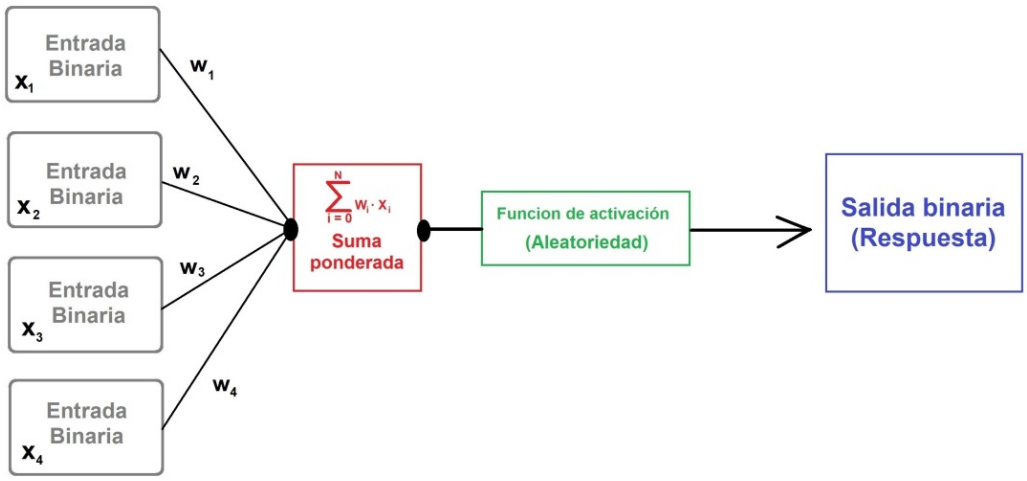


Figura 3.1: Organización del Perceptrón. (Elaboración propia)

Se desarrollaron otros sistemas como ADALINE, que implementaba el aprendizaje mediante una regla de mínimos cuadrados. Sin embargo, el perceptrón destacó y acabó evolucionando en 1960 a Mark I, el primer ordenador con la posibilidad de aprender mediante prueba y error.

En 1965, la estructura de una red neuronal evolucionó hasta ser un conjunto de neuronas, conocido como Perceptrón Multicapa. En esta estructura, la entrada y salida de datos se mantiene binaria, pero surge el concepto de las capas neuronales. Ahora la estructura sería una capa de entrada, una capa de salida y, entre medias, una sucesión de capas ocultas que procesan la información. Hasta este momento, prevalecían modelos lineales donde los científicos debían asignar manualmente los pesos de ponderación.

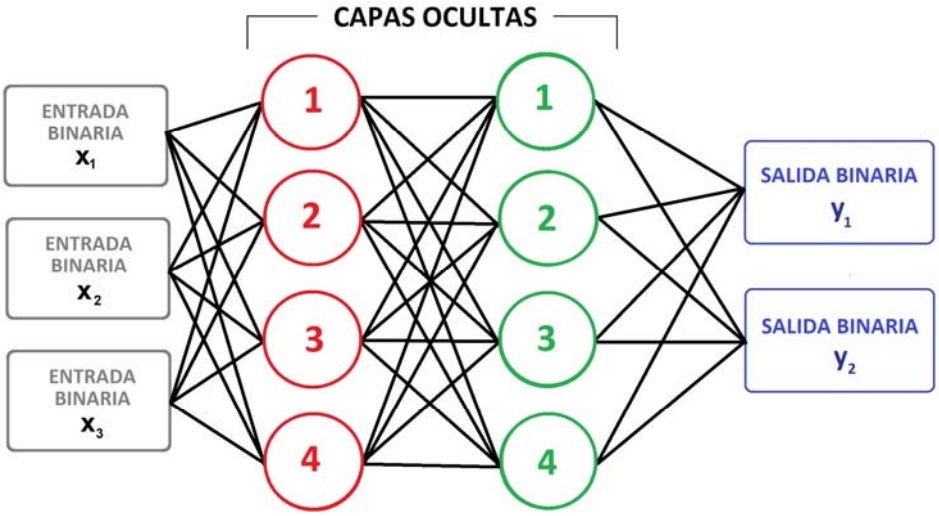


Figura 3.2: Organización del Perceptrón Multicapa. (Elaboración propia)

En 1969, Marvin Minsky y Seymour Papert publicaron ”Perceptrons”, un libro en el que presentaron los límites del perceptrón utilizando conceptos de geometría computacional. Aunque, a día de hoy se ha demostrado que las redes compuestas por múltiples perceptrones sí que pueden aproximar cualquier función continua desde un intervalo compacto de los números reales al intervalo [−1, 1], los avances en el campo se paralizaron tras la publicación mencionada.

El periodo de estancamiento duró hasta 1982, cuando John Hopfield planteó un enfoque innovador basado en la creación de dispositivos con aplicaciones prácticas. Esta nueva perspectiva marcó el inicio de grandes avances en el área del aprendizaje automático y fue el punto de partida para la realización de conferencias internacionales sobre inteligencia artificial.

Durante este proceso surgieron nuevos tipos de neuronas, como las neuronas sigmoides o las ’Bias’. Las neuronas sigmoides destacan por no estar limitadas a una entrada de datos binaria y por dar lugar al concepto de función de activación.

La función de activación se encarga cumple con la tarea de romper la linealidad de la respuesta mediante la introducción de pequeñas alteraciones que acercan los datos a la realidad. La función sigmoide es la función de activación principal por su carácter suave, como se puede observar en **Fig. 3.3:**

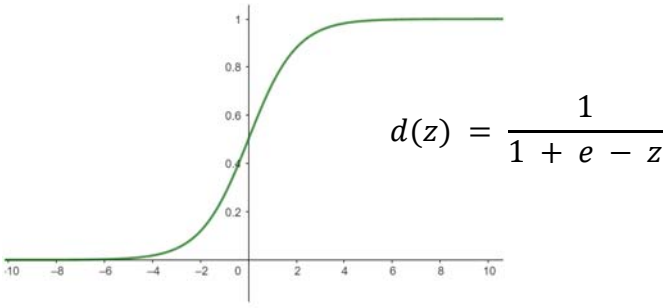


Figura 3.3: Función de activación sigmoide. (Elaboración propia)

Por otro lado, en la búsqueda de avances en el aprendizaje automático, surgieron las redes ”Feedforward” y el método ”Backpropagation”. Las redes ”Feedforward” se fundamentan en la idea de la retroalimentación, donde las capas están interconectadas de manera que los resultados de una capa se utilizan como

entradas para la siguiente capa. Cabe destacar que, en este tipo de redes, la información fluye en un solo sentido, sin existir bucles de retroceso. Además, es posible conectar todas las neuronas de entrada con la siguiente capa, denominada “Fully Connected”.

Posteriormente, en 1986, se desarrolló el algoritmo ”Backpropagation”, el cual hizo posible el entrenamiento supervisado de las redes neuronales multicapa. Dicho algoritmo calcula el error entre la respuesta generada y los datos reales, propagándolo en dirección opuesta a la red neuronal. De esta forma se pueden realizar pequeñas alteraciones en las ponderaciones asumidas, ajustando la red para que logre los resultados buscados. Para optimizar el aprendizaje se utiliza el descenso de gradiente estocástico, una técnica que permite estimar, de forma numérica, los mínimos de la función de pérdida.

Mientras se desarrollaba el aprendizaje automático, K. Fukushima publicó un nuevo modelo de red neuronal para el reconocimiento de patrones, el Neocognitrón. Este se basaba en una rígida jerarquía de la extracción de características, empezando a extraer patrones simples y aumentando la complejidad con cada capa. Para facilitar la comprensión, se incluye **Fig. 3.4** para observar la aplicación del Neocognitrón a un ejemplo sencillo de reconocimientos de patrones.

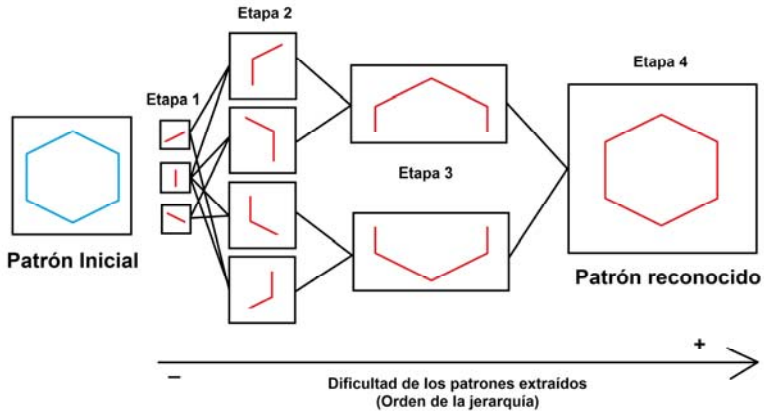


Figura 3.4: Ejemplo del funcionamiento del Neocognitrón. (Elaboración propia)

El Neocognitrón derivó a la creación de las Redes Neuronales Convolucionales, las cuales se basan en el funcionamiento del córtex visual de un cerebro humano. En 1995, Yann LeCun publicó un artículo sobre el uso de redes neuronales convolucionales para procesamiento de imágenes, lenguaje y series temporales.

LeCun planteó una arquitectura de CNN consistente en dos niveles: el nivel convolucional, donde se aplican filtros a la imagen mediante operaciones matriciales llamadas convoluciones; y el nivel de clasificación, donde se clasifica el grupo de nodos resultado del nivel convolucional. Por ejemplo, supongamos que un perro es la imagen de entrada. El primer nivel extraería las características relevantes del perro y, posteriormente, el segundo nivel clasificaría dicha información entre un grupo de nodos respuesta fijos, como podría ser el conjunto: $\{Gato, Perro, Conejo\}$, proporcionando la respuesta final de nuestra CNN. Véase el ejemplo en **Fig. 3.5.**

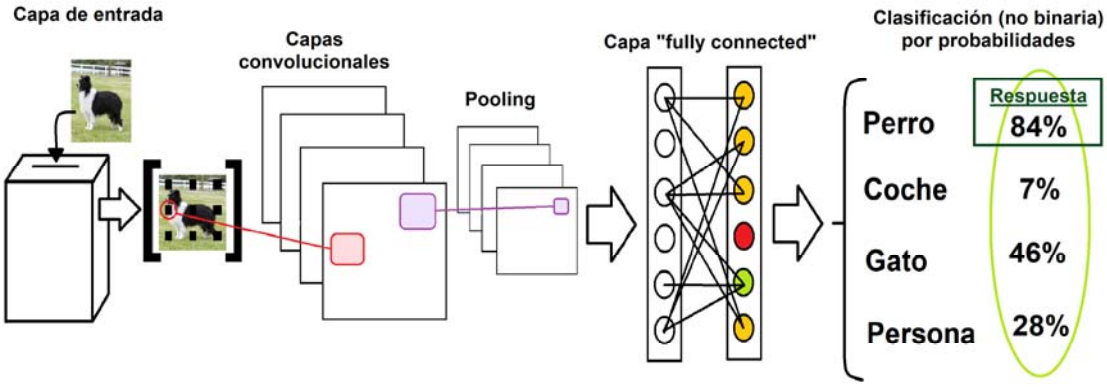


Figura 3.5: Ejemplo de estructura de red de una CNN de clasificación. (Elaboración propia)

También en la década de 1990, surgió la idea de una red siamesa. Esta idea consiste en la réplica de una parte de la arquitectura de la red neuronal con el objetivo de aplicar la red a dos conjuntos de datos en las mismas condiciones y poder fusionar ambas informaciones posteriormente en una o varias capas comunes. Estas se utilizan para comparar y verificar la similitud entre dos conjuntos de datos. En el caso de las CNN Siamesas, la comparación de imágenes resulta una herramienta muy útil en diversos campos. Es por ello que, en la década de los 90, se empezaron a utilizar como sistema de comparación de huellas dactilares y reconocimiento de rostros. Además en esta década surge el término “big data”.
Finalmente, a partir de 2010, cuando el “big data” ya había provocado un gran desarrollo en el área del procesamiento de imágenes y el aprendizaje automático, empezaron a utilizarse las CNN siamesas para tareas como recomendación musical o detección de plagio en música y documentos.

3.2. Concepto de CNN

Las Redes Neuronales Convolucionales, o CNN, son un tipo de arquitecturas dentro del campo del aprendizaje automático que fueron diseñadas para procesar datos que tienen una estructura espacial, principalmente imágenes. Estas redes utilizan una técnica matemática llamada “convolución” para extraer características significativas de los datos de entrada. El concepto de convolución tuvo sus orígenes en la rama del análisis matemático. En este campo, un filtro se define como una transformación entre dos sucesiones x, y:

$$x \to y, \qquad y_j = f_0 x_j + f_1 x_{j-1} + f_2 x_{j-2} + \cdots + f_n x_{j-n} = \sum_{k \in \mathbb{Z}} x_{(j-k)} f_k$$

Definiendo el vector coeficientes $f = (f_0, f_1 \dots, f_n)$, surge el operador convolución, $*$, el cual simplifica la expresión general de los filtros a: $y = x * f$

En las CNN se aplican las convoluciones a las imágenes, para lo cual se emplean los ‘mapas de características’. Estos son una forma matemática de representar una imagen mediante el uso de matrices, reduciendo a su vez la complejidad de los datos y presentándolos de una manera más visual. En el caso de las imágenes, cada pixel está asociado con un elemento del mapa de características y tiene un valor entero entre 0 y 255, quedando así definidas la intensidad y la ubicación precisa de cada pixel en la imagen. Además, si la imagen es en color, el mapa de características consiste en una matriz tridimensional donde la tercera dimensión indica la cantidad de cada componente de color (RGB) presente en dicho pixel. De esta forma quedan definidos la posición y el color de cada pixel.

La idea básica de una red convolucional es aplicar diferentes filtros a la imagen de entrada, y realizar una clasificación porcentual con el mapa de características de salida, ya sea binaria o de clases. La principal ventaja de las CNN es que tienen la capacidad de encontrar patrones y características sin necesidad de predefinirlos, es decir, se definen las características buscadas de manera automática durante el entrenamiento.

3.3. Funcionamiento y arquitectura de una CNN

Las CNN son un tipo de redes cuyo funcionamiento fue creado basándose en el córtex visual del cerebro humano. Por lo tanto, son capaces de extraer información y aprender en los diferentes niveles de abstracción de una imagen.

Una Red Neuronal Convolucional se compone de varias capas que abordan funciones específicas diferentes. Las capas más comunes en el procesamiento de imágenes son: la capa de entrada, “Input Layer”; las capas de convolución, “Convolutional Layer”; las capas de submuestreo, “Pooling”; las capas totalmente conectadas, “Fully Connected”; y la capa de salida, “Output Layer”. Entre ellas están conectadas por el producto escalar de los nodos activados y una serie de pesos asociados, los cuales son los parámetros de la red neuronal.
En **Fig. 3.5** se observa un esquema de las capas de una CNN sobre el ejemplo de clasificación. Veamos más en profundidad cada capa:

- **Capa de Entrada**:

  Es la primera capa de la red, en la cual se introducen los datos de entrada a la red. Esta capa no realiza ningún tipo de procesamiento o transformación en los datos, simplemente los almacena y los transmite a la red. Sin embargo, los datos de entrada pueden ser previamente normalizados con el objetivo de optimizar el rendimiento de la red. Como cada pixel tiene asociado un valor entre 0 y 255 en el mapa de características, la normalización consiste en dividir el mapa de características entre 255, obteniendo así un mapa de características con valores en el intervalo [0,1].

- **Capas Convolucionales:**

  Estas capas se encuentran en la capa oculta de la red y son aquellas en las que tienen lugar las convoluciones. Estas se utilizan para la detección de patrones y características en los datos de entrada. Tras cada convolución se aplica la función de activación, encargada de introducir no-linealidad en la red para aumentar la complejidad del aprendizaje sobre los datos de entrada. Se suelen utilizar varias capas para obtener la información buscada, de modo que entre ellas están conectadas mediante la aplicación del kernel, es decir, el filtro.

Por ejemplo en una red de clasificación se tiene que: en la primera capa convolucional se diferencian formas simples, colores o bordes; en las siguientes se empiezan a distinguir combinaciones de bordes y colores; y en la capa final se analizan las formas hasta identificar el objeto

- **Capas “Pooling”:**

  Ubicadas tras las capas convolucionales en la capa oculta de la red, son las encargadas de reducir la dimensionalidad. Es decir, reducen la resolución espacial de los datos de salida de una capa convolucional, lo cual evita que se produzca un sobreajuste y optimiza la memoria de la red reduciendo los parámetros a almacenar.

- **Capas “Fully Connected”:**

  Estas se encuentran al final de una CNN, tras las capas convolucionales y el pooling. Son un tipo de capa en la cual todos los nodos de una capa están conectados a todos los nodos de la siguiente capa. Suelen utilizarse en redes “Feedforward” ya que permiten que la información fluya por la red. Su función principal es realizar tareas de clasificación, regresión y procesamiento de imágenes.

- **Capa de Salida:**

  Se trata de la última capa de la red, y su función es generar una salida a partir de las capas anteriores. Suele estar precedida de una capa “fully connected” cuyos datos de entrada han sido activados. La activación es una medida de la probabilidad de que la imagen de entrada pertenezca a una clase en particular. En **Fig. 3.5** se observa que la capa “fully connected” devuelve una probabilidad de pertenencia de la imagen de entrada asociada a cada clase posible. Por tanto, la salida de la red es la clase con mayor probabilidad de contener la entrada inicial.

Una vez la CNN está creada, hay que entrenarla mediante un algoritmo de aprendizaje automático. La función del entrenamiento es ajustar los pesos que conectan las capas entre sí, de manera que se optimice la capacidad de reconocimiento de patrones en las imágenes de entrada.

3.4. Concepto de CNN Siamesas

Las redes neuronales convolucionales siamesas se basan en la idea de comparar dos entradas de datos utilizando una arquitectura de red neuronal compuesta por dos ramas simétricas. Estas dos ramas comparten los mismos pesos y estructura. Dicha estructura consiste en un conjunto determinado de capas convolucionales y "pooling", que definen los filtros aplicados y reducen la dimensión de la imagen; y una capa "fully connected" que reúne todas las características extraídas en un vector característico. Se observa la estructura en **Fig. 3.4**.

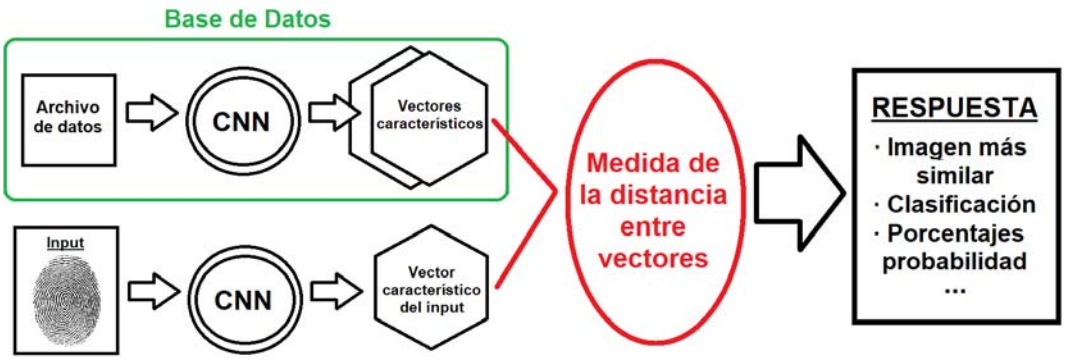


Figura 3.6: Ejemplo de estructura de red de una CNN de clasificación. (Elaboración propia)

En el ejemplo se quiere comparar una huella dactilar, introducida como input, con una base de datos con el objetivo de localizar a quién pertenece. Las huellas registradas en el archivo de datos han sido previamente procesadas con una CNN y se tiene archivado un vector característico de cada una de ellas, el cual describe las características y patrones encontrados en cada una. Por otro lado, al introducir nuestra huella input a una red CNN idéntica a la anterior, se obtiene el vector característico correspondiente. Posteriormente se calcularía la distancia entre el vector input y los vectores del archivo de datos, obteniéndose un porcentaje de similitud por cada par. Sabiendo los porcentajes por pares, el dueño de la huella será aquel con mayor porcentaje de similitud.

Actualmente las CNN siamesas se usan en tareas relacionadas con la comparación de imágenes, como la fusión de imágenes, el seguimiento de objetos y el auto-recorte de imágenes

Capítulo 4

Fundamentación teórica

4.1. Métodos de optimización no restringida.

La optimización se define, según la RAE, como la “acción de buscar la mejor manera de realizar una actividad”. A nivel matemático se puede llevar a cabo esta tarea mediante la búsqueda de un mínimo o máximo en una función que modelice nuestro problema. Dicha función se conoce como función de perdida y su función es evaluar la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas en el aprendizaje. Es decir, cuanto menor es el resultado de la función de pérdida, más eficientes es la red neuronal.
En el ámbito de las redes neuronales, la función de perdida mide la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas para el aprendizaje de nuestra red. Podemos considerar dos maneras de medir dicha desviación:

- Error Local: Mide la desviación de la predicción centrándose en una observación dada.

$$Error = Y_i - \hat{Y}_i$$

- Error Cuadrático Medio (ECM): Es una función de error global que mide el porcentaje de error de aproximación cometido por la red neuronal en la totalidad de observaciones.

$$ECM = \sum_{i=1}^{N} \frac{\left(Y_i - \hat{Y}_i\right)^2}{N}$$

Cuanto menor sea el valor de la función de pérdida, la red neuronal será más eficiente. Existen diferentes métodos para llevar acabo la optimización, en este caso minimización, de la función de pérdida. Por un lado están los métodos de búsqueda directa, destacando el método de ”Nelder-Mead”, los cuales utilizan únicamente valores de la función objetivo. Por otro lado, los métodos de búsqueda indirecta requieren valores exactos de las primeras derivadas de la función objetivo. Entre estos últimos métodos destacan el método ”Descenso del Gradiente” y el método ”Gradiente Conjugado”. Veamos en qué consisten estos métodos:

4.1.1. Método Nelder-Mead.

El método Nelder-Mead es un algoritmo iterativo diseñado para resolver el problema de optimización, en concreto de minimización, de una función no lineal sin restricciones . Además, se considera un algoritmo de búsqueda directa, ya que puede aplicarse a aquellas funciones de las cuales no podemos calcular sus derivadas. MATLAB utiliza este método para la minimización sin restricciones, en concreto con el comando: $fminsearch(f, x_0)$.

