IMPLEMENTACIÓN DE UN SISTEMA DE PROCESAMIENTO DE LA VOZ HUMANA CON DEEP LEARNING QUE PERMITA EL RECONOCIMIENTO DE VOCABLOS

Trabajo de Grado

ANDRÉS FELIPE CAMARGO CAICEDO
LUIS GUSTAVO MORENO CAMARGO

Director: MsC. Harold Armando Romo Romero

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Grupo de Nuevas Tecnologías en Telecomunicaciones - GNTT
Línea de Investigación: Procesamiento de señales
Popayán, 2022.

IMPLEMENTACIÓN DE UN SISTEMA DE PROCESAMIENTO DE LA VOZ HUMANA CON DEEP LEARNING QUE PERMITA EL RECONOCIMIENTO DE VOCABLOS

ANDRÉS FELIPE CAMARGO CAICEDO
LUIS GUSTAVO MORENO CAMARGO

Trabajo de Grado presentado a la facultad de Ingeniería Electrónica y Telecomunicaciones de la Universidad del Cauca como requisito para optar al título de: Ingenieros en Electrónica y Telecomunicaciones

Director: MsC. Harold Armando Romo Romero

Universidad del Cauca
Facultad de Ingeniería Electrónica y Telecomunicaciones
Departamento de Telecomunicaciones
Grupo de Nuevas Tecnologías en Telecomunicaciones - GNTT
Línea de Investigación: Procesamiento de señales
Popayán, 2022

AGRADECIMIENTOS

Agradecemos a nuestros padres y familiares por el apoyo y sacrificio en nuestra formación profesional y personal a lo largo de todos estos años durante nuestra carrera, a nuestros compañeros y amigos con los que pudimos formar relaciones increíbles, donde aprendimos a fomentar valores de respeto y trabajo en equipo, a nuestros docentes quienes nos impartieron su conocimiento y dedicaron su tiempo a formarnos y guiarnos en este camino profesional. Un inmenso agradecimiento a nuestro director el MsC. Harold Romo quien nos impulsó a afrontar y culminar este reto, y a lo largo de nuestro paso por el campus nos ofreció su sabiduría y amistad.

RESUMEN

El presente trabajo de grado de investigación tiene como objetivo principal el implementar un sistema de transcripción de comandos de voz, basado en algoritmos de Aprendizaje Profundo (DL, *Deep Learning*), capaz de reconocer vocablos específicos, para evaluar el desempeño del sistema implementado en el Reconocimiento Automático del Habla (ASR, *Automatic Speech Recognition*). Para su funcionamiento, se crea una base de datos conteniendo ficheros tipo WAVE de 7 vocablos diferentes, se realiza el pre-procesado de las señales de audio y se extraen los atributos de interés, en este caso, los Coeficientes Cepstrales en las Frecuencias de MEL (MFCC, *Mel Frequency Cepstral Coefficients*), los cuales servirán como entrada del algoritmo de DL.

Para su desarrollo se utiliza la herramienta de programación Python, la cual ofrece librerías de mucha utilidad tanto en el manejo de archivos de audio, como en el desarrollo de aplicaciones en el campo del Aprendizaje Automático (ML, *Machine Learning*). Así mismo, Python permite con sus librerías el análisis de los resultados obtenidos a partir de las métricas seleccionadas para la evaluación del desempeño de este algoritmo en el reconocimiento de vocablos. Finalmente se concluye sobre los resultados obtenidos y se enumeran posibles adaptaciones y trabajos futuros.

TABLA DE CONTENIDO

Lista de Figuras ..... **VII**
Lista de Tablas ..... **IX**
Lista de Acrónimos ..... **X**
INTRODUCCIÓN **1**

1. GENERALIDADES **3**
1.1. INTELIGENCIA ARTIFICIAL 3
1.2. MACHINE LEARNING 3
1.3. DEEP LEARNING 4
1.4. RECONOCIMIENTO AUTOMÁTICO DEL HABLA 4
1.5. REDES NEURONALES 4
1.6. REDES NEURONALES RECURRENTES 5
1.7. LONG SHORT-TERM MEMORY 6
1.7.1. Forget Gate 7
1.7.2. Input Gate 8
1.7.3. Update Gate 9
1.7.4. Output Gate 9
1.8. FUNCIONES DE ACTIVACIÓN 10
1.8.1. Tanh 10
1.8.2. Softmax 11
1.9. APRENDIZAJE SUPERVISADO 12
1.9.1. Backpropagation 12
1.10. MÉTRICAS DE EVALUACIÓN 12
1.10.1. Precisión 13
1.10.2. Pérdida 13
1.11. OPTIMIZACIÓN DEL DESCENSO DEL GRADIENTE 14
1.11.1. Adam 14
1.12. REGULARIZACIÓN 15
1.12.1. Dropout 15
1.13. PRE-PROCESADO Y EXTRACCIÓN DE LOS MFCCs DEL AUDIO 15
1.13.1. Eliminación de silencio y Preénfasis 16
1.13.2. Enmarcado 18

1.13.3. Función de Ventana Hamming 18
1.13.4. Transformada de Fourier 19
1.13.5. Banco de Filtros de Escala Mel 19
1.13.6. Coeficientes Cepstrales en Frecuencias de Mel 20

2. HERRAMIENTAS 23
2.1. LENGUAJE DE PROGRAMACIÓN 23
2.2. LIBRERÍAS DE MANIPULACIÓN DE DATOS 23
2.2.1. NumPy 24
2.2.2. Librosa 24
2.2.3. CSV 24
2.3. LIBRERÍAS DE DEEP LEARNING 24
2.3.1. TensorFlow 24
2.3.2. SciKit-Learn 24
2.3.3. Keras 25
2.4. HARDWARE 25

3. DISEÑO 27
3.1. BASE DE DATOS 27
3.2. PRE-PROCESADO Y EXTRACCIÓN DE LOS MFCCs 36
3.3. ESTRUCTURA DE LA BASE DE DATOS 44
3.3.1. Datos de entrenamiento, validación y pruebas 44
3.4. ARQUITECTURA DE LA RED NEURONAL 45
3.5. ENTRENAMIENTO DE LA RED NEURONAL 45
3.6. ESTRUCTURA DEL PROYECTO 46

4. EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS 47
4.1. MODELO DE LA RED NEURONAL 47
4.2. MODELO DE LA RED NEURONAL CON USO DE PREÉNFASIS EN LA BASE DE DATOS 49
4.3. PRUEBAS 50

5. CONCLUSIONES Y TRABAJOS FUTUROS 55
5.1. CONCLUSIONES 55
5.2. TRABAJOS FUTUROS 56

Bibliografía 59
Apéndices 63
Apéndice A 63
Apéndice B 64
Apéndice C 75
Apéndice D 77

LISTA DE FIGURAS

Figura 1.1. Red Neuronal 5
Figura 1.2. Red Neuronal Recurrente 5
Figura 1.3. RNN estándar vs red LSTM 6
Figura 1.4. Cell state de una red LSTM 7
Figura 1.5. Función sigmoide 7
Figura 1.6. Forget gate 8
Figura 1.7. Input gate 8
Figura 1.8. Update gate 9
Figura 1.9. Output gate 10
Figura 1.10. Representación de la función tanh 11
Figura 1.11. Representación de la métrica de pérdida 13
Figura 1.12. Proceso de obtención de los MFCCs 16
Figura 1.13. Señal ejemplo sin silencios 17
Figura 1.14. Forma de onda señal original vs señal luego de aplicarle preénfasis 18
Figura 1.15. Aplicación de ventana Hamming a un marco 19
Figura 1.16. Banco de filtros en escala Mel 20
Figura 1.17. Representación de los MFCCs obtenidos de la señal ejemplo 21

Figura 3.1. Forma de onda del vocablo abrir 29
Figura 3.2. Forma de onda del vocablo adelante 29
Figura 3.3. Forma de onda del vocablo atrás 30
Figura 3.4. Forma de onda del vocablo cerrar 30
Figura 3.5. Forma de onda del vocablo derecha 31
Figura 3.6. Forma de onda del vocablo izquierda 31
Figura 3.7. Forma de onda del vocablo parar 32
Figura 3.8. Forma de onda del vocablo abrir sin el silencio inicial 33
Figura 3.9. Forma de onda del vocablo adelante sin el silencio inicial 33
Figura 3.10. Forma de onda del vocablo atrás sin el silencio inicial 34
Figura 3.11. Forma de onda del vocablo cerrar sin el silencio inicial 34
Figura 3.12. Forma de onda del vocablo derecha sin el silencio inicial 35
Figura 3.13. Forma de onda del vocablo izquierda sin el silencio inicial 35
Figura 3.14. Forma de onda del vocablo parar sin el silencio inicial 36

Figura 3.15. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo abrir 37
Figura 3.16. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo adelante 37
Figura 3.17. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo atrás 38
Figura 3.18. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo cerrar 38
Figura 3.19. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo derecha 39
Figura 3.20. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo izquierda 39
Figura 3.21. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo parar 40
Figura 3.22. MFCCs del vocablo abrir 41
Figura 3.23. MFCCs del vocablo adelante 41
Figura 3.24. MFCCs del vocablo atrás 42
Figura 3.25. MFCCs del vocablo cerrar 42
Figura 3.26. MFCCs del vocablo derecha 43
Figura 3.27. MFCCs del vocablo izquierda 43
Figura 3.28. MFCCs del vocablo parar 44
Figura 3.29. Arquitectura de la NN 45

Figura 4.1. Métricas de evaluación del modelo 48
Figura 4.2. Métricas de evaluación del modelo con aplicación de preénfasis 50

LISTA DE TABLAS

Tabla 4.1. Resultados del modelo 48
Tabla 4.2. Comparación de los resultados de los modelos 49
Tabla 4.3. Predicción de las pruebas del locutor 1 51
Tabla 4.4. Predicción de las pruebas del locutor 2 52
Tabla 4.5. Predicción de las pruebas del locutor 3 52
Tabla 4.6. Predicción de las pruebas del locutor 4 75
Tabla 4.7. Predicción de las pruebas del locutor 5 76

LISTA DE ACRÓNIMOS

Adam	"*Adaptive Moment Estimation*" - Estimación de Momento Adaptativo
AI	"*Artificial Intelligence*" - Inteligencia Artificial
ANN	*"Artificial Neural Networks"* - Redes Neuronales Artificiales
ASR	"*Automatic Speech Recognition*" - Reconocimiento Automático del Habla
CMVN	"*Cepstral Mean and Variance Normalization*" - Normalización de Media y Varianza Cepstral
CPU	"*Central Processing Unit*" - Unidad de Procesamiento Central
DCT	"*Discrete Cosine Transform*" - Transformada Discreta del Coseno
DL	"*Deep Learning*" - Aprendizaje Profundo
FFT	"*Fast Fourier Transform"* - Transformada Rápida de Fourier
GPU	"*Graphics Processing Unit*" - Unidad de Procesamiento Gráfico
MFCC	"*Mel Frequency Cepstral Coefficients*" - Coeficientes Cepstrales en las Frecuencias de MEL
ML	"*Machine Learning*" - Aprendizaje Automático
NN	"*Neural Network"* - Red Neuronal
RNN	*"Recurrent Neural Network"* - Red Neuronal Recurrente
SNR	"*Signal Noise Ratio*" - Relación Señal a Ruido
STFT	*"Short Time Fourier Transform"* - Transformada de Fourier de Tiempo Reducido
TICs	Tecnologías de la Información y las Comunicaciones

INTRODUCCIÓN

La Inteligencia Artificial (AI, *Artificial Intelligence*) y todas las nuevas tecnologías están revolucionando el mundo, siendo utilizadas muchas técnicas y aplicaciones en diversas disciplinas, obteniendo resultados sorprendentes gracias al aumento de la capacidad de cómputo y a la aparición de herramientas que permiten realizar cálculos complejos de una forma más eficiente.

Asistentes virtuales de voz como Alexa, Siri, Cortana, Bixby, Google Assistant, demuestran que las tecnologías de Reconocimiento Automático del Habla (ASR, *Automatic Speech Recognition*) se han integrado de una manera rápida y progresiva en ámbitos sociales como el familiar, laboral y también en el ámbito de la salud. Estos asistentes son herramientas muy atractivas, ya que, la posibilidad de interacción a través del habla da un paso hacia la comodidad, dinamización y cumplimiento de funciones específicas en la velocidad y/o inmediatez que requieren las nuevas tecnologías [1].

El uso de las Tecnologías de la Información y las Comunicaciones (TICs) representa una posibilidad para la inclusión y normalización de las personas con discapacidad, ya sea en un entorno de casa, educativo y/o laboral, dado que pueden contribuir al mejoramiento de la calidad de vida, fortaleciendo su independencia e individualidad [2].

Por lo anterior, en este trabajo se analizó el desempeño de un modelo de Aprendizaje Profundo (DL, *Deep Learning*) para el reconocimiento de vocablos, para ello se realizó primero un trabajo de campo que consistió en realizar las grabaciones de vocablos específicos pronunciados por diferentes personas; esto, con el objetivo de construir la base de datos para el algoritmo propuesto, seguidamente se desarrolló en la herramienta software Python una etapa de pre-procesamiento a las grabaciones, eliminando el silencio y un filtro de preénfasis para suavizar el espectro. Posteriormente se extraen los Coeficientes Cepstrales en las Frecuencias de MEL (MFCC, *Mel Frequency Cepstral Coefficients*). Una vez construida la base de datos se determinan los porcentajes respectivos de los datos para entrenamiento, validación y pruebas. Finalmente se entrena el algoritmo de reconocimiento de vocablos y se evalúa su desempeño a partir de métricas de evaluación consignadas en una *tabla de precisión y pérdidas.*

Para el desarrollo del proyecto se sigue la guía de referencia del Modelo Lineal Secuencial [3]. Este modelo sugiere un enfoque sistemático y secuencial que se basa en 5 fases:

1. Fase de recolección de información
2. Fase de análisis y síntesis de información
3. Fase de diseño e implementación
4. Fase de pruebas y análisis de resultados
5. Fase de entrega

El desarrollo de este trabajo de grado se estructura en 4 secciones: la primera sección comprende los capítulos 1 *(Generalidades)* y 2 *(Herramientas)*, donde se mencionan y se explican cada uno de los conceptos y herramientas necesarias para entender e implementar el sistema planteado. La segunda sección comprende el capítulo 3 *(Diseño)*, donde se detallan los pasos necesarios para concebir el sistema planteado, metodología de simulación y desarrollo del sistema. En la tercera sección, se explica el modelo de DL utilizado, se analizan los resultados obtenidos y se realizan las pruebas finales del sistema, expuestos en el capítulo 4 *(Experimentación y análisis de resultados)* y finalmente en la cuarta sección, expuesta en el capítulo 5 *(Conclusiones y trabajos futuros)*, se concluye sobre los resultados obtenidos y se proponen trabajos futuros para optimizar resultados y aplicar el sistema desarrollado.

Capítulo 1

GENERALIDADES

1.1. INTELIGENCIA ARTIFICIAL

El concepto de Inteligencia Artificial (AI, *Artificial Intelligence*), fue propuesto por John McCarthy por primera vez en el año 1956, siendo esta una nueva ciencia y tecnología que se enfoca en la búsqueda y desarrollo de teorías, métodos, técnicas y sistemas de aplicación para simular y expandir la inteligencia humana. Incluyendo tareas como la comprensión de lenguaje, reconocimiento de imágenes, sonidos y aprendizaje para resolución de problemas [4,5].

La AI se clasifica según la finalidad de su investigación como: Inteligencia Artificial débil y fuerte: definiendo que la AI débil solo considera el razonamiento de la máquina sin llegar a tener conciencia, obedeciendo órdenes orientadas a una única tarea. La AI fuerte, considera la conciencia de la máquina, pudiendo igualar las capacidades humanas adaptándose a diferentes tareas, ambientes y situaciones [6].

1.2. MACHINE LEARNING

El Aprendizaje Automático (ML, *Machine Learning*) es un área central de la AI. Se centra en el estudio de cómo un sistema adquiere nuevos conocimientos o habilidades simulando o realizando un comportamiento del aprendizaje humano y reorganizando la arquitectura del conocimiento existente para mejorar su desempeño a partir del *entrenamiento* de un algoritmo, y así poder aprender. El entrenamiento conlleva alimentar al algoritmo con grandes cantidades de datos y a medida que se entrenan, estos algoritmos mejoran la precisión de sus resultados en el tiempo [4,5].

El manejo de los algoritmos de ML depende en gran medida del formato de representación de los datos que se les da, ya que éste define los fragmentos en los cuales se tiene la posibilidad de dividir los datos, denominados atributos o *features* [7].

1.3. DEEP LEARNING

El Aprendizaje Profundo (DL, *Deep Learning*) es uno de los muchos posibles enfoques del ML que entrena a una computadora, basado en procedimientos algorítmicos para lograr que realice labores similarmente como las hace un ser humano, tales como el reconocimiento del habla, la identificación de imágenes o hacer predicciones. El DL fue inspirado en la estructura y funcionamiento del cerebro, es decir, la interconexión de muchas neuronas. Las redes neuronales artificiales (ANN, *Artificial Neural Networks*) son algoritmos que imitan la estructura biológica del cerebro [4,8].

El DL configura parámetros básicos sobre los datos y entrena a la computadora para que aprenda por cuenta propia, reconociendo patrones por medio de la utilización de muchas capas de procesamiento, siendo ésta la gran diferencia con ML. El DL requiere muchos más datos iniciales y potencia de cómputo que el ML [8].

1.4. RECONOCIMIENTO AUTOMÁTICO DEL HABLA

El Reconocimiento Automático del Habla (ASR, *Automatic Speech Reconigtion*) es una disciplina dentro de la AI, la cual brinda a las máquinas la capacidad de recibir señales sonoras, resultado del habla humana, por medio de un micrófono, donde una vez captado el mensaje se extraen las características fundamentales en su forma numérica para realizar un análisis de su significado y así generar una respuesta adecuada [6].

La máquina toma como entrada una expresión hablada y entrega una cadena de caracteres que representan lo grabado, es decir frases habladas en forma de texto como salida. Para el desarrollo de un buen sistema de reconocimiento se hace necesario disponer de una gran cantidad de datos para obtener los resultados esperados [6].

1.5. REDES NEURONALES

Las Redes Neuronales (NN, *Neural Networks*) son modelos matemáticos que siguen el enfoque de DL, el cual es una estructura de conexión de nodos o neuronas que permiten la fluidez de la información, de tal forma que cada neurona identifica un determinado atributo, para así filtrar la información y obtener una salida de la red [5], tal como se muestra en la figura 1.1.

Las NN son sistemas basados en la estructura del cerebro humano, el cual puede ser definido como un modelo de 2 capas mínimo: entrada y salida, entre más complejo sea el problema que la NN debe solucionar, más capas intermedias (*Hidden Layers*) serán necesarias. Cada capa está compuesta de neuronas que recogen, por un lado, los valores de salida de la capa anterior, multiplicándolos por unos pesos (w), los cuales permiten el impulso o disparo de la entrada;

estos pesos son generados de manera aleatoria en un rango de [-1,1], luego de esto, a este valor obtenido se le suma el valor de un sesgo (*bias*), que controla qué tan predispuesta está la neurona a disparar un 1 o un 0, independiente de los pesos; y una función de activación, que se encarga de calcular y ponderar el valor de salida de la neurona. Estas neuronas suelen modelarse con funciones de activación no-lineales como las funciones *sigmoideas* [9].

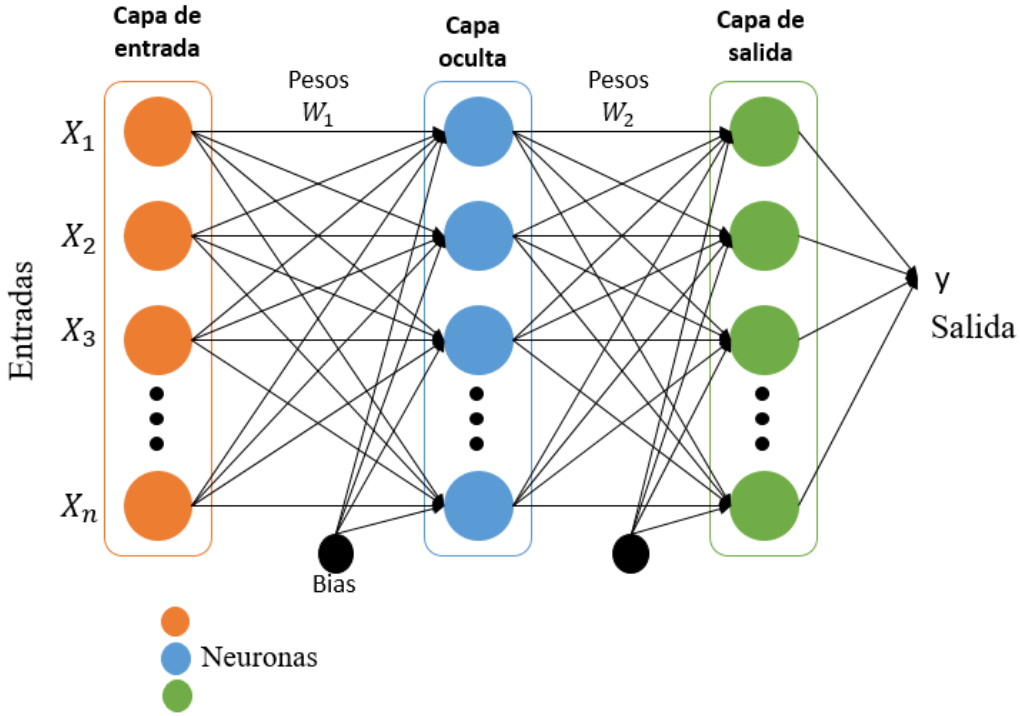

Figura 1.1. Red Neuronal (elaboración propia).

1.6. REDES NEURONALES RECURRENTES

Las Redes Neuronales Recurrentes (RNN, *Recurrent Neural Networks*), son aquellas que tienen una memoria interna, esto quiere decir que una determinada capa, además de utilizar la entrada actual, utiliza la salida que ha aprendido la capa anterior como su entrada, de tal manera que la salida de una capa determinada se propaga a la primera capa oculta, haciendo que esta funcione como capa de entrada. En otras redes neuronales, cada una de las entradas son independientes entre sí. Pero en RNN, las entradas están relacionadas entre sí [10].

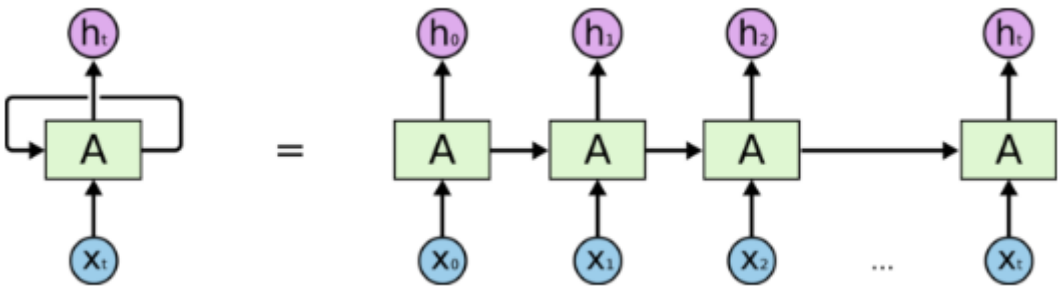

Figura 1.2. Red Neuronal Recurrente. Tomado de [10].

En la figura 1.2, se observa al lado izquierdo de forma simplificada la arquitectura de una RNN y al lado derecho la arquitectura de manera desenrollada, donde primero, toma X_0 de la secuencia de entrada y luego genera h_0 que junto con X_1 es la entrada para el siguiente paso. De manera similar, h_1 es la entrada con X_2 para el siguiente paso y así sucesivamente, siendo t el instante actual [10].

1.7. LONG SHORT-TERM MEMORY

Las redes de Memoria a Corto y Largo Plazo (LSTM, *Long Short - Term Memory*) son un tipo de RNN, las cuales tienen la capacidad de memoria a corto y largo plazo, diseñadas con la finalidad específica de poder recordar información relevante durante varios periodos de tiempo. Esto para dar solución al problema que tenían las RNN estándar las cuales presentaban el denominado Desvanecimiento del Gradiente (*Vanishing Gradient*) o dificultad de aprendizaje de dependencias a largo plazo [11].

Tanto las RNN estándar como las redes LSTM poseen una estructura de módulo en cadena. Por un lado, las RNN estándar poseen una composición simple en sus módulos, una única capa que compone a la función de activación Tangente Hiperbólica (*tanh*). Por otro lado, las LSTM se componen de algo un poco más complejo, cuatro capas las cuales conforman lo que se llamaría una *celda LSTM*, tal como se muestra en la figura 1.3.

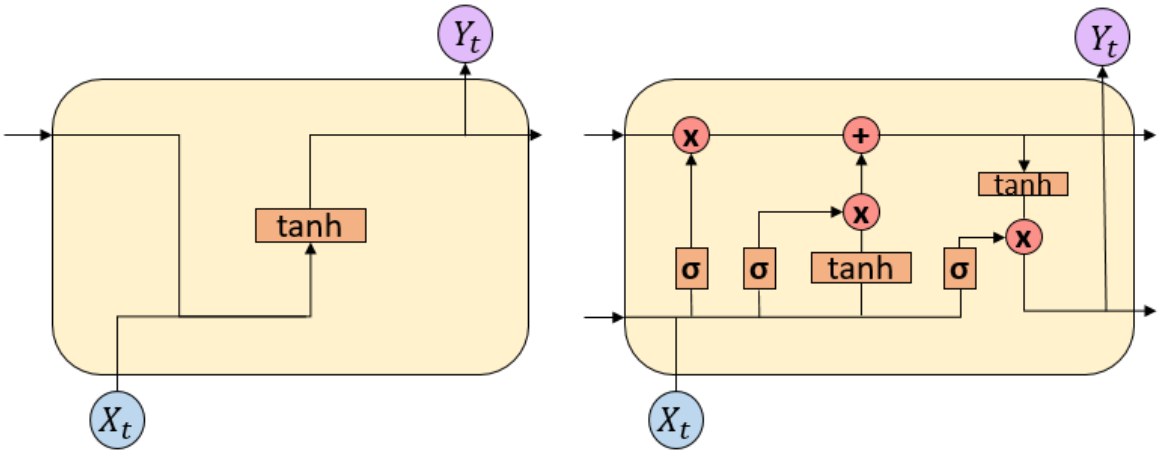

Figura 1.3. RNN estándar vs Red LSTM (elaboración propia).

En comparación con la unidad de una RNN básica, la unidad LSTM tiene una entrada y una salida adicional. Esta relación adicional se denomina como Estado de Celda o (*Cell State*), tal como se muestra en la figura 1.4. La *cell state* es clave para el funcionamiento de una red LSTM, dado que es la que permite la memoria a largo plazo, esto mediante la eliminación u olvido y adición o actualización de los datos que se requieren mantener en la memoria, para así generar una salida óptima en cada celda. Estos procesos de eliminación / adición de datos son posibles gracias a la cuidadosa regulación de unas compuertas especiales que componen a las capas de la celda [11, 12].

Mientras que las RNN estándar sólo mantienen un vector de estado h_t, las celdas LSTM dividen el estado actual de la celda en h_t y C_t, interpretándose así el primero como el *short-term* y el segundo como el *long-term*.

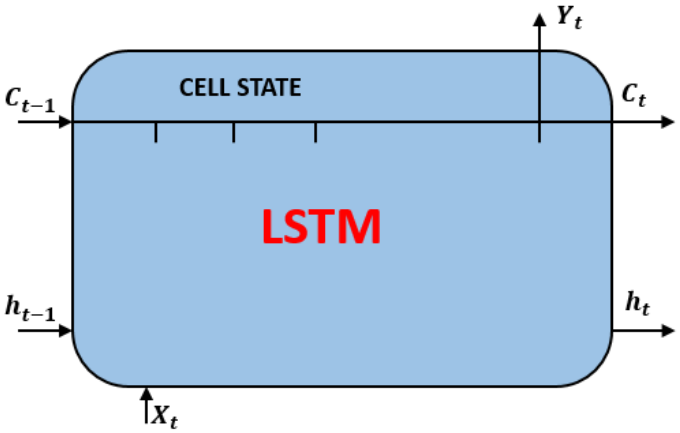

Figura 1.4. Cell State de una red LSTM (elaboración propia).

Cada capa de la celda se compone de los datos entrantes de la red neuronal, una función sigmoidal, la cual funciona como una válvula que decide la cantidad de información que seguirá el flujo, además de una operación producto o multiplicación puntual. Dicho sigmoide genera valores entre cero y uno, como se observa en la figura 1.5, los cuales definen la cantidad de información que se agrega o elimina de la *cell state* [13].

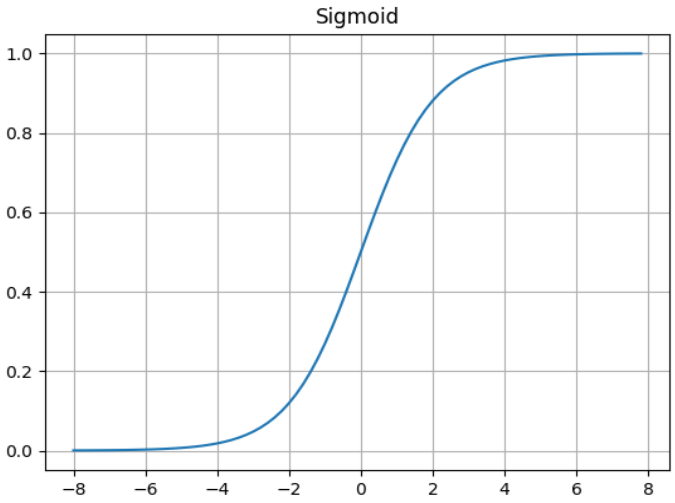

Figura 1.5. Función sigmoide (elaboración propia).

1.7.1. Forget Gate

La capa *Forget Gate* es la encargada de descartar u olvidar la información que no se requiere que pase a formar parte del *cell state*. Para esto la capa toma la entrada actual de la celda X_t y el estado oculto anterior h_{t-1} devolviendo valores entre cero y uno actuando directamente en el

flujo del C_t, dicha salida de la capa se expresa matemáticamente como f_t, según la ecuación 1.1 y como se observa en la figura 1.6 [13].

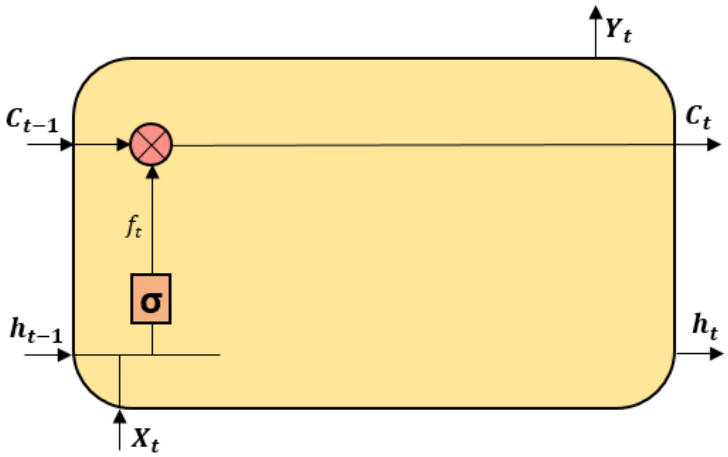

Figura 1.6. Forget gate (elaboración propia)

$$f_t = \text{Sigmoide}\big(W_f[h_{t-1}\,,\,X_t] + b_f\big), \tag{1.1}$$

donde los coeficientes presentes en la ecuación (W_f y b_f) son valores que la red aprenderá durante el proceso de entrenamiento, los cuales representan la matriz de pesos de la celda actual y el *bias* de la celda respectivamente [12].

1.7.2. Input Gate

La capa *Input Gate* es la que se encarga de decidir qué información se puede adicionar al *cell state*, para esto se toma nuevamente el estado oculto anterior h_{t-1} y la entrada actual X_t, llevándolos nuevamente a una función sigmoidal, la cual decide qué tanta información se adiciona al estado actual de la celda, al generar valores entre cero y uno. Esta capa genera una salida que se denomina i_t según la ecuación 1.2 y como se observa en la figura 1.7, la cual controla la cantidad de información saliente de la siguiente capa (*Update Gate*) qué se adicionará al *long-term* C_t [12, 13].

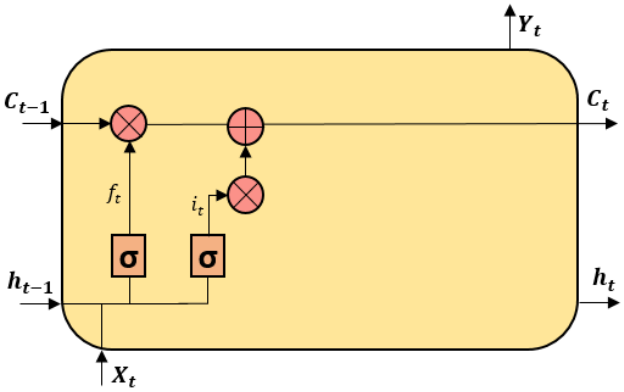

Figura 1.7. Input gate (elaboración propia)

$$i_t = \text{Sigmoide}(W_i[h_{t-1}\,,\,X_t] + b_i). \tag{1.2}$$

1.7.3. Update Gate

La capa *Update Gate* es la que se encarga de generar un vector de los nuevos datos que se van a adicionar a la *cell state* generando una salida g_t como se muestra en la figura 1.8, esto regulado por la capa previa (*Input Gate*), así se conseguirá la actualización del estado de la celda actual, según la ecuación 1.3. Al ser una función de activación tanh, genera valores entre -1 y 1 (el rango que manejan los estados ocultos en cada celda LSTM) [13].

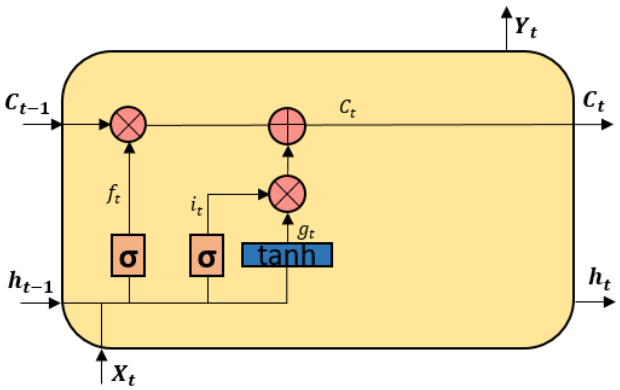

Figura 1.8. Update gate (elaboración propia)

$$g_t = \tanh\left(W_g[h_{t-1}\,,\, X_t] + b_g\right). \tag{1.3}$$

Ahora, en una visión matemática para realmente actualizar el *cell state* anterior y generar el de la celda actual, según la ecuación 1.4, primeramente, se realiza la multiplicación de la salida de la primera capa f_t con C_{t-1} , para luego adicionarlo al producto entre la salida de las dos capas siguientes, $i_t \cdot g_t$, y así generar los valores escalados a formar parte del C_t actual [7].

$$C_t = f_t \cdot C_{t-1} + i_t \cdot g_t\,. \tag{1.4}$$

1.7.4. Output Gate

La capa *Output Gate* es la que entrega la salida de la celda LSTM, se realiza un escalamiento mediante la función de activación tanh, la cual genera valores entre -1 y 1, para luego definir qué porciones de dicho *cell state* escalado formarán parte del nuevo estado oculto, esto mediante la función sigmoide, según la ecuación 1.5. El nuevo estado oculto es el valor que se usa como una de las entradas de la celda posterior. Este nuevo estado oculto es el producto entre el valor de o_t y el valor resultante de la función tanh del estado actual de la celda C_t, la cual se expresa matemáticamente como Y_t, en la ecuación 1.6 y se observa en la figura 1.9.

Este nuevo estado oculto puede interpretarse como una versión final filtrada del *cell state* que se generó mediante los procesos de las capas anteriores [13].

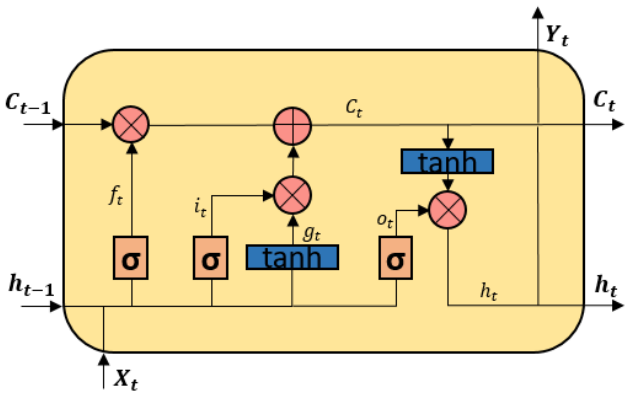

Figura 1.9. Output gate (elaboración propia)

$$o_t = \text{Sigmoide}(W_o[h_{t-1}\,,\, X_t] + b_o). \quad (1.5)$$

$$h_t = o_t \cdot \tanh(C_t) = Y_t\,. \quad (1.6)$$

1.8. FUNCIONES DE ACTIVACIÓN

La función de activación es importante para los modelos de NN porque permite la introducción de características no lineales a la red y devuelve la salida que va a ser generada por la neurona, dada una entrada o grupo de entradas. Además, se debe tener en cuenta que en la NN se utilizará una función no lineal ya que le posibilita al modelo ajustarse para trabajar con la más grande proporción de datos posible. Todas las capas que componen la NN poseen una función de activación que permitirá obtener los datos a predecir [5, 15].

1.8.1. Tangente Hiperbólica

La función tanh es utilizada en las redes LSTM como función de activación debido a la necesidad de una función cuya segunda derivada se pudiera mantener durante un largo rango de tiempo antes de caer a cero y con esto poder obtener que las celdas pudieran retener contextos a largo plazo [16]. En la figura 1.10 se observa la representación de esta función.

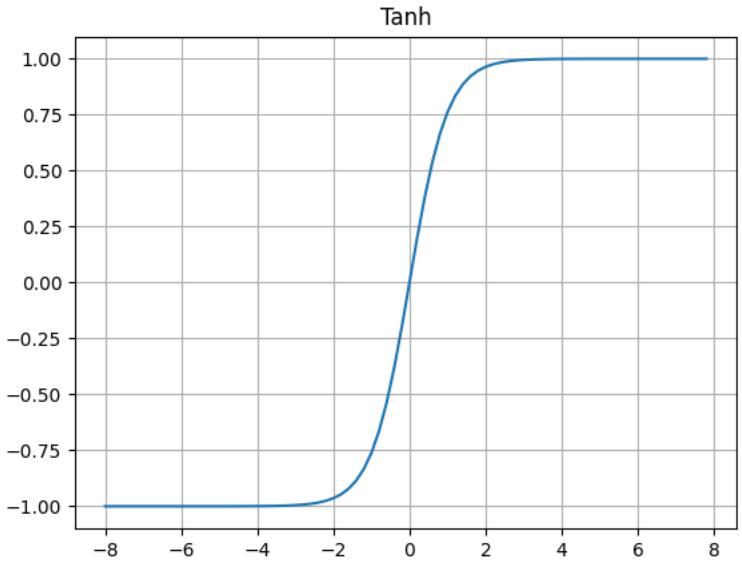

Figura 1.10. Representación de la función Tanh (elaboración propia).

El principal inconveniente de esta función es que cuando la red es muy profunda, se tiene el fenómeno llamado desvanecimiento del gradiente, es decir, que el valor del gradiente cae al área de saturación y se va haciendo cada vez más pequeño, llegando a ser cercano a cero, esto debido al algoritmo de propagación hacia atrás (*Backpropagation*) [5].

1.8.2. Softmax

Softmax es una función de activación normalmente usada para hacer clasificación de múltiples clases, es utilizada en la capa de salida de la red, donde está retorna un vector k-dimensional de valores reales como se muestra en la ecuación 1.7, siendo este un vector de probabilidades perteneciente a cada una de las clases posibles y asignando la salida al valor con mayor probabilidad. Al sumar todos los elementos del vector da como resultado un valor de 1 [5].

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}, \tag{1.7}$$

donde:

- $\vec{\mathbf{z}}$: El vector de entrada a la función softmax, compuesto por (z_0, ... z_k).
- $\mathbf{e^{z_i}}$**:** La función exponencial estándar se aplica a cada elemento del vector de entrada. Esto da un valor positivo por encima de 0, que será muy pequeño si la entrada fue negativa y muy grande si la entrada fue grande. Sin embargo, todavía no está fijo en el rango (0, 1) que es lo que se requiere de una probabilidad.
- $\mathbf{\sum_{j=1}^{K} e^{z_j}}$: El término en la parte inferior de la fórmula es el término de normalización. Asegura que todos los valores de salida de la función sumarán 1 y cada uno estará en el rango (0, 1), constituyendo así una distribución de probabilidad válida.
- **K:** El número de clases en el clasificador multiclase [30].

Las etiquetas utilizadas para entrenar una red con la función de activación softmax en la capa de salida serán vectores con 1 para la clase de destino y 0 para todas las demás clases [17].

1.9. APRENDIZAJE SUPERVISADO

El Aprendizaje Supervisado es una metodología de entrenamiento. En las NN se puede optar por 4 tipos de aprendizaje: El supervisado, semi-supervisado, no supervisado o por refuerzo. El más adecuado para este caso es el aprendizaje supervisado, esto con el fin de disminuir al máximo el tiempo de entrenamiento [18].

Con esta metodología se obtiene un modelo óptimo, con el rendimiento requerido en entrenamiento y el aprendizaje basado en las muestras de categorías conocidas, es decir, se entrena al algoritmo pasándole las preguntas, denominadas *características*, y las respuestas, denominadas *etiquetas*, con el fin de que el algoritmo pueda combinarlas y hacer las predicciones [5].

1.9.1. Backpropagation

El *Backpropagation* es un método de cálculo del gradiente para realizar el entrenamiento de la red, donde se debe encontrar el valor óptimo de los pesos con el objetivo de minimizar el error en la salida, re-ajustando los pesos de las conexiones entre las capas de neuronas. Para este fin se utiliza el algoritmo de entrenamiento de la red llamado Backpropagation [19].

El algoritmo funciona de forma iterativa, multiplicando los valores de la entrada con los pesos calculados, avanzando así hasta la capa de salida de la red y comparando el valor obtenido con el esperado para así obtener un error de salida. A partir de ahí empieza a re-ajustar los pesos propagando su error hacia atrás hasta llegar a la capa de entrada [19].

Para el reajuste de los pesos se usa un algoritmo llamado descenso de gradiente (*Gradient Descent*). La dirección negativa del gradiente es la dirección de descenso más rápida de la función, es decir, que los pesos de la red se modifican siguiendo la dirección donde se espera exista el punto mínimo del error [5, 19].

1.10. MÉTRICAS DE EVALUACIÓN

Las métricas de evaluación tienen como objetivo determinar el desempeño del sistema a estudiar, en este caso, la capacidad de predicción. Para esto, se hace uso de una tabla de precisión (*accuracy)* y pérdida, las cuales dan noción del porcentaje de aciertos del sistema en cuanto al total de las predicciones y en cuanto al margen de error [19].

1.10.1. Precisión

La precisión es un tipo de métrica qué representa el porcentaje de muestras correctamente clasificadas, dada por la proporción del número de predicciones correctas entre el número total de predicciones como se observa en la ecuación 1.8. Para el uso de esta métrica se debe tener en cuenta que no funciona bien con las clases desequilibradas que pueden tener muchos elementos de una misma clase e incluir pocos elementos de otras clases [19, 20].

$$Precisión = \frac{Predicciones\ correctas}{Total\ de\ predicciones}\ . \tag{1.8}$$

1.10.2. Pérdida

La Pérdida es un tipo de métrica qué se relaciona con la probabilidad de predicción de una entrada, la cual resulta adecuada, dado que al usar la función de activación Softmax en la capa de salida del modelo, ésta da como resultado las probabilidades de las clases a ser predichas. En esta métrica, mientras la probabilidad predicha sea mas cercana a cero, la pérdida será mucho mayor, por lo tanto, la predicción realizada será diferente de la etiqueta real. Por consiguiente, una pérdida menor es mejor [20].

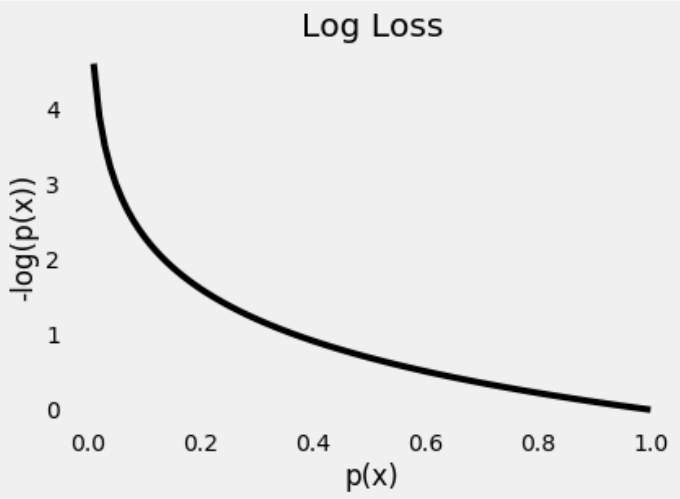

Figura 1.11. Representación de la función de pérdida

En la figura 1.11 se observa de forma gráfica como se puede calcular la pérdida logarítmica, donde esta métrica lo que hace es aplicar un menos logaritmo en base e o también conocido como el logaritmo natural a la probabilidad *p(x)* predicha por la capa de salida, esto se expresa matemáticamente en la ecuación 1.9 [20],

$$Loss = -\log_e\big(p(x)\big) = -\ln\big(p(x)\big)\ [nats], \tag{1.9}$$

donde *nat* es la unidad natural de información, se utiliza para medir la información o la entropía, es decir, mide la cantidad de información de una distribución de probabilidad o evento [32, 33].

1.11. OPTIMIZACIÓN DEL DESCENSO DEL GRADIENTE

La Optimización del Descenso del Gradiente tiene como objetivo la reducción de la función de error y modificar durante el entrenamiento y de forma automática el valor de los híper parámetros. También es el encargado del reajuste de los pesos basándose en el cálculo del gradiente de la función de coste por cada peso de la red, modificándolo en la dirección negativa del gradiente [7, 21].

Hay varios tipos de algoritmos de optimización del descenso de gradiente en la convergencia de la función de pérdida. En este caso, la mejor opción como optimizador es la Estimación de Momento Adaptativo (Adam, *Adaptive Moment Estimation*), donde los autores muestran de forma empírica que *Adam* tiene un buen funcionamiento en diferentes problemas, en comparación con otros algoritmos [21].

1.11.1. Algoritmo de Estimación de Momento Adaptativo (Adam)

Adam es el algoritmo de optimización de mayor efectividad usado en DL para obtener los parámetros de la red como la matriz de pesos y *bias*. [21].

Dicho algoritmo de optimización sigue el gradiente negativo de la función objetivo hasta alcanzar el valor mínimo de la misma. *Adam* es una extensión del descenso del gradiente que a diferencia de otros optimizadores adapta una única tasa de aprendizaje para cada variable de la función objetivo y así acelerar el proceso de optimización, esto disminuyendo el número de iteraciones en la red para alcanzar los parámetros deseados [5, 31].

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \Delta W. \quad (1.10)$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\Delta W)^2, \quad (1.11)$$

donde:

- ***m* y *v*:** representan a los dos momentos, siendo *m* el que modela la media de los gradientes a lo largo del tiempo, mientras que *v* lo hace para las varianzas.
- ***β1:*** representa la tasa de disminución exponencial de la estimación de momento de primer orden, tomando un valor en la mayoría de los casos de 0.9.
- ***β2:*** representa la tasa de disminución exponencial de la estimación de momento de segundo orden, tomando un valor en la mayoría de los casos de 0.99.
- ΔW**:** representa la derivada de la matriz de pesos, es decir el gradiente.

En la ecuación 1.12 se muestra de forma matemática como se calculan los pesos a partir de los momentos *m* y *v*, donde α a la tasa de aprendizaje y ε ($1e^{-8}$, valor predeterminado) un valor muy pequeño que evita divisiones por cero [31].

$$W_t = W_{t-1} - \frac{\propto \cdot m}{\sqrt{v+\varepsilon}}. \quad (1.12)$$

1.12. REGULARIZACIÓN

La regularización es una técnica importante y efectiva para mitigar el sobre entrenamiento en los sistemas de ML y DL. Esta técnica es especialmente utilizada en modelos de DL debido a la gran cantidad de parámetros qué se manejan para las entradas de la red neuronal. [5, 19]. Hay varias técnicas de regularización efectivas para mitigar el sobre entrenamiento. Para este trabajo de grado, se optó por utilizar *Dropout* ya que con esta técnica se obtuvieron buenos resultados en el sistema final.

1.12.1. Dropout

El *Dropout* es una técnica de regularización común y simple, la cual consiste en desactivar aleatoriamente cierto número de neuronas durante el proceso de entrenamiento. Al utilizar *Dropout,* la red se ve forzada a aprender nuevas formas de correlacionar las variables, ya qué dependiendo del porcentaje aplicado para esta técnica, cierta cantidad de neuronas quedarán desactivadas [5, 19].

1.13. PRE-PROCESADO Y EXTRACCIÓN DE LOS MFCCs DEL AUDIO

Antes de la ingesta de datos a la NN, se define como primer paso en el tratamiento de datos, las fases de pre-procesamiento y extracción de los MFCCs, las cuales permiten el acondicionamiento de los archivos de audio y la respectiva extracción de sus características principales. Esto debido a que se debe asegurar que la NN reciba la versión más adecuada y clara de la información contenida en cada fichero de audio, ya que la NN no trabaja con los datos en bruto, es decir, con los ficheros de audios como entrada.

Para el proceso de extracción de los MFCCs se requiere realizar una serie de operaciones de forma secuencial sobre cada una de las señales de audio contenidas en la base de datos. En la figura 1.12 se puede observar el orden de la secuencia a seguir.

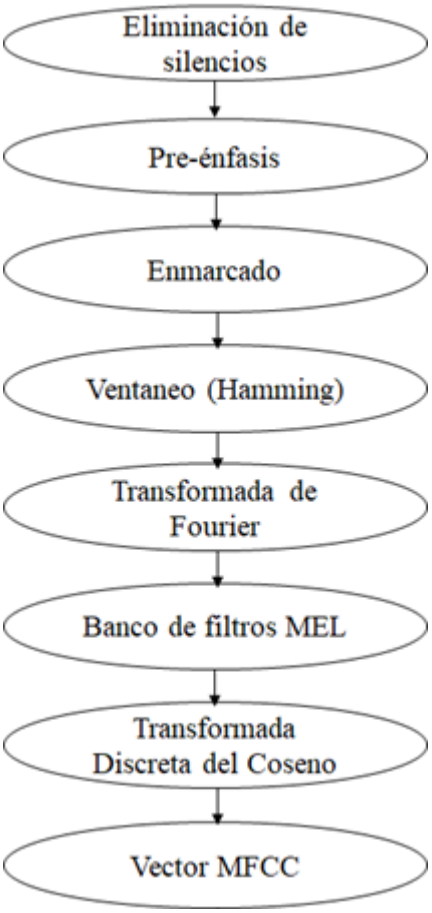

Figura 1.12. Proceso de obtención de los coeficientes MFCC (elaboración propia).

1.13.1. Eliminación de silencios y Preénfasis

Dentro del pre-procesamiento de los datos, se tuvo como primer paso la eliminación de tiempos sin información o silencios. Para este caso, se eliminó únicamente el silencio inicial, dejando una cola o tiempo de guarda para garantizar que no se borraran muestras del audio qué tuvieran un bajo nivel de amplitud pero qué auditivamente son parte de la vocalización de la palabra, de igual manera, dicha cola es relativamente corta, debido a qué hay palabras largas con una duración de vocalización amplia que podían superar un segundo y no ser tenidas en cuenta en la base de datos, debido a que se tomó únicamente un segundo de audio por cada fichero, con el fin de obtener la misma cantidad de segmentos en cada uno de éstos. En la figura 1.13, se puede observar la forma de onda de una señal ejemplo sin el silencio inicial. Estas señales ejemplo fueron grabadas a una frecuencia de 44100 Hz.

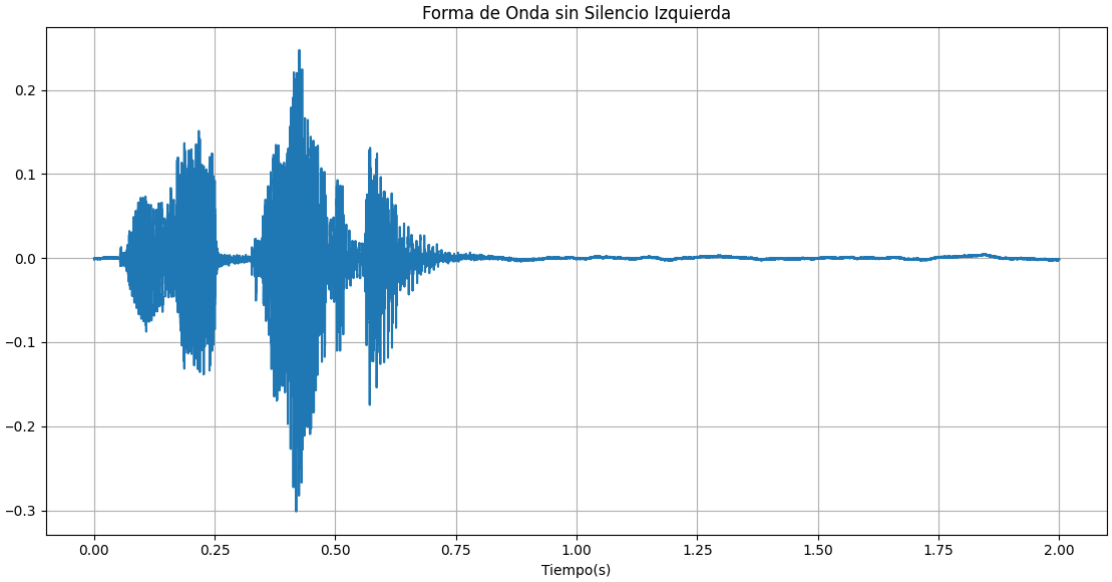

Figura 1.13. Señal ejemplo sin silencios (elaboración propia).

Luego de esto, se aplica un filtro de preénfasis con el objetivo de amplificar las frecuencias altas y compensar la atenuación de aproximadamente 20 dB/década que se produce en la señal al pasar a través del tracto vocal [22]. Lo expresado anteriormente corresponde a un filtraje pasa-altas.

El filtrado puede aplicarse a una señal *x(n)* tal y como se muestra en la ecuación 1.13, donde la expresión $x(n-1)$ hace referencia a la señal *x(n)* con un corrimiento de una muestra hacia la derecha; *n* representa las muestras de la señal; siendo α el coeficiente del filtro con valores entre 0.9 y 1.0; en este caso se toma un $\alpha = 0{,}97$ [23]:

$$y(n) = x(n) - \alpha\, x(n-1). \tag{1.13}$$

En la figura 1.14, se puede observar el resultado sobre una señal luego de aplicarle el filtro de preénfasis, donde a simple vista son apreciables los cambios en la forma de onda.

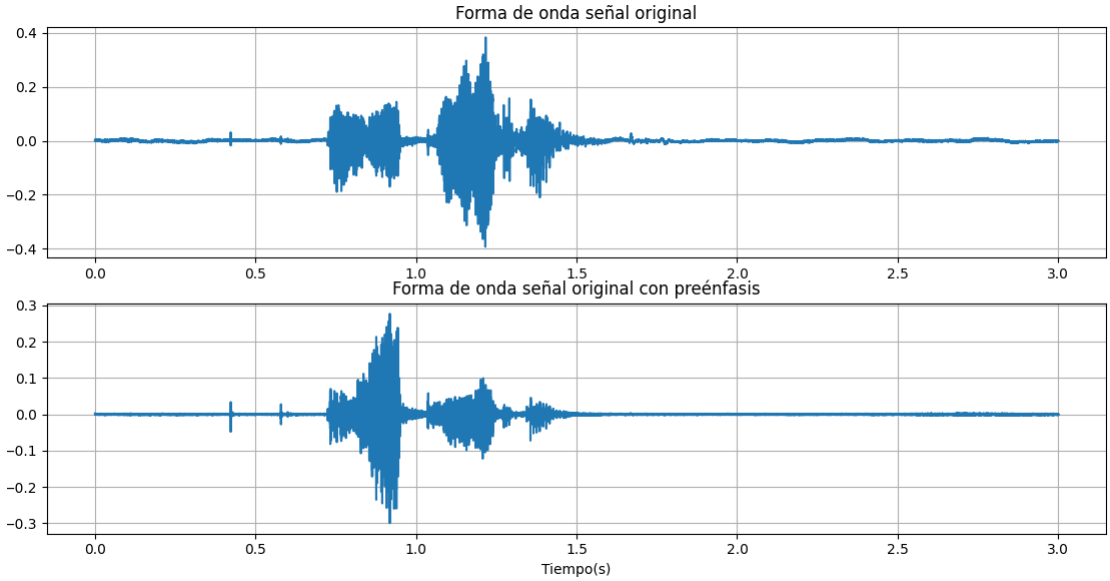

Figura 1.14. Forma de onda señal original vs señal luego de aplicarle preénfasis (elaboración propia).