Este método es óptimo en problemas con pequeñas dimensiones, ya que presenta pocas evaluaciones de función en cada iteración. Sin embargo, solo se asegura la convergencia del mismo para funciones estrictamente convexas en $\mathbb{R}^n$.

En términos geométricos este método se basa en el algoritmo del simplex, un conjunto de métodos de optimización en los cuales se maximiza, o minimiza, una función sobre un conjunto de valores que cumplen una serie de inecuaciones. Considerando dicha función objetivo a minimizar $f: \mathbb{R}^n \to \mathbb{R}^n$, el simplex es la envoltura convexa de un conjunto $\{x_0, \ldots, x_n\}$ de $(n+1)$ puntos independientes no coplanarios y afines en un espacio euclídeo.

El método Nelder-Mead consiste en realizar una secuencia de transformaciones en el simplex de manera que la función objetivo, en nuestro caso la función de pérdida, se vea reducida en cada iteración hasta llegar a tener un valor lo suficientemente pequeño. Dichas transformaciones son la expansión, la contracción o el encogimiento del simplex. Cabe destacar que la diferencia principal entre el método Nelder-Mead y el algoritmo del simplex convencional es que para ejecutar este útlimo son necesarias unas restricciones.

4.1.1.1. Aplicación del Método Nelder-Mead:

Para comenzar a usar el método, se tiene que construir primero el simplex inicial. Partiendo de un punto arbitrario x_0 , se generan los n vértices restantes teniendo en cuenta que todas sus aristas deben tener la misma longitud y que debe cumplirse:

$$x_i = x_0 + h_i \cdot e_i \quad con\ i \in \{1, \dots, n\}$$

donde h_i es el tamaño de paso en la dirección del vector unitario e_i en $\mathbb{R}^n$.

Una vez se tiene el simplex inicial, para hallar la transformación óptima en cada iteración se siguen los pasos que se muestran a continuación:

- **Primer paso: Ordenación**

  Se evalúan los nodos x_i del simplex en la función de pérdida, de manera que se obtienen los $f_i \equiv f(x_i)\ \ \forall i \in [0, n]$. Después, se ordenan los valores f_i de manera ascendente, de forma que si $f_j \ll f_k$ entonces consideraremos que x_j es mejor punto que x_k, puesto que presenta menor pérdida. Esto da pie a definir x_m como el mejor vértice, x_p como el peor, y x_{2p} como el segundo peor.

  Vamos a tomar como ejemplo el caso de un simplex en $\mathbb{R}^2$, por tanto compuesto por tres puntos $\{x_1, x_2, x_3\}$. Además suponemos $f_3 < f_2 < f_1$, por tanto se tiene que $x_m \equiv x_3$ y $x_p \equiv x_1$.

- **Segundo paso: Centroide y reflexión**

  En este paso buscamos un nuevo simplex, a partir del anterior, de modo que se optimice la pérdida. Por teoría, a partir de un simplex se puede generar uno nuevo mediante la proyección de uno de sus vértices a través del centroide de los demás vértices.

  Para ello, primero vamos a calcular el centroide, o baricentro, del conjunto total de vértices del simplex excluyendo a x_p, puesto que es el que presenta mayor pérdida.

$$c \equiv \frac{1}{n}\sum_{i \neq p} x_i \quad \forall i \in [1, n]$$

En nuestro ejemplo, se tiene que el centroide de dos puntos es su punto medio.

Una vez calculado el centroide, c, vamos a calcular la reflexión simple de x_p sobre c. Se trata de una proyección, es decir, en la dirección del vector que une x_p y c, se genera el punto x_r a una determinada distancia de c. En el caso de la reflexión simple, dicho punto se genera a la misma distancia de c que x_p pero al lado opuesto. Sin embargo, según la transformación que se aplique durante la reflexión, dicha distancia varía completamente.

Esto se observa mejor en **Fig. 4.1:**

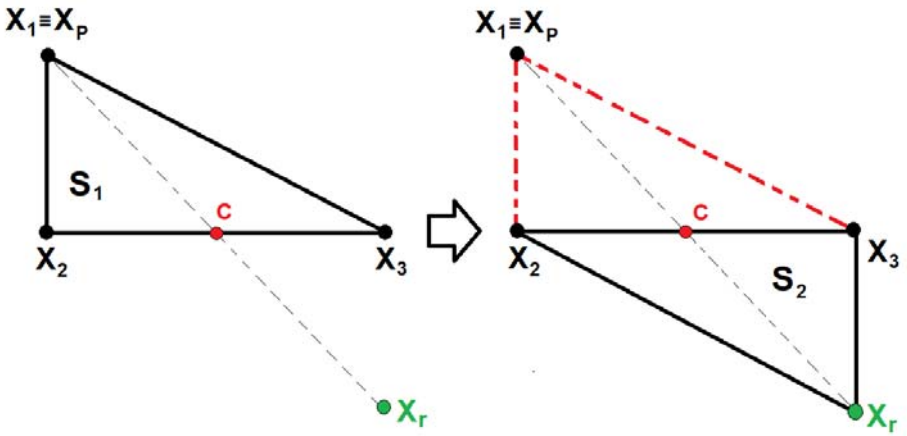


Figura 4.1: Reflexión simple del simplex a través del centroide. (Elaboración propia)

Ahora que hemos calculado x_r, evaluamos este dentro de f para observar su pérdida y compararla con la pérdida del resto de puntos del simplex. Teniendo en cuenta que el paso de reflexión viene dado por:

$$x_r = x_p + (1+\theta)(c - x_p) = c + \theta(c - x_p)$$

se presentan diferentes casos de transformaciones dentro de la reflexión:

- **Si** $f_m < f_r < f_p$, entonces el punto de reflexión es mejor que el proyectado, por lo que se considera a partir de ahora el nuevo simplex y se termina la iteración. Esto corresponde con el caso $\theta = 1$, la reflexión simple de **Fig. 4.1.**

- **Si** $f_r < f_m$, entonces el punto de reflexión contiene suficiente información sobre el mínimo buscado como para realizarse una **expansión** del simplex en su dirección. Esto corresponde al caso $\theta > 1$. Observar **Fig. 4.2.a**

Para ello hay que calcular el punto de expansión x_e de la forma:

$$x_e = c + \mu \cdot (x_r - c)$$

Evaluando x_e en f se tiene: si $f_e < f_r$, reemplazamos el punto x_p por x_e; en caso contrario, omitimos x_e y reemplazamos x_p por el punto x_r, como una reflexión simple. En ambos casos se termina la iteración.

- Si $f_r > f_{2p}$, entonces la dirección hacia x_r no es la correcta para la optimización, por lo que se realiza una **contracción**. Esto corresponde al caso $\theta \in [-1,1)$. Observar **Fig. 4.2.b** y **Fig. 4.2.c.**

Si se tiene que $\boldsymbol{f_r < f_p}$**,** se realiza una contracción hacia fuera, de la forma:

$$x_{cf} = c + \gamma \cdot (x_r - c)$$

Si, por el contrario, se tiene que $\boldsymbol{f_r \geq f_p}$**,** se realiza una contracción hacia dentro, de la forma:

$$x_{cd} = c + \gamma \cdot (c - x_p)$$

Tras calcular el punto de contracción correspondiente, se comparan sus imágenes con $\boldsymbol{f_r}$ **y** $\boldsymbol{f_p}$**,** respectivamente. Si se observa una mejora, se reemplaza x_p por el punto contraído calculado; y si no se mejora la pérdida del simplex, se ejecuta el tercer paso.

- **Tercer paso: Encogimiento**

Finalmente, si $f_r > f_{2p}$ y no se ha aplicado ninguna contracción, se efectuará un encogimiento del simplex. Esto consiste en mantener invariante x_m y generar un nuevo simplex tomando n nuevos puntos de la forma: $y_i = x_m + \beta \cdot (x_i - x_m)$ con $i \in [1, n+1] \setminus \{m\}$. Se muestra en **Fig. 4.2.d:**

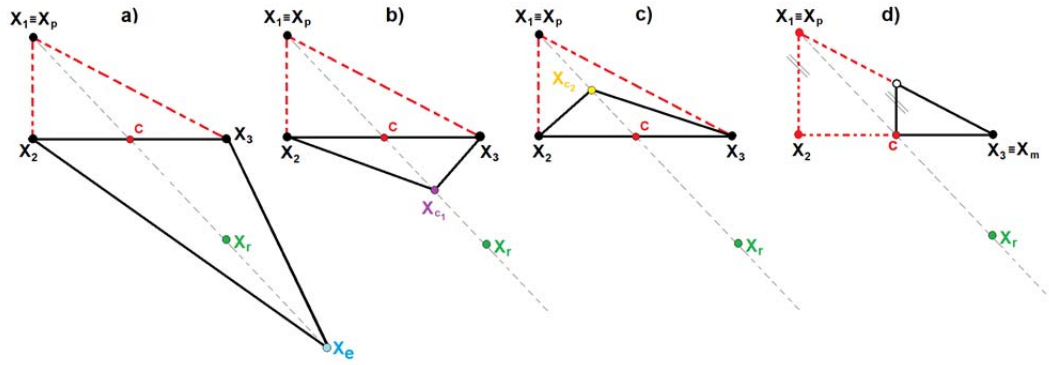


Figura 4.2: Transformaciones del simplex en el proceso de reflexión. (Elaboración propia)

Como se ha observado, el algoritmo va modificando los 3 puntos pertenecientes al simplex durante todo el proceso. Cuando el valor de la función a optimizar, en nuestro caso la función de pérdida, alcanza un valor mínimo preestablecido, el algoritmo finaliza habiendo encontrado el mínimo absoluto de la función $\boldsymbol{f}$.

Aplicado a la red neuronal, los parámetros que minimizan la función pérdida son aquellos cuyas predicciones se acercan más a la realidad.

4.1.2. Método del Descenso del Gradiente.

El método del Descenso del Gradiente es un algoritmo iterativo diseñado para resolver numéricamente el problema de minimización de una función multivariante no lineal sin restricciones. Este se considera un algoritmo de búsqueda indirecta, ya que para su aplicación es necesario el cálculo de la primera derivada de la función de pérdida.

Se trata de los métodos más utilizados en la programación de Deep Learning con Matlab debido a su escalabilidad, ya que existen comandos predeterminados que ejecutan este método de manera inmediata para un número elevado de dimensiones. Para aplicar esté método en Matlab, se utiliza la función:

$$[trainedNet, tr] = train(net, net.trainFcn)$$

la cual nos devuelve la red entrenada y el registro de entrenamiento.

Sea una función $f: A \subset \mathbb{R}^n \to \mathbb{R}$ tal que sus derivadas parciales existen en cierto punto $X_0 \subset A$, el gradiente de f en el punto X_0 es el vector conjunto de las derivadas parciales de dicha función evaluadas en X_0.
Por notación, esto se expresa:

$$\nabla f(X_0) \equiv \left(\frac{\partial f(X_0)}{\partial x_1}, \dots, \frac{\partial f(X_0)}{\partial x_n} \right)$$

El gradiente, evaluado en un punto inicial, indica la dirección en la cual la función crece de manera más rápida partiendo de dicho punto. Además, este indica la dirección ortogonal a las curvas de nivel de f, que son aquellas en las que la función tiene un valor constante.

Sin embargo, el objetivo del presente método es hallar el mínimo global de la función f. Por tanto, la dirección de búsqueda utilizada en el algoritmo será $d^k = -\nabla f(x^k)$, donde x^k refleja el punto inicial de la iteración k-ésima.

La iteración k-ésima del algoritmo es la transición entre los puntos x^k y x^{k+1} , y se expresa matemáticamente de la forma:

$$x^{k+1} = x^k + \lambda^k d^k = x^k - \lambda^k \cdot \nabla f(x^k)$$

donde λ^k es un escalar que determina la longitud de paso en la dirección d^k, más conocido como razón de aprendizaje.

La aplicación del algoritmo consiste, resumidamente, en los siguientes pasos:

- **Primer paso:** Elección de un punto inicial x^0
- **Segundo paso:** Cálculo de las derivadas parciales de f y de la dirección d^k.
- **Tercer paso:** Determinación del punto x^{k+1}.
- **Cuarto paso:** Comparación de las imágenes $f(x^k)$ y $f(x^{k+1})$. Si la diferencia entre estas es menor que una tolerancia previamente determinada, finaliza el algoritmo. En caso contrario, se ejecuta una nueva iteración desde el segundo paso, considerando como punto inicial el x^{k+1} calculado en el tercer paso.

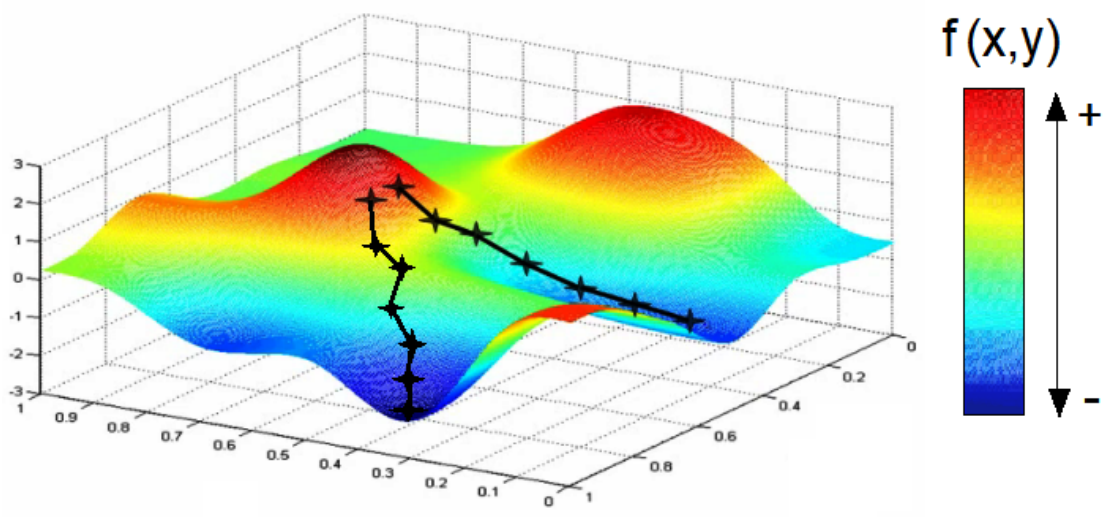


Figura 4.3: Algoritmo del Descenso del Gradiente hallando mínimo local y global. (Imagen de (Pérez Gómez, 2021) modificada)

La desventaja de utilizar el método de “Descenso del Gradiente” es que, a medida que el algoritmo se aproxima al punto mínimo buscado, se observa que su velocidad de convergencia se va disminuyendo, ya que $\nabla f \to 0$.
Además, siempre se tiene que verificar que el punto mínimo encontrado no sea un punto de silla. Esto es un punto en el cual el gradiente se anula y, sin embargo, no se trata de un extremo local de la función.

4.1.3. Método del Gradiente Conjugado.

El método del Gradiente Conjugado es un algoritmo iterativo de búsqueda indirecta utilizado para la minimización de funciones cuadráticas convexas. Se trata de una variación del método del Descenso del Gradiente en la cual la dirección de búsqueda ha sido optimizada mediante la estimación de la curvatura de la función.

Una función cuadrática convexa es una función $f: \mathbb{R}^n \to \mathbb{R}$ de la forma:

$$f(x) = \frac{1}{2}x^t Ax - x^t b$$

donde $x \in \mathbb{R}^n, b \in \mathbb{R}^n$ y $A \in \mathbb{R}^{n \times n}$ es una matriz simétrica definida positiva, lo cual significa que se cumple $x^* Ax > 0 \ \forall x \in \mathbb{R}^n$, siendo x^* el conjugado del vector x.
Como propiedad de las funciones cuadráticas se tiene que las sucesivas direcciones de búsqueda son conjugadas, lo que implica que el mínimo buscado se halla en un total de n pasos, donde n es el número de variables.

La diferencia respecto al método del Descenso del Gradiente es que este calcula los gradientes en cada iteración de manera independiente mientras que el método del Gradiente Conjugado realiza la combinación lineal de los gradientes, ∇f, en dicha iteración y la anterior. Esto provoca que la dirección de cada iteración sea siempre linealmente independiente de todos los vectores dirección anteriores, agilizando así la aproximación al valor buscado.

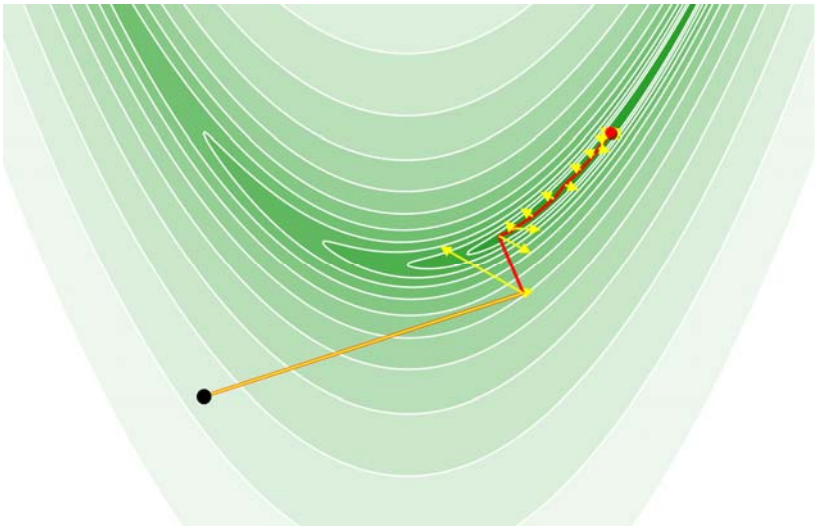


Figura 4.4: Algoritmo del Gradiente conjugado. (Frederickson, 2016)

La aplicación del algoritmo consiste, resumidamente, en los siguientes pasos:

- **Primer paso:** Elección de un punto inicial x^0 y cálculo del vector dirección en dicho punto. Como no existe una iteración anterior, se mantiene el cálculo realizado en el método de Descenso del Gradiente: $d^0 = -\nabla f(x^0)$
- **Segundo paso:** Determinación del punto inicial de la siguiente iteración:

$$x^{k+1} = x^k + \lambda^k d^k \;\; con\; k \in [0, n]$$

  siendo λ^k un escalar que determina la longitud de paso en la dirección d^k, más conocido como razón de aprendizaje.

- **Tercer paso:** Cálculo de la dirección de búsqueda de la siguiente iteración. Para ello primero se evalúa el punto inicial en la función, $f(x^{k+1})$, y en el gradiente, $\nabla f(x^{k+1})$. Posteriormente se realiza la combinación lineal de direcciones de la forma:

$$d^{k+1} = -\nabla f(x^{k+1}) + d^k \cdot \frac{\left(\nabla f(x^{k+1})\right)^t \cdot \nabla f(x^{k+1})}{\left(\nabla f(x^k)\right)^t \cdot \nabla f(x^k)}$$

- **Cuarto paso:** Verificación de la convergencia del método, esto es, asegurarse de que las imágenes $f(x^k)$ y $f(x^{k+1})$ progresan hacia el mínimo buscado. Este algoritmo finaliza cuando el módulo de la dirección, $|d^k|$, es tan pequeño como el margen elegido. En caso contrario, se itera de nuevo el algoritmo desde el segundo paso.

4.1.4. Método ADAM:

El método ADAM, Adaptive Moment Estimation, es un algoritmo iterativo de optimización estocástica cuyo principal objetivo son los problemas no convexos. Este método combina conceptos del método del descenso del gradiente estocástico y del método de momentos.

Su ventaja principal es que tiene la capacidad de adaptar automáticamente la razón de aprendizaje durante la optimización. Además, ADAM solo requiere gradientes de primer orden, por lo que se optimiza la memoria que necesita la red. Según sus creadores, ADAM está diseñado para combinar los métodos AdaGrad, especializado en gradientes dispersos; y RMSProp, especializado en el cálculo mediante momentos. ADAM se diferencia del método RMSProp porque calcula los parámetros mediante la media del primer y segundo gradiente de los parámetros. Además, ADAM incluye una corrección bias.

Los momentos del gradiente se definen como:

$$\boldsymbol{m_t = \beta_1 \cdot m_{\{t-1\}} + (1 - \beta_1) \cdot g_t} \quad ; \quad \boldsymbol{v_t = \beta_2 \cdot v_{\{t-1\}} + (1 - \beta_2) \cdot g_t^2}$$

Siendo $\boldsymbol{g_t}$ el gradiente de paso $\boldsymbol{t}$, $\boldsymbol{\beta_1, \beta_2} \in [\boldsymbol{0, 1}]$ un hiperparámetro que controla el peso del momento anterior

La aplicación del algoritmo consiste, resumidamente, en los siguientes pasos:

- **Primer paso:** Inicializar los momentos $\boldsymbol{m_o}$ y $\boldsymbol{v_0}$ como vectores de ceros.

- **Segundo paso:** En cada iteración $\boldsymbol{t}$ se realizan las siguientes acciones:

  · Calcular el gradiente $\boldsymbol{g_t}$ usando una muestra aleatoria de datos.

  · Actualizar el primer y el segundo momento con las fórmulas previas.

  · Corregir los momentos por el sesgo (bias) inicial de la forma:

$$\boldsymbol{m'_t = \frac{m_t}{1 - \beta_1^t}} \qquad \boldsymbol{v'_t = \frac{v_t}{1 - \beta_2^t}}$$

  · Por último, actualizar el valor los parámetros recogidos en $\boldsymbol{x}$:

$$\boldsymbol{x_t = x_{\{t-1\}} - \frac{\alpha}{\sqrt{v'_t + \varepsilon}} \cdot m'_t}$$

Siendo $\boldsymbol{\alpha}$ la razón de aprendizaje y $\boldsymbol{\varepsilon}$ una constante cercana a 0.

4.2. Convoluciones

Las Redes Neuronales Convolucionales son un modelo de Deep Learning cuya función principal es la detección de características y patrones en imágenes. Esta tarea se lleva a cabo en las capas convolucionales de la red, explicadas en la **Sección 3.3**, las cuales consisten en una sucesión de convoluciones que actúan como filtros sobre la señal de entrada. La señal de entrada puede ser de imagen o de audio.

En el área del análisis funcional, una convolución es un operador matemático que realiza la combinación de dos señales, $f(t)$ y $g(t)$, en una nueva señal, $h(t)$, la cual representa una medida de la magnitud en la que se superponen f y una versión trasladada e invertida de g.