1.13.2. Enmarcado

Teniendo en cuenta que las frecuencias de una señal van variando con el paso del tiempo, se puede decir que la señal de audio no es periódica, por lo tanto, es necesario realizar un proceso de segmentación en las señales de audio, asumiendo que el porcentaje de la señal contenido en cada uno de los segmentos representa que las frecuencias son estacionarias, es decir que en cortos periodos de tiempo la señal es periódica, y así, poder realizar un correcto cálculo de componentes en frecuencia mediante la Transformada de Fourier [7, 24].

Para este proceso es de suma importancia definir dos parámetros: tamaño de los marcos y el solapamiento entre ellos. Para el procesamiento de la voz, generalmente los valores típicos son una longitud de marco de 25ms y un solapamiento de 15ms.

1.13.3. Función de ventana Hamming

La aplicación de una ventana *Hamming* tiene como objetivo disminuir la fuga espectral o hacer un *alisado* de la señal, esto debido a las frecuencias espurias resultantes por las transiciones o solapamiento entre los marcos anteriormente mencionados, conocidas como discontinuidades, las cuales generan versiones distorsionadas de las señales en el análisis espectral [24, 25].

La ventana *Hamming* hace una disminución de la amplitud de dichas discontinuidades, variando suavemente los bordes a cero como se observa en la figura 1.15.

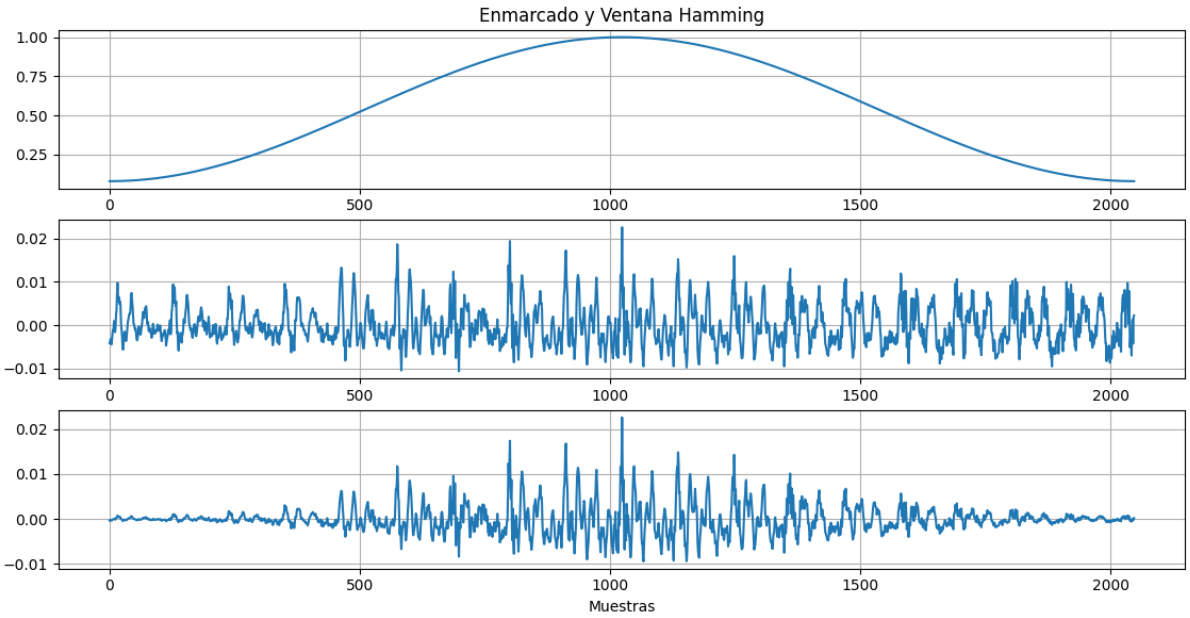

Figura 1.15. Aplicación de ventana Hamming a un marco (elaboración propia).

1.13.4. Transformada de Fourier

El siguiente paso es la aplicación de la Transformada Rápida de Fourier (FFT, *Fast Fourier Transform).* La ecuación 1.14 expresa dicha transformada de Fourier, la cual suma una lista de *N* valores sobre cada marco para calcular el espectro en frecuencia, donde x_n es la *n*-ésima muestra de la señal *x*, es decir, se suman los valores resultantes de la FFT de cada marco y así se obtiene la componente en frecuencia de la totalidad del fichero [25]

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k\frac{n}{N}}, k = 0, \dots, N-1 \quad (1.14)$$

donde:

- ***N*** es el número total de muestras
- ***n*** es la muestra actual, $n \in \{0 \dots N-1\}$
- ***k*** frecuencia actual, *k* de *0Hz* a *N – 1Hz*
- $\boldsymbol{X_k}$ cantidad de la frecuencia *k* en la señal, es un numero complejo.
- $\boldsymbol{i} = \sqrt{-1}$

1.13.5. Bancos de filtros de escala Mel

Con el objetivo de extraer bandas de frecuencia, se usan filtros triangulares en la escala de MEL (por lo general se usan 40filtros), así se emula la forma no lineal de percepción del oído humano, siendo más discriminatorio con las frecuencias bajas y menos con las altas. Es decir, para las frecuencias bajas, los filtros abarcan menos cantidad de frecuencias, pero con una mayor amplitud; y en las frecuencias altas, se tiene filtros con mayor rango de frecuencias, pero

con una amplitud más baja, dado que se guarda relación en cuanto a la intensidad de percepción del oído [7, 24]. En la figura 1.16 se puede observar lo dicho anteriormente.

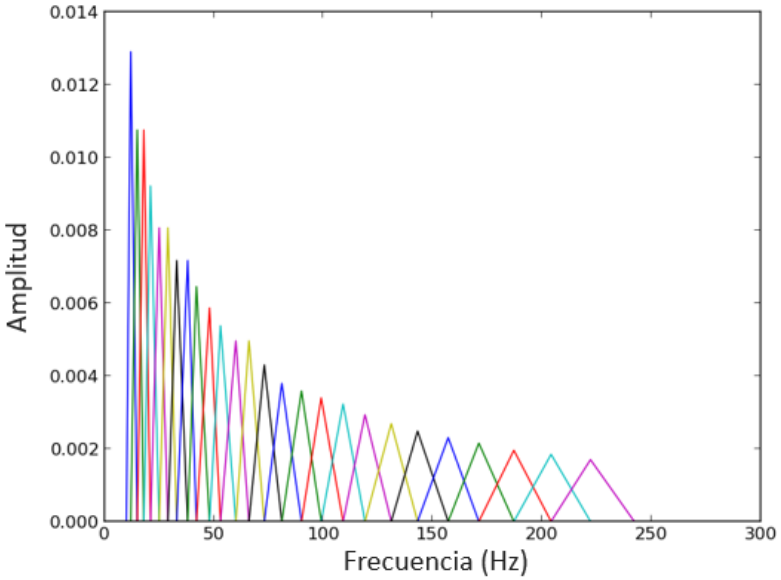

Figura 1.16. Banco de filtros en escala de MEL. Tomado de [24].

1.13.6. Coeficientes Cepstrales en Frecuencias de Mel

Mediante los MFCC se identifican los componentes de la señal de audio que son adecuados para determinar el contenido lingüístico y descartar todo lo demás. Estos coeficientes se hallan a partir del banco de filtros, al aplicarles la Transformada Discreta del Coseno (DCT, *Discrete cosine transform*) como se expresa matemáticamente en la ecuación 1.15, la cual es una transformada basada en la Transformada Discreta de Fourier, pero la DCT solo trabaja con cosenos mientras que la DFT lo hace con exponenciales complejos [34]

$$X_k = \sum_{n=1}^{N} x_n \cos\left(\frac{\pi}{N} k \left(n - \frac{1}{2}\right)\right), \quad (1.15)$$

donde:

- $\boldsymbol{X_k}$ representa el vector de coeficientes por cada ventana
- $\boldsymbol{n}$ representa el número de filtros en el banco de filtros MEL
- $\boldsymbol{N}$ representa el número total de coeficientes MFCC
- $\boldsymbol{k}$ representa la ventana actual sobre la que se está trabajando

Una vez se tiene la señal en la escala de Mel se aplica la DCT para volver los coeficientes obtenidos del banco de filtros a el dominio del tiempo y así obtener los coeficientes cepstrales, esto se logra al comprimir los bancos y eliminar la correlación que hay entre ellos. Se toman únicamente los MFCCs del 2 al 13, descartando el resto ya que solo representan cambios rápidos en los coeficientes del banco de filtros y estos datos no aportan nada para el

reconocimiento de voz [24,25]. En la figura 1.17 se observan los MFCCs obtenidos en una señal ejemplo.

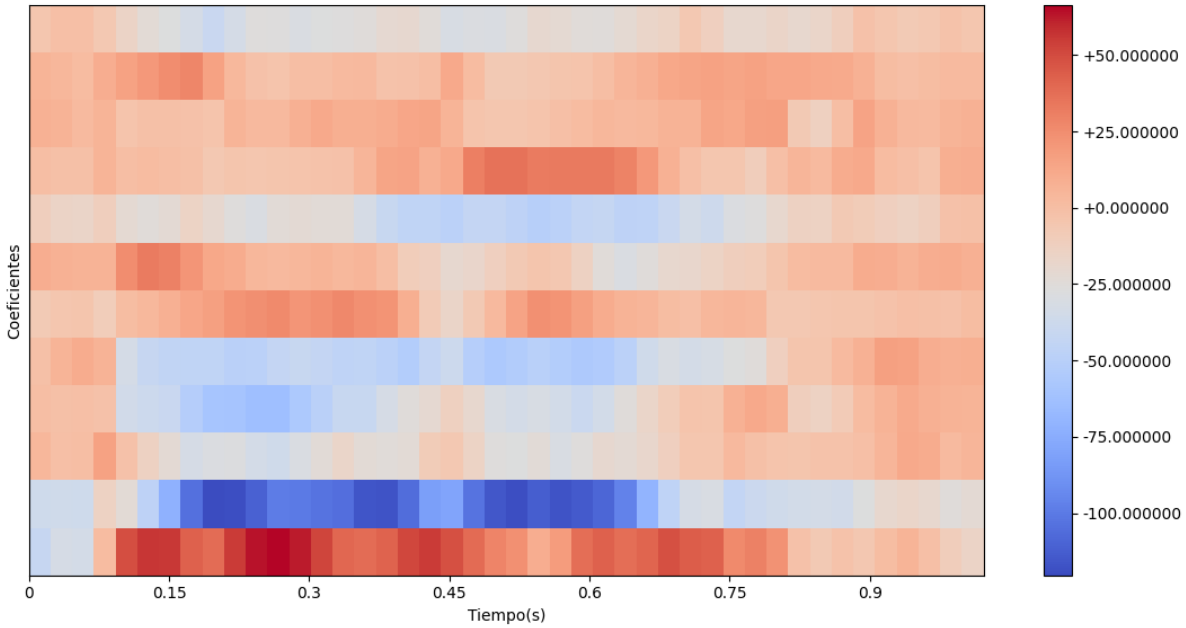

Figura 1.17. Representación de los MFCCs obtenidos de la señal ejemplo (elaboración propia).

Capítulo 2

HERRAMIENTAS

2.1. LENGUAJE DE PROGRAMACIÓN

El lenguaje de programación más utilizado para el diseño e implementación de prototipos basados en AI, ML y DL en particular es Python, según la documentación encontrada durante la investigación; por lo tanto, al ser Python un lenguaje de sintaxis sencilla, qué facilita el desarrollo y la evaluación de modelos de DL, se optó por este lenguaje de programación para cumplir los objetivos del proyecto.

Todas las librerías relacionadas, ya sea para la manipulación de datos, ML o DL, son compatibles con este lenguaje, o directamente han sido desarrolladas en este mismo. Cabe destacar que para el diseño del sistema se optó por la versión de Python 3, ya que esta es la versión qué más soporte tiene en la actualidad.

Para el manejo de este lenguaje de programación, haciendo referencia al entorno de desarrollo, la elección de este no tiene ninguna influencia sobre los resultados a obtener. Con el entorno de desarrollo se busca poder trabajar de forma sencilla y eficiente. A partir de estas características se tomó la decisión de trabajar con Visual Studio Code (VSC), siendo este compatible con Python e incluyendo varias características para el manejo y creación de algoritmos.

2.2. LIBRERÍAS DE MANIPULACIÓN DE DATOS

Estas librerías son utilizadas para la creación de la base de datos, el pre-procesado del mismo y la obtención de las características de los ficheros de audio. Estas librerías son de gran utilidad al momento de operar sobre los datos de una manera más sencilla y óptima.

2.2.1. NumPy

La librería *NumPy* se considera como el paquete fundamental para la computación científica en Python. En el núcleo de *NumPy* se encuentra el objeto *ndarray*, el cual, proporciona un objeto de matriz multidimensional y una variedad de módulos para procesos rápidos en los arreglos de matrices [27].

2.2.2. Librosa

La librería *Librosa* es un paquete de Python dedicado al análisis de audio y música, el cual provee una gran variedad de módulos para el procesado del audio. Esta librería resultó muy útil para las operaciones necesarias en cuanto al pre-procesado y la extracción de las características de los ficheros de audio [28].

2.2.3. CSV

Los *CSV* (Valores separados por comas), son el formato más común de importación y exportación de hojas de cálculo y bases de datos. El módulo csv en Python implementa clases para leer y escribir datos en formato *CSV* [29]. Con este módulo se logró almacenar las características y etiquetas de los ficheros en un archivo .csv, para luego ser utilizado en el entrenamiento de la NN.

2.3. LIBRERÍAS DE DEEP LEARNING

2.3.1. TensorFlow

TensorFlow es un *framework* de código abierto desarrollado por Google para desarrollar y entrenar modelos de ML y especialmente de DL. Se basa en el concepto de grafos de computación, donde los datos en forma de tensores fluyen entre distintas operaciones, este destaca porque permite un rápido entrenamiento de modelos de DL con bases de datos de gran tamaño [5].

2.3.2. SciKit-Learn

SciKit-Learn es una librería de Python para hacer análisis predictivo, qué incluye clasificadores, regresión y *clustering*, qué partiendo de las funcionalidades de otras librerías como *NumPy, SciPy y Matplotlib* permite ejecutarlos de manera sencilla [19].

2.3.3. Keras

Keras es una API desarrollada en Python, qué se ejecuta sobre la plataforma de ML *TensorFlow*, la cual permite crear NN de manera sencilla, a partir de abstracciones y bloques de construcción esenciales para el desarrollo de las NN. Los datos se preparan en tensores y sus modelos se basan en capas, con una capa para la entrada y otra para la salida y un número indeterminado de capas ocultas [26].

2.4. HARDWARE

El hardware, al contrario del entorno de desarrollo, si influye en los resultados a obtener debido a que los algoritmos de DL requieren de una potencia de cómputo muy grande para el procesamiento de los datos y las operaciones a realizar y entre más amplia sea la base de datos, en cuestión de datos, es recomendable recurrir a la aceleración por Unidad de Procesamiento Gráfico (GPU, *Graphics Processing Unit*), esto para la fase de entrenamiento de la NN, debido qué el entrenamiento por Unidad de Procesamiento Central (CPU, *Central Processing Unit*) puede resultar demasiado lento.

Este trabajo de grado se diseñó e implementó en un equipo con procesador *Intel Core i5-8250U* de *4 núcleos* a una frecuencia de *1.6GHz* y *8GB* de memoria *RAM* y no se hizo uso de GPU.

Capítulo 3

DISEÑO

Un sistema de reconocimiento de vocablos tiene como entrada los MFCCs de los ficheros de audio, los cuales tienen una duración acotada, una mínima cantidad de ruido y cada uno contiene únicamente la pronunciación del vocablo a reconocer, esto con el fin de que al hacer una implementación futura del sistema, éste funcione de tal manera que permita activar funciones específicas que en otro entorno se harían mediante el uso de teclados o botones, reemplazando esta acción por el uso de la voz humana.

Para el diseño de este sistema se optó por dividirlo en fases y abordar una a una de la forma más eficiente. Estas fases son las siguientes y se exponen en los siguientes sub-apartados.

- Construcción de la base de datos
- Pre-procesado y extracción de los MFCCs
- División de la base de datos
- Arquitectura de la red neuronal
- Entrenamiento de la red neuronal
- Estructura del proyecto

3.1. BASE DE DATOS

En el diseño de un modelo de ML o DL es fundamental construir o elegir correctamente la base de datos a utilizar, ya que ésta influye directamente en los resultados obtenidos. Para este proyecto se tomó la decisión de construir la base de datos desde cero, es decir, realizar una base de datos propia.

Para la construcción de la base de datos se realizaron grabaciones de la pronunciación de los vocablos a trabajar con la ayuda de varias personas (tanto hombres como mujeres).

Inmediatamente grabados se hace la eliminación del silencio inicial, se guardan y organizan en carpetas etiquetadas con el respectivo nombre de cada vocablo; por tanto, el nombre de la carpeta en el que está contenido el fichero de audio que se está procesando, en un determinado momento es utilizado como la etiqueta del mismo. El utilizar una etiqueta por cada fichero de audio es necesario debido a que se trabaja con aprendizaje supervisado.

La base de datos contiene 7 vocablos diferentes, los cuales son los comandos a ser reconocidos por la red neuronal:

- Abrir
- Adelante
- Atrás
- Cerrar
- Derecha
- Izquierda
- Parar

La base de datos cuenta con un total de 2800 ficheros de audio, todos en formato WAVE (.wav). Se creó un código, con el cual se grabó y se eliminó el silencio inmediatamente el locutor realizo la grabación el vocablo. Para la creación de la base de datos se optó por una duración de grabación de 3 segundos, con el fin de que en el momento de realizar la grabación el locutor cuente con un tiempo suficiente para pronunciar cualquiera de los vocablos, los ficheros de audio se grabaron a una frecuencia de 44100 Hz y a dos canales, se tomó esta frecuencia con el fin de tener una mejor calidad en la grabación del audio y se optó por dos canales debido a que las grabaciones se realizan tanto con el código como en dispositivos celulares y estos últimos realizan las grabaciones a dos canales.

En las figuras 3.1 a la 3.7 se observa la forma de onda en el tiempo de una grabación por cada uno de los vocablos a trabajar, contenidas en el lapso de 3 segundos, evidenciando la diferencia de fuerza de pronunciación de cada sílaba que compone al vocablo.

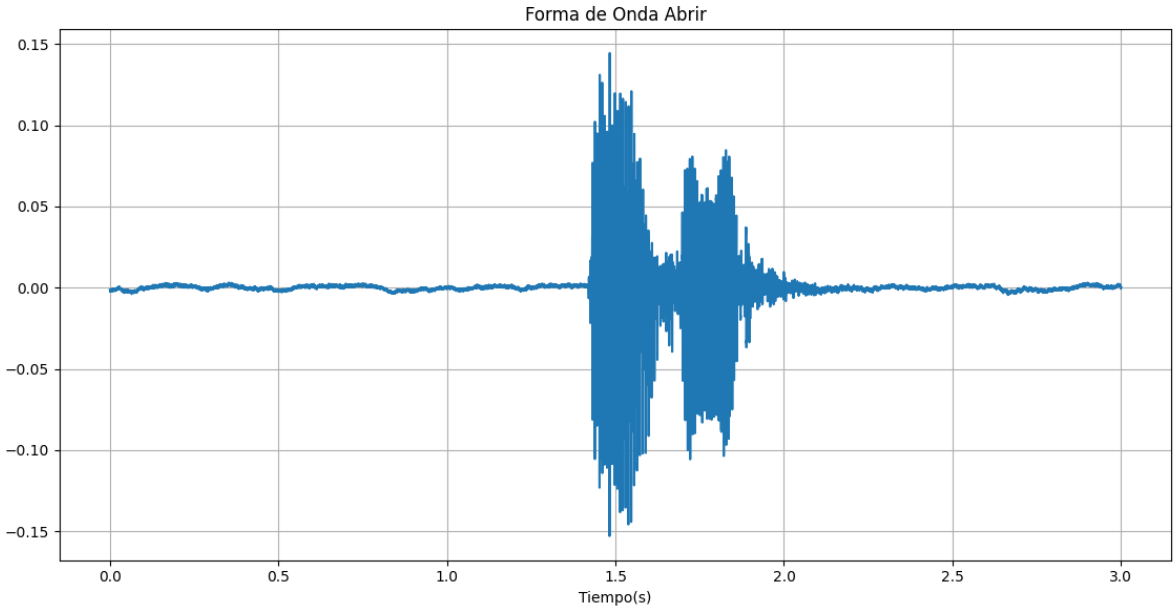

Figura 3.1. Forma de onda del vocablo abrir (elaboración propia).

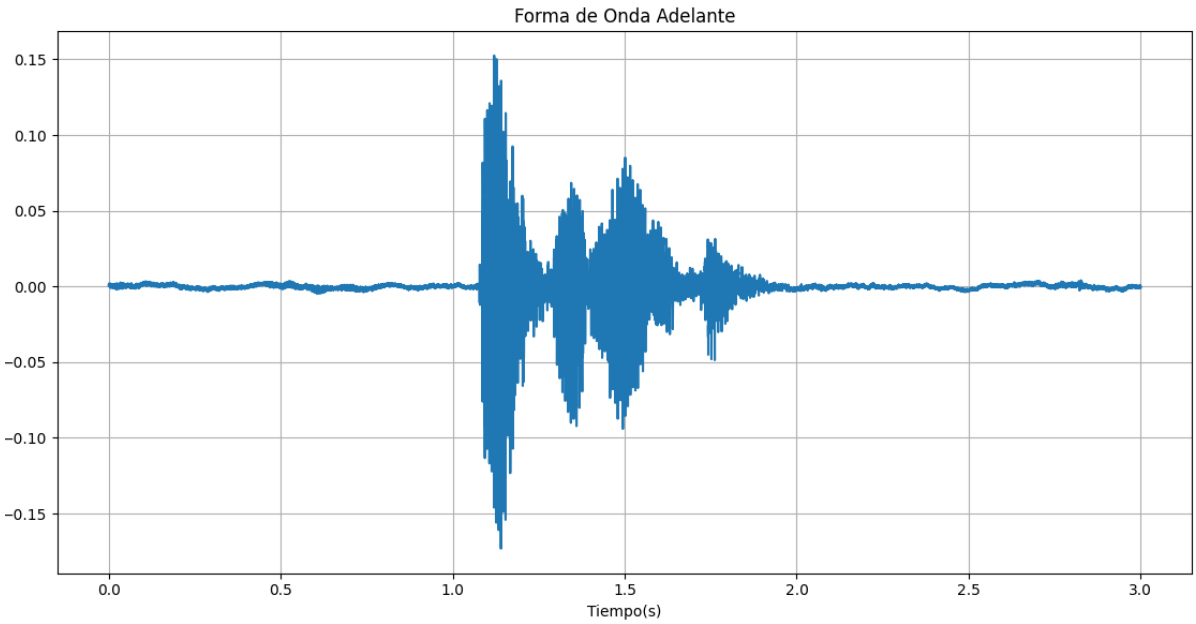

Figura 3.2. Forma de onda del vocablo adelante (elaboración propia).

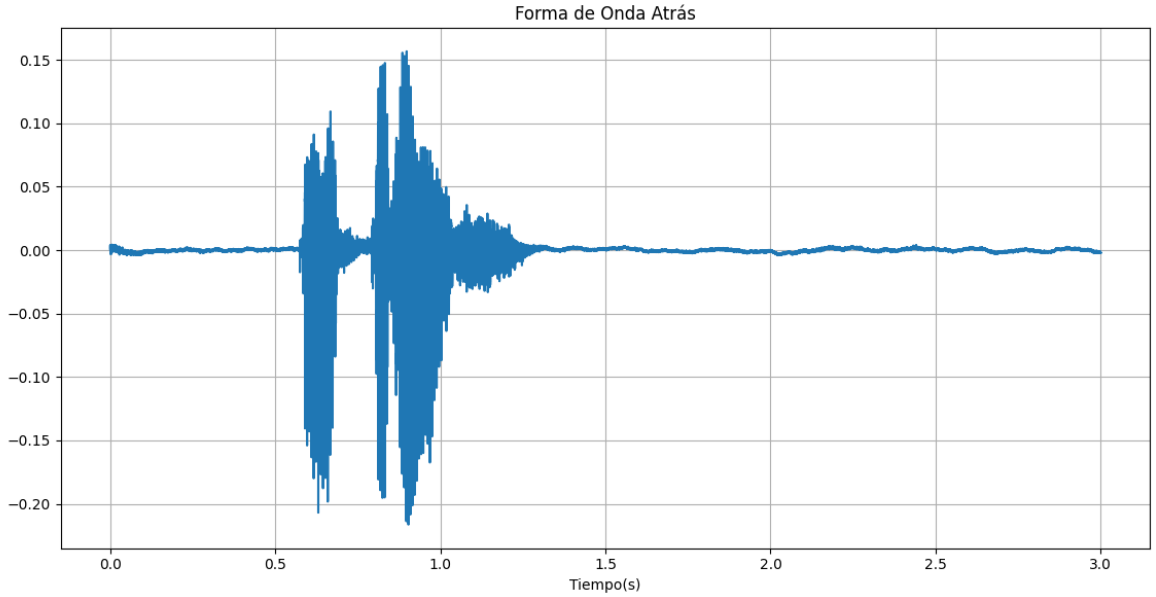

Figura 3.3. Forma de onda del vocablo atrás (elaboración propia).

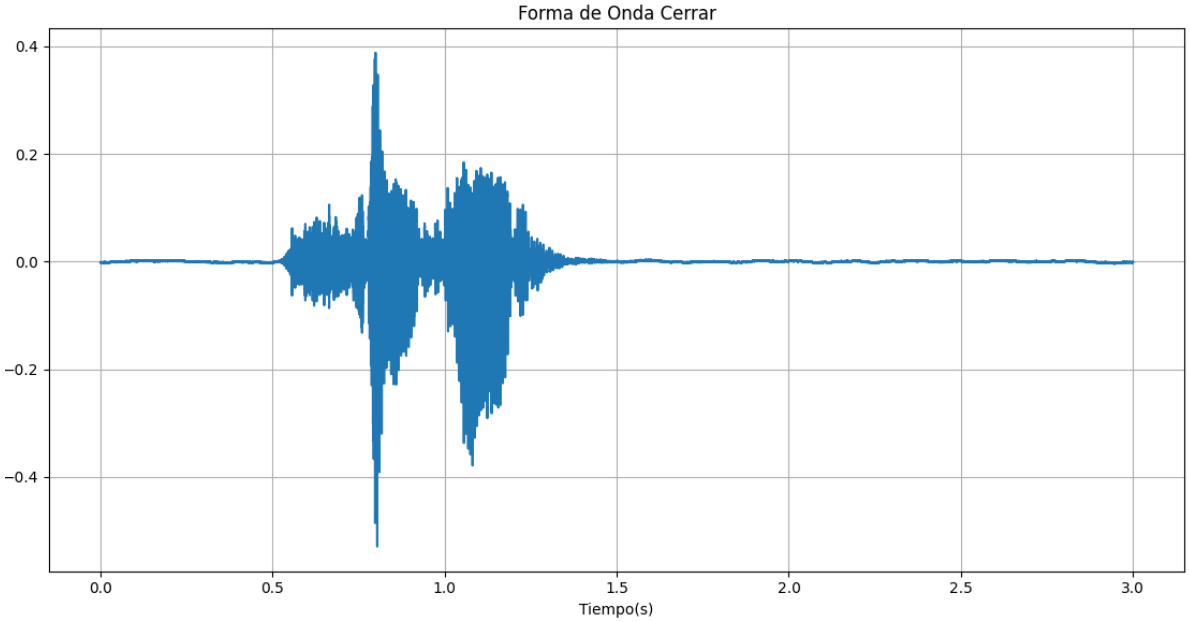

Figura 3.4. Forma de onda del vocablo cerrar (elaboración propia).

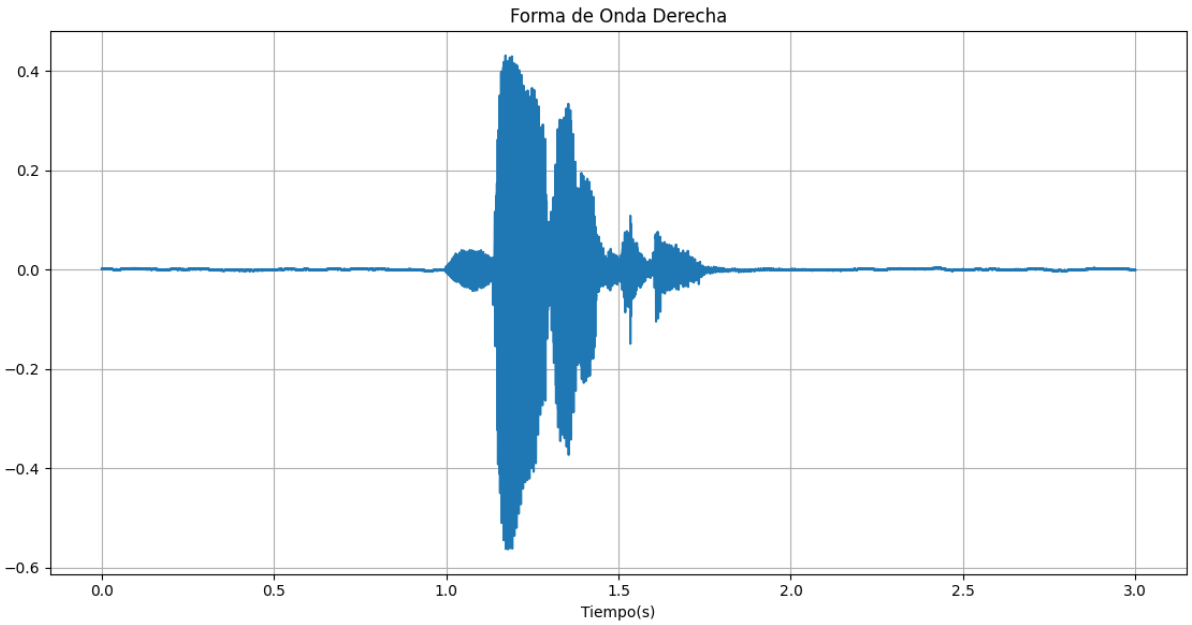

Figura 3.5. Forma de onda del vocablo derecha (elaboración propia).

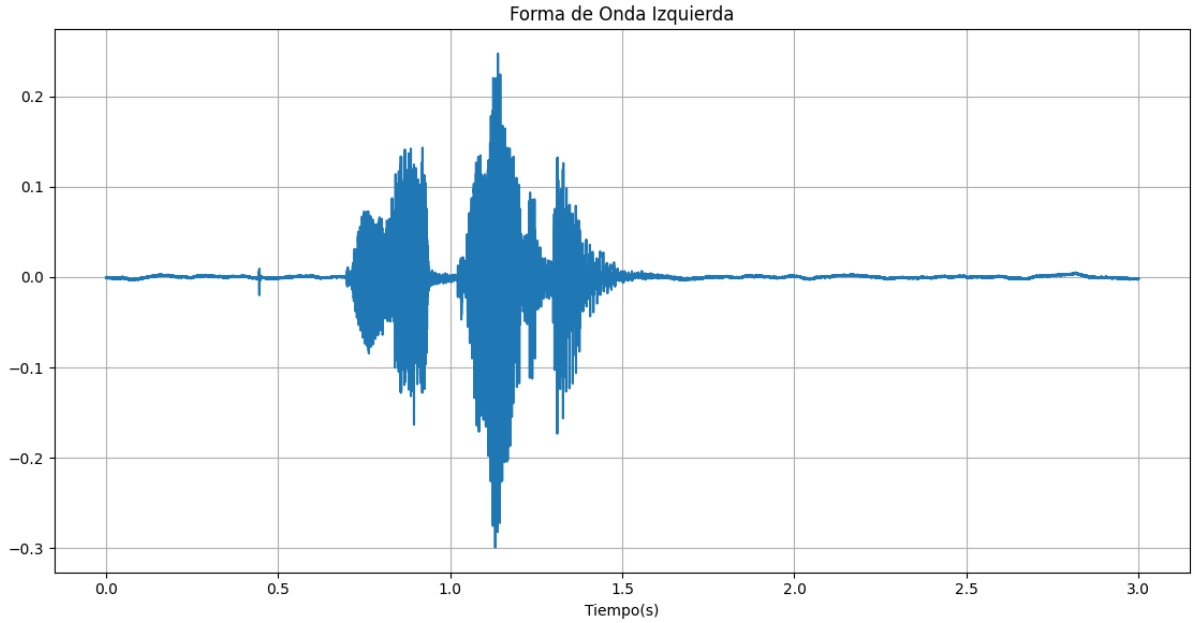

Figura 3.6. Forma de onda del vocablo izquierda (elaboración propia).

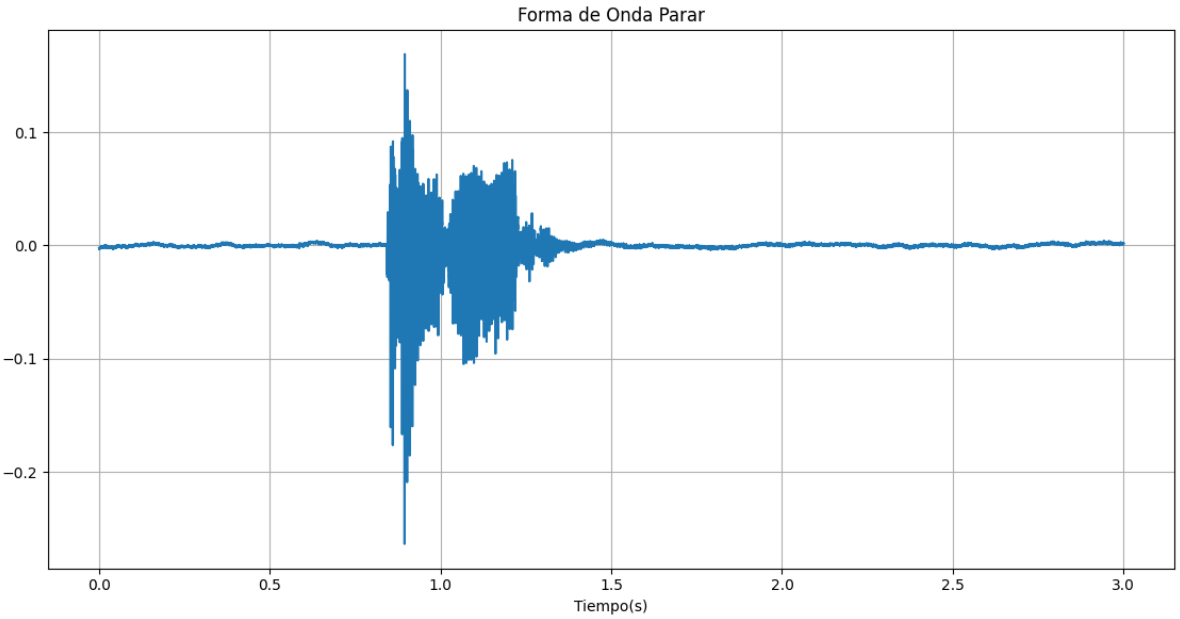

Figura 3.7. Forma de onda del vocablo parar (elaboración propia).

- ***Detección de silencio:***

Se definió la función *detect_silence()* la cual recibe como parámetros de entrada: *signal*, que representa a la señal que se grabó previamente, *threshold*, que define el umbral de intensidad de señal para el corte de la grabación, *chunk_size*, define una ventana de barrido, es decir, en ventanas de 15ms se recorren las muestras de la señal hasta superar dicho umbral.

La función *while* recorre el audio iniciando en la muestra número cero y en marcos de *chunk_size*, así hasta encontrar la primera muestra que supere el *threshold*, a su vez se va aumentando un contador definido como *trim*, donde este será el retorno de la función *detect_silence()*, el cual representa el tiempo en *ms* a recortar de la grabación. (ver Apéndice B).

- ***Eliminación de silencio:***

La eliminación del silencio se realiza con el fin de eliminar los tiempos de la señal que no contienen nada de información, esto para que al realizar la extracción de las características se obtengan simplemente las características respecto a la pronunciación del vocablo y evitar extraer características que no aportarán nada para el sistema. Al momento de eliminar el silencio inicial, se optó por dejar un margen o "cola" de las primeras muestras de audio (100 muestras), esto con el fin de prevenir cortes en la pronunciación de la primera sílaba de cada palabra (no tienen la misma fuerza de acentuación o realce, posiblemente no superando el *threshold*), dando seguridad de que se toma la totalidad del vocablo.

La eliminación de silencio realizada en código, donde se recibe el valor retornado de la función *detect_silence()*, es decir, el tiempo a eliminar de la grabación, como se mencionó se deja un margen de 100 muestras extra del silencio y finalmente se exporta el archivo con el silencio

eliminado *trimmed_sound*, dónde *file* representa la ruta donde se va a guardar. (ver Apéndice B).

En las figuras 3.8 a la 3.14 se observa la forma de onda en el tiempo de una grabación por cada uno de los vocablos a trabajar con la respectiva cola que se mencionó anteriormente de 100 muestras. Se observa que en algunos vocablos la cola es más corta que en otros, esto es debido a la fuerza de acentuación que se tiene en la primera sílaba de cada vocablo.

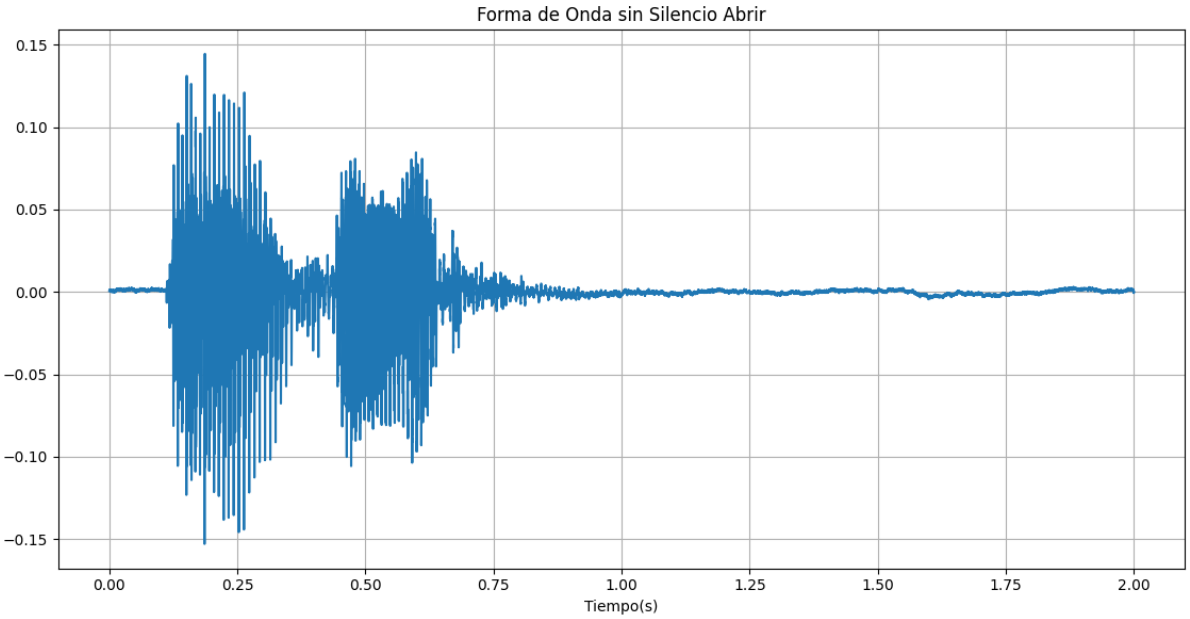

Figura 3.8. Forma de onda del vocablo abrir sin el silencio inicial (elaboración propia).

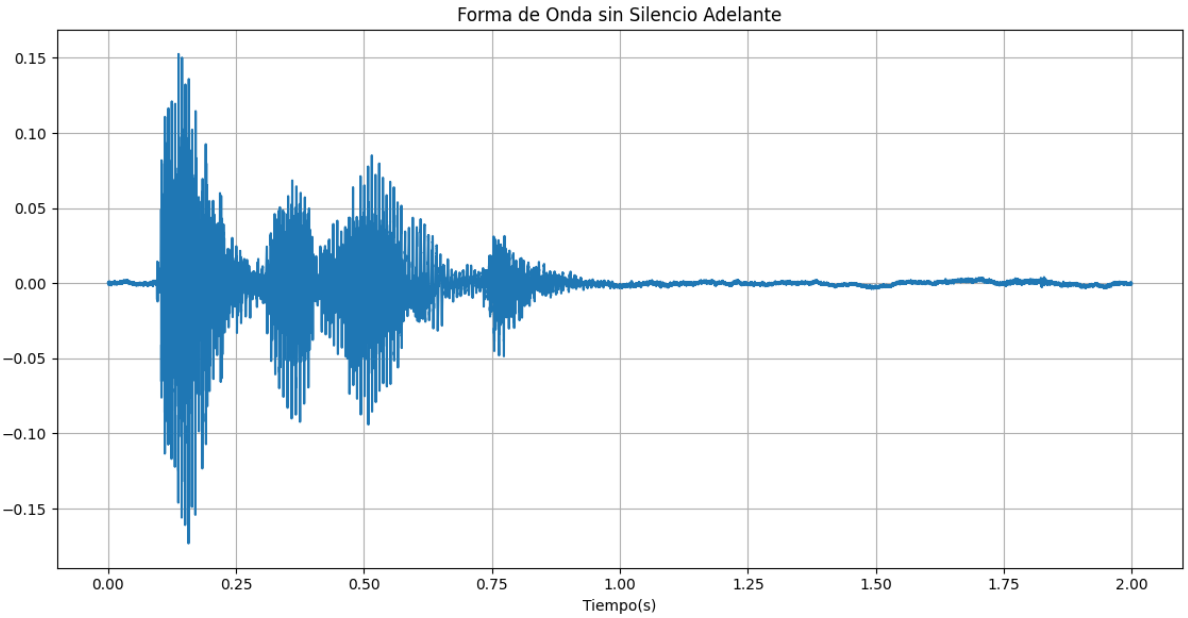

Figura 3.9. Forma de onda del vocablo adelante sin el silencio inicial (elaboración propia).

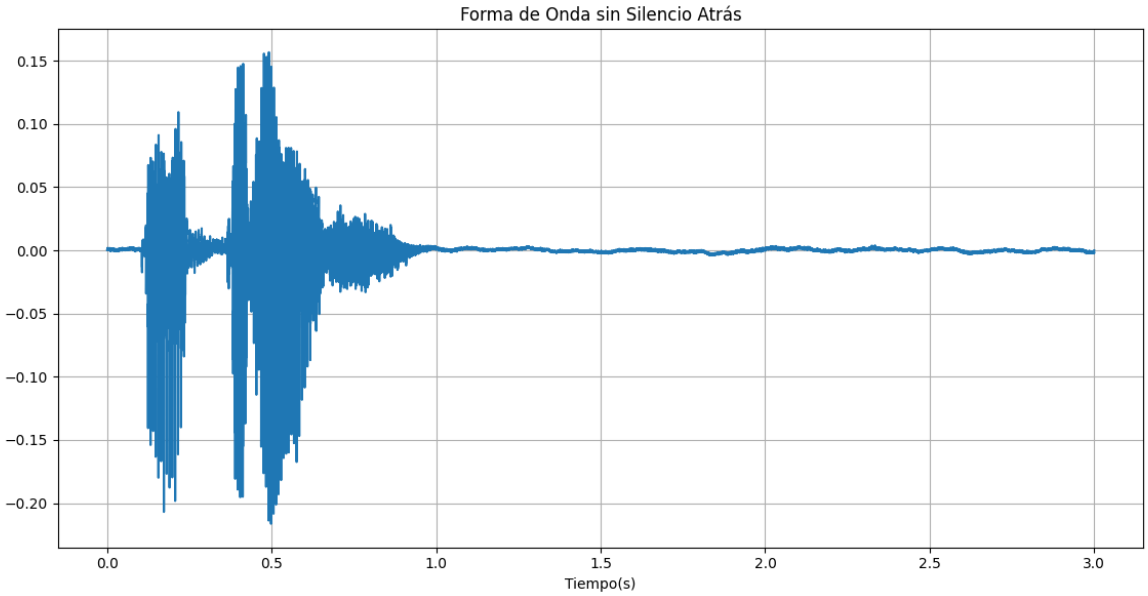

Figura 3.10. Forma de onda del vocablo atrás sin el silencio inicial (elaboración propia).

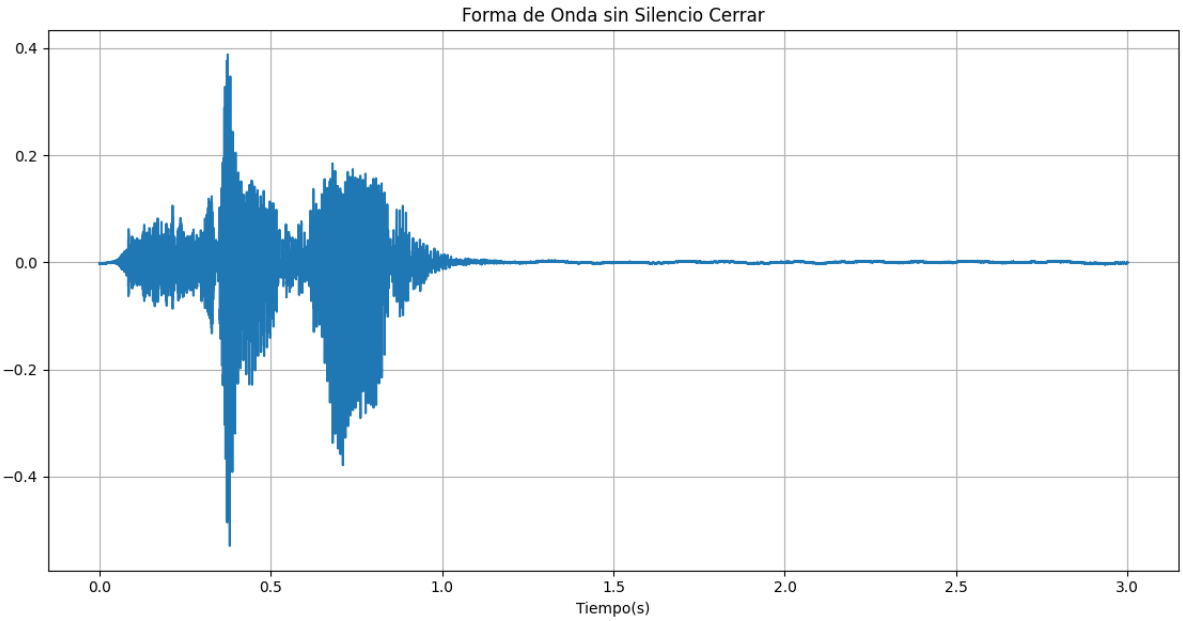

Figura 3.11. Forma de onda del vocablo cerrar sin el silencio inicial (elaboración propia).

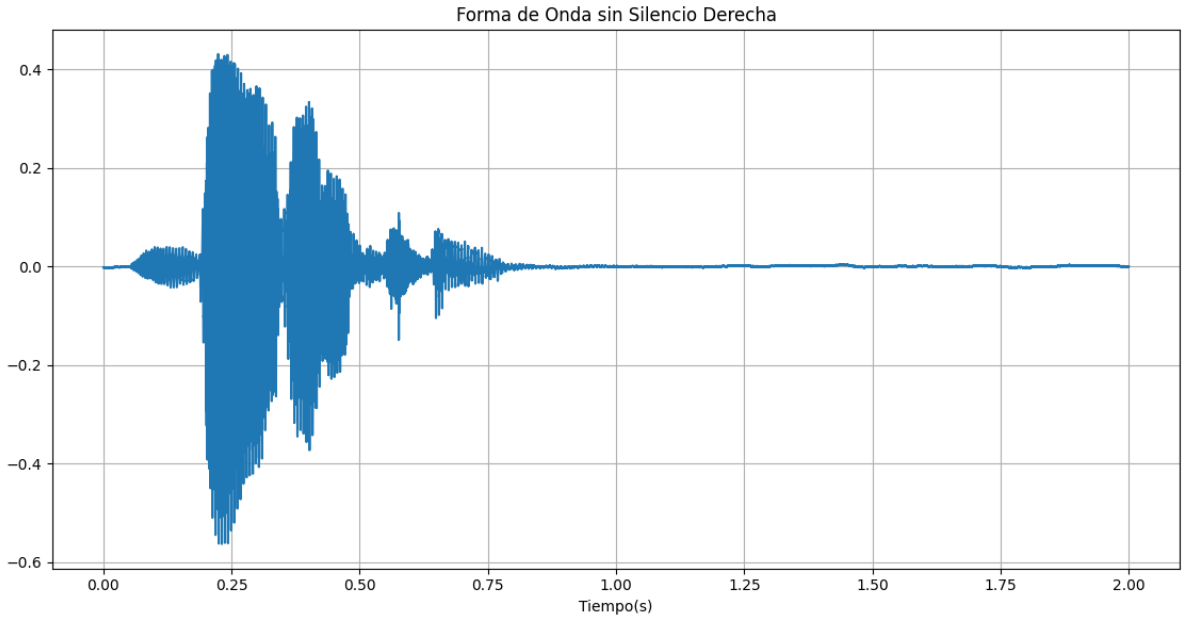

Figura 3.12. Forma de onda del vocablo derecha sin el silencio inicial (elaboración propia).

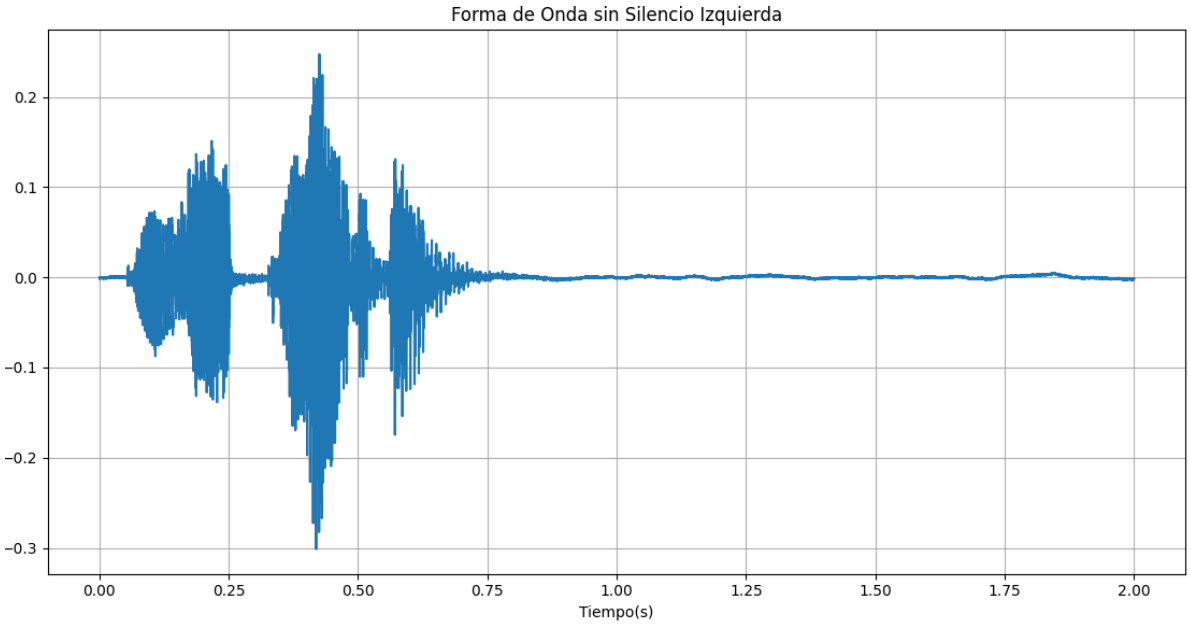

Figura 3.13. Forma de onda del vocablo izquierda sin el silencio inicial (elaboración propia).

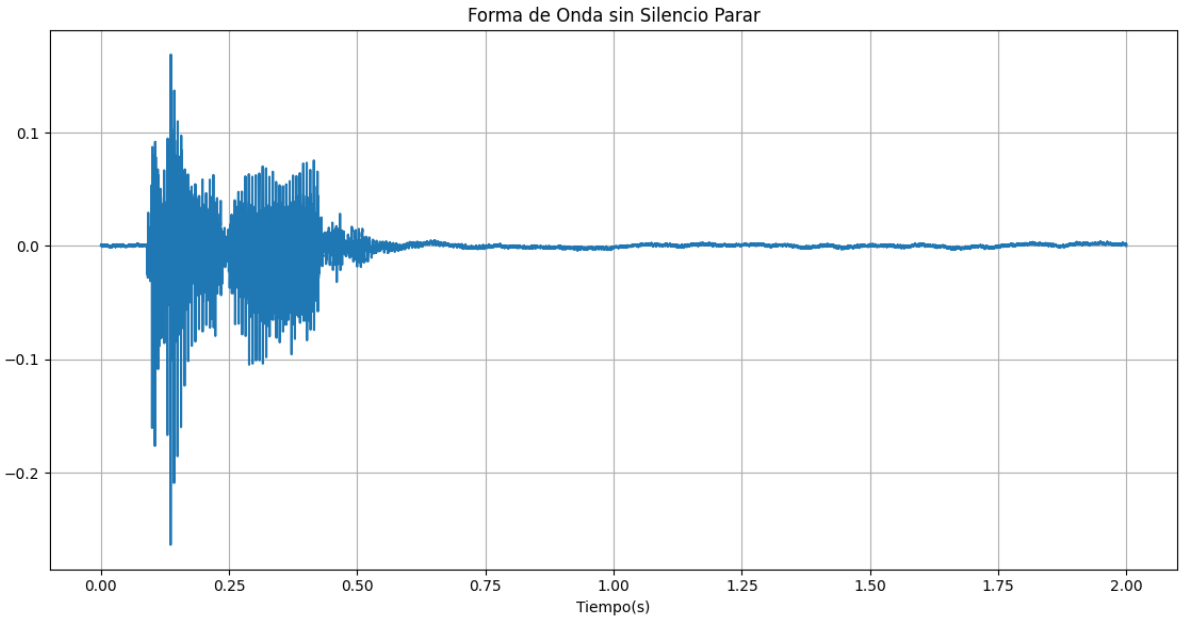

Figura 3.14. Forma de onda del vocablo parar sin el silencio inicial (elaboración propia).

3.2. PRE-PROCESADO Y EXTRACCIÓN DE LOS MFCCs

Una vez lista la base de datos, se tomaron los ficheros de audio, se procesaron y se extrajeron las características MFCCs que componen a cada fichero de audio. Para ello, se realizaron las siguientes operaciones:

1. Preénfasis, con un coeficiente de filtro (α) de 0,97.

   El filtro de preénfasis como se explicó en el *Capítulo 1* es para amplificar las frecuencias altas y para compensar la atenuación que se produce en la señal al pasar a través del tracto vocal.

   Para la implementación del filtro de preénfasis se empleó la función *roll* de la librería *numpy* para representar la señal desplazada en una muestra $x(n-1)$ de la ecuación 1.13. Esta función hace un corrimiento en las muestras de la señal de una posición, es decir, pasando el último valor a la posición cero, y el valor en la posición cero, estará en la posición uno y así sucesivamente.

   A continuación, en las figuras 3.15 a la 3.21, se muestra una comparación entre el espectro en frecuencia de la señal original y el espectro de la señal al aplicársele un filtro de preénfasis, donde se puede evidenciar cómo las frecuencias de la señal se acentúan realzando las altas frecuencias y directamente aumentando la amplitud de ruido en dichas frecuencias.

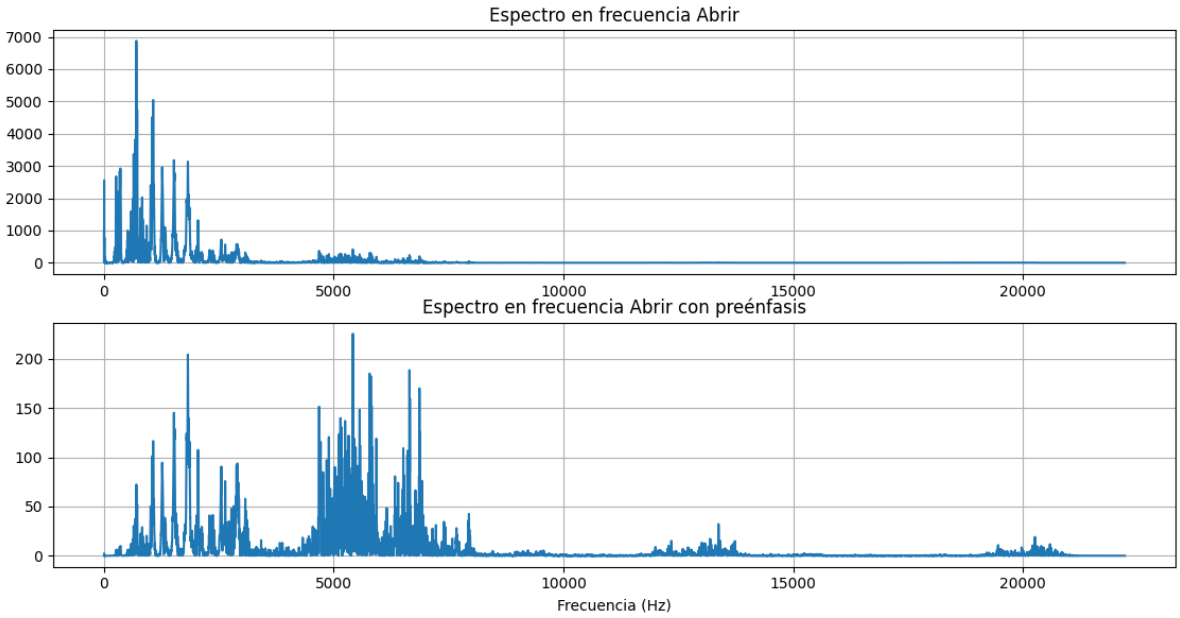

Figura 3.15. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo abrir (elaboración propia).