Sean las señales $f(t)$ y $g(t)$, la convolución de f y g en un instante de tiempo, $t = \tau$, se calcula siguiendo tres pasos clave. A continuación se ilustran dichos pasos con un ejemplo sencillo en el cual:

$$f(t) = \begin{cases} 1 & si\ -1 \leq t \leq 1 \\ 0 & en\ otro\ caso \end{cases} \qquad g(t) = \begin{cases} t & -1 \leq t \leq 1 \\ -t+2 & 1 \leq t \leq 2 \\ -t-2 & -2 \leq t \leq -1 \\ 0 & en\ otro\ caso \end{cases}$$

- **Primer paso: Ajuste de señales.**

  Primero se fija la señal $f(t)$. Por otro lado se refleja y traslada la señal $g(t)$ un cierto tiempo τ, de manera que se realiza el cambio de variable: $t \rightarrow -(t-\tau)$:

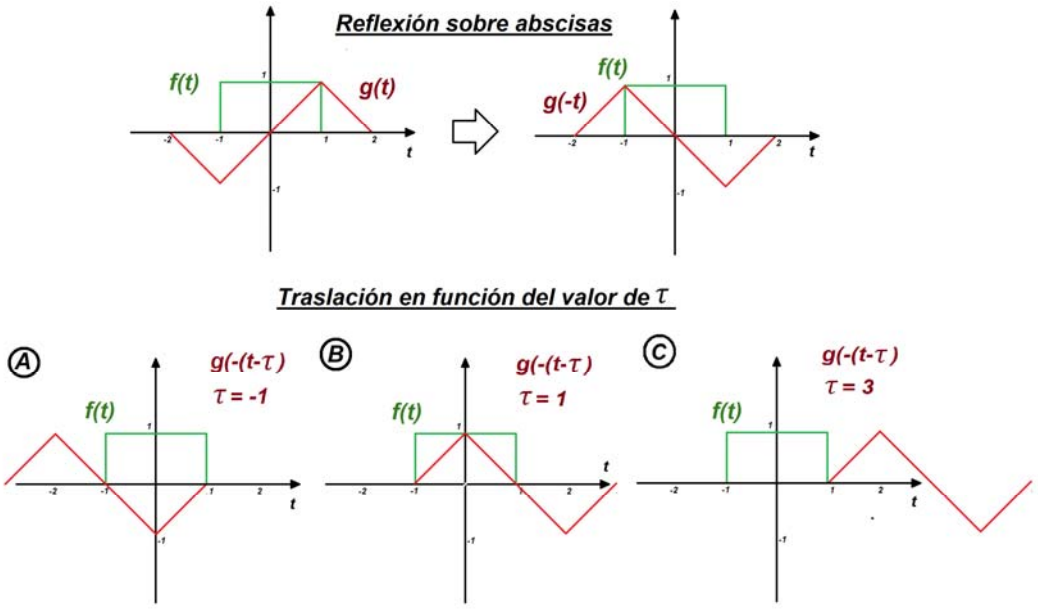


Figura 4.5: Ejemplo del ajuste de señales para la convolución. (Elaboración propia)

- **Segundo paso: Producto de señales.**

Una vez calculadas las señales $f(t)$ y $g(-(t-\tau))$, se realiza la multiplicación de las señales como un producto de funciones normal, es decir, punto a punto.

$$f(t) \cdot g(-(t-\tau))$$

Se observa en la **Fig. 4.6., que** el producto resulta trivial en nuestro ejemplo, coincidiendo este con la función $g(-(t-\tau)$ en el intervalo $[-1,1]$.

- **Tercer paso: Cálculo del área del producto.**

Mediante la integración en la variable temporal, se calcula el área del producto $f(t) \cdot g(-(t-\tau))$ en todo su dominio, obteniendo así la convolución buscada en el instante $t = \tau$:

$$h(\tau) = (f * g)(\tau) \equiv \lim_{T\to\infty} \int_{-T}^{T} f(t) \cdot g(-(t-\tau))\, dt$$

En la **Fig. 4.6.** se muestra en morado el área calculada en cada instante τ de manera gráfica, así como su valor concreto al lado:

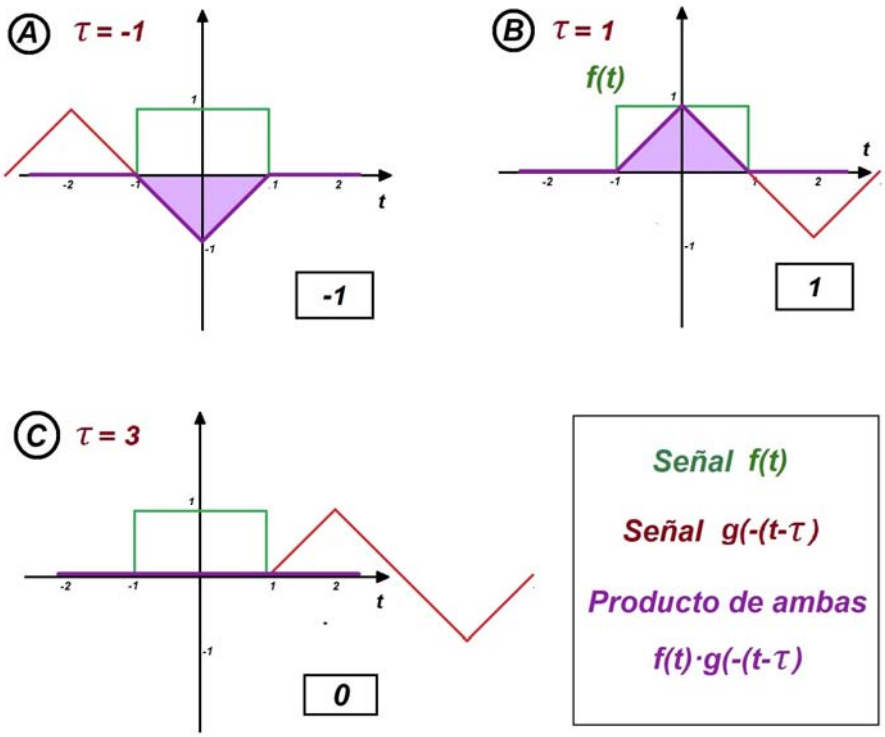


Figura 4.6: Ejemplo del producto de señales y el cálculo de su área. (Elaboración propia)

Normalmente, las convoluciones interesa aplicarlas para todo instante de tiempo, es decir, tomando $\tau \in (-\infty, \infty)$. Representando los valores de las áreas calculadas en función de ciertos valores del parámetro τ, se obtiene una aproximación discreta a la convolución general de f y g:

$$h(t) = (f * g)(t) \equiv \lim_{T\to\infty} \sum_{\tau=-(T-1)}^{T-1} f(t) \cdot g\big(-(t-\tau)\big) \quad \forall t$$

Para obtener la convolución exacta, es decir continua, se debe aplicar cálculo infinitesimal. Para ello, resulta muy útil realizar el cambio de variable de doble dirección: $t \leftrightarrow \tau$, lo cual simplifica la notación.

De esta manera, la señal producto de la convolución, $h(t)$, se puede expresar de la forma:

$$h(t) = (f * g)(t) \equiv \int_{-\infty}^{\infty} f(\tau) \cdot g(t-\tau)\, d\tau \quad \forall t$$

Se presentan en **FIg. 4.6.** la aproximación discreta de la convolución general y su equivalente infinitesimal:

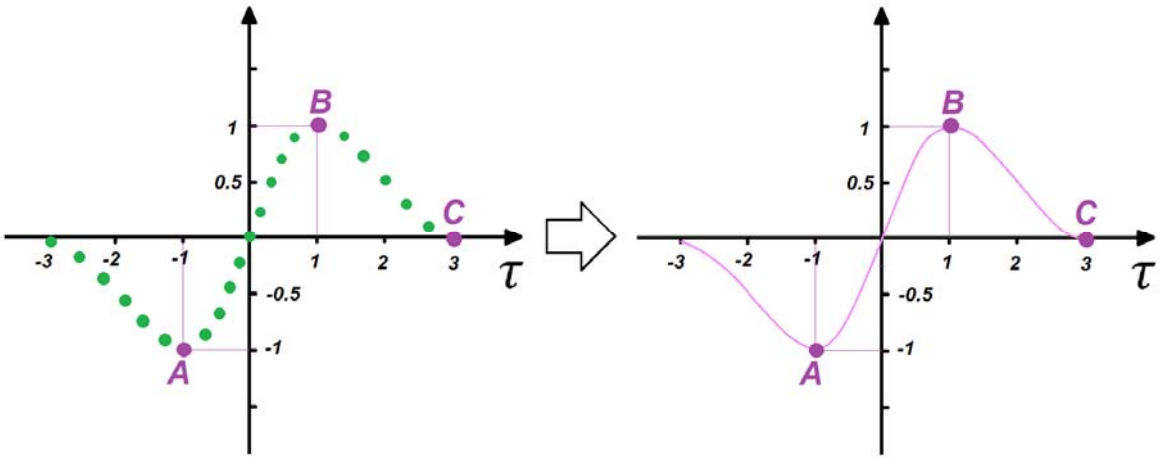


Figura 4.7: Convoluciones discreta y continua de las señales ejemplo $f(t)$ y $g(t)$.
(Elaboración propia)

4.2.1. Convolución aplicada a señales de audio

Una señal de audio analógica consiste en una función de carácter sinusoidal $s(t) : \mathbb{R} \to \mathbb{R}$, continua en el dominio temporal, la cual contiene información sobre la amplitud de la señal en cada instante t.

Estas señales pueden ser periódicas, de forma simple o compleja, o aperiódicas, las cuales comprenden señales continuas y señales transitorias. Vamos a reducir la explicación al caso sencillo de una señal periódica simple, la cual se expresa como:

$$s(t) = A \cdot sen(2\pi f t + \varphi)$$

donde A es la amplitud de la onda, f la frecuencia (Hz) y φ la fase de la onda.

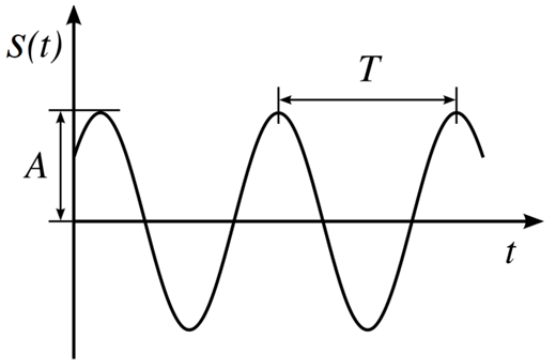


Figura 4.8: Señal de audio analógica s(t)
(Elaboración propia)

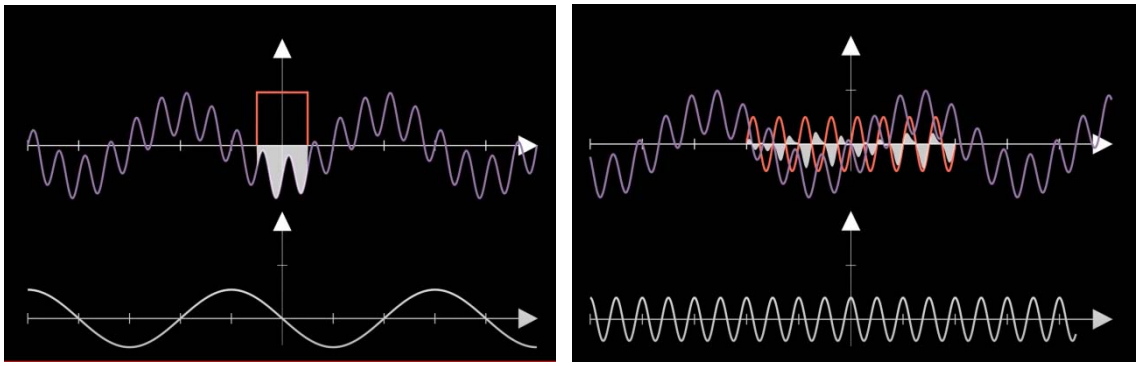
Por un lado, la amplitud representa la perturbación de la energía de la onda, lo que se traduce al volumen de la onda sonora. Por otro lado, la frecuencia representa el número de perturbaciones de la onda por unidad de tiempo, lo cual se interpreta como el tono de la señal sonora. Además, la fase de la onda representa la medida en la que la onda se ve desplazada respecto al origen de coordenadas, de modo que será la responsable del movimiento de la onda sobre el eje temporal.

El operador convolución se utiliza sobre una señal de audio con el objetivo de aplicar a dicha señal sistemas lineales invariantes en el tiempo, entre los que destacan los filtros digitales.

Como se explica en el apartado anterior, la convolución se realiza entre dos señales temporales $f(t)$ y $g(t)$, las cuales consisten en la señal de audio que se pretende estudiar y una señal fija que define el filtro a aplicar.

El resultado de este proceso es una tercera señal de audio que contiene la señal de entrada transformada en función del filtro.

Esto se entiende mejor de manera visual:



Figuras 4.9: Ejemplos de convolución sobre una señal simple de audio. (Sanderson, 2018)

En las figuras previas se observa la aplicación de diferentes filtros mediante convolución. Estas consisten en: una señal morada, la cual representa el audio de entrada; una señal roja, la cual representa el filtro o 'kernel'; y, en la parte inferior de las figuras, la señal de salida del proceso de convolución.

En el caso de la **Fig. 4.9.a,** se trata de la aplicación de un filtro de eliminación de agudos, es decir, de disminución de la frecuencia. Por otro lado, en **Fig. 4.9.b** se muestra la aplicación de un filtro de eliminación de graves. Cabe tener en cuenta que los filtros digitales no consisten únicamente en modificaciones tonales de una señal, sino que se tienen otros filtros como el eco, la reverberación o la modulación de frecuencia.

En el ejemplo anterior, mediante la comparación de las señales de salida respecto a la señal original, se pueden apreciar los cambios conseguidos en la frecuencia de las ondas. Sin embargo, esto no resulta tan sencillo cuando tratamos con ondas sonoras más complejas. Una onda sonora compleja consiste en la superposición de varias ondas sinusoidales simples, de manera que cada una de ellas, conocidas como ondas parciales, describe un sonido puro diferente.

Por ello se utilizan métodos matemáticos, principalmente las transformadas de Fourier, para fragmentar la señal compleja en un conjunto de señales simples.

4.2.1.1. La transformada de Fourier

La transformada de Fourier es una aplicación matemática consistente en la transformación de una señal entre el dominio temporal y el dominio de frecuencia. De esta manera se puede visualizar desde diferentes perspectivas la información de la onda y, además, puede llevarse a cabo la fragmentación de una señal compleja en un conjunto de señales simples

El aspecto más importante que relaciona la transformada de Fourier y las convoluciones es el Teorema de Convolución. Este establece que la convolución de dos señales en el dominio temporal es proporcional a la multiplicación de las señales en el dominio de frecuencia por el escalar $\sqrt{2\pi}$. Matemáticamente esto se expresa como:

$$\mathcal{F}\big(s(t) * g(t)\big) = \sqrt{2\pi} \cdot \mathcal{F}\big(s(t)\big) \cdot \mathcal{F}\big(g(t)\big) = \sqrt{2\pi} \cdot \hat{s}(f) \cdot \hat{g}(f)$$

La idea base de los estudios de Fourier es que toda función periódica puede ser expresada como una suma trigonométrica de senos y cosenos. Sea una señal $s \in C^{\infty}(\mathbb{R})$ periódica, con periodo T, esta se puede expresar como una serie de Fourier de la forma:

$$s(t) \cong \frac{A_0}{2} + \sum_{j=1}^{\infty} A_j \cdot \cos\left(\frac{2n\pi}{T} t\right) + B_j \cdot sen\left(\frac{2n\pi}{T} t\right)$$

Veamos cómo se define matemáticamente la transformada de Fourier de $s(t)$:

$$\hat{s}(f) = \int_{-\infty}^{\infty} s(t) \cdot e^{-2\pi i f t} dt$$

El concepto básico de su acción es, al igual que en la convolución, cuantificar en qué medida dos ondas se superponen. En concreto, la superposición de la señal $s(t)$ con las ondas simples asociadas a cada frecuencia, lo cual nos permitirá identificar en qué señales simples se descompone nuestra señal compleja $s(t)$.

Cabe recordar que $e^{-2\pi i f t}$ es un número complejo expresado en forma exponencial, el cual se puede expresar trigonométricamente de la forma:

$$e^{-2\pi i f t} = \cos(-2\pi f t) + i \cdot sen(-2\pi f t) = \cos(2\pi f t) - i \cdot sen(2\pi f t)$$

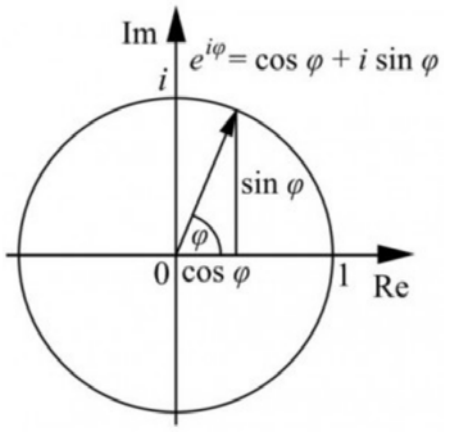


Figura 4.10: Vista en el plano complejo (Elaboración propia)

Visto en el plano complejo, $e^{-2\pi ift}$ representa el recorrido de una partícula dando una vuelta completa a un círculo de radio unidad con un periodo $T = \frac{1}{f}s$.

Por tanto, dada una frecuencia fija f_0, se observa que el número complejo $e^{-2\pi ift}$ representa la posición de la partícula en el círculo en el instante t $\forall t \in \mathbb{R}$.

Entonces, el producto $s(t) \cdot e^{-2\pi ift}$ indicará la posición de la señal $s(t)$ enrollada alrededor del origen de coordenadas. Veamos **Fig. 4.9** para entenderlo mejor:

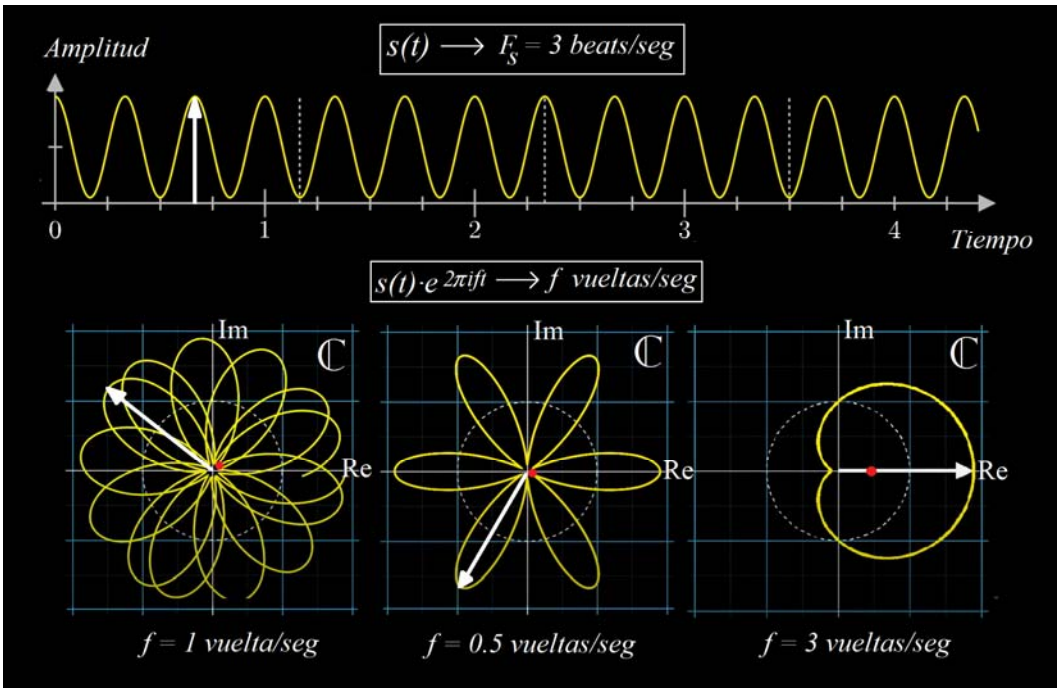


Figura 4.11: Señal enrollada en el plano complejo. (Sanderson, 2018)

Teniendo en cuenta que la señal de entrada $s(t)$ está fija, el dominio de frecuencia de la transformada $\hat{s}(t)$ se tratará de la frecuencia de enrollado f.

Se observa una diferencia en las distribuciones de la onda en el plano complejo en función de dicha frecuencia, las cuales definen de manera abstracta la superposición de la señal $s(t)$ y la onda simple asociada a la frecuencia f.

Una manera de cuantificar dichas diferencias es tomar un conjunto de puntos a lo largo de las figuras inferiores y calcular el centro de masas, o centroide, asociado a cada figura. Este viene representado en **Fig. 4.9** como un punto rojo.

Tomado dentro de cada figura un conjunto de $\boldsymbol{N}$ puntos, $\boldsymbol{P_f} = \{\boldsymbol{P_{f_1}}, .., \boldsymbol{P_{f_N}}\}$, de manera que cada uno de ellos tiene una dirección asociada $\vec{\boldsymbol{e}}$, el cálculo matemático del centroide consiste en la expresión matemática:

$$\boldsymbol{C(f)} = \frac{\boldsymbol{1}}{\boldsymbol{N}} \sum_{\boldsymbol{k=1}}^{\boldsymbol{N}} \boldsymbol{P_{f_k}} \cdot \vec{\boldsymbol{e}}_{\boldsymbol{k}}$$

Ahora bien, tomando un número de puntos elevado, de forma que $\boldsymbol{N} \rightarrow \infty$, el conjunto de puntos se convierte en la señal $\boldsymbol{s(t)}$ continua a lo largo del dominio temporal, por lo que se puede expresar el centroide como:

$$\boldsymbol{C(f)} = \frac{\boldsymbol{1}}{\boldsymbol{t_2 - t_1}} \int_{\boldsymbol{t_1}}^{\boldsymbol{t_2}} \boldsymbol{s(t)} \cdot \boldsymbol{e^{-2\pi i f t} dt}$$

donde $\boldsymbol{e^{-2\pi i f t}}$ es la dirección compleja asociada a $\boldsymbol{s(t)}$ $\forall \boldsymbol{t} \in [\boldsymbol{t_1}, \boldsymbol{t_2}]$.