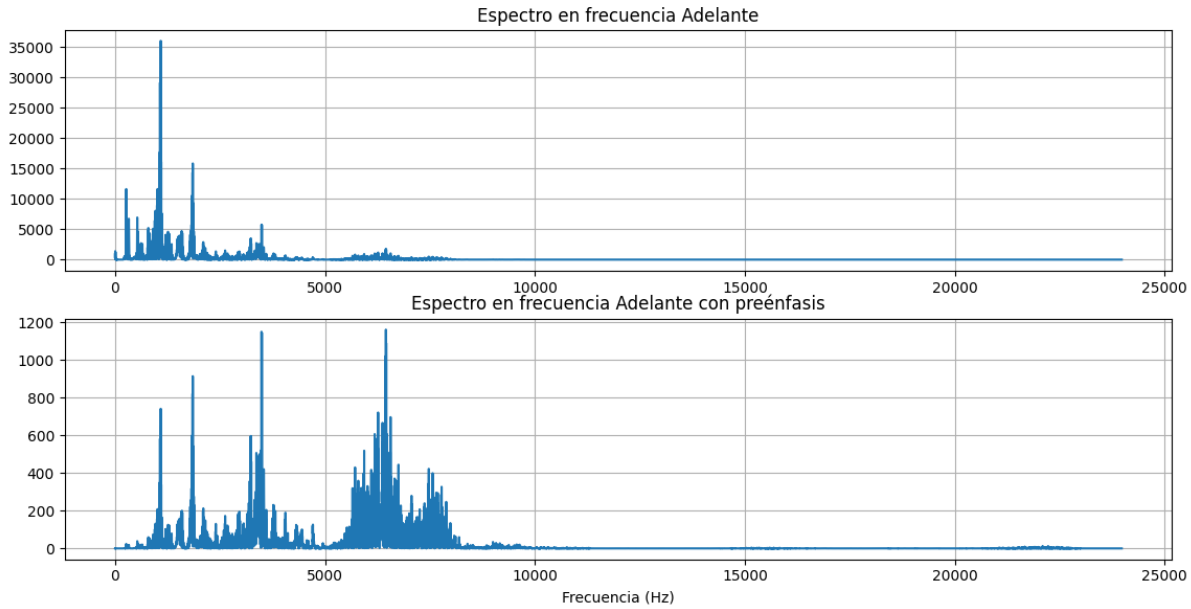

Figura 3.16. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo adelante (elaboración propia).

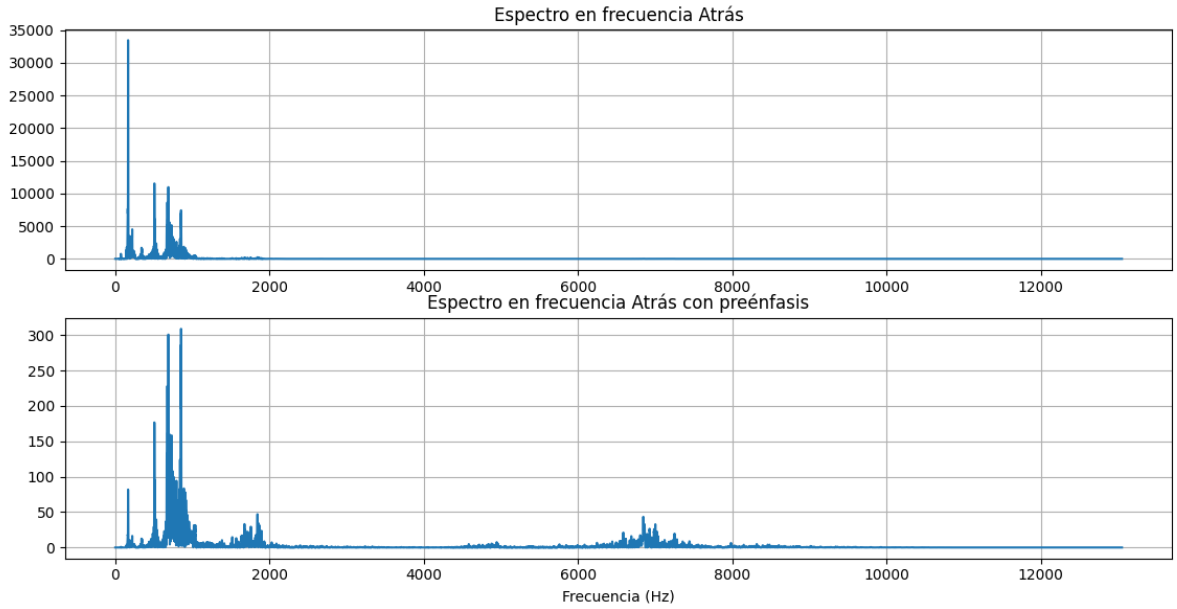

Figura 3.17. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo atrás (elaboración propia).

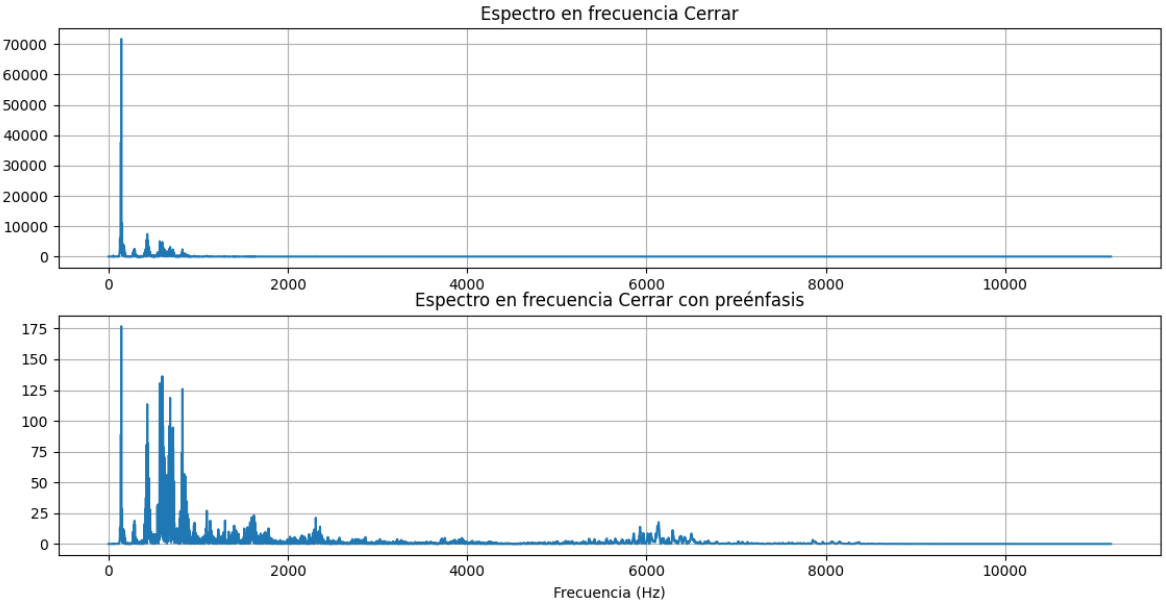

Figura 3.18. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo cerrar (elaboración propia).

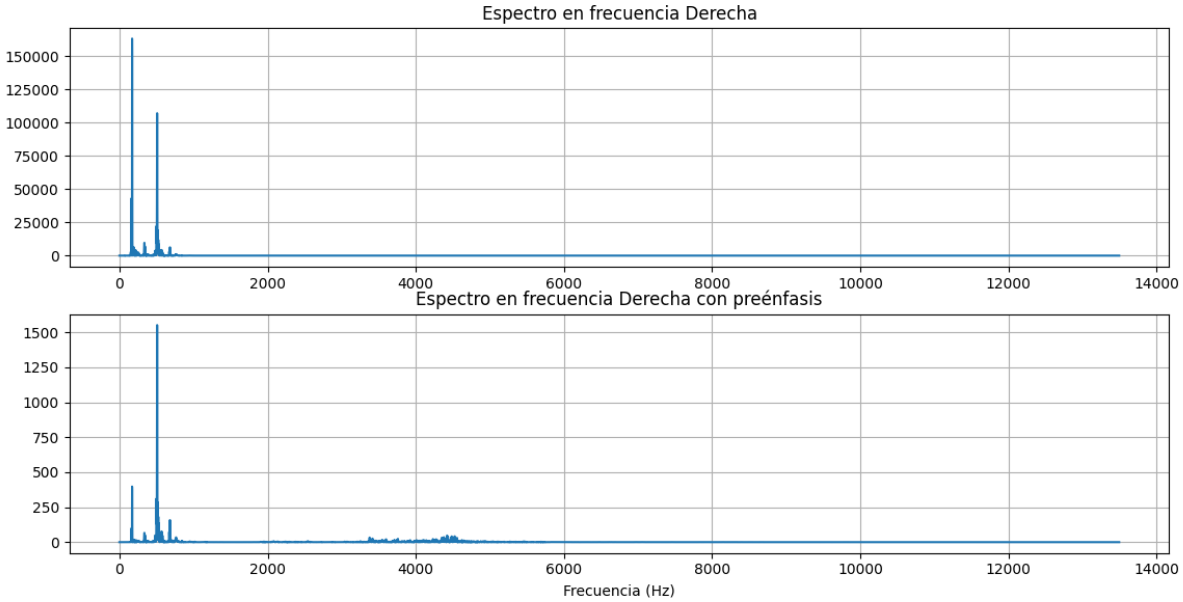

Figura 3.19. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo derecha (elaboración propia).

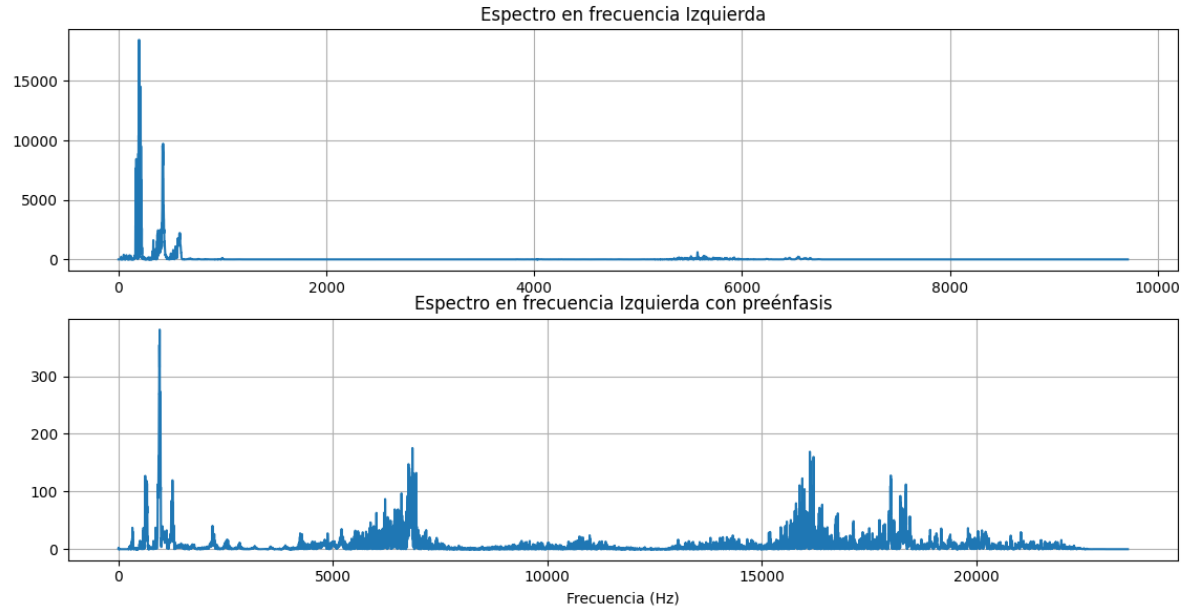

Figura 3.20. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo izquierda (elaboración propia).

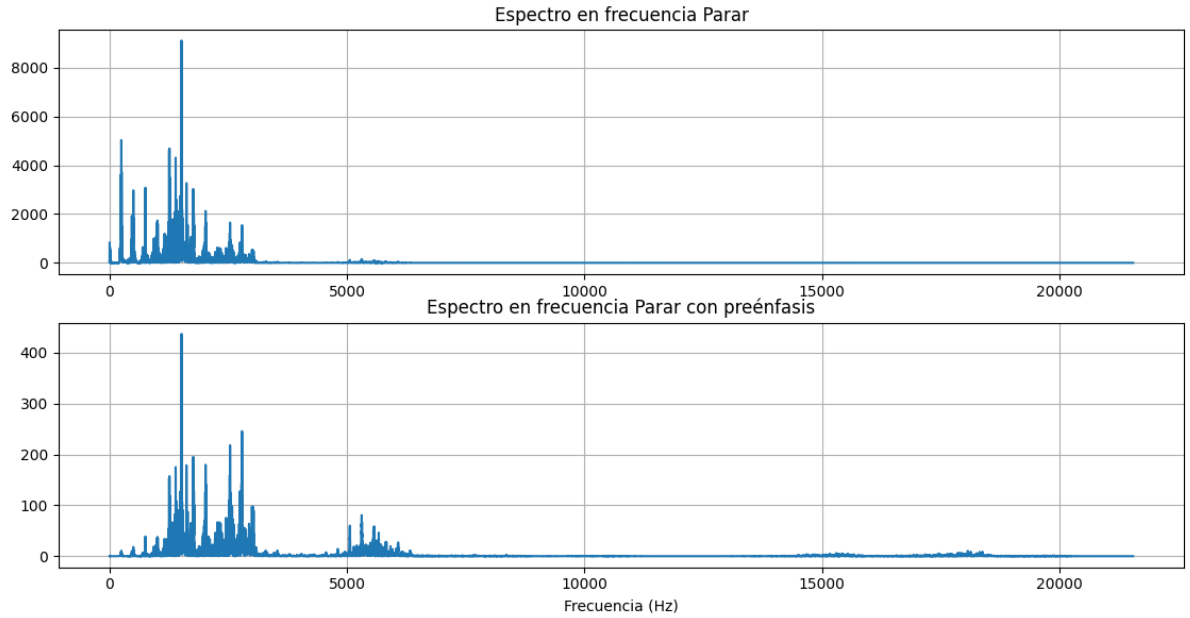

Figura 3.21. Espectro en frecuencia señal original vs espectro de señal con preénfasis del vocablo parar (elaboración propia).

2. Enmarcado, con un tamaño de marco de 2048 muestras y un solapamiento de 512 muestras.
3. Aplicación de la FFT obteniendo el espectro de potencia.
4. Cálculo y aplicación del banco de filtros en escala Mel, obtenidos mediante filtros triangulares. El número de filtros que se utilizaron de 40.
5. Cálculo de los MFCCs mediante la aplicación de la DCT, se tomaron solo los valores del 2 a 13, debido a que el primer coeficiente representa la componente en DC de la señal y el resto de coeficientes a partir del 13 solo representan cambios rápidos en los coeficientes del banco de filtros.

Para la implementación de las operaciones anteriores se tuvo la facilidad en Python de usar la librería *Librosa,* la cual permite obtener los MFCCs utilizando la función *feature.mfcc.* (ver Apéndice B).

La función *librosa.feature.mfcc(),* aplica la FFT, utiliza 40 filtros para el banco de filtros de Mel y para obtener los MFCCs aplica la DCT. Esta función recibe como entrada para el enmarcado de la señal y obtención de los coeficientes los siguientes parámetros:

- *signal*, es el fichero de audio al cual se le extraen los coeficientes,
- *sr*, es la tasa de muestreo de la señal, que es 22050 Hz, siendo este valor el que utiliza por defecto la librería *Librosa* al abrir un archivo de audio, y también representa un segundo de audio del fichero a tratar.
- *n_mfcc*, representa el número de coeficientes a ser extraídos (13 coeficientes).
- *n_fft*, representa el tamaño del marco en el cual será aplicada la FFT, medido en muestras, para este caso se toman 2048 muestras para cada marco.

- *hop_length*, es el tamaño de solapamiento entre los marcos, medido en muestras, para este caso 512 muestras.

En las figuras 3.22 a la 3.28, se puede observar la variación de los 12 coeficientes cepstrales en el tiempo a lo largo de la escala de Mel. Se observa también en el eje *x* el tiempo de 1 segundo por cada vocablo y este eje también representa la cantidad de segmentos para 1 segundo, que en este caso son 44 y en el eje *y* se tienen los MFCCs que en este caso son 12, formando así el vector de características de cada audio de (44 × 12).

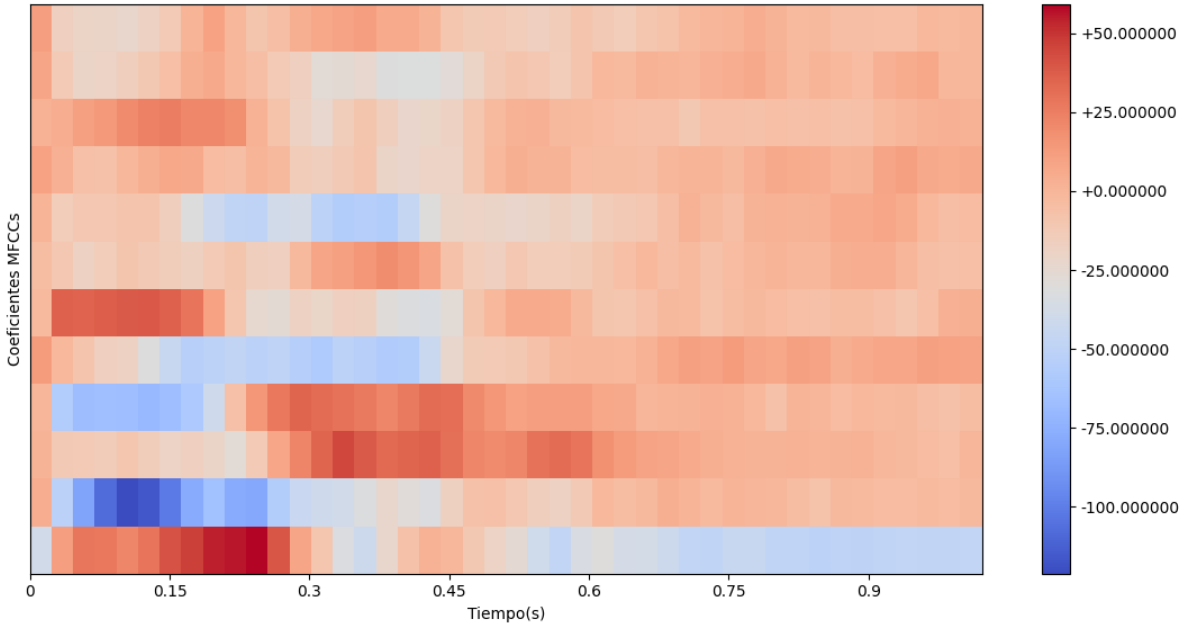

Figura 3.22. MFCCs del vocablo abrir (elaboración propia).

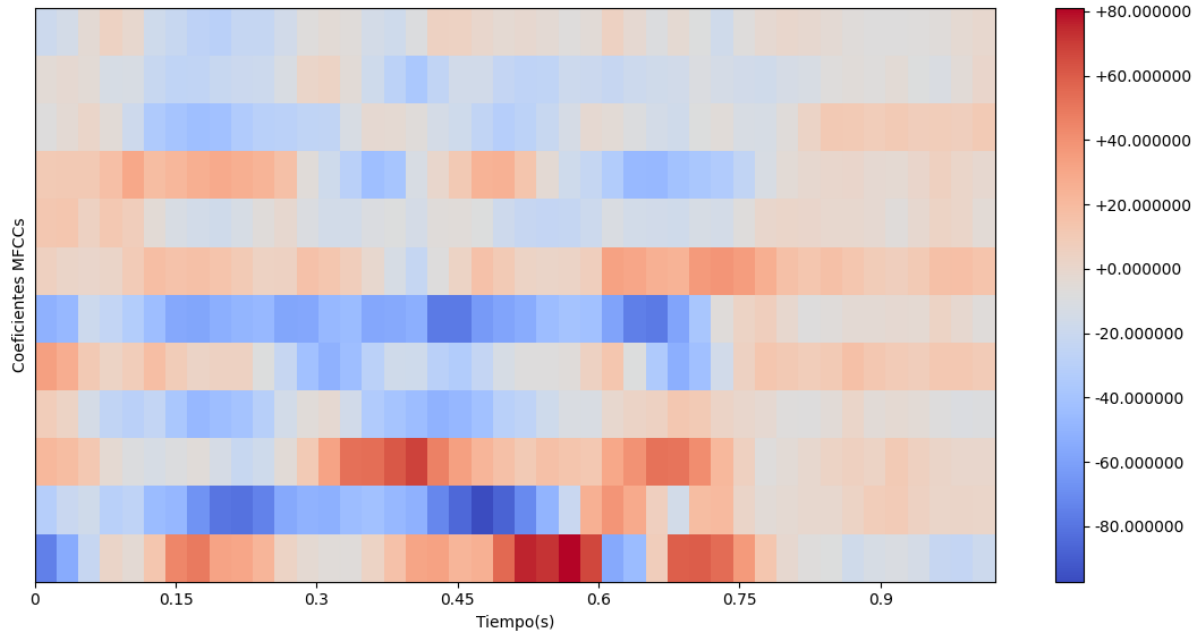

Figura 3.23. MFCCs del vocablo adelante (elaboración propia).

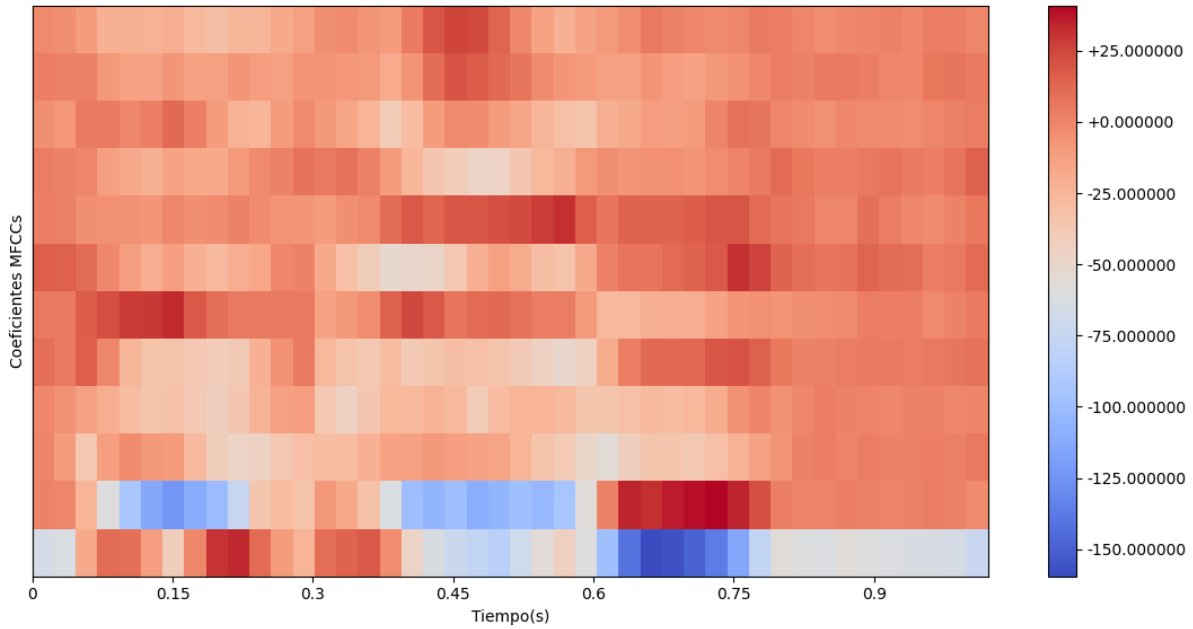

Figura 3.24. MFCCs del vocablo atrás (elaboración propia).

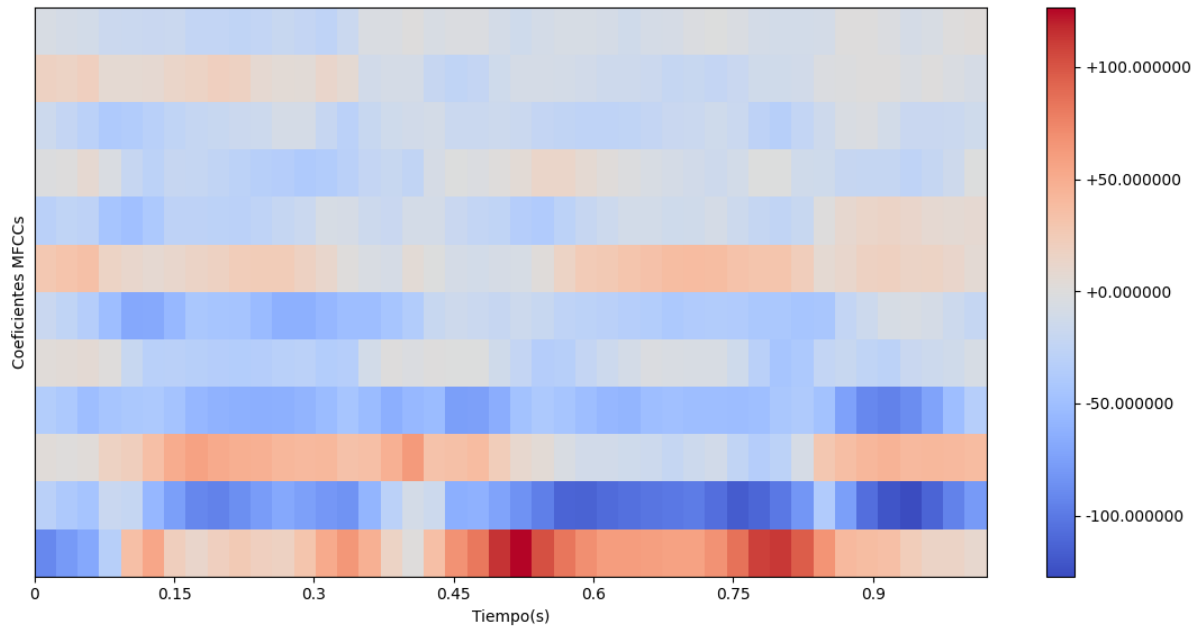

Figura 3.25. MFCCs del vocablo cerrar (elaboración propia).

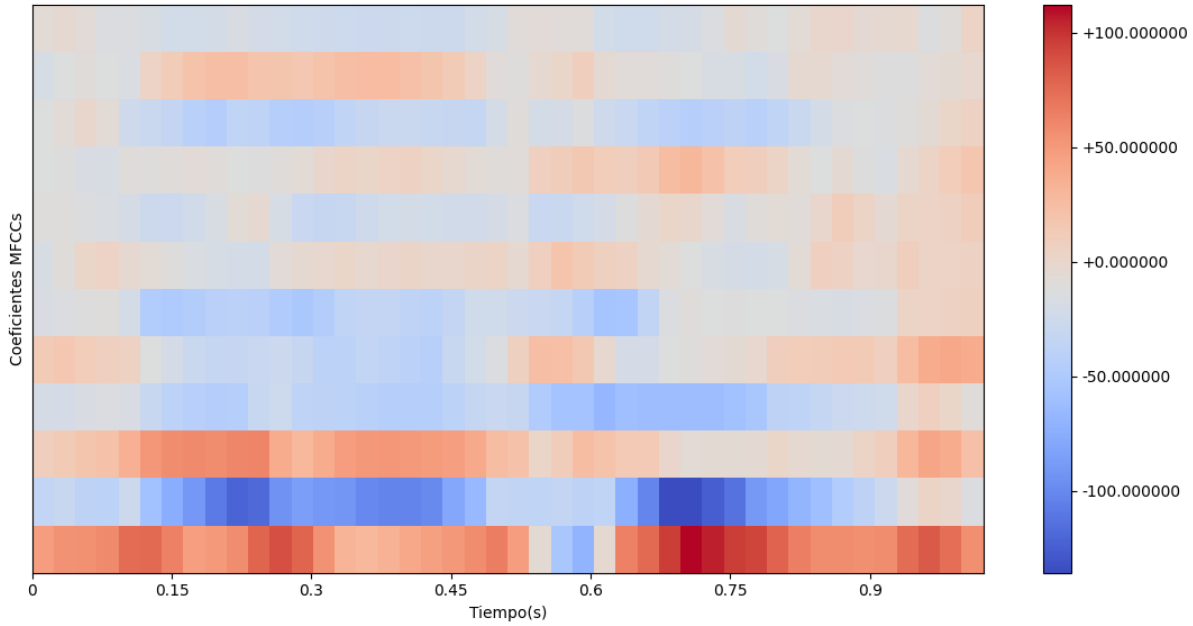

Figura 3.26. MFCCs del vocablo derecha (elaboración propia).

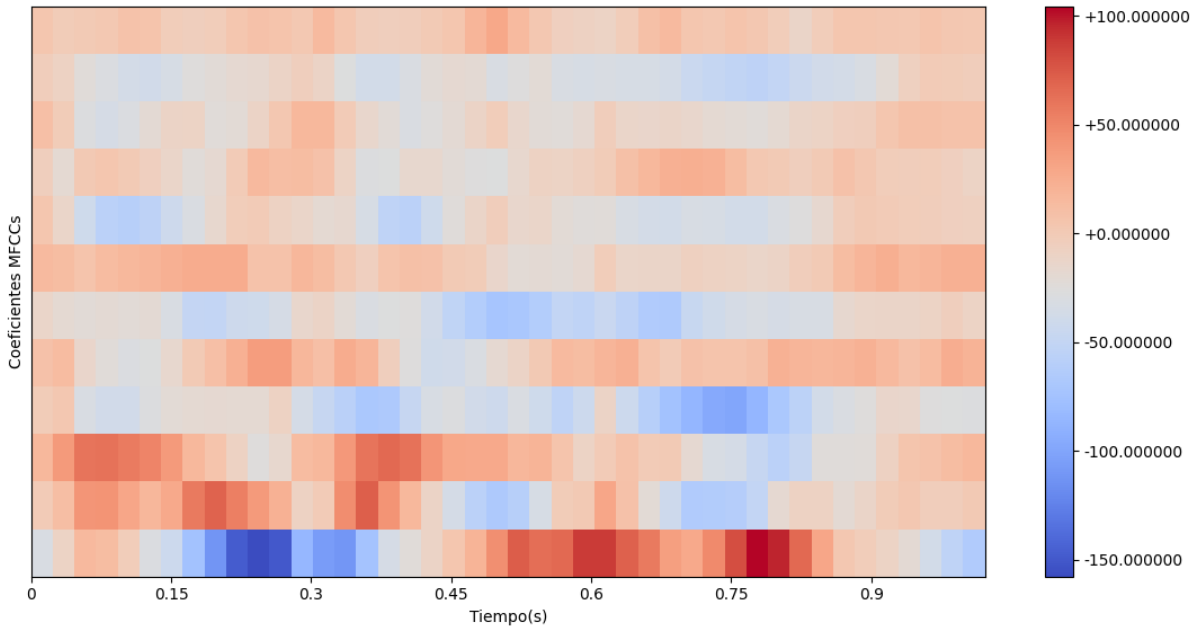

Figura 3.27. MFCCs del vocablo izquierda (elaboración propia).

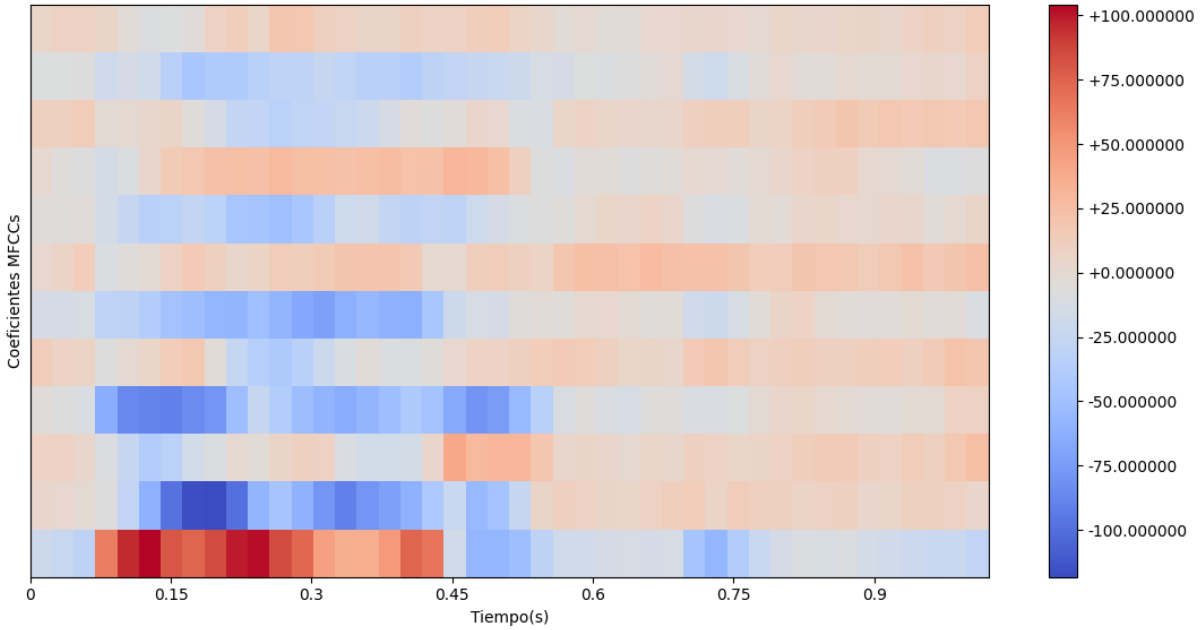

Figura 3.28. MFCCs del vocablo parar (elaboración propia).

3.3. ESTRUCTURA DE LA BASE DE DATOS

Una vez obtenidos los MFCCs y las etiquetas de todos los ficheros de audio a reconocer, se deben guardar o almacenar los datos en una estructura que permita operar con ellos fácilmente. La naturaleza de los datos hace que la estructura más idónea para almacenar los coeficientes y las etiquetas sea un archivo .csv, donde se representó cada fichero de audio como una fila. La primera columna se utilizó para almacenar los coeficientes extraídos de cada fichero, compuesto por un vector de *floats* de dimensiones (44 × 12), y la segunda columna toma el valor de la etiqueta, en formato *string*.

3.3.1. Datos de entrenamiento, validación y pruebas

En esta etapa se dividió la base de datos en tres subconjuntos: entrenamiento, validación y pruebas. El conjunto de entrenamiento es el que contiene los ejemplos con los cuales el modelo aprendió las características e identificó los patrones en los datos. El conjunto de validación permitió identificar si el modelo estaba aprendiendo de una manera correcta y no se estaba teniendo sobreajuste (*overfitting*) durante el entrenamiento. El conjunto de pruebas o test permitió comprobar el desempeño del modelo final a partir de las métricas de evaluación.

Para la separación de los conjuntos de datos de entrenamiento, validación y pruebas se hizo uso de la función *train_test_split*, perteneciente a la librería *scikit-learn*. Esta función recibe como parámetros las variables de entrada *x* y de salida *y*, que contienen los coeficientes y las etiquetas respectivamente y la variable *ramdon_state* la cual ayuda a controlar el factor aleatorio con que la función *train_test_split* separa los datos. El porcentaje de separación por el cual se optó fue un 64% para los datos de entrenamiento, 16% para los datos de validación

y 20% para los datos de prueba. La función retorna seis *numpy array* qué contiene los datos de *x* y *y* separados en los conjuntos de entrenamiento, validación y pruebas. (ver Apéndice B).

3.4. ARQUITECTURA DE LA RED NEURONAL

En la creación de la arquitectura de la NN se implementaron dos capas de celdas LSTM, siendo una la capa de entrada con 150 neuronas y la otra una capa oculta con 100 neuronas. En las dos capas se utilizó la función de activación *tanh*, la cual es la función por defecto del planteamiento original de las celdas LSTM, además de aplicarse sobre las dos capas el método de regularización Dropout al 20%. La capa de salida es una densamente conectada de 7 neuronas, una neurona por cada posible clase (vocablo) a reconocer, esta con función de activación Softmax. En la figura 3.29 se observa la arquitectura de la NN utilizada para este trabajo de grado.

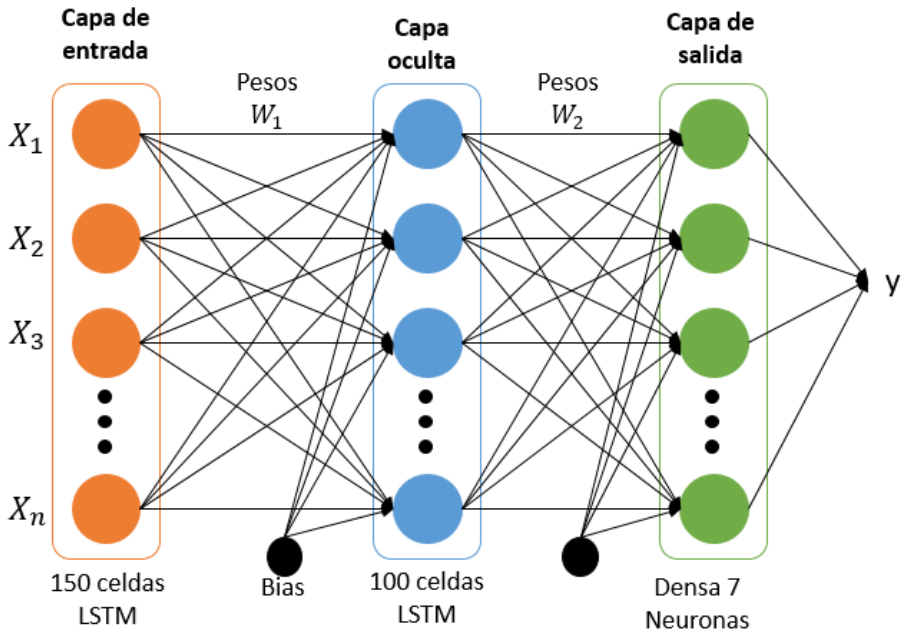

Figura 3.29. Arquitectura de la NN (elaboración propia).

3.5. ENTRENAMIENTO DE LA RED NEURONAL

Una vez pre-procesada y estructurada la base de datos, los datos están listos para ser introducidos a la capa de entrada de la NN diseñada. El objetivo de entrenar el modelo es encontrar de manera iterativa los valores de los pesos (W) y *bias* (b) más adecuados para la entrada de cada una de las neuronas de la red, con el fin de que la respuesta de la capa de salida se ajuste lo más posible a los datos esperados. Para esto se hizo uso del algoritmo de optimización *Adam*, el cual tiene una mejor velocidad de convergencia en la búsqueda de una solución para W y b que otros algoritmos de optimización.

3.6. ESTRUCTURA DEL PROYECTO

Para la estructura del proyecto, teniendo en cuenta organización y optimización en procesos, se decidió dividir en 4 *scripts* el desarrollo software, asignando así tareas específicas para cada *script*, las cuales son:

Create_db.py es un script en el cual, mediante la función de *PyAudio,* se realizaron las grabaciones de la pronunciación de los vocablos que conformaron la base de datos, almacenándolos como un archivo *wave*. En paralelo a cada grabación, se realizó el proceso de detección de silencio inicial de cada fichero para su posterior eliminación y la aplicación del filtro de preénfasis.

Prepare_dataset_CSV.py es un script el cual recibe la base de datos previamente grabada y realiza todo lo referente al pre-procesamiento del audio, obteniendo así los coeficientes MFCC y sus respectivas etiquetas, los cuales fueron las entradas de la red neuronal, además de esto se genera el archivo de almacenamiento *.csv*.

NN_LSTM.py un script con cual se realizó la separación de los datos para el proceso de entrenamiento, validación y pruebas, se diseñó el modelo de la red neuronal y se realizó el respectivo entrenamiento y evaluación del modelo, obteniendo los resultados según las métricas definidas previamente.

Recognition_App.py un script con el cual se realizaron las pruebas finales del sistema, éste engloba las funciones principales de los códigos mencionados anteriormente, donde se realiza una grabación, para ser sometida a detección / eliminación de silencios y preénfasis, obtención de MFCCs del audio y pasando dichos coeficientes como entrada al modelo ya entrenado de la NN y así obtener la respectiva predicción.

Capítulo 4

EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS

En este capítulo se exponen los experimentos llevados a cabo para entrenar, optimizar y evaluar el modelo de la NN a partir de los resultados obtenidos. Además de esto, se realizó la experimentación y comparación del sistema, cuando éste recibió como entrada las características de los audios con preénfasis y las mismas sin la aplicación de preénfasis, para evaluar si dicho proceso tiene alguna relevancia sobre el rendimiento y efectividad del sistema.

4.1 MODELO DE LA RED NEURONAL

El modelo de la NN se implementó haciendo uso de la librería *Keras* de *TensorFlow* en *Python.* La clase *Sequential* permite agregar nuevas capas a la NN de una manera sencilla e indicar para cada capa el número de neuronas a implementar, así mismo, la función de activación a utilizar. Para la creación del modelo inicial de la NN con 3 capas, correspondientes a la capa de entrada, la capa oculta y la capa de salida. Para la capa de entrada con 150 neuronas y la capa oculta con 100 neuronas, se utilizaron celdas LSTM donde estas utilizan la función de activación *tanh*, la usada por defecto por estas celdas, respetando la implementación original; y se aplicó un *Dropout* de 0.2 para evitar el sobre ajuste durante el entrenamiento. La capa de salida es una capa *Dense* con 7 neuronas (una por cada clase) y utiliza la función de activación *Softmax.*

Una vez creado el modelo este se debe compilar utilizando un algoritmo de optimización. Se utilizó el optimizador *Adam*, siendo este uno de los mejores algoritmos debido a su buen rendimiento. La función *compile* de *keras* recibe como parámetros el optimizador del modelo, la función de pérdida, qué en este caso se implementó *sparse_categorical_crossentropy*, y la métrica de desempeño, donde se utilizó *accuracy* o exactitud. (ver Apéndice B).

Lo siguiente corresponde al entrenamiento de la red, lo cual se realizó con la función *fit* de *keras.* Esta función recibe como parámetros los conjuntos de datos qué contienen *x* y *y del* conjunto de entrenamiento, el porcentaje del conjunto de datos que se utilizó para la validación corresponde al 16% de los datos, el cual está contenido en las variables *X_validation* y *y_validation,* el número de épocas (epochs) en las que se realizó el entrenamiento que corresponde a 250 iteraciones y el tamaño de lote en el que se sub-dividió el conjunto de datos de entrenamiento, qué en este caso fue de 40. La salida de la función *fit* corresponde a la información de las métricas de desempeño para cada iteración.

Una vez entrenado el modelo, se utiliza el conjunto de datos destinado como *test* para evaluar el desempeño obtenido con el modelo. Para esto se hizo uso de la función *evaluate* de *keras*, la cual recibe como parámetros las variables *X_test* y *y_test* que contienen los conjuntos de datos para las pruebas y las etiquetas respectivamente. En la tabla 4.1 se presenta el resultado obtenido del entrenamiento, validación y pruebas del modelo actual.

Tabla 4.1. Resultados del modelo (elaboración propia).

Métrica	**Entrenamiento**	**Validación**	**Test**
Precisión (acc)	98.67%	97.86%	98.09%
Pérdida (loss)*[nats]*	0.0534	0.0607	0.051

Como se puede observar en la tabla 4.1, se alcanzó un porcentaje de precisión en la evaluación del sistema de 98.09% para el conjunto de *test*, siendo este un valor considerablemente bueno con una convergencia alta para el sistema planteado, por lo que se concluye que el modelo es capaz de aprender y se puede tomar como una solución válida para el problema planteado.

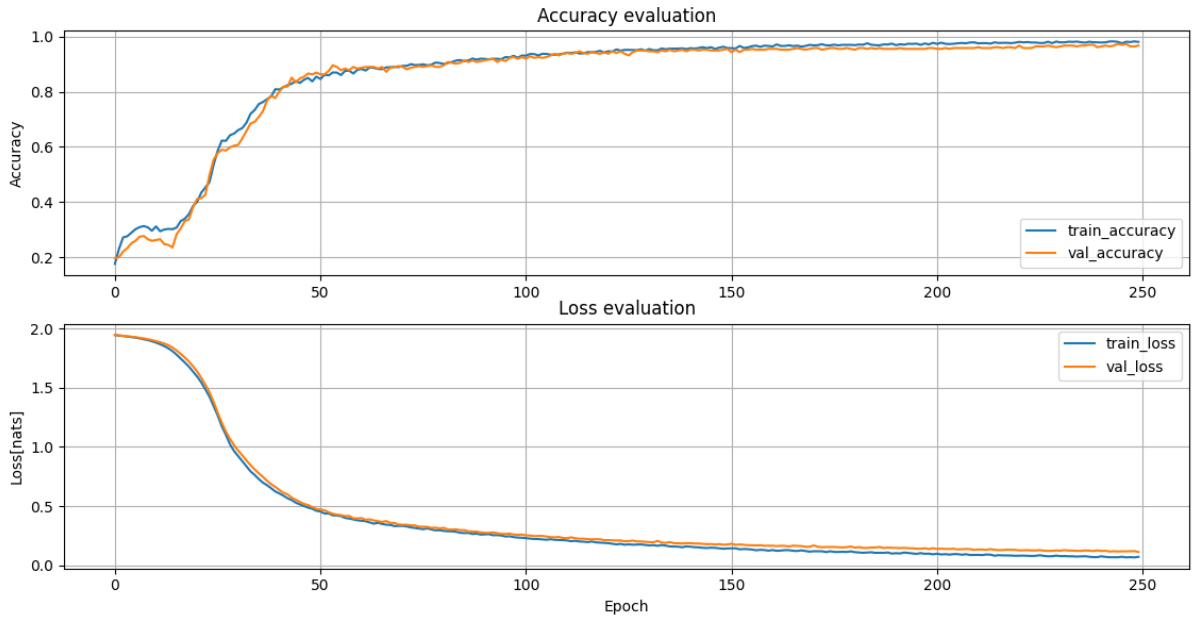

Figura 4.1. Métricas de evaluación del modelo (elaboración propia).

De igual manera, en la figura 4.1 se evidencia que a medida que van transcurriendo las épocas o iteraciones se van alcanzando valores considerables en cuanto a la precisión de predicción en la red y en la pérdida se tienen valores muy cercanos a cero, además de notar que no se tiene sobreajuste, dado que los valores para entrenamiento y validación van mejorando de una manera progresiva, paralela y sin cambios abruptos, convergiendo a un mismo punto al final del entrenamiento.

4.2 MODELO DE RED NEURONAL CON USO DE PREÉNFASIS EN LA BASE DE DATOS

Para este caso, se evaluó y se hizo una comparación de la variación en las métricas de evaluación del modelo previamente definido respecto a este modelo, al cual se le aplicó un filtro de preénfasis a los archivos de audio de la base de datos.

Tabla 4.2. Comparación de resultados de los modelos (elaboración propia).

Modelo	**Métrica**	**Entrenamiento**	**Validación**	**Test**
Sin Preénfasis	*Precisión (acc)*	98.67%	97.86%	98.09%
	Pérdida (loss) [nats]	0.0534	0.0607	0.051
Con Preénfasis	*Precisión (acc)*	98.92%	98.34%	98.72%
	Pérdida (loss) [nats]	0.0369	0.0601	0.0508

En la tabla 4.2 se visualiza la comparación de resultados de ambos modelos, donde se nota que el modelo al cual se le aplicó el filtro de preénfasis a la base de datos genera mejores resultados en las métricas de evaluación y se puede observar que los tres resultados, tanto el entrenamiento, validación y test se encuentran más cercanos entre sí con porcentajes ligeramente superiores a los del modelo sin filtro de preénfasis, el cual es un resultado esperado, debido a que al usar este filtro de preénfasis, permitió equilibrar el espectro de frecuencias al amplificar las componentes de alta frecuencia y compensó la pendiente natural de la voz antes del análisis espectral, por tanto, se tuvo un análisis espectral más claro y así una extracción de MFCCs más precisa.

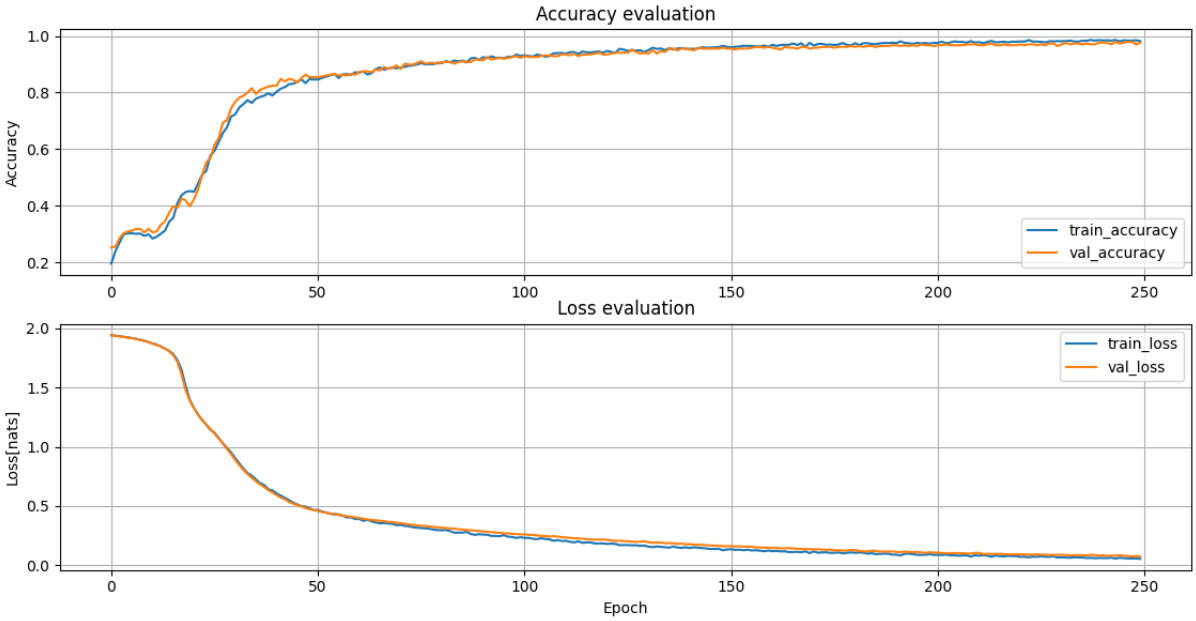

Figura 4.2. Métricas de evaluación del modelo con aplicación de preénfasis (elaboración propia).

Continuando con lo mencionado anteriormente, en la figura 4.6 se puede corroborar gráficamente los resultados del modelo con uso del filtro de preénfasis, donde se observa que los valores para entrenamiento y validación al final de la ejecución convergen a un mismo punto, tanto para la precisión como para la pérdida, estos van mejorando en paralelo y progresivamente, sin tener sobreajuste a medida que aumentan las iteraciones.

Por lo anterior se concluyó en utilizar el modelo al cual se le aplicó el filtro de preénfasis a la base de datos para realizar las correspondientes pruebas de predicción del modelo.

4.3 PRUEBAS

Una vez culminado el diseño y entrenamiento del modelo de predicción, se procedió a finalizar el desarrollo del sistema con una fase de pruebas, donde se midió la eficiencia del sistema completo, teniendo en cuenta que el funcionamiento de cada fase del desarrollo fue evaluado por separado, desde la grabación de los archivos de audio, el pre-procesado y el entrenamiento del modelo de la NN.

Para esta fase de pruebas, se hizo uso de un *script* el cual recibió grabaciones en tiempo real de 5 locutores, donde 2 de estos fueron partícipes de la creación de la base de datos y 3 son ajenos a ella, los cuales realizaron una grabación por cada vocablo, teniendo así un total de 35 grabaciones, donde se prueba el sistema y se corrobora el desempeño del mismo, desde el momento en que se generan los ficheros hasta que se recibe la predicción del vocablo. Esto se hizo mediante el uso del *script* denominado *Recognition_App.py*, con el cual se obtuvo un tiempo de ejecución promedio de 5.5 segundos.

Durante el proceso de pruebas del modelo se presentaron ciertos inconvenientes. Al realizar las pruebas con algún locutor, se evidenció qué el sistema solo lograba predecir dos o tres de los 7 vocablos, a pesar de tener un porcentaje superior al 98% durante la fase de entrenamiento y no verse afectado por sobreajuste, esto fue debido a que con el uso de la función *train_test_split*, la encargada de realizar la separación de los datos en entrenamiento, validación y pruebas, esta realizaba la separación de una manera aleatoria no controlada, lo cual se prestó para que se seleccionaran más características de ciertos vocablos y menos de otros o incluso ninguna característica, por ende el sistema no predecía correctamente todos los vocablos, dado que no se tenía un equilibrio en la distribución de los datos. Para dar solución a este inconveniente, se hizo uso de la variable *random_state* para garantizar una distribución aleatoria equilibrada de los datos, también se utilizó un valor de *learning_rate* más bajo y se incrementó el valor de *epochs* para garantizar que el sistema tuviera un mayor aprendizaje, esto conllevó a qué el proceso de entrenamiento fuera un poco más demorado al tener más iteraciones.

A continuación, en las tablas 4.3, 4.4 y 4.5 se muestran las predicciones resultantes de las pruebas realizadas.

Tabla 4.3. Predicción de las pruebas del locutor 1 (elaboración propia).

Locutor	Etiqueta	Predicción
1 (Hombre)	Abrir	Abrir
	Adelante	Adelante
	Atrás	Atrás
	Cerrar	Cerrar
	Derecha	Abrir
	Izquierda	Izquierda
	Parar	Atrás

Tabla 4.4. Predicción de las pruebas del locutor 2 (elaboración propia).

Locutor	Etiqueta	Predicción
2 (Hombre participe de la base de datos)	Abrir	Abrir
	Adelante	Adelante
	Atrás	Atrás
	Cerrar	Cerrar
	Derecha	Derecha
	Izquierda	Izquierda
	Parar	Parar

Tabla 4.5. Predicción de las pruebas del locutor 2 (elaboración propia).

Locutor	Etiqueta	Predicción
3 (Mujer)	Abrir	Abrir
	Adelante	Adelante
	Atrás	Atrás
	Cerrar	Derecha
	Derecha	Abrir
	Izquierda	Izquierda
	Parar	Cerrar

En los resultados mostrados anteriormente, se muestran las pruebas realizadas de reconocimiento a tres locutores, las pruebas de los otros dos locutores se pueden observar en el Apéndice C. Con un total de 35 grabaciones, teniendo un total de 29 predicciones correctas y 6 erróneas, lo cual representa aproximadamente 82,85% de precisión de las pruebas realizadas, el cual es un resultado que permitió validar el funcionamiento correcto del sistema a pesar de contar con una base de datos relativamente pequeña, con respecto a la cantidad de datos qué se manejan para un sistema de DL. Con esto se corroboraron los resultados obtenidos

en la fase de entrenamiento, validación y test donde se obtuvo con la métrica de precisión valores de porcentaje muy buenos, como se observó en la tabla 4.2.

La ejecución de las pruebas concluyó con 6 ficheros mal clasificados, debido a que las variaciones de los resultados dependen de la capacidad vocal, la gesticulación del locutor, también podría deberse a pronunciaciones con poca representación en la base de datos o a que el sistema no haya tomado las suficientes características de estos vocablos erróneos.

Capítulo 5

CONCLUSIONES Y TRABAJOS FUTUROS

5.1 CONCLUSIONES

La experiencia demuestra que del tamaño y calidad de la base de datos depende en gran parte el desempeño de un algoritmo de aprendizaje *Deep Learning*, por lo que recurrir a bases de datos ya construidas es una forma de investigar; sin embargo, en el presente trabajo, y con ánimo de personalizar la aplicación del sistema propuesto al reconocimiento de vocablos específicos, se tuvo que crear la base de datos desde cero, lo cual implicó un trabajo de campo de recolección de grabaciones y posterior pre-procesamiento, con lo que se pudo consolidar nuestro conocimiento en el procesamiento digital de señales de audio a partir de datos crudos.

Para la construcción de las bases de datos a partir de los datos pre-procesados, se optó por la extracción de los coeficientes Cepstrales MFCC a cada registro de audio de los vocablos, siendo estos sus características más representativas y adecuadas para su discriminación al pasar como datos de entrada del algoritmo de aprendizaje *Deep Learning*. Para esto, fue necesario considerar la misma cantidad de muestras para todas las grabaciones, con el fin de garantizar las mismas dimensiones en el vector de coeficientes de cada fichero de audio, esto para evitar inconvenientes en el momento de almacenar estos datos en el archivo tipo csv de la base de datos.