Por tanto el centroide la magnitud en la que la onda $\boldsymbol{s(t)}$ se superpone con la onda sinusoidal asociada a la frecuencia $\boldsymbol{f}$ dentro del intervalo temporal $[\boldsymbol{t_1}, \boldsymbol{t_2}]$.

Como el valor de $\boldsymbol{C(f)}$ es cercano a 0 para la mayoría de frecuencias del dominio, obviando la división entre el tamaño del intervalo temporal se consiguen destacar aquellas frecuencias con mayor magnitud. Finalmente, tomando todo el dominio temporal, se obtienen los coeficientes complejos de la Transformada de Fourier:

$$\boldsymbol{\hat{s}(f)} = \int_{\boldsymbol{t_1}}^{\boldsymbol{t_2}} \boldsymbol{s(t)} \cdot \boldsymbol{e^{-2\pi i f t} dt}$$

Aunque dichos coeficientes sean complejos, nos interesa sobretodo la parte real ellos. Esbozando sus valores reales respecto al dominio de frecuencias se obtiene el espectro de magnitudes de la onda, el cual se muestra en **Fig. 4.10:**

Figura 4.12: Espectro de magnitudes de la onda de **Fig. 4.11. (Sanderson, 2018)**

Se observa que el pico en **Fig. 4.12** coincide con la frecuencia $\boldsymbol{F_s}$ de la onda representada en **Fig.4.11**. Esto significa que la onda $\boldsymbol{s(t)}$ se puede descomponer concretamente en una onda simple sinusoidal de frecuencia 3, es decir, se descompone en sí misma, ya que se trata de una onda simple.

En **Fig.4.12** se observa un pico notable en el eje vertical, el cual señala la frecuencia de las ondas simples en las que puede ser descompuesta la señal original. En este caso únicamente existe un pico que se encuentra en la frecuencia $\boldsymbol{F_s}$, por lo que la señal $\boldsymbol{s(t)}$ sólo puede descomponerse en sí misma.

El resultado anterior resulta obvio, ya que $\boldsymbol{s(t)}$ se trata de una señal simple. Sin embargo, esta información resulta muy útil a la hora de analizar señales complejas. Veamos un ejemplo:

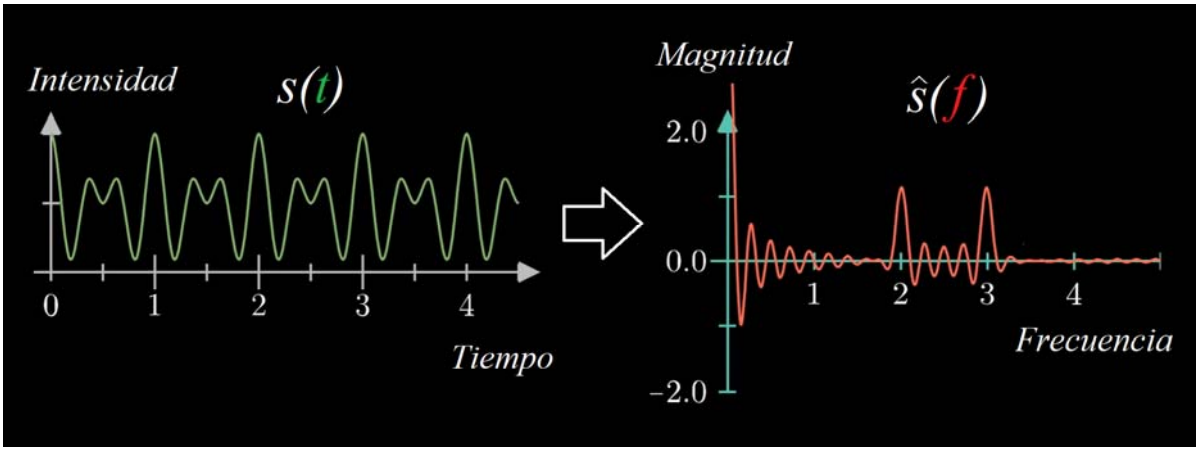


Figura 4.13: Espectro de magnitudes de una onda compleja. **(Sanderson, 2018)**

En el caso de **Fig.4.13** se tienen dos picos en el espectro de magnitudes, correspondientes a las dos ondas simples, de frecuencias 2 y 3, que componen la señal compleja $\boldsymbol{s(t)}$.

4.2.2. Convolución aplicada a señales de imagen

Aplicar una convolución a una imagen consiste en tomar como señales de entrada: el mapa de características de una imagen, I; y una señal fija conocida como núcleo o 'kernel', la cual determina el filtro que ejercerá la convolución.

Un mapa de características consiste en una estructura vectorial multidimensional que recoge los datos de cada pixel de una imagen.

En caso de que la imagen de entrada sea en blanco y negro, se utilizan matrices bidimensionales, cuyas coordenadas ubican cada píxel en cuanto a la anchura y la altura de la imagen. Los valores de la matriz se encuentras en el intervalo $[0,255]$, e indican el color de cada pixel en la escala de grises, siendo 0 el negro y 255 el blanco. En este caso, el 'kernel' va a consistir en otra matriz bidimensional, la cual normalmente es de pequeñas dimensiones como $3x3, 5x5$ o $7x7$. Como las operaciones de convolución deben realizarse entre matrices de mismas dimensiones, la imagen de entrada se divide en regiones conocidas como campos receptivos. Posteriormente se realiza la convolución iteradamente sobre los diferentes campos receptivos mediante el producto celda a celda.

Se observa el proceso en **Fig.4.14.:**

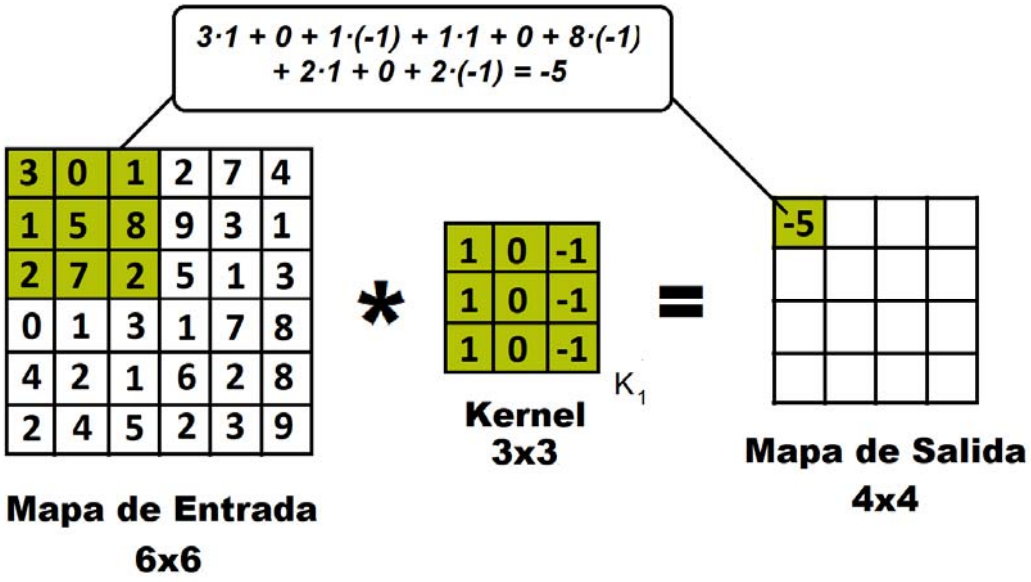


Figura 4.14: Convolución bidimensional de una imagen en Blanco y Negro.
(Elaboración propia)

En cuanto a las dimensiones de los mapas se tiene el lema: Sea la entrada de una convolución un mapa de características de tamaño $m \times n$ y sea un kernel o núcleo de tamaño $u \times v$. Entonces se tiene que, tras la convolución, las dimensiones del mapa de salida son: $(m-u+1) \times (n-v+1)$.
En el caso de usar imágenes a color, el mapa de características será una matriz tridimensional, esto es un conjunto de matrices bidimensionales superpuestas, donde la tercera dimensión representa la proporción de colores RGB (rojo, verde y azul) de cada píxel.

De esta forma se tienen datos respecto a la ubicación y el color total de cada pixel, el cual se calcula sumando los valores en cada matriz bidimensional: R, G y B. Por su parte, el 'kernel' también será una matriz tridimensional con sus matrices bidimensionales K_1, K_2 y K_3, de manera que se cada una ellas está asociada con las matrices R, G y B. La convolución se lleva a cabo iterando sobre los campos receptivos de la entrada, de manera similar al caso bidimensional, entre los pares $(R, K_1), (G, K_2)\ y\ (B, K_3)$. Posteriormente, los valores de las convoluciones son sumados, resultando así una matriz bidimensional.
El proceso de convolución se muestra en **Fig.4.15.:**

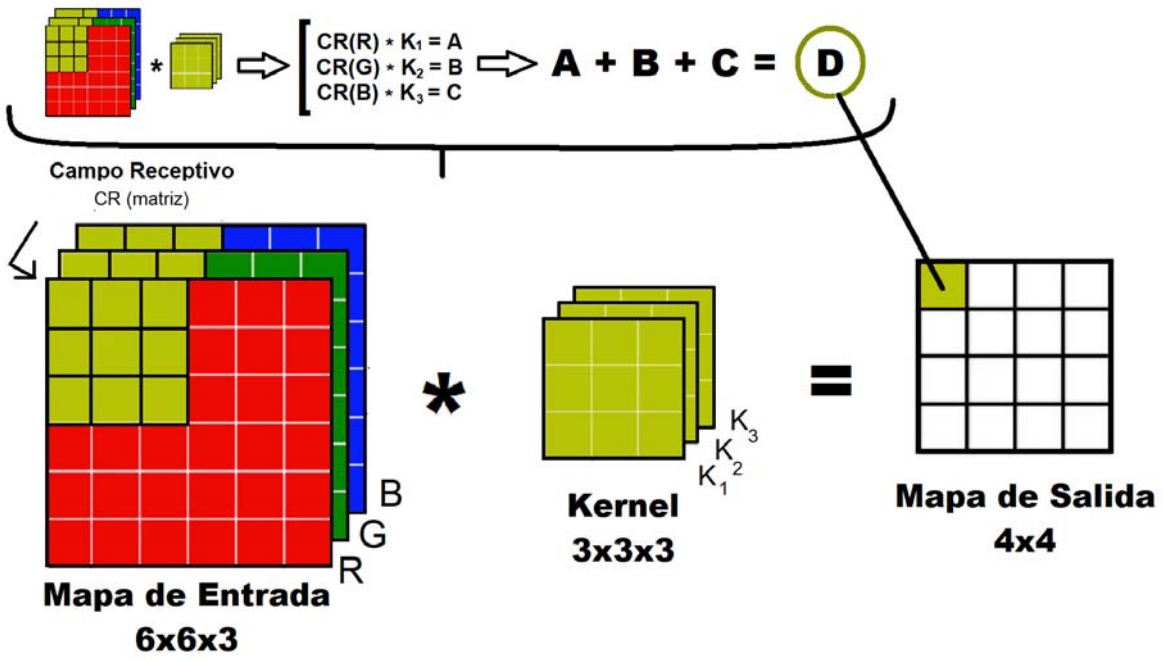


Figura 4.15: Convolución tridimensional de una imagen a color.
(Elaboración propia)

4.2.2.1. Ejemplo de convolución de imágenes

Como hemos visto anteriormente, emplear el operador convolución sobre una señal de entrada consiste en aplicarle a esta un filtro. Dicho filtro viene determinado por los valores del 'kernel' elegido. Además, las dimensiones del 'kernel' influyen en la calidad de aplicación del filtro ya que, como las dimensiones del mapa de salida son $(m-u+1)\times(n-v+1)$, cuanto más pequeño sea el 'kernel', más grande será la imagen de salida.

En el caso de las señales imagen, un ejemplo de filtros son el enfoque, el desenfoque o la de detección de bordes. Para ilustrar estos ejemplos vamos a tomar como señal de entrada una imagen, a color, del Taj Mahal. Además vamos a considerar los siguientes 'kernels' de dimensiones 5×5:

$$K_1=\begin{pmatrix}0&0&0&0&0\\0&1&1&1&0\\0&1&1&1&0\\0&1&1&1&0\\0&0&0&0&0\end{pmatrix},\ K_2=\begin{pmatrix}0&0&0&0&0\\0&0&-1&0&0\\0&-1&5&-1&0\\0&0&-1&0&0\\0&0&0&0&0\end{pmatrix},\ K_3=\begin{pmatrix}0&0&0&0&0\\0&0&1&0&0\\0&1&-4&1&0\\0&0&1&0&0\\0&0&0&0&0\end{pmatrix}$$

Como la imagen de entrada es un mapa tridimensional, se consideran 'kernels' tridimensionales. En este caso se trata de filtros simples que se aplican a cada color RGB por igual, por lo que consideramos cada capa de los kernels igual.

a) K_1: Difuminación b) K_2: Enfoque c) K_3: Detección de bordes

Figura 4.16: Ejemplos de filtros a imagen por convolución (GIMP, 2002)

4.3 Funciones de activación

El concepto de función de activación, dentro del contexto de las redes neuronales, consiste en una aplicación matemática utilizada para eliminar la linealidad del modelo. El objetivo de su uso es acercar los datos del modelo a la realidad, de forma que se aumenta la complejidad del aprendizaje en las capas internas de la red.

Para ello se introducen pequeñas alteraciones en los datos tras cada convolución. Además, estas pueden ser diferentes en cada capa, ya que cada tipo de función de activación tiene sus propias características y aplicaciones.

Las funciones de activación más utilizadas son las funciones: Sigmoide, Tangente Hiperbólica, ReLU, Leaky ReLU, ELU y Softmax. Veamos cada una de ellas:

- **Función de activación Sigmoide:**

Se trata de una función no lineal que toma cualquier número real como entrada y lo devuelve comprimido a un valor dentro del intervalo $[\mathbf{0}, \mathbf{1}]$. Es por ello que su uso principal reside en la clasificación binaria de valores, de forma que el valor de salida representa la probabilidad, del valor de entrada, de pertenecer a una de las clases.

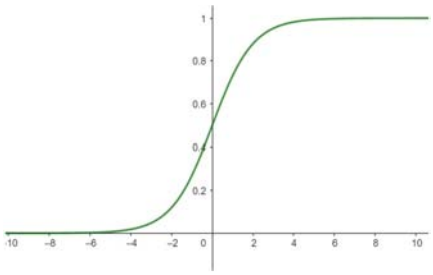


Figura 4.17: Función sigmoide (Elaboración propia)

Su expresión matemática es: $\boldsymbol{\sigma(x) = \frac{1}{1+e^{-x}}}$

Como se observa en la **Fig. 4.17**, cuando los valores de entrada son muy altos o muy bajos, las salidas se acumulan en los valores 0 y 1. Dicha acumulación provoca que, durante el entrenamiento de la red mediante el Método del Descenso del Gradiente, los gradientes de los pesos tiendan a 0, dificultando la convergencia del algoritmo. Dicho en otras palabras, la función se vuelve insensible a pequeñas variaciones en la entrada, lo cual limita el aprendizaje de patrones. Esto se conoce como 'Problema de Saturación en las regiones extremo'.

Tangente Hiperbólica (tanh):

Se trata de una función no lineal que toma cualquier número real como entrada y lo devuelve comprimido dentro del intervalo $[-\mathbf{1}, \mathbf{1}]$. Su característica principal es la simetría respecto al origen, lo cual es óptimo para su aplicación en las capas intermedias de la red en las que se busque una media nula de activaciones.

Su expresión matemática es:

$$\boldsymbol{tanh}(\boldsymbol{x}) = \frac{\boldsymbol{e^x - e^{-x}}}{\boldsymbol{e^x + e^{-x}}}$$

Una ventaja de la función $\boldsymbol{tanh}$ es que es diferenciable en todo el dominio, por lo que ayuda al buen rendimiento de la red a la hora de calcular los gradientes.

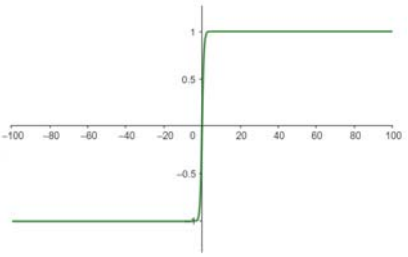


Figura 4.18: Función sigmoide
(Elaboración propia)

Sin embargo, puede presentar problemas de acumulación en los extremos, como la función Sigmoide.

- **Función de activación ReLU:**

La función Rectified Linear Unit, conocida por su abreviación ReLU, es una función no lineal la cual transforma los valores de entrada negativos a 0 y devuelve los valores positivos sin modificaciones.

Su expresión matemática es:

$$\boldsymbol{f(x) = \max(0, x) = \begin{cases} 0 & si\ x < 0 \\ x & si\ x \geq 0 \end{cases}}$$

La función ReLU es popular por ser simple, eficiente y evitar la saturación en la región positiva. Además, su gradiente, constante en cada región del dominio, previene el crecimiento exponencial del tiempo de entrenamiento.

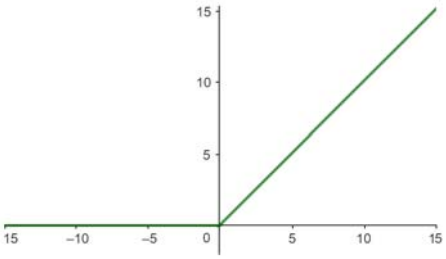


Figura 4.19: Función ReLU
(Elaboración propia)

Sin embargo, la función ReLU también tiene sus limitaciones. Por un lado, se encuentra el problema de la 'muerte de neuronas', el cual consiste en que, cuando la entrada de una neurona es continuamente negativa, la neurona no se activa nunca. Esto unido a la falta de simetría de la función, puede provocar cierto sesgo y mala calidad en la salida de la red. Por otro lado, la función ReLU no está acotada superiormente, por lo que puede tener valores de salida excesivamente grandes que no resultan útiles en ciertas aplicaciones.

- **Funciones de activación Leaky ReLU y ELU:**

Ambas funciones son versiones mejoradas de la función ReLU, solucionándose el problema de la 'muerte de neuronas'. En el caso de la función Leaky, esto se consigue mediante un coeficiente rectificativo $\boldsymbol{\alpha}$, esto es, una pequeña pendiente fija $\boldsymbol{\alpha > 0}$ en la región negativa del dominio. En cuanto a la función ELU, la pendiente de la región negativa se trata de una pendiente exponencial, lo cual suaviza la función haciéndola totalmente diferenciable.

Sus expresiones matemáticas son:

$$\boldsymbol{Leaky\ ReLU(x) = \begin{cases} \alpha x & si\ x < 0 \\ x & si\ x \geq 0 \end{cases}}$$

$$\boldsymbol{ELU(x) = \begin{cases} \alpha(e^x - 1) & si\ x < 0 \\ x & si\ x \geq 0 \end{cases}}$$

donde $\boldsymbol{\alpha} \in \mathbb{R}$ es el coeficiente rectificador.

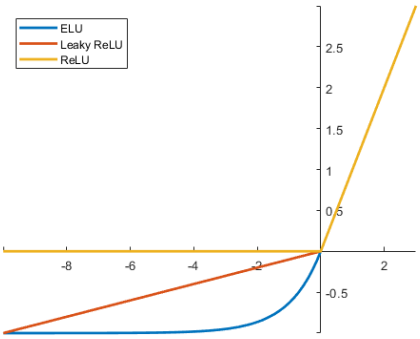


Figura 4.20: Comparación de las funciones ReLU, ELU y Leaky ReLU (Shenoy, 2019)

Se aprecian las modificaciones en la pendiente de la región negativa en **Fig. 4.20.**

Cabe destacar que ambas versiones destacan por su buen desempeño en las capas intermedias de la Redes Neuronales Convolucionales.

- **Función SoftMax:**

La función SoftMax suele encontrarse en la capa de salida de una red neuronal en aquellos casos que se busca un distribución de probabilidad. Esta toma un vector de entrada con números reales y devuelve un vector probabilidad cuyos elementos se encuentran en el intervalo $[\mathbf{0}, \mathbf{1}]$ y entre todos ellos suman 1.

$$\boldsymbol{SoftMax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}}$$

Su uso más frecuente es en problemas de clasificación multiclase, donde se busca asignar una etiqueta de clase a la entrada en función de sus características. En estos casos, la función devuelve la probabilidad de la entrada de pertenecer a cada una de las clases, lo cual se puede observar en el ejemplo de **Fig. 3.5.**

4.3.1. Batch Normalization[2]

La normalización por lotes es una técnica de optimización utilizada frecuentemente en redes convolucionales con el objetivo de mejorar la velocidad y la estabilidad del entrenamiento de la red.

En Deep Learning, el entrenamiento de las redes no se realiza para todo el conjunto de datos de entrenamiento, si no que se lleva a cabo una división previa de dichos datos en diferentes batches, o lotes. Estos son subconjuntos de datos, de tamaño fijo, los cuales se procesan de forma independiente, permitiendo a la red ajustar gradualmente los pesos.

La normalización por lotes se realiza antes de cada capa de activación y consiste en redistribuir las activaciones de la red en busca de reducir diferencias de distribución entre los diferentes lotes. Matemáticamente esto consiste en la tipificación de los datos de la siguiente manera:

Sea $\boldsymbol{D} = \{\boldsymbol{x_1}, \dots, \boldsymbol{x_n}\}$ el conjunto de valores de un batch, primero se calculan la media y la varianza del lote:

$$\boldsymbol{\mu_D} = \frac{\boldsymbol{1}}{\boldsymbol{n}} \cdot \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x_i} \quad \boldsymbol{y} \quad \boldsymbol{\sigma_D^2} = \frac{\boldsymbol{1}}{\boldsymbol{n}} \cdot \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} (\boldsymbol{x_i} - \boldsymbol{\mu_D})^{\boldsymbol{2}}$$

Una vez calculados, se procede a tipificar los valores:

$$\widehat{\boldsymbol{x}}_{\boldsymbol{i}} = \frac{\boldsymbol{x_i} - \boldsymbol{\mu_D}}{\sqrt{\boldsymbol{\sigma_D^2}}} \quad \forall \boldsymbol{i} \in [\boldsymbol{1}, \boldsymbol{n}]$$

Por último se realiza el reescalado de los valores en función de los parámetros de aprendizaje $\boldsymbol{\beta}$ $\boldsymbol{y}$ $\boldsymbol{\theta}$, obteniendo así los valores a utilizar:

$$\boldsymbol{BN_{\beta,\theta}(x_i)} = \boldsymbol{\theta} \cdot \widehat{\boldsymbol{x}}_{\boldsymbol{i}} + \boldsymbol{\beta}$$

Cabe mencionar que la normalización por lotes ayuda a reducir el overfitting en la red actuando como regulador de los datos.