En la literatura científica, se encuentra que entre más profunda fuera la Red Neuronal (capas ocultas), mejor sería el desempeño del sistema, siempre y cuando la base de datos sea lo suficientemente grande; sin embargo, dado el tamaño limitado de la base de datos

construida para el presente trabajo, se alcanzaron unos buenos resultados con una red de aprendizaje *Deep Learning* de una capa de entrada, una de salida y una única capa oculta.

La selección y aplicación de las métricas de evaluación de un algoritmo se hacen dependiendo del propósito para el cual éste fue creado. En el presente trabajo, a partir de las métricas de precisión (función de la tasa de aciertos) y pérdidas (función de probabilidad de predicción), se comprobó que un sistema de procesamiento de la voz implementado en *Deep Learning* es efectivo para el reconocimiento de vocablos personalizados, con una precisión de aproximadamente 98% y unas pérdidas del 5%, a pesar del volumen bajo de la base de datos construida.

El aplicar un filtro de preénfasis a los archivos de audio puede mejorar el desempeño del algoritmo *Deep Learning* propuesto, subiendo del 98% a 98,7% a costa de algún tiempo extra de procesamiento, lo que obliga a sopesar su conveniencia o no.

5.2 TRABAJOS FUTUROS

A continuación, se presentan posibles ampliaciones o iteraciones sobre el proyecto aquí desarrollado, con el fin de mejorar lo ya hecho o ampliar el alcance del mismo.

- **Ampliación de la base de datos.**

Incrementar el volumen de la base de datos, tanto en cantidad de datos como en vocablos a reconocer, esto para realizar un entrenamiento más óptimo y así mismo alcanzar mejores porcentajes de predicción y con la ampliación de vocablos poder ampliar el alcance y aplicación del sistema desarrollado.

- **Mejorar la arquitectura de la red neuronal.**

Al ampliar la base de datos se propone variar los parámetros correspondientes a las redes neuronales (*learning rate*, número de capas, número de neuronas, etc), utilizar otras técnicas de regularización y hacer uso de otras funciones de activación como *ReLu*, en general ampliar la complejidad de la red neuronal, con la finalidad de conseguir un mejor desempeño del sistema.

- **Implementación de un aplicativo final.**

Al tener un sistema con un rendimiento excelente, es decir, con un porcentaje de precisión del 100%, entrenado con comandos en un entorno con ruido, se puede pensar en la integración del sistema con una aplicación final en alguna plataforma de interés, dependiendo de la finalidad

que se le dé al sistema, con el fin obtener una implementación en un entorno real y así poder medir la efectividad del sistema.

- **Comparación de arquitecturas**

Realizar una comparación del rendimiento entre el sistema diseñado con LSTM y otro tipo de RNN, en específico con una red de Unidades Recurrentes Cerradas (*GRU*, Gated Recurrent Unit), teniendo en cuenta que ambas son muy similares, pero la GRU es más simple en arquitectura debido a que maneja dos compuertas en lugar de tres, como en las LSTM, lo cual podría ser funcional para operaciones más simples, dado que usa menos parámetros para el entrenamiento de la red, usando menos memoria y obteniendo respuestas más rápidas.

Bibliografía

[1] E. Connect, "Asistentes virtuales de voz: una “revolución silenciosa” en el ámbito de la salud", *Elsevier Connect*, 2020. [Online]. Available: https://www.elsevier.com/es-es/connect/ehealth/asistentes-virtuales-de-voz-una-revolucion-silenciosa-en-el-ambito-de-la-salud.

[2] R. Miranda, *Ciapat.org*, 2007. [Online]. Available: http://ciapat.org/biblioteca/pdf/673-Discapacidad_y_Accesibilidad.pdf.

[3] I. Rodríguez, M. Salamanca and J. Márquez, *Utpuebla.edu.mx*. [Online]. Available: http://www.utpuebla.edu.mx/divisiones/tic/TIC/2_materias/2/image/2do_sist/Introduccio%CC%81n%20al%20ana%CC%81lisis%20y%20disen%CC%83o%20de%20sistemas/Portafolio%20de%20evidencias/Producto%204.pdf.

[4] C. McClelland, "The Difference Between Artificial Intelligence, Machine Learning, and Deep Learning", *Medium*, 2021. [Online]. Available: https://medium.com/iotforall/the-difference-between-artificial-intelligence-machine-learning-and-deep-learning-3aa67bff5991].

[5] Huawei, “*HCIA-AI V3.0 Training Material.pdf*”, Huawei Colombia, 2021, material de aprendizaje curso Inteligencia Artificial (HCIA-AI).

[6] A. Arias Paéz and D. Rubiano Venegas, "MÉTODO AUTOMÁTICO DE RECONOCIMIENTO DE VOZ PARA LA CLASIFICACIÓN DE VOCALES AL LENGUAJE DE SEÑAS COLOMBIANO", *repository.ucatolica.edu.co*, 2018. [Online]. Available: https://repository.ucatolica.edu.co/bitstream/10983/22809/1/M%C3%89TODO%20AUTOM%C3%81TICO%20DE%20RECONOCIMIENTO%20DE%20VOZ%20PARA%20LA%20CLASIFICACI%C3%93N%20DE%20VOCALES%20AL%20LENGUAJE%20DE%20SE%C3%91AS%20COLOMBIANO.pdf.

[7]. Vicente, "Sistema de Reconocimiento de Comandos por Voz Basado en Redes Neuronales LSTM", *Core.ac.uk*, 2018. [Online]. Available: https://core.ac.uk/download/pdf/159466484.pdf.

[8] "¿Qué es deep learning?", *Sas.com*, 2021. [Online]. Available: https://www.sas.com/es_co/insights/analytics/deep-learning.html#:~:text=El%20deep%20learning%20es%20un,de%20im%C3%A1genes%20o%20hacer%20predicciones.

[9] O. Marketing, "Neural networks: ¿de qué son capaces las redes neuronales artificiales?", *IONOS Digitalguide*, 2020. [Online]. Available: https://www.ionos.es/digitalguide/online-marketing/marketing-para-motores-de-busqueda/que-es-una-neural-network/#:~:text=Una%20red%20neuronal%20(neural%20network,los%20ordenadores%20de%20inteligencia%20artificial.

[10] A. Mittal, "Understanding RNN and LSTM", *Medium*, 2019. [Online]. Available: https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e.

[11] A. Mañas, "Capítulo 8 Métodos basados en Deep Learning | Notas sobre pronóstico del flujo de tráfico en la ciudad de Madrid", *Bookdown.org*. [Online]. Available: https://bookdown.org/amanas/traficomadrid/m%C3%A9todos-basados-en-deep-learning.html.

[12] M. Sotaquirá, "¿Qué son las Redes LSTM?", *Codificandobits.com*, 2019. [Online]. Available: https://www.codificandobits.com/blog/redes-lstm/.

[13] J. Cea, *Oa.upm.es*, 2020. [Online]. Available: http://oa.upm.es/63687/1/TFM_JUAN_JULIAN_CEA_MORAN.pdf.

[14] I. Alvarado, "Redes Neuronales", *Ml4a.github.io*, 2021. [Online]. Available: https://ml4a.github.io/ml4a/es/neural_networks/.

[15] E. Freire and S. Silva, "Redes neuronales", *Medium*, 2019. [Online]. Available: https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb.

[16] P. Cornieles, "Entendiendo las redes neuronales: De la neurona a RNN, CNN y Deep Learning - IA Latam", *IA Latam*. [Online]. Available: https://ia-latam.com/2019/02/06/entendiendo-las-redes-neuronales-de-la-neurona-a-rnn-cnn-y-deep-learning/.

[17] "Cómo elegir una función de activación para el aprendizaje profundo - Top Big Data", *Top Big Data*, 2021. [Online]. Available: https://topbigdata.es/como-elegir-una-funcion-de-activacion-para-el-aprendizaje-profundo/#:~:text=La%20funci%C3%B3n%20de%20activaci%C3%B3n%20de%20la%20tangente%20hiperb%C3%B3lica%20tambi%C3%A9n%20se,rango%20de%20%2D1%20a%201.

[18] J. Zambrano, "¿Aprendizaje supervisado o no supervisado?", *Medium*, 2018. [Online]. Available: https://medium.com/@juanzambrano/aprendizaje-supervisado-o-no-supervisado-39ccf1fd6e7b.

[19] G. Vásquez, "Clasificador con redes neuronales para el pronóstico de la enfermedad renal crónica en la población de Colombia", *Reunir.unir.net*, 2019. [Online]. Available: https://reunir.unir.net/bitstream/handle/123456789/9446/V%C3%A1zquez%20Morales%2C%20Gabriel.pdf?sequence=1&isAllowed=y.

[20] "Métricas De Evaluación De Modelos En El Aprendizaje Automático", *DataSource.ai*, 2021. [Online]. Available: https://www.datasource.ai/es/data-science-articles/metricas-de-evaluacion-de-modelos-en-el-aprendizaje-automatico.

[21] L. Velasco, "Optimizadores en redes neuronales profundas: un enfoque práctico", *Medium*, 2020. [Online]. Available: https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5.

[22] I. Martín, *Repositorio.uam.es*, 2019. [Online]. Available: https://repositorio.uam.es/bitstream/handle/10486/688634/martin_calle_irene_tfm.pdf?sequence=1&isAllowed=y.

[23] J. Molina, *Dte.us.es*. [Online]. Available: https://www.dte.us.es/ing_inf/trat_voz/Practicas/PracticaVoz1.pdf.

[24] I. Ruiz, "MFCCs", *Turing.iimas.unam.mx*. [Online]. Available: https://turing.iimas.unam.mx/~ivanvladimir/posts/mfcc/.

[25] A. Arias and D. Rubiano, "MÉTODO AUTOMÁTICO DE RECONOCIMIENTO DE VOZ PARA LA CLASIFICACIÓN DE VOCALES AL LENGUAJE DE SEÑAS COLOMBIANO.", *Repository.ucatolica.edu.co*, 2021. [Online]. Available: https://repository.ucatolica.edu.co/bitstream/10983/22809/1/M%C3%89TODO%20AUTOM%C3%81TICO%20DE%20RECONOCIMIENTO%20DE%20VOZ%20PARA%20LA%20CLASIFICACI%C3%93N%20DE%20VOCALES%20AL%20LENGUAJE%20DE%20SE%C3%91AS%20COLOMBIANO.pdf.

[26] K. Team, "Keras documentation: About Keras", *Keras.io*. [Online]. Available: https://keras.io/about/.

[27] "What is NumPy? — NumPy v1.21 Manual", *Numpy.org*. [Online]. Available: https://numpy.org/doc/stable/user/whatisnumpy.html.

[28] "librosa — librosa 0.8.1 documentation", *Librosa.org*. [Online]. Available: https://librosa.org/doc/latest/index.html.

[29] "csv — Lectura y escritura de archivos CSV — documentación de Python - 3.10.0", *Docs.python.org*. [Online]. Available: https://docs.python.org/es/3/library/csv.html.

[30] T. Wood, "Softmax Function", *DeepAI*. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/softmax-layer.

[31] "El algoritmo más utilizado para el aprendizaje profundo: el algoritmo de optimización de Adam - programador clic", *Programmerclick.com*. [Online]. Available: https://programmerclick.com/article/4434513201/.

[32] Z. Global, "Brutalk - Una suave introducción a las métricas de probabilidad para la clasificación desequilibrada", *Brutalk*. [Online]. Available: https://www.brutalk.com/en/news/brutalk-blog/view/una-suave-introduccion-a-las-metricas-de-probabilidad-para-la-clasificacion-desequilibrada-6046dad9ac5de.

[33] R. Castaño, *AbDatum*, 2021. [Online]. Available: https://abdatum.com/machine-learning/entropia-shannon#:~:text=Entrop%C3%ADa%20cruzada%20o%20%E2%80%9Ccross%2Dentropy,de%20perdida%20(loss%20function).

[34] M. Ibarra, *Maixx.files.wordpress.com*, 2011. [Online]. Available: https://maixx.files.wordpress.com/2012/10/cap12_dct_v01_01_03.pdf.

Apéndice A

BASE DE DATOS

Se entrega la base de datos pre-procesada con la eliminación del silencio inicial y aplicación del filtro de preénfasis de la pronunciación de los vocablos en español. Esta cuenta con un total de 2800 ficheros de audio, 400 ficheros por cada vocablo.

URL: https://drive.google.com/drive/folders/1MHJlkuwmZt0sbjRkrA2wwHECS-moWzoq?usp=sharing

Apéndice B

CÓDIGO FUENTE

1. CREACIÓN DE LA BASE DE DATOS, ELIMINACIÓN DE SILENCIO Y FILTRO DE PREÉNFASIS

```
'''
IMPLEMENTACIÓN DE UN SISTEMA DE PROCESAMIENTO DE LA VOZ HUMANA CON DEEP
LEARNING QUE PERMITA
                                EL RECONOCIMIENTO DE VOCABLOS

@autores: Andrés Felipe Camargo Caicedo y Luis Gustavo Moreno Camargo.
@universidad: UNIVERSIDAD DEL CAUCA.
@versión: 1.0, 01/05/2022
'''

import pyaudio, wave, time, librosa
from pydub import AudioSegment
import numpy as np
import scipy.io.wavfile as waves
#import matplotlib.pyplot as plt

TIME = 3 # Duración de la grabación
N = 1 # Número de grabaciones
CHANNELS = 2
RATE = 44100
CHUNK = 1024

# Función para detectar silencios
def detect_silence(signal, threshold=-30.0, chunk_size=15):
    '''
    threshold en dB
    chunk_size en ms
```

```
    '''
    trim = 0 # ms

    assert chunk_size > 0 # evita loop infinito
    while signal[trim:trim+chunk_size].dBFS < threshold and trim <
len(signal):
        trim += chunk_size

    return trim

# Bucle para realizar n grabaciones
for i in range(N):

    # Ruta para los archivos grabados y sin el silencio del inicio
    db_path = r'Pruebas\audio'
    number = str(i)
    file = db_path +  '.wav'

    audio=pyaudio.PyAudio() #Se inicia PyAudio

    # Se definen características de audio
    stream=audio.open(format=pyaudio.paInt16,channels=CHANNELS,
                      rate=RATE,input=True,
                      frames_per_buffer=CHUNK)

    # Se inicia grabación
    print("Grabando ...")
    frames=[]
    for i in range(0,int(RATE/CHUNK*TIME)):
        data=stream.read(CHUNK)
        frames.append(data)

    # Finaliza la grabación
    print("Grabacion terminada ")
    stream.stop_stream()
    stream.close()
    audio.terminate()

    # Se guarda archivo con sus parámetros
    waveFile=wave.open(file,'w')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(pyaudio.paInt16))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(frames))
    waveFile.close()

    time.sleep(1) # Retardo de 1seg entre cada grabación
```

```
    # Se carga el archivo de audio
    sound = AudioSegment.from_file(file, format="wav")

    # Eliminación del silencio inicial
    start_trim = detect_silence(sound)
    duration = len(sound) # ms
    trimmed_sound = sound[start_trim - 100:duration]
    trimmed_sound.export(file, format="wav")

    # Se carga el archivo
    signal, sr = librosa.load(file)

    # Preénfasis
    sound_shift = np.roll(signal, 1) # corrimiento ciclico de una posición
    sound_shift[0] = 0 # el primer valor se pone en cero
    filtered_sound = signal - 0.97*sound_shift
    waves.write(file, sr, filtered_sound)
```

2. EXTRACCIÓN DE LOS MFCCs Y CREACIÓN ARCHIVO .CSV

```
'''
IMPLEMENTACIÓN DE UN SISTEMA DE PROCESAMIENTO DE LA VOZ HUMANA CON DEEP
LEARNING QUE PERMITA
                                EL RECONOCIMIENTO DE VOCABLOS

@autores: Andrés Felipe Camargo Caicedo y Luis Gustavo Moreno Camargo.
@universidad: UNIVERSIDAD DEL CAUCA.
@versión: 1.0, 01/05/2022
'''

import csv
import librosa
import os

DATASET_PATH = "Base_Datos_pre"
CSV_PATH = "data.csv"
SAMPLES_TO_CONSIDER = 22050 # 1 seg de audio

def prepare_dataset(dataset_path, csv_path, num_mfcc = 13, hop_length = 512,
n_fft = 2048):

    # diccionario de datos
    data = {
        "mapping": [],
        "labels": [],
```

```
        "MFCCs": [],
        "files": []
    }

    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):

        if dirpath is not dataset_path:

            # actualización de la etiqueta
            category = dirpath.split(os.sep)[-1]  # Base_Datos/arriba ->
[Base_Datos, arriba]
            data["mapping"].append(category)
            print(f"processing {category}")

            # bucle para todos los comandos y extracción de los MFCCs
            for f in filenames:

                # se obtiene la ruta del archivo
                file_path = os.path.join(dirpath, f)

                # se carga el archivo de audio
                signal, sr = librosa.load(file_path)

                # se asegura que el audio este almenos en un segundo
                if len(signal) >= SAMPLES_TO_CONSIDER:


                    signal = signal[:SAMPLES_TO_CONSIDER]

                    # extracción de los MFCCs
                    MFCCs = librosa.feature.mfcc(signal, sr=sr, n_mfcc =
num_mfcc, hop_length = hop_length, n_fft = n_fft)
                    MFCCs = MFCCs[1:]

                    # se almacenan los datos
                    data["labels"].append(i-1)
                    data["MFCCs"].append(MFCCs.T.tolist())
                    data["files"].append(file_path)

    # se guardan los datos en un archivo csv
    with open(csv_path, "w", newline="") as file:
        fieldnames = ["MFCCs", "labels"]
        writer = csv.DictWriter(file, fieldnames=fieldnames)
        writer.writeheader()
        for i in range(len(data["files"])):
            writer.writerow({"MFCCs": data["MFCCs"][i], "labels":
data["labels"][i]})
```

```
if __name__ == "__main__":
    prepare_dataset(DATASET_PATH , CSV_PATH)
```

3. MODELO DE LA RED NEURONAL

```
'''
IMPLEMENTACIÓN DE UN SISTEMA DE PROCESAMIENTO DE LA VOZ HUMANA CON DEEP
LEARNING QUE PERMITA
                                EL RECONOCIMIENTO DE VOCABLOS

@autores: Andrés Felipe Camargo Caicedo y Luis Gustavo Moreno Camargo.
@universidad: UNIVERSIDAD DEL CAUCA.
@versión: 1.0, 01/05/2022
'''

import random
import csv
import json
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

DATA_PATH = "data.csv"
SAVED_MODEL_PATH = "model1.h5"
EPOCHS = 250
BATCH_SIZE = 40
LEARNING_RATE = 0.00001
NUM_COMANDS = 7

def load_data(data_path):

    # se extraen las caracteristicas y las etiquetas del archivo csv
    x = []
    y = np.array([])
    with open(data_path, 'r') as file:
        csv_file = csv.DictReader(file)
        csv_file = sorted(csv_file, key = lambda x: random.randint(0, 100))
        for row in csv_file:
            a = json.loads(row["MFCCs"])
            x.append(a)
            y = np.append(y, int(row["labels"]))

    x = np.array(x)

    return x, y
```

```
def prepare_dataset(data_path, test_size=0.2, validation_size=0.16):

    # se carga la base de datos
    x, y = load_data(data_path)

    # división en entrenamiento, validación y pruebas
    X_train, X_test, y_train, y_test = train_test_split(x, y,
test_size=test_size, random_state=15)
    X_train, X_validation, y_train, y_validation = train_test_split(X_train,
y_train, test_size=validation_size,
                                                                    random_sta
te=27)

    # normalización de los datos
    X_train = X_train/255.0
    X_test = X_test/255.0
    X_validation = X_validation/255.0

    return X_train, y_train, X_validation, y_validation, X_test, y_test


def build_model(input_shape, loss="sparse_categorical_crossentropy",
learning_rate=0.00001):

    # construcción de la arquitectura de la NN
    model = tf.keras.models.Sequential()

    # capa de entrada
    model.add(tf.keras.layers.LSTM(units=150, return_sequences=True,
input_shape=input_shape))
    model.add(tf.keras.layers.Dropout(0.2))

    # capa oculta
    model.add(tf.keras.layers.LSTM(units=100))
    model.add(tf.keras.layers.Dropout(0.2))

    # capa de salida
    model.add(tf.keras.layers.Dense(NUM_COMANDS, activation='softmax'))

    opt = tf.optimizers.Adam(learning_rate=learning_rate)

    # compilación del modelo
    model.compile(optimizer=opt,
                  loss=loss,
                  metrics=["accuracy"])

    # imprime los parametros del modelo
    model.summary()
```

```
    return model


def train(model, epochs, batch_size, X_train, y_train, X_validation,
y_validation):

    # entrenamiento del modelo
    history = model.fit(X_train, y_train, epochs=epochs,
                        batch_size=batch_size,
                        validation_data=(X_validation, y_validation))

    return history


def plot_history(history):

    fig, axs = plt.subplots(2)

    # graficación de las métricas
    axs[0].plot(history.history["accuracy"], label="train_accuracy")
    axs[0].plot(history.history['val_accuracy'], label="val_accuracy")
    axs[0].set_ylabel("Accuracy")
    axs[0].legend(loc="lower right")
    axs[0].set_title("Accuracy evaluation")
    axs[0].grid()

    axs[1].plot(history.history["loss"], label="train_loss")
    axs[1].plot(history.history['val_loss'], label="val_loss")
    axs[1].set_xlabel("Epoch")
    axs[1].set_ylabel("Loss")
    axs[1].legend(loc="upper right")
    axs[1].set_title("Loss evaluation")
    axs[1].grid()

    plt.show()


def main():
    # genera los sets de entrenamiento, validación y pruebas
    X_train, y_train, X_validation, y_validation, X_test, y_test =
prepare_dataset(DATA_PATH)

    # crea la red
    input_shape = (X_train.shape[1], X_train.shape[2])
    model = build_model(input_shape, learning_rate=LEARNING_RATE)

    # entrenamiento de la red
    history = train(model, EPOCHS, BATCH_SIZE, X_train, y_train, X_validation,
y_validation)
```

```
    # graficación de las métricas
    plot_history(history)

    # evaluación de la red
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print("\nTest loss: {}, test accuracy: {}".format(test_loss,
100*test_acc))

    # guarda el modelo
    model.save(SAVED_MODEL_PATH)



if __name__ == "__main__":
    main()
```

4. IMPLEMENTACIÓN DEL MODELO

```
'''
IMPLEMENTACIÓN DE UN SISTEMA DE PROCESAMIENTO DE LA VOZ HUMANA CON DEEP
LEARNING QUE PERMITA
                                EL RECONOCIMIENTO DE VOCABLOS

@autores: Andrés Felipe Camargo Caicedo y Luis Gustavo Moreno Camargo.
@universidad: UNIVERSIDAD DEL CAUCA.
@versión: 1.0, 01/05/2022
'''

import librosa
import tensorflow as tf
import numpy as np
import pyaudio, wave, librosa
from pydub import AudioSegment
import scipy.io.wavfile as waves
from time import time


SAVED_MODEL_PATH = "model.h5"
SAMPLES_TO_CONSIDER = 22050
TIME = 3 # Duración de la grabación
CHANNELS = 2
RATE = 44100
CHUNK = 1024

model = tf.keras.models.load_model(SAVED_MODEL_PATH)
mapping = [
    "abrir",
```

```
    "adelante",
    "atras",
    "cerrar",
    "derecha",
    "izquierda",
    "parar"
    ]

start_time = time()

def detect_silence(signal, threshold=-30.0, chunk_size=15):
    '''
    threshold en dB
    chunk_size en ms
    '''
    trim = 0 # ms

    assert chunk_size > 0 # evita loop infinito
    while signal[trim:trim+chunk_size].dBFS < threshold and trim <
len(signal):
        trim += chunk_size

    return trim


def recording(file_path):

    audio=pyaudio.PyAudio() #Se inicia PyAudio

    # Se definen características de audio
    stream=audio.open(format=pyaudio.paInt16,channels=CHANNELS,
                      rate=RATE,input=True,
                      frames_per_buffer=CHUNK)

    # Se inicia grabación
    print("Grabando ...")
    frames=[]
    for i in range(0,int(RATE/CHUNK*TIME)):
        data=stream.read(CHUNK)
        frames.append(data)

    # Finaliza la grabación
    print("Grabacion terminada ")
    stream.stop_stream()
    stream.close()
    audio.terminate()

    # Se guarda archivo con sus parámetros
    waveFile=wave.open(file_path,'w')
```

```
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(pyaudio.paInt16))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(frames))
    waveFile.close()

    # Se carga el archivo de audio
    sound = AudioSegment.from_file(file_path, format="wav")

    # Eliminación del silencio inicial
    start_trim = detect_silence(sound)
    duration = len(sound) # ms
    trimmed_sound = sound[start_trim - 100:duration]
    trimmed_sound.export(file_path, format="wav")

    # Se carga el archivo
    signal, sr = librosa.load(file_path)

    # Preénfasis
    sound_shift = np.roll(signal, 1) # corrimiento ciclico de una posición
    sound_shift[0] = 0 # el primer valor se pone en cero
    filtered_sound = signal - 0.97*sound_shift
    waves.write(file_path, sr, filtered_sound)


def preprocess(file_path, num_mfcc=13, n_fft=2048, hop_length=512):

    # carga el archivo de audio
    signal, sr = librosa.load(file_path)

    if len(signal) >= SAMPLES_TO_CONSIDER:

        filtered_sound = signal[:SAMPLES_TO_CONSIDER]

        # extracción de los MFCCs
        MFCCs = librosa.feature.mfcc(filtered_sound, sr, n_mfcc=num_mfcc,
n_fft=n_fft,
                                     hop_length=hop_length)
        MFCCs = MFCCs[1:]
    return MFCCs.T


def predict(model, file_path):

    MFCCs = preprocess(file_path)

    # se necesita una matriz de 3 dmensiones: (# muestras, # segmentos, #
coeficientes)
    MFCCs = MFCCs[np.newaxis, ...]
```

```
    # se obtiene la etiqueta a predecir
    predictions = model.predict(MFCCs)
    predicted_index = np.argmax(predictions)
    predicted_keyword = mapping[predicted_index]
    return predicted_keyword


if __name__ == "__main__":

    # hace la grabación del audio
    audio = recording("pruebas/audio.wav")

    # hace la predicción
    keyword = predict(model, "pruebas/audio.wav")
    print("\nPredicted word: {}\n".format(keyword))

    end_time = time()
    print("Runtime: ", end_time-start_time, "seg")
```

Apéndice C

OTROS RESULTADOS

Tabla 4.6. Predicción de las pruebas del locutor 4 (elaboración propia).

Locutor	Etiqueta	Predicción
4 (Hombre)	Abrir	Abrir
	Adelante	Adelante
	Atrás	Atrás
	Cerrar	Cerrar
	Derecha	Abrir
	Izquierda	Izquierda
	Parar	Atrás

Tabla 4.7. Predicción de las pruebas del locutor 5 (elaboración propia).

Locutor	Etiqueta	Predicción
5 (Mujer participe de la base de datos)	Abrir	Abrir
	Adelante	Adelante
	Atrás	Atrás
	Cerrar	Cerrar
	Derecha	Derecha
	Izquierda	Izquierda
	Parar	Parar

Apéndice D

ARTÍCULO DE INVESTIGACIÓN

IMPLEMENTACIÓN DE UN SISTEMA DE PROCESAMIENTO DE LA VOZ HUMANA CON DEEP LEARNING QUE PERMITA EL RECONOCIMIENTO DE VOCABLOS

Andrés Felipe Camargo Caicedo, Luis Gustavo Moreno Camargo
Universidad del Cauca
Popayán, Cauca
andrescamargo@unicauca.edu.com
luismoreno@unicauca.edu.com

Resumen. **Este articulo tiene como objetivo mostrar el proceso de diseño e implementación de un sistema clasificador basado en Redes Neuronales (NN, *Neural Networks*) para la transcripción de comandos de voz, con algoritmos de Aprendizaje profundo (DL, *Deep Learning*) capaz de reconocer vocablos específicos, para evaluar el desempeño del sistema implementado en el Reconocimiento Automático del Habla (ASR, *Automatic Speech Recognition*), enfocado a personas con discapacidad motora en miembros superiores para permitirles interactuar con sistemas que normalmente son accionados mediante teclados o botones. Para su diseño primero se construye una base de datos propia de la pronunciación de 7 vocablos en español, grabados en ambientes libres de ruido, se realiza el pre-procesado de las señales de audio y se extraen los atributos de interés, en este caso, los Coeficientes Cepstrales en las Frecuencias de MEL (MFCC, *Mel Frequency Cepstral Coefficients*), los cuales serán la entrada del algoritmo de DL. Una vez entrenado el sistema y aplicadas las métricas de evaluación se analizan los resultados y se realizan las pruebas correspondientes para evaluar su desempeño.**

Palabras claves. **Aprendizaje profundo, Comandos específicos, MFCCs, Reconocimiento automático del habla, Redes neuronales.**

I. INTRODUCCIÓN

La revolución tecnológica alrededor del mundo gracias a las nuevas tecnologías emergentes, como son las aplicaciones de Inteligencia Artificial (AI, *Artificial Intelligence*) y algoritmos de DL, han tenido un impacto importante gracias a la posibilidad de abordar problemas complejos y tener herramientas lo suficientemente robustas como para resolverlos.