[2] https://keepcoding.io/blog/batch-normalization-red-convolucional/

4.4. Agrupamientos (Pooling)

Las capas de agrupamiento, o pooling, son una técnica utilizada en las Redes Neuronales Convolucionales cuyo objetivo es reducir la dimensión espacial de los mapas de características generados por las capas convolucionales. Normalmente, estas capas se ubican tras las capas convolucionales con activación.

Dichas técnicas se basan en la partición del mapa de características en ventanas, generalmente no solapadas y de tamaño fijo, en las cuales se prioriza la información más relevante según el tipo de pooling elegido.

Los principales tipos de agrupamiento son: el Max Pooling, el Min Pooling y el Average Pooling. En la técnica Max Pooling se toma el valor máximo de los datos dentro de cada ventana, mientras que en el Min Pooling se toman los valores mínimos. Por otro lado, en el Average Pooling se calcula la media de los valores en cada ventana.

En **Fig. 4.21** se observa la metodología de la agrupación por ventanas, mostrándose la aplicación del Max Pooling para señales de audio, donde el valor tomado se representa en rojo en cada ventana unidimensional, y para señales de imagen, como se muestra:

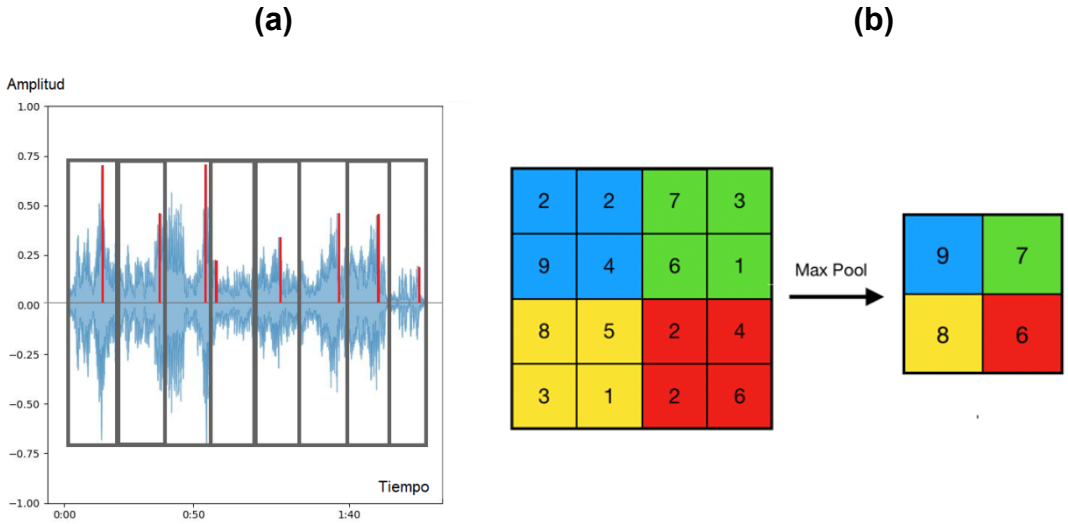


Figura 4.21: Max Pooling aplicado a señales de audio (a) e imagen (b)

(a) Elaboración propia (b) (Savyakhosla, 2023))

Cada tipo de pooling se utiliza en función de las características de la señal.
El Average Pooling se utiliza normalmente para la clasificación de objetos, ya que suaviza la señal acelerando el funcionamiento de la red. Sin embargo, esto dificulta la identificación de trazos concretos.
Por otro lado, el uso del Max Pooling y del Min Pooling es ventajoso en la identificación de objetos o trazos, pero ralentiza el funcionamiento de la red. El primero toma los sonidos con mayor volumen o los píxeles más brillantes, lo cual es útil para la detección de objetos sobre fondo oscuro. Por otro lado, el Mín Pooling toma los sonidos bajos y los píxeles más ocutos, resultando útil para la identificación de los sonidos de fondo y para la detección de objetos sobre un fondo claro.

Los resultados de estas técnicas se observan mejor sobre señales de imagen:

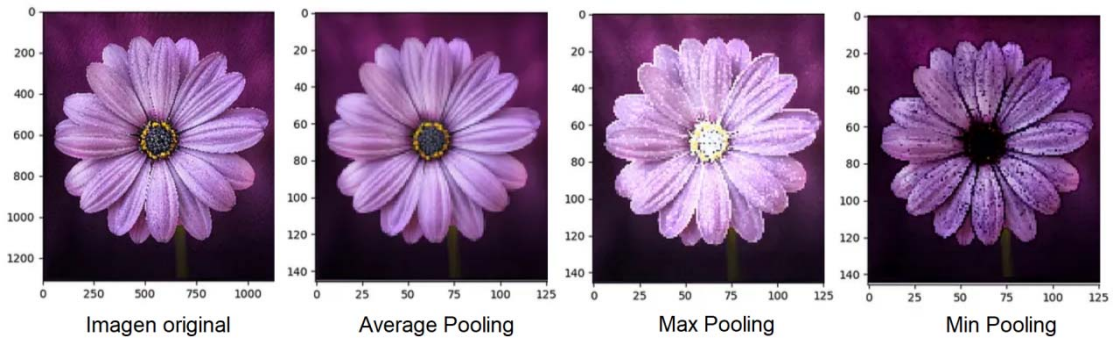


Figura 4.22: Comparación de tipos de Pooling sobre una imagen.
(Basavarajaiah, 2019)

Capítulo 5
Aplicaciones

5.1. Detección de plagio

5.1.1. Introducción al problema

En la actualidad, el avance tecnológico ha desencadenado el auge del plagio musical, esto es, el acto de apropiarse de las características principales de obras musicales ajenas.

En este contexto, las Redes Neuronales Convolucionales Siamesas son una herramienta poderosa para abordar la detección del plagio musical, ya que destacan por su habilidad para la detección de patrones y de características complejas en señales audiovisuales.

El objetivo principal de este estudio es analizar la aplicación de una Red Neuronal Convolucional Siamesa a la detección de plagio. Para ello, se estudiará cómo evoluciona la función pérdida, lo que define el rendimiento del entrenamiento; y cuál es la precisión de las predicciones, lo cual define el funcionamiento de la red.

5.1.2. Herramientas

Para llevar a cabo el estudio se utilizará el software Matlab R2023a.
Además, en la elaboración del código se han utilizado las siguientes librerías:

- Deep Learning ToolBox
- Audio Toolbox
- Wavelet Toolbox
- Computer Vision Toolbox
- Parallel Computing Toolbox

5.1.3. Dataset

Con el objetivo de crear una herramienta de detección de plagio útil, se requieren grandes cantidades de datos de audio. El dataset utilizado durante el estudio ha sido creado específicamente para esta tarea, consistiendo en un total de 3100 elementos a comparar. Se puede acceder al dataset inicial de audios a través del enlace a pie de página.[3] La estructura del dataset consiste en una serie de subconjuntos de audios relacionados entre sí por asuntos de plagio, cesión de derechos o por ser una versión del tema original. En **Fig. 5.1** se observa un ejemplo de los elementos del subconjunto 10 del dataset. Para generar la tabla de datos mostrada se ha utilizado la función $tabladataset()$ definida en el **Apéndice B.1.**

51	Eagles - Hotel california	68839182	10	wav
52	Gipsy Kings - Hotel California (Spanish Mix)	5600109	10	wav
53	Hotel California - Alabama Colts	9731565	10	wav
54	Jethro Tull - We Used to Know	45513826	10	wav
55	We Used to Know - Steven Wilson Stereo Remix	5845869	10	wav

Figura 5.1: Ejemplo de audios asociados en el dataset. (grupo 10)

Además, con el objetivo de ampliar el volumen de datos, se ha realizado la técnica de 'Data Augmentation'. Esta técnica consiste en realizar ligeras variaciones en los audios del dataset, alterando algunos parámetros como pueden ser la velocidad, la tonalidad o la traslación temporal. En el presente estudio se han generado cinco versiones de cada elemento del dataset, por lo que el grupo 10 final estaría compuesto por los elementos presentes en **Fig. 5.1.** y cinco versiones alteradas de cada uno de ellos. Para aplicar esta técnica se ha utilizado la función $data\_augmentation()$ definida en la **Sec. B.1.**

Cabe mencionar que se destinará un 80% del dataset al entrenamiento de la red y un 20% al testeo mediante la elección aleatoria de pares. Dichos pares únicamente tendrán la etiqueta de plagio si sus elementos pertenecen al mismo subconjunto del dataset.

[3]. Enlace al Google Drive con el dataset en formato audios wav, subconjuntos de imágenes y tabla de datos: https://drive.google.com/drive/folders/1cjBftvoaEcah2lddsaw0k6f3wP5LMEbM?usp=sharing

5.1.4. Preprocesamiento de audios

Como se explica en el **apartado 3.2**, las Redes Neuronales Convolucionales son un tipo de arquitecturas de red diseñadas para procesar datos con estructura espacial, principalmente imágenes. Por tanto debemos transformar nuestro dataset de audios a un dataset de imágenes características, las cuales representen los rasgos distintivos de cada audio.

Primero se realiza un filtrado del ruido de las señales de audio. Para ello se aplican los filtros de media móvil, mediana móvil y el filtro de Wiener. Para aplicar esta técnica se ha utilizado la función $cleaner()$ definida en el **Apéndice B.1.**

En el **Apéndice A.2** se encuentran explicadas las características más relevantes a la hora de detectar plagio musical. Para el estudio llevado a cabo se han utilizado el Zero Crossing Rate, el cual permite diferenciar la percusión de los tonos; la media cuadrática de la energía (RMSE), la cual facilita la fragmentación del audio; el espectrograma de MEL, que permite la identificación de tonos a lo largo de la señal; y el histograma, el cual facilita el reconocimiento de picos de volumen y áreas de amplitud constante. Para la extracción de características se ha utilizado la función $feature\_extraction()$ definida en el **Apéndice B.1.**

Una vez extraídas las características principales de las señales de audio, se concatenan generando la imagen característica asociada a cada audio. Se muestra un ejemplo de imagen característica en **Fig. 5.2.** Para generar la imagen característica se ha utilizado la función $feature\_concat()$ definida en **Apéndice B.1.**

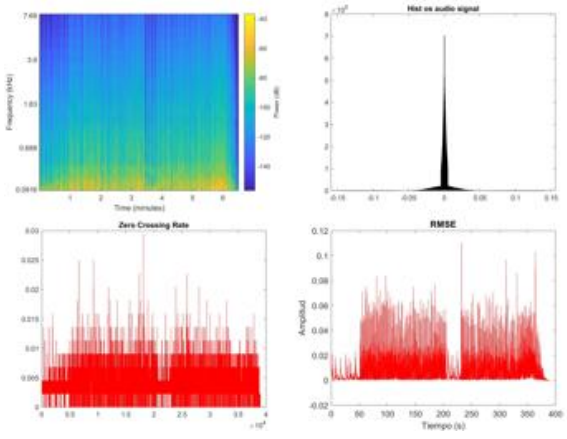


Figura 5.2: Ejemplo de imagen característica (audio del grupo 10)

Por ultimo, todas las imágenes del dataset deben tener las mismas dimensiones, por lo que debe ajustarse su tamaño. En el caso de este studio, las imágenes resultants tenían una gran resolución y se comprimieron todas a las dimensiones: 262 x 350 píxeles.

5.1.5. Entrenamiento

El entrenamiento de una red neuronal es el proceso mediante el cual se determinan los pesos de las conexiones entre las neuronas con el fin de optimizar las predicciones de la red. En los problemas de clasificación, como el caso de este estudio, se emplea un enfoque de entrenamiento supervisado. Esto implica introducir un conjunto de datos etiquetados como entrada y ajustar reiteradamente los pesos de las conexiones de la red hasta que el modelo se adapte de manera efectiva. El algoritmo más usado para realizar esta tarea es el Back Propagation, el cual calcula el gradiente de la función de pérdida para optimizar los pesos de la red en función de las diferencias entre la predicción y la etiqueta del par.
Se observa un esquema en **Fig. 5.3:**

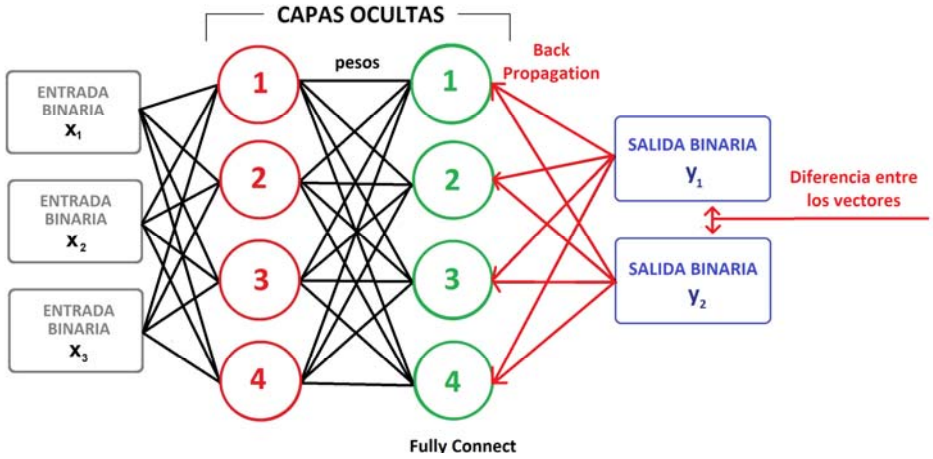
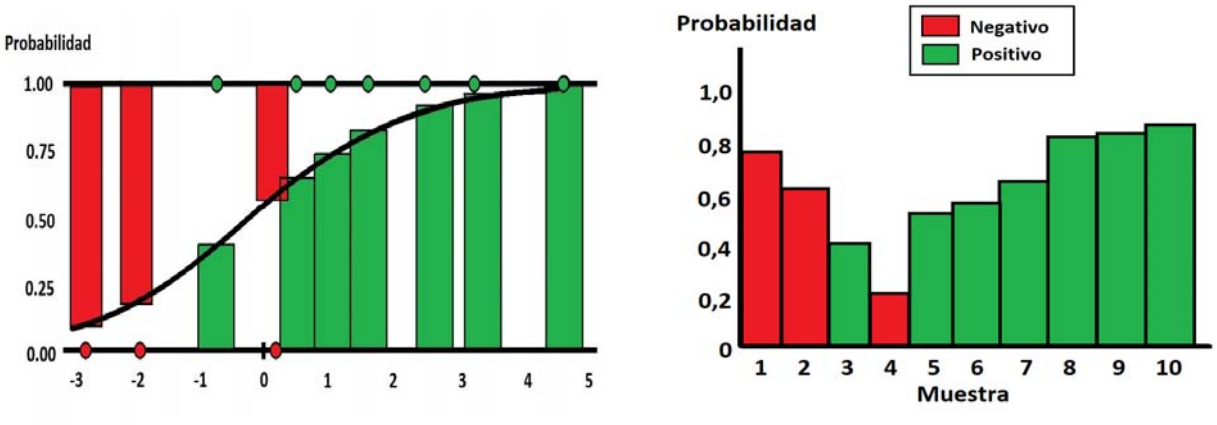


Fig. 5.3: Esquema del algoritmo Back Propagation (elaboración propia)

En cada iteración del proceso de entrenamiento, se introduce una muestra compuesta por pares aleatorios formados a partir de los elementos del dataset de entrenamiento. A estos pares se les asigna su correspondiente vector etiqueta, que determina si los elementos del par se consideran como plagio o no plagio,según su relación en el conjunto de datos. Para extraer la muestra del dataset se utilizan las funciones: $getSiameseBatch()$ $getSimilarPair()$ y $getDissimilarPair()$ definidas en el **Apéndice B.2.**

Una vez se ha seleccionado la muestra, se inicializan los pesos de la red y se calcula la pérdida mediante la función $modelLoss()$. En primer lugar, se llama a la función $forwardSiamese()$, la cual define la estructura de la red siamesa. En esta etapa, cada elemento del par se introduce en una rama de la red siamesa, obteniéndose el vector característico de cada elemento. Después, se calcula el vector distancia, efectuando la resta de los vectores característicos del par. Posteriormente, se ealiza la operación "fully connect", que consiste en una suma ponderada del vector distancia utilizando los pesos de la red. Luego, se aplica la función de activación sigmoide, la cual transforma el resultado en una predicción probabilística sobre la similitud de los pares de entrada en el intervalo $[0,1]$.

Se muestra en **Fig. 5.4.a** cómo se calculan las probabilidades con la sigmoide y en **Fig. 5.4.b** las probabilidades finales. En rojo se marcan las predicciones de similitud fallidas y en verde las predicciones que coinciden con la etiqueta real del par:



(a) **(b)**

Fig. 5.4: Aplicación de la función sigmoide para el cálculo de la probabilidad (Elaboración propia a partir de (Godoy, 2021))

Una vez obtenida la predicción de la red en formato probabilístico, se llama a la función auxiliar $binarycrossentropy()$, la cual calcula la pérdida binaria de entropía cruzada. En el contexto de la teoría de la información, la entropía es una medida de la incertidumbre asociada a la distribución $q(y)$ en relación con las etiquetas de los pares de la muestra.

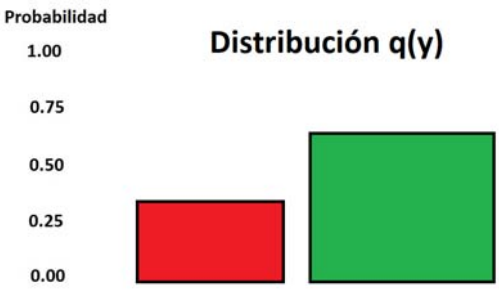


Fig. 5.5: Distribución q(y) (Godoy, 2021)

Esta entropía, utilizada como función de pérdida, se expresa de la forma:

$$H(q) = -\frac{1}{N} \cdot \sum_{i=1}^{N} q(y_i) \cdot \log(q(y_i))$$

Siendo N el número de subconjuntos del dataset y q(y) la distribución de predicciones fallidas y acertadas que se menciona previamente. Si la distribución fuese equilibrada, esto es una proporción 50/50, la entropía se reduciría a:

$$H(q) = \log(2)$$

En nuestro caso queremos relacionar le entropía de la distribución de las etiquetas reales, $q(y) \in \{0,1\}$, con la entropía de la distribución de las predicciones, $\hat{y} \in [0,1]$. Para ello se utiliza la entropía cruzada, la cual se expresa como:

$$\widehat{H}(q) = -\frac{1}{N} \cdot \sum_{i=1}^{N} q(y_i) \cdot \log(\widehat{y_i}) + (1 - q(y_i)) \cdot \log(1 - \widehat{y_i})$$

donde $\widehat{y_i}$ es la predicción probabilística de que el par pertenezca a la clase plagio, $q(y_i)$ es la etiqueta binaria real y N es el número de muestras.

La idea consiste en penalizar las predicciones cuanto más alejadas estén de las etiquetas reales. Se observa en la expresión que, cuando la etiqueta del par sea plagio, se tendrá $q(y_i) = 1$, por lo que se añadirá $\log(\widehat{y_i})$ a la pérdida de la red. Por otro lado, en caso de que la etiqueta sea de no plagio, q$(y_i) = 0$ implicará que se añada $\log(1 - \widehat{y_i})$ a la pérdida de la red.

Se utiliza el logaritmo porque el objetivo es incrementar la pérdida cuando la probabilidad asociada con la etiqueta real es menor, lo cual se ajusta bien a la función $-log(\widehat{y_i})$ en el intervalo $[0,1]$. Se observa en **Fig. 5.6.:**

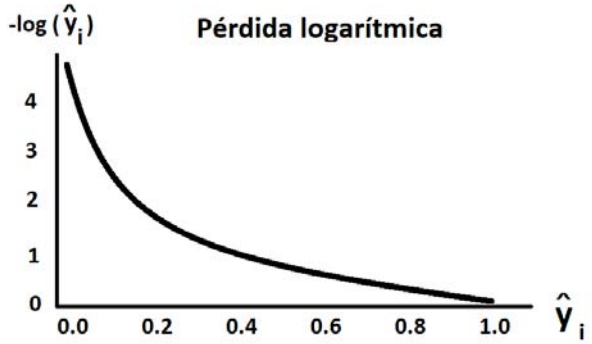


Figura 5.6: Pérdida logarítmica de la probabilidad predicha (Godoy, 2021)

Una vez calculado el valor de la pérdida como la entropía cruzada, se determina la pérdida media de la muestra. Esta se empleará a la hora de calcular los gradientes de los parámetros de la red mediante la función $dlgradient()$.

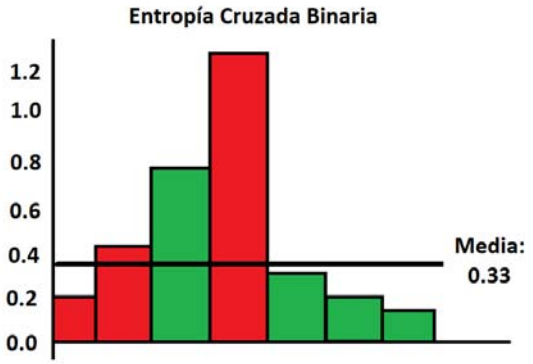


Figura 5.7: Entropía cruzada binaria media (Godoy, 2021)

Para concluir cada iteración, se actualizan los pesos de la red considerando el valor medio de la pérdida y los gradientes. En la **Sección 4.1** se han presentado los métodos de optimización más comunes a la hora de entrenar una Red Neuronal Convolucional. En este estudio se ha aplicado el método de optimización ADAM, que en MATLAB corresponde a la función: $adamupdate()$.

Además, se grafica iterativamente la evolución de la función de pérdida respecto al número de iteraciones. Su evolución se observa en **Fig. 5.8.**:

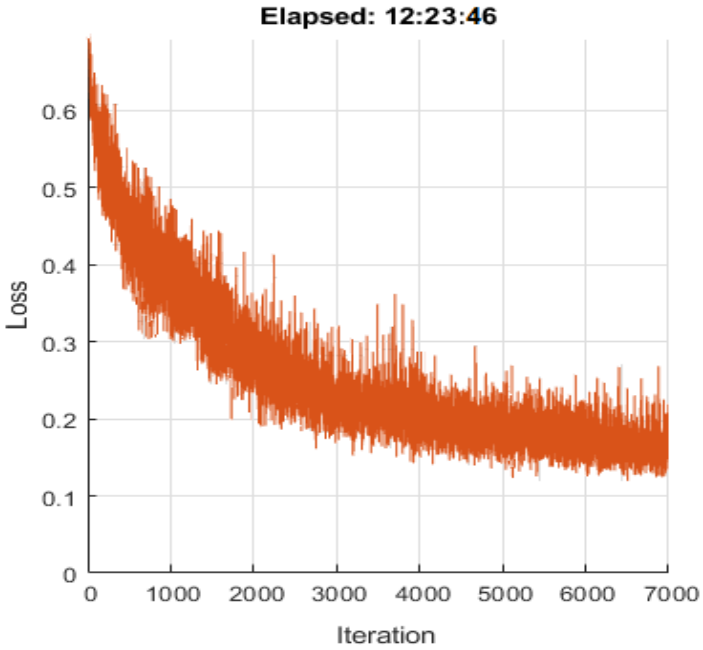


Figura 5.8: Evolución de la función pérdida durante el entrenamiento de la red

Como se aprecia en la **Fig. 5.8,** el valor de la función de pérdida disminuye de manera hiperbólica a medida que aumentan el número de iteraciones, es decir, el número de veces que se toma una muestra aleatoria de datos para entrenar la red.

5.1.6. Test

Para concluir la evaluación del funcionamiento de la red siamesa, se realiza una prueba del modelo. Para llevar a cabo la evaluación se utiliza el 20% restante del dataset con el que ha sido entrenada la red. Este proceso se ejecuta mediante la función $\boldsymbol{predictSiamese}()$, la cual se encuentra definida en la **Sección B.2.**

Luego, las predicciones probabilísticas se transforman en valores binarios, asignando 1 cuando se predice 'plagio' y 0 cuando se predice 'no plagio'. Finalmente, se calcula la precisión de las predicciones de la muestra mutilizando el siguiente ratio:

$$\boldsymbol{precisión} = \frac{\boldsymbol{nº\ de\ predicciones\ correctas}}{\boldsymbol{nº\ total\ de\ predicciones}}$$

Durante las pruebas realizadas en la red de estudio, se ha obtenido una precisión media de **74%.**

```
averageAccuracy = 74.0170
```

Capítulo 6
Conclusiones

6.1. Análisis de resultados

Tras analizar la evolución de la función pérdida en **Fig. 5.4**, se observa que esta disminuye a medida que aumenta el número de iteraciones de entrenamiento. Este patron sugiere la convergencia del modelo hacia un valor mínimo local, lo que demuestra la efectividad del proceso de entrenamiento. Además, la pendiente inicial de la curva es bastante pronunciada, lo que indica una rápida identificación de patrones y, por lo tanto, una convergencia rápida.

Sin embargo, en algunos puntos de la gráfica, se observa que la función de pérdida presenta valores significativamente más altos en comparación con su entorno. Esto suele deberse a problemas de estabilidad en el entrenamiento, como un tamaño de muestra inadecuado o una mala configuración de los hiperparámetros.

En el context de este estudio, los resultados obtenidos no se ajustan a las expectativas, ya que la precisión promedio en todas las muestras de prueba apenas alcanza el **74%,** mientras que se esperaba una precision cercana al 90%.

6.2. Limitaciones

Previamente se ha comentado que la precisión de las predicciones de la red no es la esperada al inicio del estudio. Esto se puede deber a diferentes razones:

- **Dataset:**

  La calidad de los datos de entrenamiento es crucial al entrenar una red neuronal. En el caso de este estudio, el dataset ha sido creado manualmente mediante la búsqueda de casos de plagio registrados, versiones de canciones y el uso de técnicas de aumento de datos. Se han utilizado alrededor de 3.000 imágenes, siendo una décima parte del volumen de datos habitual en un dataset de alta calidad.

- **Razón de aprendizaje:**

  El valor de este parámetro influye en la estabilidad y la convergencia del entrenamiento. En este caso, la velocidad del proceso de entrenamiento indica que se ha utilizado un valor bajo como razón de aprendizaje.

- **Recursos computacionales:**

  El principal problema durante el estudio ha sido la falta de memoria RAM en el ordenador utilizado. Un modelo complejo requiere recursos de memoria y potencia que superan los límites de una CPU convencional.

6.3. Propuestas de mejora

Para abordar las limitaciones identificadas, se presentan las siguientes propuestas:

- **Mejora del dataset:**

  Para mejorar el rendimiento de la red, se sugiere ampliar el dataset y realizar un filtrado más preciso de las versiones y los casos de plagio presentes en los datos.

- **Explorar características de audio:**

  Se propone explorar diferentes características de audio, como son el ritmo, el tono, el volumen, etc., con el objetivo de determinar cuáles son las más relevantes en la tarea de reconocimiento de patrones.

- **Usar GPU:**

  La utilización de una GPU, interna o externa, puede acelerar significativamente el entrenamiento, reduciendo el tiempo requerido, así como proporcionar más memoria RAM.

- **Considerar el contexto legal y ético:**

  A la hora de etiquetar datos es importante considerar el contexto legal y ético de cada elemento. Existen grupos del dataset en los que se han incluido diferentes versiones de canciones bajo la misma etiqueta de plagio. Sin embargo habrá casos de cesión de derechos, versionado y plagio. Por tanto, se propone buscar una manera de incorporar este contexto en las predicciones de la red.

Capítulo 7
Bibliografía

Aprende Machine Learning. (29 de Noviembre de 2018). Recuperado el 2023 de Enero de 7, de https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

Basavarajaiah, M. (8 de Febrero de 2019). *Medimum*. Recuperado el 4 de Abril de 2023, de Maxpooling vs minpooling vs average pooling: https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9#:~:text=Average%20pooling%20method%20smooths%20out,lighter%20pixels%20of%20the%20image.

Brownlee, J. (03 de 07 de 2017). *Machine Learning Mastery*. Recuperado el 29 de 08 de 2023, de https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

Burgsteiner, H. (2007). *A learning rule for very simple approximators consisting of a single layer of perceptron.* Graz University. Graz: Elsevier.

Frederickson, B. (25 de November de 2016). *An Interactive Tutorial on Numerical Optimization*. Recuperado el 2023 de Enero de 20, de http://www.benfrederickson.com/numerical-optimization/

GIMP. (Noviembre de 2002). Recuperado el 16 de Marzo de 2023, de GNU Image Manipulation Program: https://docs.gimp.org/2.10/en/gimp-filter-convolution-matrix.html

Godoy, D. (18 de Noviembre de 2021). *Towards DataScience*. Recuperado el 3 de Septiembre de 2023, de Understanding binary cross-entropy: a visual explanation: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a

IBM. (s.f.). *IBM Cloud*. Recuperado el 3 de Septiembre de 2023, de ¿Qué es el aprendizaje supervisado?: https://www.ibm.com/es-es/topics/supervised-learning

J.C. Lagarias, J. A. (1998). *Convergence properties of the Nelder-Mead simplex method in low dimensions.* SIAM Journal on Optimization.

Keep Coding. (22 de Diciembre de 2022). Obtenido de ¿Qué es Batch Normalization para red convolucional?: https://keepcoding.io/blog/batch-normalization-red-convolucional/

LeCun, Y. (1995). *Convolutional Networks for Images, Speech, and Time-Series.* Holmdel, New Jersey.

Martinez, A. A. (2020). *Redes Neuronales Convolucionales Siamesas aplicadas a la Verificación Facial.* Madrid: UC3M.

MathWorks. (s.f.). Recuperado el 12 de 02 de 2023, de Aplique la convolución a Deep Learning, procesamiento de imágenes y señales: https://es.mathworks.com/discovery/convolution.html

MathWorks. (s.f.). Recuperado el 20 de 05 de 2023, de Convolución: https://es.mathworks.com/discovery/convolution.html

McCulloch, W. (1943). A logical calculus of the ideas immanent in nervous activity.

Mead, J. N. (1965). *A simplex method for function minimization.* Harpenden: Rothamsted Research.

Métodos matemáticos de optimización no restringida. (s.f.). Recuperado el 2 de Enero de 2023, de Unicen: https://www.fio.unicen.edu.ar/usuario/cgely/q13-0/Apuntes/unidad4b.pdf

Moreno, M. J. (2010). *Analisis de algunas metaheurísticas creadas a partir de "Optimización Gravitatoria".* Burgos: Universidad de Burgos.

Nakano, M. (2019). *Redes Convolucionales Siamesas y Tripletas para la Recuperación de Imágenes Similares en Contenido.* Ciudad de Mexico: ESIME.

Nelder, S. S. (2009). *Nelder-Mead Algortihm.* Recuperado el 24 de Enero de 2023, de Scholarpedia: http://var.scholarpedia.org/article/Nelder-Mead_algorithm#The_Nelder-

Mead_simplex_algorithm
P., R. S. (2008). *Teoría de Convolución.* Cartagena: Universidad de Cartagena.
Papert, M. M. (1969). *Perceptrons: An Introduction to Computational Geometry.*
Pérez Gómez, S. (31 de 05 de 2021). *Huawei Forum*. Recuperado el 19 de 05 de 2023, de Aprendizaje automático en IA: método del descenso del gradiente: https://forum.huawei.com/enterprise/es/Aprendizaje-autom%C3%A1tico-en-IA-m%C3%A9todo-de-descenso-de-gradiente/thread/667225135943401472-667212895009779712
Rosenblatt, F. (1958). *The Perceptron: A probabilistic model for information storage and organization in the brain.* Cornell University, Cornell.
Sanderson, G. (26 de Enero de 2018). *YouTube*. Recuperado el 20 de Marzo de 2023, de ero ¿qué es la Transformada de Fourier? Una introducción visual.: https://www.youtube.com/watch?v=spUNpyF58BY&ab_channel=3Blue1Brown
Santina, M. S. (2005). *Handbook of networked and embedded control systems.* Birkhäuser.
Savyakhosla. (21 de Abril de 2023). *geeksforgeeks*. Recuperado el 15 de Mayo de 2023, de CNN, Introduction to Pooling Layer: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/
Shenoy, A. (Julio de 2019). *Research Gate.* Recuperado el 15 de Marzo de 2023, de Feature optimization of contact map predictions based on inter-residue distances and U-Net++ architecture: https://www.researchgate.net/publication/334389306_Feature_optimization_of_contact_map_predictions_based_on_inter-residue_distances_and_U-Net_architecture
SignalSense. (s.f.). *Visualizando la CONVOLUCION de señales continuas*. Recuperado el 12 de Febrero de 2023, de Youtube: https://www.youtube.com/watch?v=IOXzLMqcLbg&ab_channel=SignalSense
Sotaquirá, M. (30 de Marzo de 2019). *La Convolución en las Redes Convolucionales*. Recuperado el 12 de Febrero de 2023, de https://www.codificandobits.com/blog/convolucion-redes-convolucionales/
Vannieuwenhuyze, A. (s.f.). *Inteligencia artificial fácil.* Re.

Apéndice A:

Pre-Procesamiento de señales de audio

Las señales de audio analógicas son funciones continuas que representan, en función de la variable temporal, la presión generada por la interacción de las partículas del medio. Para digitalizar una señal analógica se llevan a cabo los procesos de muestreo y cuantificación.

El muestreo, o sampling, de una señal consiste en tomar un conjunto $\boldsymbol{t_n} = \{\boldsymbol{t_1}, \dots, \boldsymbol{t_n}\}$ de n nodos equiespaciados en el tiempo, los cuales permiten representar la señal continua para su uso en un sistema digital. Es decir: $\boldsymbol{t_k} = \boldsymbol{k} \cdot \boldsymbol{\tau}$, siendo $\boldsymbol{\tau}$ el tiempo entre dos nodos consecutivos, es decir, el tamaño del simple. Se define entonces el sampling rate, $\boldsymbol{S_r}$, que es el número de puntos por unidad de tiempo, es decir:

$$\boldsymbol{S_r} = \frac{\mathbf{1}}{\boldsymbol{\tau}}\ \boldsymbol{Hz}$$

. Para la elección de $\boldsymbol{\tau}$ se tiene el teorema de muestreo de Nyquist-Shannon, por el cual la forma más óptima de caracterizar una señal continua consiste en tomar dos muestras por ciclo. Esto significa que dichos puntos se deben tomar a una frecuencia de muestreo: $\boldsymbol{f_N} = \frac{\boldsymbol{S_r}}{\mathbf{2}}$. Por ejemplo, para las grabaciones en CD se toma una frecuencia de muestreo $\boldsymbol{f_N} = \mathbf{22050}\ \boldsymbol{Hz}$, que es la frecuencia límite del oído humano. Por tanto: $\boldsymbol{S_r} = \mathbf{44.1}\ \boldsymbol{KHz}$, lo cual supone un tiempo de espaciado $\boldsymbol{\tau} = \mathbf{2,3} \cdot \mathbf{10^{-5}}\boldsymbol{s}$.

Por otro lado, la cuantificación, o 'quantization', consiste en aproximar los valores de la señal continua para una partición del intervalo de amplitud. El número de niveles de dicha partición se determina en función del número de bits del sistema. De esta forma, un CD con una resolución de 16 bits permite un total de $\mathbf{2^{16}} = \mathbf{65536}$ niveles discretos.

Además, conociendo el número de bits, $\boldsymbol{Q}$, se puede calcular matemáticamente el ratio de ruido originado por la cuantificación de la señal de la forma:

$$\boldsymbol{SQNR} = \mathbf{6,02} \cdot \boldsymbol{Q}\ \boldsymbol{dB}$$

En el caso del CD de 16 bits se tiene que la intensidad de la entrada de audio ha sido modificada un total de $\boldsymbol{SQNR \approx 96\,dB}$ para su ajuste a los niveles de cuantificación.

Para llevarse a cabo la extracción de características de una señal digital, primero se realizar un framing de la señal. Esto es una partición temporal de la señal de forma que cada nivel, o frame, esté compuesto por $\boldsymbol{K = 2^j}$ samples con $\boldsymbol{j} \in \mathbb{N}$. Entonces cada frame tendrá una duración $\boldsymbol{d_f = \frac{K}{S_r}}$. Además, se suelen tomar frames solapados, de forma que se evita la pérdida de características en los extremos de cada frame.

Existen diversas categorías de características, o features, que pueden ser extraídos de una señal de audio. Los features de nivel bajo de abstracción son las características básicas del audio, necesarias para su procesamiento mediante computación, como pueden ser el centroide espectral o la envolvente de la amplitud. Las features con nivel medio de abstracción se basan en el análisis del tono y la búsqueda de patrones, como son los coeficientes cepstrales. Por último, el máximo nivel de abstracción consiste en la identificación de elementos muy integrados en el audio como son los instrumentos, acordes, género, ritmo o melodía.

Para comprender visualmente cómo funciona la extracción de cada característica, utilizaremos como ejemplo el concierto nº3 de Mozart en Mi Mayor. La representación de su señal en el dominio temporal es:

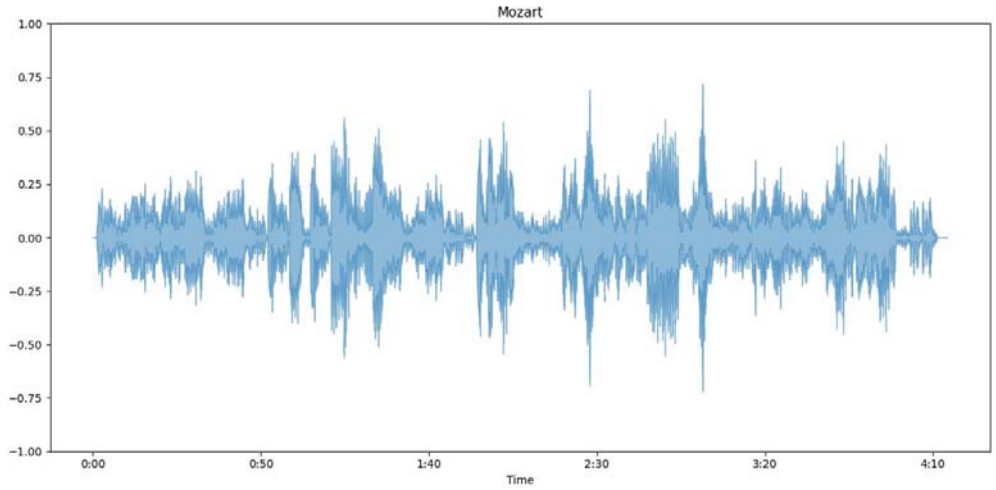


Fig A.1: Representación de la señal ejemplo en el dominio temporal.

A.1. Técnicas de extracción de ruido

Existen diferentes técnicas para la eliminación de ruido en una señal de audio. La elección de la técnica a usar depende del tipo de ruido presente en la señal y de los objetivos del usuario al aplicar el procesamiento de audios.

- **Técnica de filtrado por media móvil:**

Esta técnica resulta útil para eliminar el ruido de baja frecuencia y para suavizar la señal. El filtrado por media móvil consiste en calcular la media de los valores del audio que se encuentran dentro de una ventana móvil de tamaño predefinido. Se debe tener en cuenta que cuanto menor sea el tamaño de la ventana móvil, el filtrado mantendrá los detalles más finos del audio. La técnica finaliza cuando la ventana ha recorrido la totalidad de valores del audio. Para aplicar esta técnica mediante MATLAB se utiliza la función: $\boldsymbol{movmean}(\boldsymbol{X})$

- **Técnica de filtrado por mediana:**

Esta técnica resulta útil para eliminar el ruido impulsivo, es decir, los valores atípicos. El filtrado por mediana móvil también involucra una ventana móvil que recorre el audio entero pero, en este caso, se calcula la mediana de los valores recogidos en la ventana. La técnica finaliza cuando se ha recorrido la totalidad del audio. Para aplicar esta técnica mediante MATLAB se utiliza la función: $\boldsymbol{medfilt1}(\boldsymbol{X})$

- **Técnica de filtrado Wiener:**

El filtrado de Wiener es útil para los ruidos de alta y baja frecuencia. Esta técnica se basa en estimar la relación señal-ruido en diferentes frecuencias, minimizando el error entre la señal original y la filtrada. Para aplicar esta técnica mediante MATLAB se utiliza la función: $\boldsymbol{medfilt2}(\boldsymbol{X})$

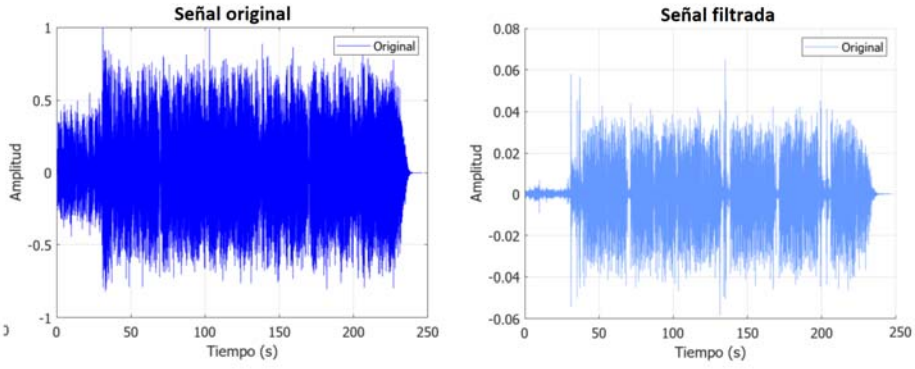


Fig A.2: Señal ejemplo filtrada por los tres filtros

A.2. Extracción de características en el dominio temporal

Una vez se tiene la señal digitalizada y dividida en frames, se pueden calcular la envolvente de la amplitud, el RSM y el ZCR:

- **Envolvente de la amplitud:**

La envolvente de la amplitud es similar al Max Pooling que se realiza dentro de la red pero tomando como ventanas cada frame. Por tanto, consiste en tomar la amplitud máxima entre las amplitudes de cada sample del frame. Su objetivo principal es la detección de valores outliers sobre el volumen, aunque también resulta útil a la hora de detectar fonemas y de clasificar una canción según su género musical.

Su expresión matemática para cada frame $\boldsymbol{F_t} = [\boldsymbol{t} \cdot \boldsymbol{K}, (\boldsymbol{t}+\boldsymbol{1})\boldsymbol{K} - \boldsymbol{1}]$ es:

$$\boldsymbol{AE_t} = \max_{\boldsymbol{k} \in \boldsymbol{F_t}} \boldsymbol{S(k)}$$

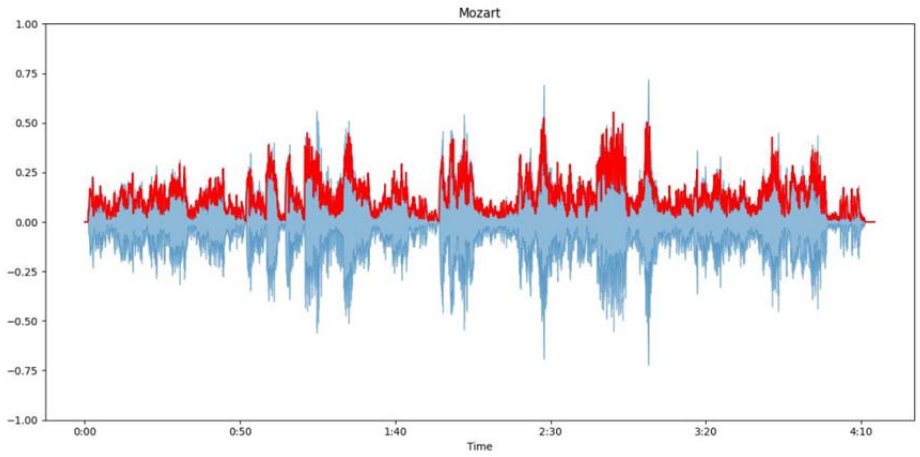


Fig A.3: Envolvente de la amplitud de la señal ejemplo.

Para obtener la envolvente de la amplitud de una señal de audio mediante MATLAB se utiliza la función: $\boldsymbol{envelope(X)}$

- **Media cuadrática de la energía:**

La media cuadrática de la energía, conocida como Root Mean Square Energy o RMSE, mide la energía media de la onda en cada frame, siendo esta la media de la energía en cada sample del frame. La energía está directamente relacionada con el volumen del audio, por lo que resulta una herramienta útil para la segmentación de audio y la clasificación por géneros.