Asistentes virtuales de voz como Alexa, Siri, Cortana, Bixby, Google Assistant, demuestran que las tecnologías de ASR se han integrado de una manera rápida y progresiva en ámbitos sociales como el familiar, laboral y también en el ámbito de la salud. Estos asistentes son herramientas muy atractivas, ya que, la posibilidad de interacción a través del habla da un paso hacia la comodidad, dinamización y cumplimiento de funciones específicas en la velocidad y/o inmediatez que requieren las nuevas tecnologías [1].

El uso de las Tecnologías de la Información y las Comunicaciones (TICs) representa una posibilidad para la inclusión y normalización de las personas con discapacidad, ya sea en un entorno de casa, educativo y/o laboral, dado que pueden contribuir al mejoramiento de la calidad de vida, fortaleciendo su independencia e individualidad [2].

Se propone el uso de una red neuronal basada en técnicas de DL, mediante aprendizaje supervisado, la cual tenga la capacidad de hacer reconocimiento de comandos específicos de voz, esto con el enfoque de aplicación en sistemas de ayuda a personas con discapacidad motora en miembros superiores, para poder ejecutar tareas mediante su voz en sistemas que normalmente requieren pulsación de teclas y/o accionadores táctiles.

Por lo anterior, en este artículo se analiza el desempeño de un modelo de DL para el reconocimiento de vocablos, para ello se realizó primero un trabajo de campo que consistió en realizar las grabaciones de vocablos específicos pronunciados por diferentes personas; esto, con el objetivo de construir la base de datos para el algoritmo propuesto, seguidamente se desarrolló en la herramienta software Python una etapa

de pre-procesamiento a las grabaciones, eliminando el silencio y un filtro de preénfasis para suavizar el espectro. Posteriormente se extraen los Coeficientes Cepstrales en las Frecuencias de MEL (MFCC, *Mel Frequency Cepstral Coefficients*). Una vez construida la base de datos se determinan los porcentajes respectivos de los datos para entrenamiento, validación y pruebas. Finalmente se entrena el algoritmo de reconocimiento de vocablos y se evalúa su desempeño a partir de métricas de evaluación.

II. OBJETIVOS Y METODOLOGÍA

El objetivo principal de este proyecto es implementar un sistema de procesamiento de la voz humana con Deep Learning que permita el reconocimiento de vocablos.

Los objetivos específicos son:

1. Identificar las características fundamentales de las señales de audio correspondientes a los vocablos a reconocer.
2. Diseñar un algoritmo basado en técnicas Deep Learning para el reconocimiento de vocablos específicos.
3. Evaluar el desempeño del sistema implementado a partir del análisis de su función objetivo.

Para el desarrollo de este proyecto se sigue la metodología lineal, la cual sugiere un enfoque sistemático y secuencial. Dicha metodología se basa en las siguientes fases: Recolección de la información, Análisis y síntesis de la información, Diseño e implementación, Evaluación y entrega de resultados [3].

III. DISEÑO DEL SISTEMA

Para el diseño de este sistema se optó por dividirlo en fases y abordar una a una de la forma más eficiente. Estas fases se exponen en los siguientes sub-apartados.

A. Base de datos

En el diseño de un modelo de Aprendizaje Automático (ML, *Machine Learning*) o DL es fundamental construir o elegir correctamente la base de datos a utilizar, ya que, esta influye directamente en los resultados obtenidos. En este caso se tomó la decisión de construir una base de datos desde cero, es decir, una base de datos propia.

Para el desarrollo de un modelo de DL es necesario contar con una cantidad de datos lo suficientemente grande para poder obtener resultados de precisión satisfactorios.

Con la ayuda de locutores se realizan grabaciones de 7 vocablos, donde estos serán los comandos a reconocer por parte de la NN, los vocablos a trabajar son:

- Abrir
- Adelante
- Atrás
- Cerrar
- Derecha
- Izquierda
- Parar

La base de datos cuenta con un total de 2800 ficheros de audio, todos en formato WAVE (.wav). Se creó un código, con el cual se grabó y se eliminó el silencio inmediatamente el locutor realizo la grabación el vocablo. Para la creación de la base de datos se optó por una duración de grabación de 3 segundos, las grabaciones se realizaron a una frecuencia de 44100 Hz a dos canales. Se tomó esta frecuencia con el fin de tener una mejor calidad en la grabación del audio y se optó por dos canales debido a que las grabaciones se realizan tanto con el código como en dispositivos celulares y estos últimos realizan las grabaciones a dos canales.

Luego, se realizó eliminación de silencios, esto con el fin de eliminar tiempos sin información, para que al momento de extraer las características del vocablo no se obtengan datos que no aporten al sistema. Al momento de eliminar el silencio inicial, se opta por dejar un margen o “cola” de las primeras muestras de audio (100 muestras) como se muestra en la Figura 1, esto con el fin de prevenir cortes en la pronunciación de la primera sílaba de cada palabra (no tienen la misma fuerza de acentuación o realce) dando seguridad de que se toma la totalidad del vocablo.

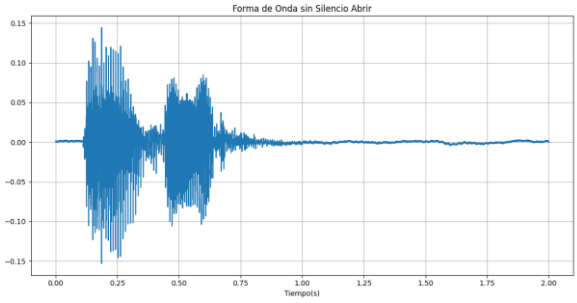

Fig. 1. Forma de onda del vocablo abrir sin el silencio inicial (elaboración propia).

B. Pre-procesamiento y extracción de los MFCCs

Una vez lista la base de datos, se tomaron los ficheros de audio, estos se procesaron y se extrajeron las características MFCCs que componen a cada

fichero de audio. Para ello, se realizaron las siguientes operaciones:

1) *Pre-énfasis:* Se usó con el objetivo de amplificar las frecuencias altas y compensar la atenuación de aproximadamente 20 dB/década que se produce en la señal al pasar a través del tracto vocal [4]. Siendo alpha el coeficiente del filtro, en este caso se toma un α = 0,97 tal como se muestra en la ecuación 1,

$$y(n) = x(n) - \alpha\, x(n-1). \quad (1)$$

2) *Enmarcado y ventana Hamming:* Teniendo en cuenta que las frecuencias de una señal van variando con el paso del tiempo, se puede decir que la señal de audio no es periódica, por lo tanto, fue necesario realizar un proceso de segmentación en las señales de audio, asumiendo que el porcentaje de la señal contenido en cada uno de los segmentos representa que las frecuencias son estacionarias, es decir que en cortos periodos de tiempo la señal es periódica. Para este caso se tomaron marcos de 2048 muestras y un solapamiento de 512 muestras. Luego de esto se aplicó la ventana Hamming con el objetivo de hacer un alisado sobre la señal, es decir, realizar una disminución en la amplitud de las discontinuidades producidas por el solapamiento entre marcos, como se muestra en la Figura 2 [5, 6].

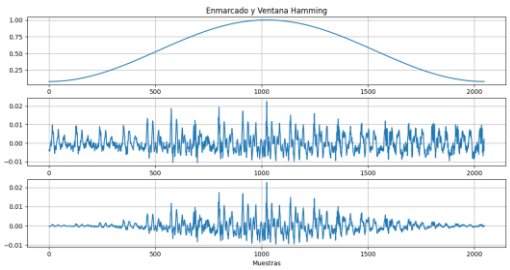

Fig. 2. Aplicación de ventana Hamming a un marco (elaboración propia).

3) *Transformada de Fourier:* La transformada de Fourier suma una lista de N valores sobre cada marco para calcular el espectro en frecuencia, esto se puede observar en la ecuación 2, donde x_n es la n-ésima muestra de la señal x, es decir, se suman los valores resultantes de la FFT de cada marco para así obtener las componentes en frecuencia de la totalidad del fichero [6]

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}}, k = 0, \dots, N-1, \quad (2)$$

donde:

- ***N*** es el número total de muestras
- ***n*** es la muestra actual, $n \in \{0 \dots N-1\}$
- ***k*** frecuencia actual, *k* de *0Hz* a *N – 1Hz*
- $\boldsymbol{x_k}$ cantidad de la frecuencia *k* en la señal, es un numero complejo.
- $\boldsymbol{i} = \sqrt{-1}$

4) *Banco de filtros en escala Mel:* Con el objetivo de extraer bandas de frecuencia se usan filtros triangulares en la escala de MEL (por lo general 40), así se emula la forma no lineal de percepción del oído humano, siendo más discriminatorio con las frecuencias bajas y menos con las altas, es decir, para las frecuencias bajas los filtros abarcan menos cantidad de frecuencias, pero con una mayor amplitud, y en las frecuencias altas se tiene filtros con mayor rango de frecuencias, pero con una amplitud más baja, como se visualiza en la Figura 3 [7].

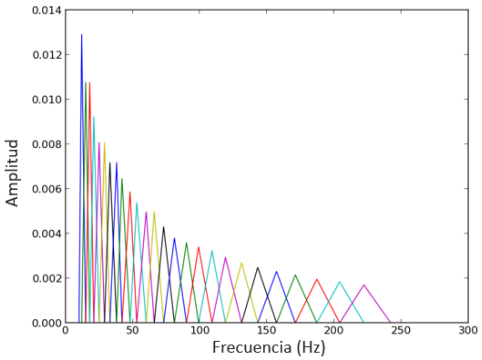

Fig. 3. Banco de filtros en escala de MEL. Tomado de [5]

5) *MFCCs:* Mediante los MFCC se identifican las componentes de la señal de audio pertinentes para determinar el contenido lingüístico y descartar todo lo demás. Estos coeficientes se hallan a partir del banco de filtros, aplicando la Transformada Discreta del Coseno (DCT, *Discrete Cosine Transform*) como se expresa en la ecuación 3, con el fin de comprimir dichos bancos y poder eliminar la correlación entre ellos y así obtener los MFCCs, donde se toman únicamente del 2 al 13, descartando el resto ya que solo representan cambios rápidos en los coeficientes del banco de filtros y estos datos no aportan nada para el reconocimiento de voz. En la Figura 4 se pueden visualizar los MFCCs de una señal [5, 6].

$$X_k = \sum_{n=1}^{N} x_n \cos\left(\frac{\pi}{N} k \left(n - \frac{1}{2}\right)\right), \quad (3)$$

donde:

- X_k representa el vector de coeficientes por cada ventana
- n representa el número de filtros en el banco de filtros MEL
- N representa el número total de coeficientes MFCC
- k representa la ventana actual sobre la que se está trabajando

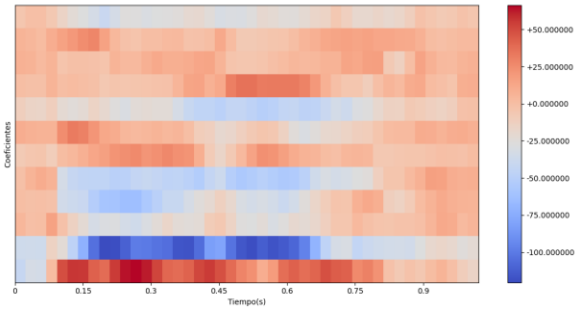

Fig. 4. Representación de los MFCCs obtenidos de una señal ejemplo (elaboración propia).

C. *División de los datos*

En esta etapa se dividió la base de datos en tres subconjuntos: entrenamiento, validación y pruebas. El conjunto de entrenamiento es el que contiene los datos con los cuales el modelo aprendió las características e identificó los patrones en los datos. El conjunto de validación permitió identificar si el modelo está aprendiendo de una manera correcta y no se está teniendo un sobreajuste (overfitting) durante el entrenamiento. El conjunto de pruebas o test permitió comprobar el desempeño del modelo final a partir de las métricas de evaluación. El porcentaje de separación por el cual se optó fue un 64% para los datos de entrenamiento, 16% para los datos de validación y 20% para los datos de prueba.

D. *Arquitectura de la NN*

En la creación de la arquitectura de la NN se implementaron dos capas de Memoria a Corto y Largo Plazo (LSTM, Long Short - Term Memory), estas celdas son un tipo de RNN, las cuales tienen la capacidad de memoria a corto y largo plazo, diseñadas con la finalidad específica de poder recordar información relevante durante varios periodos de tiempo [8]. Siendo así, una la capa de entrada con 150 neuronas y la otra una capa oculta con 100 neuronas. En las dos capas se utiliza la función de activación tanh, la cual es la función por defecto del planteamiento original de las celdas LSTM, además de aplicarse sobre las dos capas el método de regularización Dropout al 20%. La capa de salida es una densamente conectada de 7 neuronas, una neurona por cada posible clase (vocablo) a reconocer, esta con función de activación Softmax. La arquitectura de la NN se observa en la Figura 5.

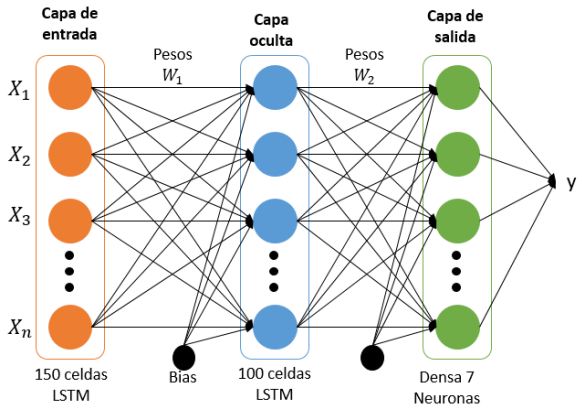

Fig. 5. Arquitectura de la NN (elaboración propia).

E. *Entrenamiento de la NN*

Una vez pre-procesada y estructurada la base de datos, los datos están listos para ser introducidos a la capa de entrada de la NN diseñada. El objetivo de entrenar el modelo es encontrar de manera iterativa los valores de los pesos (W) y *bias* (b) más adecuados para la entrada de cada una de las neuronas de la red, con el fin de que la respuesta de la capa de salida se ajuste lo más posible a los datos esperados. Para esto se hizo uso del algoritmo de optimización *Adam*, el cual tiene una mejor velocidad de convergencia en la búsqueda de una solución para W y b que otros algoritmos de optimización.

F. *Métricas de evaluación*

Las métricas de evaluación tienen como objetivo determinar el desempeño del sistema a estudiar, en este caso, la capacidad de predicción. Para esto, se hace uso de una tabla de precisión (*accuracy)* y pérdida, las cuales dan noción del porcentaje de aciertos del sistema en cuanto al total de las predicciones y en cuanto al margen de error.

- *Accuracy:* La precisión es un tipo de métrica qué representa el porcentaje de muestras correctamente clasificadas, dada por la proporción del número de predicciones correctas entre el número total de predicciones como se observa en la ecuación 4 [9].

$$Precisión = \frac{Predicciones\ correctas}{Total\ de\ predicciones}. \qquad (4)$$

- *Pérdida:* La Pérdida es un tipo de métrica qué se relaciona con la probabilidad de predicción de

una entrada, la cual resulta adecuada, dado que al usar la función de activación Softmax en la capa de salida del modelo, ésta da como resultado las probabilidades de las clases a ser predichas. En esta métrica, mientras la probabilidad predicha sea más cercana a cero, la pérdida será mucho mayor, por lo tanto, la predicción realizada será diferente de la etiqueta real. Esta métrica lo que hace es aplicar un menos logaritmo en base e o también conocido como el logaritmo natural a la probabilidad *p(x)* predicha por la capa de salida, esto se expresa matemáticamente en la ecuación 5 [9]

$$Loss = -\log_e\big(p(x)\big) = -\ln\big(p(x)\big)\ [nats] \quad (5)$$

donde *nat* es la unidad natural de información, se utiliza para medir la información o la entropía, es decir, mide la cantidad de información de una distribución de probabilidad o evento [10].

IV. EXPERIMENTACIÓN

A. *Modelo inicial*

Haciendo uso de la librería *Keras* de *TensorFlow* en Python se implementó el modelo de la NN, el cual se definió con 3 capas: la capa de entrada (150 neuronas), la capa oculta (100 neuronas) de celdas LSTM y la capa de salida (capa densa con 7 neuronas). Se aplicó un Dropout de 0.2 el cual previene el sobreajuste en la etapa de entrenamiento de la red.

Mediante el algoritmo de optimización *Adam* se realizó la compilación del modelo. Luego de esto, se entrenó y se validó el modelo, inyectando los datos obtenidos de la fase de procesamiento y extracción de los MFCCs del audio. Una vez entrenado el modelo se utiliza el conjunto de datos destinado como pruebas para evaluar el desempeño obtenido con el modelo, a partir de las métricas de precisión y perdida.

Tabla I
Resultados del modelo

Métrica	Entrenamiento	Validación	Pruebas
Precisión (acc)	98.67%	97.86%	98.09%
Pérdida (loss)	0.0534	0.0607	0.051

Como se observa en la tabla I se obtienen resultados óptimos en cuanto a las métricas de evaluación del modelo, con un porcentaje de precisión de 98.67% en el entrenamiento de la red teniendo una convergencia alta, concluyendo que el modelo es capaz de aprender y puede ser tomado como una solución válida para el proyecto.

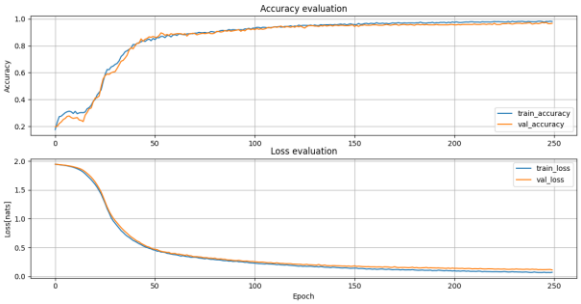

Fig. 6. Métricas de evaluación del modelo (elaboración propia).

De igual manera en la Figura 6 se evidencia que a medida que van transcurriendo las épocas se van alcanzando valores óptimos en cuanto a la precisión de predicción en la red y en la pérdida se tienen valores muy cercanos a cero, además de notar que no se tiene sobreajuste, dado que los valores para entrenamiento y validación van mejorando de una manera progresiva, paralela y sin cambio abruptos, convergiendo a un mismo punto al final del entrenamiento.

B. *Modelo con uso de preénfasis en la base de datos*

Para este segundo caso, se evaluó el cambio en el desempeño del modelo de red, mediante la aplicación de un filtrado de preénfasis a los archivos de la base de datos.

Tabla II
Comparación de resultados de los modelos

Modelo	Métrica	Entrenamiento	Validación	Test
Sin filtro	Precisión	98.67%	97.86%	98.09%
	Pérdida	0.0534	0.0607	0.051
Con filtro	Precisión	98.92%	98.34%	98.72%
	Pérdida	00.0369	0.0601	0.050

En la tabla II se visualiza la comparación entre los resultados de ambos modelos, notando que en el segundo modelo, es decir, al cual se le aplicó el filtro de preénfasis se obtienen mejores resultados en cuanto a las métricas de evaluación y se puede observar que los tres resultados, tanto el entrenamiento, validación y test se encuentran más cercanos entre sí con porcentajes ligeramente superiores a los del modelo sin filtro de preénfasis, el cual se toma como un resultado esperado, dado que el filtro de preénfasis permitió equilibrar el espectro de frecuencias, al ampliar las frecuencias altas y compensar la pendiente natural de la voz antes del

análisis espectral, obteniendo valores para los MFCCs más precisos.

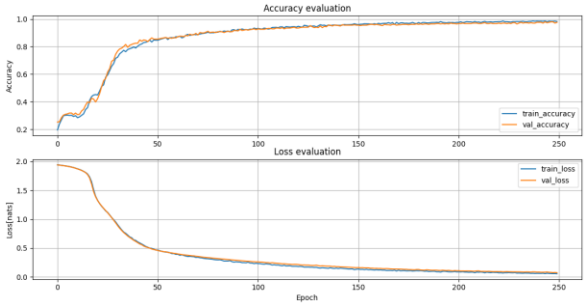

Fig. 7. Métricas de evaluación del modelo con aplicación de preénfasis (elaboración propia).

De igual manera que para el modelo anterior se observa en la Figura 7 que no se presentó sobreajuste, al ver que los valores para entrenamiento y validación al final de la ejecución convergen a un mismo punto tanto para la precisión como para la pérdida. Estos van mejorando en paralelo y progresivamente, a medida que aumentan las épocas.

Por lo anterior se concluyó en utilizar el modelo al cual se le aplicó el filtro de preénfasis a la base de datos para realizar las correspondientes pruebas de predicción del modelo.

V. PRUEBAS

Una vez culminado el diseño y entrenamiento del modelo de predicción, se procedió a finalizar el desarrollo del sistema con una fase de pruebas, donde se midió el desempeño del sistema completo.

Durante el proceso de pruebas del modelo se presentaron ciertos inconvenientes. Al realizar las pruebas con algún locutor, se evidenció qué el sistema solo logra predecir dos o tres de los 7 vocablos, a pesar de tener un porcentaje superior al 98% durante la fase de entrenamiento y no verse afectado por sobreajuste, esto es debido a que con el uso de la función *train_test_split*, la encargada de realizar la separación de los datos en entrenamiento, validación y pruebas, dicha función de Python realiza la separación de una manera aleatoria, lo cual se presta para que se seleccionen más características de ciertos vocablos y menos de otros vocablos o incluso ninguna característica, por ende el sistema no predecía correctamente todos los vocablos, dado que no se tenía un equilibrio en la distribución de los datos. Para dar solución a este inconveniente lo qué se hace es que sobre dicha función se usó de la variable *random_state* qué está incluida en la función de Python *train_test_split* para garantizar una distribución aleatoria equilibrada de los datos, y también se utilizó un valor de *learning_rate* más bajo y se incrementó el valor de las épocas para garantizar que el sistema tuviera un mayor aprendizaje, esto conllevó a qué el proceso de entrenamiento fuera un poco más demorado al tener más épocas.

A continuación, en las tablas III y IV se muestran las predicciones resultantes de las pruebas realizadas.

Tabla III
Predicción de las pruebas del locutor 1

Locutor	Etiqueta	Predicción
1 (Hombre)	Abrir	Abrir
	Adelante	Adelante
	Atrás	Atrás
	Cerrar	Cerrar
	Derecha	Abrir
	Izquierda	Izquierda
	Parar	Atrás

Tabla IV
Predicción de las pruebas del locutor 2

Locutor	Etiqueta	Predicción
2 (Mujer participe de la base de datos)	Abrir	Abrir
	Adelante	Adelante
	Atrás	Atrás
	Cerrar	Cerrar
	Derecha	Derecha
	Izquierda	Izquierda
	Parar	Parar

La ejecución de las pruebas concluyó con 2 ficheros mal clasificados, debido a que las variaciones de los resultados dependen de la capacidad vocal y la gesticulación del locutor, también podría deberse a pronunciaciones con poca representación en la base de datos, además de que si el locutor tuvo participación en

la construcción de la base de datos o es un tercero cuya voz es ajena al sistema.

VI. CONCLUSIONES

La experiencia demuestra que del tamaño y calidad de la base de datos depende en gran parte el desempeño de un algoritmo de aprendizaje *Deep Learning*, por lo que recurrir a bases de datos ya construidas es una forma de investigar; sin embargo, en el presente trabajo, y con ánimo de personalizar la aplicación del sistema propuesto al reconocimiento de vocablos específicos, se tuvo que crear la base de datos desde cero, lo cual implicó un trabajo de campo de recolección de grabaciones y posterior pre-procesamiento, con lo que se pudo consolidar nuestro conocimiento en el procesamiento digital de señales de audio a partir de datos crudos.

Para la construcción de las bases de datos a partir de los datos pre-procesados, se optó por la extracción de los coeficientes Cepstrales MFCC a cada registro de audio de los vocablos, siendo estos sus características más representativas y adecuadas para su discriminación al pasar como datos de entrada del algoritmo de aprendizaje *Deep Learning*. Para esto, fue necesario considerar la misma cantidad de muestras para todas las grabaciones, con el fin de garantizar las mismas dimensiones en el vector de coeficientes de cada fichero de audio, esto para evitar inconvenientes en el momento de almacenar estos datos en el archivo tipo csv de la base de datos.

En la literatura científica, se encuentra que entre más profunda fuera la Red Neuronal (capas ocultas), mejor sería el desempeño del sistema, siempre y cuando la base de datos sea lo suficientemente grande; sin embargo, dado el tamaño limitado de la base de datos construida para el presente trabajo, se alcanzaron unos buenos resultados con una red de aprendizaje *Deep Learning* de una capa de entrada, una de salida y una única capa oculta.

La selección y aplicación de las métricas de evaluación de un algoritmo se hacen dependiendo del propósito para el cual éste fue creado. En el presente trabajo, a partir de las métricas de precisión (función de la tasa de aciertos) y pérdidas (función de probabilidad de predicción), se comprobó que un sistema de procesamiento de la voz implementado en *Deep Learning* es efectivo para el reconocimiento de vocablos personalizados, con una precisión de aproximadamente 98% y unas pérdidas del 5%, a pesar del volumen bajo de la base de datos construida.

El aplicar un filtro de preénfasis a los archivos de audio puede mejorar el desempeño del algoritmo *Deep Learning* propuesto, subiendo del 98% a 98,7% a costa de algún tiempo extra de procesamiento, lo que obliga a sopesar su conveniencia o no.

VII. REFERENCIAS

[1] E. Connect, "Asistentes virtuales de voz: una “revolución silenciosa” en el ámbito de la salud", *Elsevier Connect*, 2020.

[2] R. Miranda, *Ciapat.org*, 2007. [Online]. Available: http://ciapat.org/biblioteca/pdf/673Discapacidad_y_Accesibilidad.pdf.

[3] I. Rodríguez, M. Salamanca and J. Márquez, *Utpuebla.edu.mx*.

[4] I. Martín, Repositorio.uam.es, 2019. [Online]. Available: https://repositorio.uam.es/bitstream/handle/10486/688634/martin_calle_irene_tfm.pdf?sequence=1&isAllowed= y.

[5] I. Ruiz, "MFCCs", Turing.iimas.unam.mx. [Online]. Available: https://turing.iimas.unam.mx/~ivanvladimir/posts/mfcc/.

[6] A. Arias and D. Rubiano, "MÉTODO AUTOMÁTICO DE RECONOCIMIENTO DE VOZ PARA LA CLASIFICACIÓN DE VOCALES AL LENGUAJE DE SEÑAS COLOMBIANO.", Repository.ucatolica.edu.co, 2021. [Online]. Available: https://repository.ucatolica.edu.co/bitstream/10983/22809/1/M%C3%89TODO%20AUTOM%C3%81TICO%20DE%20RECONOCIMIENTO%20DE%20VOZ%20PARA%20LA%20CLASIFICACI%C3%93N%20DE%20VOCALES%20AL%20LENGUAJE%20DE%20SE%C3%91AS%20COLOMBIANO.pdf.

[7] Vicente, "Sistema de Reconocimiento de Comandos por Voz Basado en Redes Neuronales LSTM", Core.ac.uk, 2018. [Online]. Available: https://core.ac.uk/download/pdf/159466484.pdf.

[8] A. Mañas, "Capítulo 8 Métodos basados en Deep Learning | Notas sobre pronóstico del flujo de tráfico en la ciudad de Madrid", Bookdown.org. [Online]. Available: https://bookdown.org/amanas/traficomadrid/m%C3%A9todos-basados-en-deep-learning.html.

[9] "Métricas De Evaluación De Modelos En El Aprendizaje Automático", *DataSource.ai*, 2021. [Online]. Available: https://www.datasource.ai/es/data-science-articles/metricas-de-evaluacion-de-modelos-en-el-aprendizaje-automatico.

[10] Z. Global, "Brutalk - Una suave introducción a las métricas de probabilidad para la clasificación desequilibrada", *Brutalk*. [Online]. Available: https://www.brutalk.com/en/news/brutalk-blog/view/una-suave-introduccion-a-las-metricas-de-probabilidad-para-la-clasificacion-desequilibrada-6046dad9ac5de.