Su expresión matemática para cada frame t es:

$$\boldsymbol{RMSE_t = \sqrt{\frac{1}{K} \cdot \sum_{k=tK}^{(t+1)K-1} s(k)^2}}$$

siendo K el número de samples por frame.

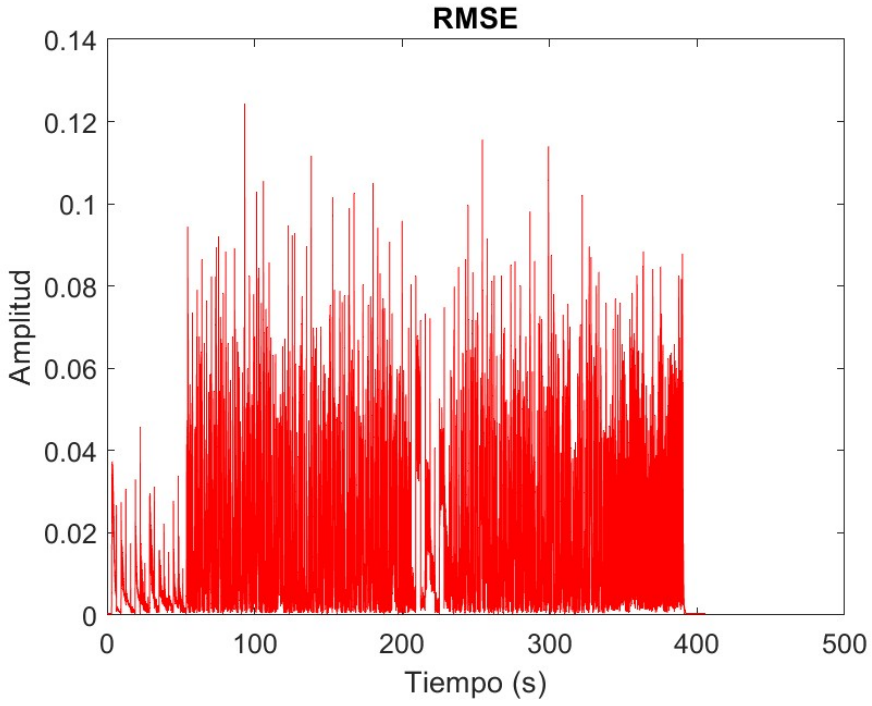


Fig A.4: RMSE de la señal ejemplo.

Para obtener la media cuadrática de la energía de una señal de audio mediante MATLAB se utiliza la función: $\boldsymbol{envelope(X, 'rms')}$

- **Zero Crossing Rate (ZCR):**

La tasa de cruce por cero mide la tasa en la que la señal cambia entre las regiones positiva y la negativa. Dicha información resulta útil en problemas de reconocimiento del habla y de identificación de sonidos de percusión frente a tonos musicales. En concreto, la técnica ZCR permite reconocer tonos monofónicos, permitiendo clasificar el audio en función de si tiene voz o no.

Su expresión matemática para cada frame es:

$$\boldsymbol{ZCR_t = \frac{1}{2} \cdot \sum_{k=tK}^{(t+1)K-1} \left| signo\big(s(k)\big) - signo(s(k+1)) \right|}$$

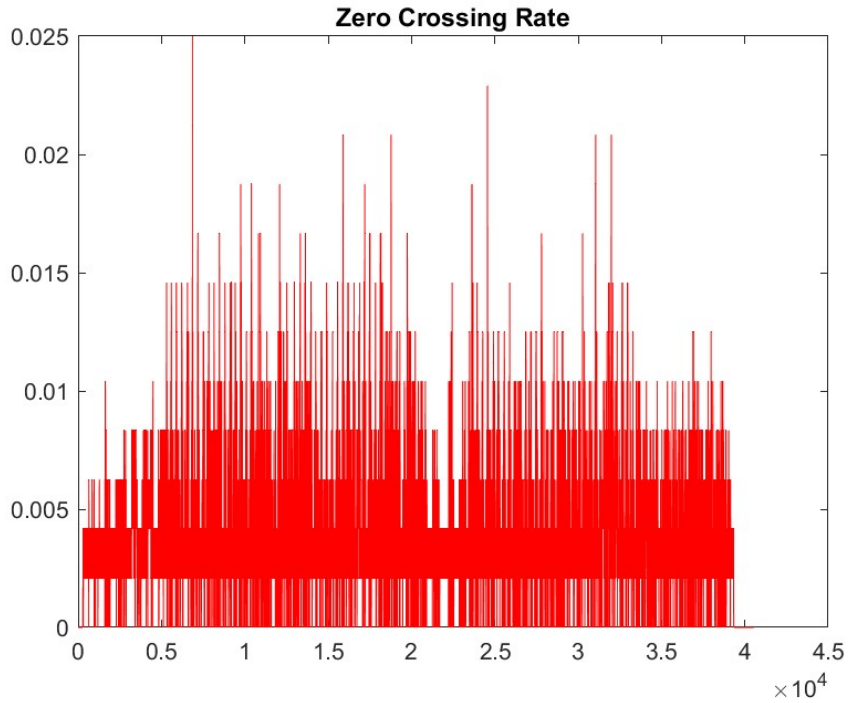


Fig A.5: ZCR de la señal ejemplo.

Para obtener el zero crossing rate de una señal de audio mediante MATLAB se utiliza la función:$\boldsymbol{zerocrossrate(X)}$

A.3. Extracción en los dominios de tiempo y frecuencia.

- **Espectrograma:**

El espectrograma representa cómo la intensidad de la señal varía en cada frecuencia a lo largo del tiempo. Para ello, se divide la señal en varios fragmentos temporales y se les aplica la transformada de Fourier, explicada en la **Sección 3.2.1.1**. La característica resultante es una gráfica 2D en la cual el eje OX es el eje temporal, el eje OY la frecuencia y los colores de cada punto representan la potencia de la onda en el momento y frecuencia correspondientes.

Para obtener el espectrograma estándar de una señal de audio mediante MATLAB se utiliza la función: $\boldsymbol{spectrogram(X)}$

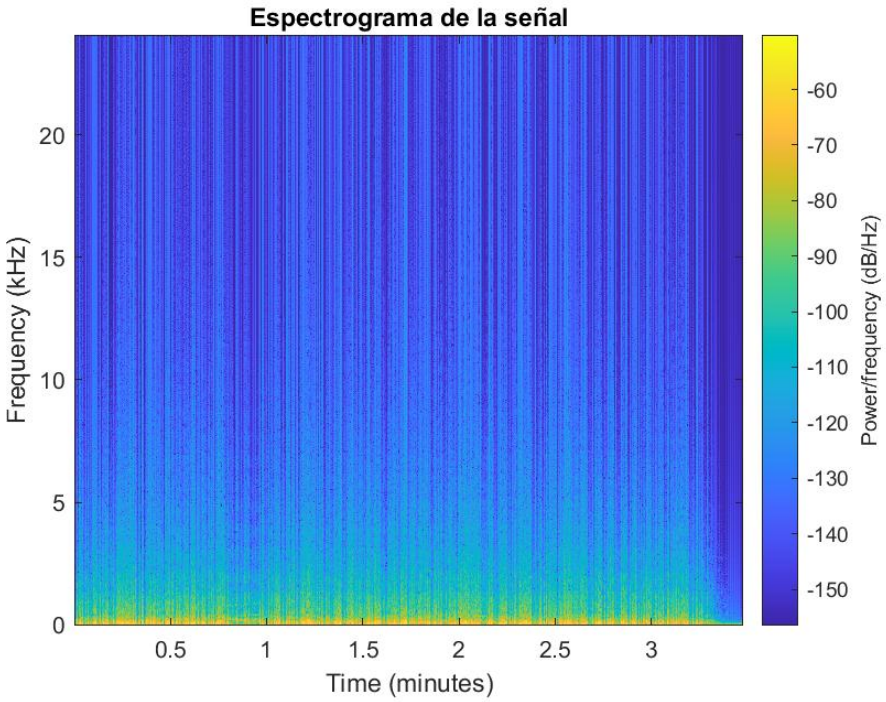


Fig A.6: Espectrograma de la señal ejemplo

- **Espectrograma de MEL:**

El espectrograma de MEL es una variación del espectrograma estándar cuya principal diferencia es la escala de frecuencias utilizada. Esta característica emplea la escala MEL, una transformación no lineal que ajusta la escala lineal de frecuencias a la percepción tonal característica del oído humano. De esta forma, las pequeñas diferencias serán más visibles en las frecuencias bajas. La transformación MEL se expresa matemáticamente como:

$$\boldsymbol{M(f) = 2595 \cdot \log_{10}\left(1 + \frac{f}{700}\right)}$$

Siendo $\boldsymbol{f}$ la frecuencia en $\boldsymbol{Hz}$ y $\boldsymbol{M(f)}$ su frecuencia asociada en la escala MEL.

Para extraer esta característica, primero se transforma la escala lineal en $\boldsymbol{Hz}$ a escala MEL, y posteriormente se genera el espectrograma estándar.

Para obtener el espectrograma de MEL de una señal de audio mediante MATLAB se utiliza la función: $\boldsymbol{melSpetrogram(X)}$

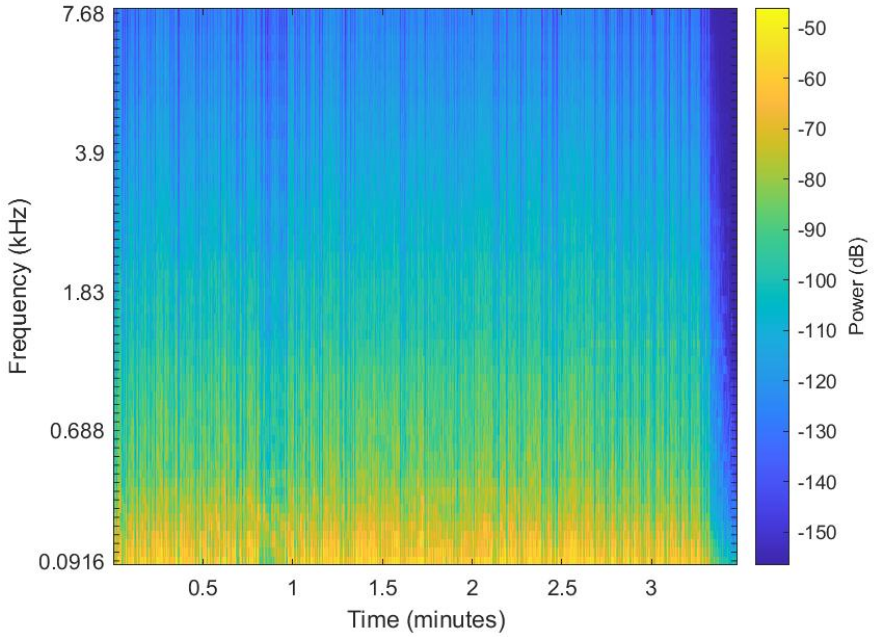


Fig A.7: Espectrograma de MEL de la señal ejemplo.

Apéndice B:

Programación en MATLAB

Para la aplicación de una Red Neuronal Convolucional Siamesa a la detección de plagio se tienen dos códigos principales (main): el primero procesa los audios, extrayendo las imágenes características y creando la tabla de datos sobre el dataset; y la segunda en la cual se define la red, se entrena con parte del dataset y se testea con los datos restantes. Cada una de ellas utiliza funciones auxiliares definidas con el objetivo de organizar y simplificar el código.

La sección B1 ha sido programada desde cero y la sección B2 se ha basado en el tutorial sobre cómo entrenar una red siamesa para la comparación de imágenes, ofrecida por MATLAB en https://es.mathworks.com/help/deeplearning/ug/train-a-siamese-network-to-compare-images.html.

B.1. Preprocesamiento de datos de audio

- **Función principal: Main**

El siguiente código llama a la función auxiliar tabladataset() para generar la tabla de datos y, posteriormente, itera sobre todos los elementos del dataset. En el bucle se llama a la función auxiliar cleaner() para eliminar el ruido del audio, se extraen y concatenan sus características principales con las funciones auxiliares feature_extraction() y feature_concat(), y se realiza la aumentación de datos mediante la función auxiliar data_augmentation(). De esta forma se completa el dataset que se utilizará para el entrenamiento y testeo de la red.

```
clc

% Definición del path con los audios del dataset en formato wav
path_file = "dataset_wav\";

% Uso de la función auxiliar tabladataset() para generar la tabla de datos:
data = tabladataset();


% Dimensiones de la tabla data
```

```
l = height(data)

% Definición del número de iteraciones
numiteraciones = 62



for k = 1:numiteraciones
    muestra = 10
    path_file = "dataset_wav\";
    data = tabladataset();
    l = height(data)

    for h = 1:muestra

        i = k*10 + h

        % Parar el bucle cuando se llegue al final del dataset
        if i == height(data)
            break
        end

        % Lectura del audio
        arch = data.archivo(i);
        audio_file = strcat(path_file,arch);
        [X,fs] = audioread(audio_file);

        % Especificar el uso de un solo canal de audio
        X = X(:,1);

        % Creación de carpetas para la organización del dataset
        name = data.nombre(i)
        group = string(data.grupo(i));
        name = string(name);

        if not(isfolder(name))
            mkdir('features',name);
        end

        if not(isfolder(group))
            mkdir('pair',group);
        end

        % Definición de los path donde guardar las características extraídas

        folderpath1 = "C:\2\TFG\TFG programa\features\";
        featurepath = strcat(folderpath1,name);

        folderpath2 = "C:\2\TFG\TFG programa\pair\";
        pairpath = strcat(folderpath2,group);
```

```
        %%% Extracción de las características físicas del audio

        % Duración de un sample
        sample_duration = 1/fs;

        % Número de samples
        N = length(X);
        t = (0:N-1)/fs;

        % Duración del audio:
        duracion = sample_duration*length(X);



        %%% Limpieza del audio

        % Uso de la función auxiliar cleaner() para la eliminación de ruido
        X_clean = cleaner(X,t)

        %%% Extracción de características

        % Extracción de características con la función auxiliar
        % feature_extraction()
        feature_extraction(X_clean,t,fs,N,featurepath);

        % Generación de la imagen característica con la función
        % auxiliar feature_concat()
        feature_concat(data,i,featurepath,pairpath,0);

        % Aumentación de datos con la función data_augmentation()
        data_augmentation(X,fs,data,i,pairpath)
    end
end
```

- **Función auxiliar: tabladataset()**

El siguiente código extrae los nombres de los archivos en la carpeta 'dataset_wav' y genera una tabla en un documento Excel. Posteriormente se limpia la tabla, renombrando las columnas y excluyendo aquellos archivos que no pertenecen al dataset.

```
%%%% TABLADATASET %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Creación y ETL de la tabla data con las columnas:{nombre, tamaño, fecha,
%  año, grupo, archivo} recogiendo los datos de cada audio del dataset.
%
%%%% INPUT:
%
%%%% OUTPUT:
%
%   - data ----------> Table ---------> Tabla con el dataset
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function data = tabladataset()

    % Creación de un excel con el dataset actualizado
    info = dir('dataset_wav');
    writetable(struct2table(info),'dataset_to_clean.xlsx');


    % Lectura del excel creado
    data = readtable('dataset_to_clean.xlsx');

    % Cambio de nombre de las columnas
    data.Properties.VariableNames =
    {'nombre','carpeta','fecha','bytes','isdir','something'};

     % Eliminación de archivos que no pertenecen al dataset
    data(1,:)=[];
    data(1,:)=[];
    rush = contains(data.nombre,'xlsx');
    rushrow = [];
    for j = 1:length(rush)
        if rush(j)==1
            rushrow=[rushrow,j];
        end
    end
    data(rushrow,:)=[];

    % Se realizan la extracción del grupo y del año de cada elemento del dataset,
    % la creación de la columna archivo con el nombre del archivo original
    % y se limpia la columna nombre.
    for i=1:size(data,1)
        name = data.nombre{i};
        l = length(name);
        group = str2num(name(1:3));
        format = convertCharsToStrings(name(l-2:l));
```

```
        archivo = convertCharsToStrings(name);
        % disp(class(group))
        % disp(data.grupo(i))
        data.grupo(i) = group;
        data.nombre(i) = cellstr(name(5:l-4));
        data.formato(i) = format;
        data.archivo(i) = archivo;
    end

    % Eliminación de variables no pertinentes
    data.fecha = [];
    data.isdir = [];
    data.something = [];
    data.carpeta = [];

    % Eliminación de duplicados
    [~,rows] = unique(data.nombre);
    data = data(rows,:);

    % Guardado de la tabla final
    writetable(data,'dataset.xlsx')

end
```

- **Función auxiliar: cleaner()**

El siguiente código aplica los filtros de media móvil, mediana móvil y el filtro de Wiener a la señal con el objetivo de eliminar el ruido del audio.

```
%%%% CLEANER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% · Normalización del audio
% · Filtrado de ruido mediante los filtros: media movil, mediana y Wiener
%.
%%%% INPUT
% X --------------> Array ---------> Vector de audio
% t --------------> Array ---------> Vector tiempo
%
%%%% OUTPUT:
% X_clean --------> Array ---------> Vector de audio filtrado
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function X_clean = cleaner(X,t)

    % Normalización del audio:
    X = X / max(abs(X));

    % Filtro media movil (ruido de alta frecuencia)
    X_clean = movmean(X,1000)

    % Filtro mediana (ruido impulsivo)
    X_clean = medfilt1(X_clean,100)
```

```
    % Filtro de Wiener (ruido de baja frecuencia y ruido blanco)
    X_clean = wiener2(X_clean)

    % Además se grafica la superposición del audio original y el filtrado:
    figure;
    plot(t,X,'b')
    hold on
    plot(t,X_clean,'Color',[0.4,0.6,1])
    grid on
    set(gca, 'FontName','Time New Roman','FontSize',12)
    legend('Original','Filtrada')
    title('Filtros de ruido: Media, Mediana y Wiener')
    xlabel('Tiempo (s)')
    ylabel('Amplitud')
    hold off

end
```

- **Función auxiliar: feature_extraction()**

El siguiente código extrae las características del audio y guarda las representaciones 2D correspondientes como archivos jpg en el directorio 'features'.

```
%%%% FEATURE_EXTRACTION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% · Extracción de las características:
%       - Envolvente de la amplitud
%       - Root Mean Square Energy
%       - Zero Crossing Rate
%       - Espectro de frecuencias
%       - Espectrograma
%       - Periodograma
%       - Histograma
%       - Espectro de MEL
%       - Coeficientes Cepstrales de MEL (MFCC)
%
%%%% INPUT:
%
%   - X_clean ----> Array -----> Vector de audio
%   - t ----------> Array -----> Vector tiempo
%   - fs ---------> Int -------> Sampling Rate (Hz)
%   - N ----------> Int -------> Longitud de X_clean
%   - filepath ---> String ----> Path con las características jpg
%
%%%% OUTPUT:
%
% ** Extrae las características y las guarda en formato jpg.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function feature_extraction(X_clean,t,fs,N,filepath)

    % Envolvente de la amplitud
    [up,~] = envelope(X_clean);
    hold on
    en = plot(t,up,'linewidth',0.5,'Color','r');
    set(gca, 'FontName','Time New Roman','FontSize',12)
    xlabel('Tiempo (s)')
    ylabel('Amplitud')
    legend('audio','up')
    title('Envolvente de la Amplitud')
    hold off
    saveas(en,fullfile(filepath,'envolvente_amplitud'),'jpg')

% Root Mean Square Energy (RMSE):
    figure;
    rms = envelope(X_clean,512,'rms');
    rmse = plot(t,rms,'Color','r');
    set(gca, 'FontName','Time New Roman','FontSize',12);
    xlabel('Tiempo (s)');
    ylabel('Amplitud');
    title('RMSE');

    saveas(rmse,fullfile(filepath,'root_mean_square_energy'),'jpg');
    % Zero crossing rate
    win = fs*0.01;
    rate_features = zerocrossrate(X_clean, WindowLength=win,Method="comparison");
    zero = plot(rate_features,'Color','r');
    title('Zero Crossing Rate')

    saveas(zero,fullfile(filepath,'zero_crossing_rate'),'jpg')

    % Dominio de Frecuencia
    X_fft = fft(X_clean);
    X_shiffft = fftshift(fft(X_clean));
    ff = linspace(-fs/2,fs/2,length(X_fft));
    ff_2 = linspace(-fs/2,fs/2,length(X_shiffft));
    mag_X = abs(X_fft);
    mag_X_2 = abs(X_shiffft);

    figure();
    frec = plot(ff, mag_X);
    title('Espectro de frecuencias de la señal');
    xlabel('Frecuency (Hz)');
    ylabel('Amplitud');
    grid on, grid minor; % cuadricula

    saveas(frec,fullfile(filepath,'espectro_frecuencias'),'jpg');

    esp = plot(ff_2, mag_X_2/max(mag_X_2));
    title('Espectro de frecuencias de la señal');
    xlabel('Frecuency (Hz)');
    ylabel('Amplitud');
    grid on, grid minor; % cuadricula
    saveas(esp,fullfile(filepath,'espectro_frecuencias_2'),'jpg');

    % Poner exponente 3
    ax = gca;
    ax.XAxis.Exponent = 3;
```

```
    % Espectrograma
    h=figure();
    spectrogram(X_clean,1024, 512, 1024,fs,'yaxis');
    title('Espectrograma de la señal')
    saveas(h, fullfile(filepath,'espectrograma'),'jpg')

    % Plot histograma
    hist = figure();
    histogram(X_clean);
    title('Hist os audio signal');

    saveas(hist, fullfile(filepath,'histograma'),'jpg');

    % Espectrograma de MEL:
    mel = figure();

       melSpectrogram(X_clean,fs,'Window',hann(2048,"periodic"),'OverlapLength',1024
       ,'FFTLength',4096,'NumBands',64,'FrequencyRange',[62.5,8e3]);
    %coefs = melSpectrogram(X_clean,fs);

    saveas(mel, fullfile(filepath,'espectrograma_mel'),'jpg');
end
```

- **Función auxiliar: feature_concat()**

El siguiente código genera la imagen característica de un audio concatenando las características del audio elegidas para el estudio. Dicha imagen característica se guarda en el directorio ‘features’ así como se incluye en el número de grupo correspondiente.

```
%%%% FEATURE_CONCAT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% · Concatenación de las características principales de un elemento
%
%%%% INPUT:
%
%   - data -------> Table -----> Tabla con información sobre los audios
%   - i ----------> Int -------> Número de iteración
%   - filepath ---> String ----> Path con las imagenes características
%   - pairpath ---> String ----> Path con los pares de imagenes caract.
%   - var --------> Int -------> Numero de variación (0 si original)
%
%%%% OUTPUT:
%
%   ** Genera la imagen característica y la guarda en jpg.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
function feature_concat(data,i,filepath,pairpath,var)

    song = data.nombre(i);

    % Lectura de características
    A = imread(fullfile(filepath,"espectrograma_mel.jpg"));
    B = imread(fullfile(filepath,"histograma.jpg"));
    C = imread(fullfile(filepath,"zero_crossing_rate.jpg"));
    D = imread(fullfile(filepath,"root_mean_square_energy.jpg"));

    % Redimensionado de las características para su concatenación
    M = min([size(A,1),size(B,1),size(C,1),size(D,1)]);
    N = min([size(A,2),size(B,2),size(C,2),size(D,2)]);
    A_resized = imresize(A,[M,N]);
    B_resized = imresize(B,[M,N]);
    C_resized = imresize(C,[M,N]);
    D_resized = imresize(D,[M,N]);

    % Concatenación de características en cuadrícula
    fila_sup = [A_resized, B_resized];
    fila_inf = [C_resized, D_resized];
    caracteristicas = [fila_sup; fila_inf];
    feat = imresize(caracteristicas,[262,350]);
    imshow(feat)

    % Guardado de las características concatenadas en 'features':
    imwrite(feat,fullfile(filepath,'features_concat.jpg'));

    % Guardado de las características concatenadas en 'pair':
    if var==0
        featname = strcat('features_',song,'.jpg');
        imwrite(feat,fullfile(pairpath,featname));
    else
        var_str = string(var);
        featname = strcat('features_',song,'_var_',var_str,'.jpg''.jpg');
        imwrite(feat,fullfile(pairpath,featname));
    end
end
```

- **Función auxiliar: data_augmentation()**

El siguiente código genera cinco variaciones de cadata audio, modificando algunas de sus características físicas como el volumen, el ruido o la velocidad. Dichas variaciones son procesadas de la misma manera que los audios originales, es decir, utilizando las funciones auxiliares feature_extraction y feature_concatenation. Las características se guardan en el directorio ‘augmentation’ y la imagen característica se guarda en el grupo correspondiente en el directorio ‘pair’.

```
%%%% DATA_AUGMENTATION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%    · Generación de 5 variaciones de los audios (pith, volumen, ruido...)
%    · Extracción y concatenación de las características de cada variación.
%
%%%% INPUT:
%
%    - X ----------> Array -----> Vector de audio
%    - fs ---------> Int -------> Sampling rate (Hz)
%    - data -------> Table -----> Dataset
%    - i ----------> Int -------> Número de iteración
%    - pairpath ---> String ----> Path con los pares de imagenes caract.
%
%%%% OUTPUT:
%
%    ** Genera la imagen característica de las versiones y las guarda en jpg.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



function data_augmentation(X,fs,data,i,pairpath)

% Nombre de las canciones originales
    name = data.nombre(i);
    name = string(name);



    % Definición del aumentador de datos
    augmenter = audioDataAugmenter( ...
        "AugmentationMode","sequential", ...
        "NumAugmentations",5, ...
        ...
        "TimeStretchProbability",0.8, ...
        "SpeedupFactorRange", [1.3,1.4], ...
        ...
        "PitchShiftProbability",0, ...
        ...
        "VolumeControlProbability",0.8, ...
        "VolumeGainRange",[-5,5], ...
        ...
        "AddNoiseProbability",0, ...
        ...
        "TimeShiftProbability",0.8, ...
        "TimeShiftRange", [-500e-3,500e-3]);

    % Aumentación de datos
    data_var = augment(augmenter,X,fs);

    % Ajustes organizativos
    mkdir('augmentations',name)
    folderpath = "C:\2\TFG\TFG programa\augmentations\";
    filepath = strcat(folderpath,name);
```

```
    for j=1:5
        % Definir el numero de variación
        var = string(j);
        % Crear una carpeta para cada variación de cada elemento
        mkdir(filepath,var)
        filepath2 = strcat(filepath,"\");
        filepath3 = strcat(filepath2,var);
        data_var.AugmentationInfo(j);

        % Definir la variación asociada a la iteración y su vector tiempo
        % asociado (puede variar al modificarse la velocidad del audio)
        aug = data_var.Audio{j};
        N = length(aug);
        taug = (0:(numel(aug)-1))/fs;

        % Extracción de características con la función auxiliar
         % feature_extraction()
        feature_extraction(aug,taug,fs,N,filepath3);

       % Generación de la imagen característica con la función
       % auxiliar feature_concat()
        feature_concat(data,i,filepath3,pairpath,j);
    end
end
```

B.2. CNN Siamesa. Entrenamiento y testeo.

- **Función principal: Main**

El siguiente código primero carga el dataset y define la red CNN, inicializando sus pesos y su sesgo (bias). Una vez separado el dataset en subconjuntos de entrenamiento y testeo, se utilizan las funciones auxiliares getSiameseBatch(), dlfeval() y adamupdate() para entrenar la red, generándose una gráfica sobre la evolución de la pérdida respecto al número de iteraciones del entrenamiento. Por último, se llama a funciones auxiliares getSiameseBatch() y predictSiamese() para realizar el testeo de la red, calculándose la precisión media de la red.

```
clc
clear all
```

1 Carga de datos

```
% Definición del path con los grupos del dataset en función de su
% similitud
dataFolderTrain = 'C:\2\TFG\TFG programa\pair'
```

```
% Carga del dataset completo
imds_total = imageDatastore(dataFolderTrain, ...
    IncludeSubfolders=true, ...
    LabelSource="none");


% Se permuta aleatoriamente el dataset para la posterior separación en dos
% subconjuntos para entrenamiento y testeo

files = imds_total.Files(randperm(length(imds_total.Files)));
parts = split(files,filesep);
labels = join(parts(:,(end-2):(end-1)),"-");
imds_total.Labels = categorical(labels);
```

2 Arquitectura de la CNN

```
% Definición de las capas de la CNN:

layers = [
    imageInputLayer([262 350 3],Normalization="zscore",NormalizationDimension =
    "auto", Mean = 0, StandardDeviation = 1, Name = "Input")
    convolution2dLayer(10,64,WeightsInitializer="narrow-
    normal",BiasInitializer="narrow-normal", WeightLearnRateFactor = 1,
    BiasLearnRateFactor = 1)
    reluLayer
    maxPooling2dLayer(2,Stride=2)
    convolution2dLayer(7,128,WeightsInitializer="narrow-
    normal",BiasInitializer="narrow-normal")
    reluLayer
    maxPooling2dLayer(2,Stride=2)
    convolution2dLayer(4,256,WeightsInitializer="narrow-
    normal",BiasInitializer="narrow-normal")
    reluLayer
    maxPooling2dLayer(2,Stride=2)
    convolution2dLayer(5,512,WeightsInitializer="narrow-
    normal",BiasInitializer="narrow-normal")
    reluLayer
    fullyConnectedLayer(256,WeightsInitializer="narrow-
    normal",BiasInitializer="narrow-normal")
    ]

% Visualización de la CNN
lgraph = layerGraph(layers)
figure
plot(lgraph)


% Creación de la red CNN
net = dlnetwork(lgraph)
```

```
% Inicialización de los pesos de la red
fcWeights = dlarray(0.01*randn(1,256));
fcBias = dlarray(0.01*randn(1,1));
fcParams = struct(...
    "FcWeights",fcWeights,...
    "FcBias",fcBias);
```

3) Entrenamiento de la SCNN

```
% Definición del número de iteraciones, el tamaño de cada muestra y el
% porcentaje de dataset destinado al entrenamiento.
numIterations = 1000;
miniBatchSize = 10;
training_perc = 80;
```

```
% Definición del subconjunto de entrenamiento del dataset (80% de los datos)
num_training = roung(training_perc * numel(imds.Files /100));
imds_train = subset(imds_total, 1:num_training)
files_train = imds_train.Files;
parts_train = split(files_train,filesep);
labels_train = join(parts_train(:,(end-2):(end-1)),"-");
imds_train.Labels = categorical(labels_train);
```

```
% Especificación de los parámetros para la optimización ADAM:
learningRate = 6e-5;
gradDecay = 0.9;
gradDecaySq = 0.99;
```

```
% Definición del modo de ejecución (para GPU poner "auto" sino poner "cpu")
executionEnvironment = "auto";
```

```
% Inicialización de la gráfica dinámica: Función de pérdida vs. Iteración
figure
C = colororder;
lineLossTrain = animatedline(Color=C(2,:));
ylim([0 inf])
xlabel("Iteration")
ylabel("Loss")
grid on
```

```
% Inicialización de los parámetros del solver y del recuento temporal:
trailingAvgSubnet = [];
trailingAvgSqSubnet = [];
```

```
trailingAvgParams = [];
trailingAvgSqParams = [];
start = tic;

% Generación iterativa del gráfico Función de pérdida vs. Iteración
for iteration = 1:numIterations

    % Extracción de una muestra de tamaño 10
    [X1,X2,pairLabels] = getSiameseBatch(imds_train,miniBatchSize);

    % Conversión de los datos de la muestra a dlarray
    % (parámetro "SSCB" (spatial, spatial, channel, batch) for image data)
    X1 = dlarray(X1,"SSCUB");
    X2 = dlarray(X2,"SSCUB");


    % Conversión a gpuarray si se utiliza una GPU
    if (executionEnvironment == "auto" && canUseGPU)
            || (executionEnvironment == "gpu")
        X1 = gpuArray(X1);
        X2 = gpuArray(X2);
    end
    % Uso de la función dlfeval() para calcular la pérdida del modelo
    % y los gradientes, evaluando la red en la función pérdida definida en la
    % función auxiliar modelLoss().
    [loss,gradientsSubnet,gradientsParams] =
          dlfeval(@modelLoss,net,fcParams,X1,X2,pairLabels);


    % Actualización de los parámetros de la CNN siamesa con adamupdate()
    [net,trailingAvgSubnet,trailingAvgSqSubnet] =
          adamupdate(net,gradientsSubnet, …
          trailingAvgSubnet,trailingAvgSqSubnet,iteration,
          learningRate,gradDecay,gradDecaySq);


    % Actualización de los parámetros de la capa fully connected mediante la
    función adamupdate()
    [fcParams,trailingAvgParams,trailingAvgSqParams] =
           adamupdate(fcParams,gradientsParams, ...
           trailingAvgParams,trailingAvgSqParams,iteration,
           learningRate,gradDecay,gradDecaySq);
    % Graficado de la función de pérdida
    D = duration(0,0,toc(start),Format="hh:mm:ss");
    lossValue = extractdata(loss);
    lossValue = double(lossValue)
    addpoints(lineLossTrain,iteration,lossValue);
    title("Elapsed: " + string(D))
    drawnow
end
```

4) Test

```
% Inicialización de parámetros
accuracy = zeros(1,10);
accuracyBatchSize = 150;


% Definición del subconjunto de datos para el testeo
imds_test = subset(imds_total, num_training+1:numel(files))
files_test = imds_test.Files;
parts_test = split(files_test,filesep);
labels_test = join(parts_test(:,(end-2):(end-1)),"-");
imds_test.Labels = categorical(labels_test);

for i = 1:10
    % Extracción de la muestra
    [X1,X2,pairLabelsAcc] = getSiameseBatch(imds_test,accuracyBatchSize);


    % Conversión de los datos de la muestra a dlarray
    % (parámetro "SSCB" (spatial, spatial, channel, batch) for image data)
    X1 = dlarray(X1,"SSCUB");
    X2 = dlarray(X2,"SSCUB");


    % Conversión a gpuarray si se utiliza una GPU
    if (executionEnvironment == "auto" && canUseGPU) || executionEnvironment ==
"gpu"
        X1 = gpuArray(X1);
        X2 = gpuArray(X2);
    end


    % Generación de predicción con la red CNN siamesa
    Y = predictSiamese(net,fcParams,X1,X2);


    % Conversión de las predicciones a valores binarios
    Y = gather(extractdata(Y));
    Y = round(Y);


    % Cálculo de la precisión media de cada muestra
    accuracy(i) = sum(Y == pairLabelsAcc)/accuracyBatchSize;
end


averageAccuracy = mean(accuracy)*100
```

- **Función auxiliar: getSiameseBatch()**

El siguiente código escoge un determinado número de pares del dataset, eligiendo de manera aleatoria si se selecciona un par con etiqueta de plagio o no plagio. Para ello utiliza las funciones auxiliares getSimilarPair() y getDissimilarPair().

```
%%%% GETSIAMESEBATCH %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% · Extracción de una muestra del dataset
%
%%%% INPUT:
%
%   - imds ------------> ImageDataStore ----> Dataset de pares de plagio
%   - miniBatchSize ---> Int ---------------> Tamaño de la muestra
%
%%%% OUTPUT:
%
%   - X1 --------------> Array -------------> Imagen 1
%   - X2 --------------> Array -------------> Imagen 2
%   - pairLabels ------> Array -------------> Etiqueta del par
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [X1,X2,pairLabels] = getSiameseBatch(imds,miniBatchSize)

    % Definición de parámetros
    pairLabels = zeros(1,miniBatchSize);
    imgSize = size(readimage(imds,1));
    X1 = zeros([imgSize 1 miniBatchSize],"single");
    X2 = zeros([imgSize 1 miniBatchSize],"single");

    for i = 1:miniBatchSize

        % Selección aleatoria entre par con plagio o sin plagio
        choice = rand(1);
        if choice < 0.5
            [pairIdx1,pairIdx2,pairLabels(i)] = getSimilarPair(imds.Labels);
        else
            [pairIdx1,pairIdx2,pairLabels(i)] = getDissimilarPair(imds.Labels);
        end

        X1(:,:,:,i) = imds.readimage(pairIdx1);
        X2(:,:,:,i) = imds.readimage(pairIdx2);
    end

end
```

- **Función auxiliar: getSimilarPair()**

El siguiente código extrae aleatoriamente un par de elementos del dataset cuya etiqueta es 'plagio', es decir, pertenecen al mismo grupo en el dataset.

```
%%%% GETSIMILARPAIR %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% · Extracción de un par con la etiqueta de plagio
%
%%%% INPUT:
%
%   - classLabel -------> Array -------> Vector con las clases (pares)
%
%%%% OUTPUT:
%
%   - pairIdx1 ---------> Int ---------> ID de la imagen 1 del par
%   - pairIdx2 ---------> Int ---------> ID de la imagen 2 del par
%   - pairLabel --------> Int ---------> 1 es plagio, 0 no plagio
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [pairIdx1,pairIdx2,pairLabel] = getSimilarPair(classLabel)

    % Lista única de clases
    classes = unique(classLabel);

    % Elección aleatoria de una clase
    classChoice = randi(numel(classes));
    classes(classChoice);

    % Extracción de los indices de la clase elegida
    idxs = find(classLabel==classes(classChoice));

    % Elección de dos imágenes aleatorias de la clase elegida
    pairIdxChoice = randperm(numel(idxs),2);
    pairIdx1 = idxs(pairIdxChoice(1));
    pairIdx2 = idxs(pairIdxChoice(2));

    % Marcamos que el par es plagio con un 1
    pairLabel = 1;

end
```

- **Función auxiliar: getDissimilarPair()**

El siguiente código extrae aleatoriamente un par de elementos del dataset cuya etiqueta es 'no plagio', es decir, no pertenecen al mismo grupo en el dataset.

```
%%%% GETDISSIMILARPAIR %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% · Extracción de un par con la etiqueta de NO plagio
%
%%%% INPUT:
%
%   - classLabel -------> Array -------> Vector con las clases (pares)
%
%%%% OUTPUT:
%
%   - pairIdx1 ---------> Int ---------> ID de la imagen 1 del par
%   - pairIdx2 ---------> Int ---------> ID de la imagen 2 del par
%   - pairLabel --------> Int ---------> 1 es plagio, 0 no plagio
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function  [pairIdx1,pairIdx2,label] = getDissimilarPair(classLabel)

    % Lista única de clases
    classes = unique(classLabel);

    % Elección aleatoria de una clase
    classesChoice = randperm(numel(classes),2);

    % Extracción de los indices de la clase elegida
    idxs1 = find(classLabel==classes(classesChoice(1)));
    idxs2 = find(classLabel==classes(classesChoice(2)));

    % Elección de dos imágenes aleatorias de la clase elegida
    pairIdx1Choice = randi(numel(idxs1));
    pairIdx2Choice = randi(numel(idxs2));
    pairIdx1 = idxs1(pairIdx1Choice);
    pairIdx2 = idxs2(pairIdx2Choice);

    % Marcamos que el par es plagio con un 0
    label = 0;

end
```

- **Función auxiliar: modelLoss()**

El siguiente código calcula la pérdida de la red. Primero llama a la función auxiliar forwardSiamese() para calcular una predicción sobre la similitud de los input. Posteriormente llama a la función auxiliar binarycrossentropy() para obtener la pérdida de entropía cruzada. Finalmente actualiza los gradientes de la red teniendo en cuenta los parámetros de aprendizaje.

```
%%%% MODELLOSS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   · Cálculo de pérdida de la red utilizando la fórmula:
%
%       loss = -t·log(y)-(1-t)log(1-y), y es la predicción, t€{0,1}
%
%%%% INPUT:
%
%   - net --------------> dlnetwork --------> Arquitectura de la red CNN
%   - fcParams ---------> Array ------------> Parámetros de la capa fc
%   - X1 ---------------> Array ------------> Imagen 1
%   - X2 ---------------> Array ------------> Imagen 2
%   - pairLabels -------> Array ------------> Etiquetas de plagio
%
%%%% OUTPUT:
%
%   - loss -------------> Double -----------> Pérdida media se la muestra
%   - gradientsSubnet --> Array ------------> Gradientes de la red
%   - gradientsParams --> Array ------------> Gradientes de los parámetros
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [loss,gradientsSubnet,gradientsParams] =
modelLoss(net,fcParams,X1,X2,pairLabels)

    % Obtención de la probabilidad de plagio
    Y = forwardSiamese(net,fcParams,X1,X2);

    % Cálculo binario de la pérdida cross-entropy media de la muestra
    loss = binarycrossentropy(Y,pairLabels);

    % Cálculo de los gradientes respecto a los parámetros de aprendizaje
    [gradientsSubnet,gradientsParams] = dlgradient(loss,net.Learnables,fcParams);

end
```

- **Función auxiliar: forwardSiamese()**

El siguiente código define la Red Neuronal Convolucional Siamesa, conectando dos redes idénticas y calculando una predicción probabilística sobre la similaridad de los datos input.

```
%%%% FORWARDSIAMESE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%   · Cálculo de la probabilidad de plagio de un par
%
%%%% INPUT:
%
%   - net ---------> dlNetwork ------> Arquitectura de la red CNN
%   - fcParams ----> Array ----------> Parámetros de la fc
%   - X1 ----------> Array ----------> Imagen 1
%   - X2 ----------> Array ----------> Imagen 2
%
%%%% OUTPUT:
%
%   - Y -----------> Double ---------> Probabilidad de plagio €[0,1]
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Y = forwardSiamese(net,fcParams,X1,X2)

    % Obtención de los vectores característicos de las dos imágenes
    % utilizando una misma arquitectura de red (red siamesa)
    Y1 = forward(net,X1);
    Y1 = sigmoid(Y1);
    Y2 = forward(net,X2);
    Y2 = sigmoid(Y2);

    % Calcula la distancia de los vectores característicos mediante su resta
    Y = abs(Y1 - Y2);

    % Cálculo de la suma ponderada del vector distancia utilizando los pesos y el
    % sesgo (bias) especificados:
    Y = fullyconnect(Y,fcParams.FcWeights,fcParams.FcBias);

    % Conversión de la probabilidad al intervalo [0,1]
    Y = sigmoid(Y);

end
```

- **Función auxiliar: binarycrossentropy()**

El siguiente código mide la diferencia entre las probabilidades predichas y las etiquetas binarias asociadas al par de datos input.

```
%%%% BINARYCROSSENTROPY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  · Cáclulo de la pérdida media de la muestra
%
%%%% INPUT:
%
%   - Y ------------> Double --------> Probabilidad del plagio
%   - pairLabels --> Array ---------> Etiquetas de plagio
%
%%%% OUTPUT:
%
%   - loss --------> Double --------> Pérdida media de la muestra
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function loss = binarycrossentropy(Y,pairLabels)

    % Definición del tipo de dato de la predicción (single o double)
    precision = underlyingType(Y);

    % Si el valor de una predicción se encuentra en los intervalos [0,eps]
    % o [1-eps, eps], siendo eps un valor pequeño asociado a la precisión de
    % los datos, se le asigna el valor eps o 1-eps, respectivamente.
    Y(Y < eps(precision)) = eps(precision);
    Y(Y > 1 - eps(precision)) = 1 - eps(precision);

    % Cálculo de la entropía cruzada binaria de cada par
    loss = -pairLabels.*log(Y) - (1 - pairLabels).*log(1 - Y);

    % Cálculo de la pérdida media en la muestra
    loss = sum(loss)/numel(pairLabels);

end
```

- **Función auxiliar: predictSiamese()**

El siguiente código utiliza la red entrenada para realizar una predicción sobre la similitud de las dos imágenes características introducidas como input.

```
%%%% PREDICTSIAMESE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% · Predicción de la probabilidad de plagio a través de la red
%
%%%% INPUT:
%
%   - net --------> dlnetwork ----------> Arquitectura de la red
%   - fcParams ---> Array --------------> Parámetros de la fully connect
%   - X1 ---------> Array --------------> Imagen 1
%   - X2 ---------> Array --------------> Imagen 2
%
%%%% OUTPUT:
%
%   - Y ----------> Array -------------->
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Y = predictSiamese(net,fcParams,X1,X2)

    % Obtener el vector característico de las imágenes del par input
    Y1 = predict(net,X1);
    Y1 = sigmoid(Y1);

    Y2 = predict(net,X2);
    Y2 = sigmoid(Y2);

    % Resta de los vectores característicos
    Y = abs(Y1 - Y2);

    % Operación fullyconnect
    Y = fullyconnect(Y,fcParams.FcWeights,fcParams.FcBias);

    % Transformación de la probabilidad en [0,1]
    Y = sigmoid(Y);

end
```