Proyecto Fin de Máster

Máster en Ingeniería Electrónica, Robótica y Automática

Reconocimiento de Emociones a través del Habla

Autor: Juan Salinas Hernández

Tutor: Rubén Martín Clemente

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2024

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Reconocimiento de Emociones a través del Habla

Autor:
Juan Salinas Hernández

Tutor:
Rubén Martín Clemente
Catedrático de Universidad

Dpto. de Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Autor: Juan Salinas Hernández

Tutor: Rubén Martín Clemente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

A mi familia, a mis amigos y a mis compañeros de clase, todos habéis formado parte de este camino, ofreciéndome siempre todo lo que ha estado en vuestras manos para que siga creciendo personal y profesionalmente.

Especialmente a Alejandra, mi incansable compañera, por ser el pilar fundamental en mi vida estos últimos años, sin el que estoy seguro de que no habría llegado hasta donde estoy ahora mismo.

Finalmente, a mis docentes, por no solo enseñarme de la mejor forma posible, sino por sembrar en mi el afán de querer seguir aprendiendo, comprender el mundo que nos rodea y seguir ingeniando para construir el futuro de nuestra sociedad. Especialmente a Rubén, por confiar una vez más en mí y creer en mis capacidades.

Juan Salinas Hernández

Sevilla, 2024

Resumen

En este proyecto se tratará de estudiar la eficacia de los modelos de *deep learning* para la tarea de clasificación de emociones en el habla, comparándola con otro tipo de soluciones de *machine learning.* Esta es un área de especial interés dentro de la ingeniería, por ser una potencial herramienta con gran cantidad de aplicaciones tanto en la industria como a nivel científico. El objetivo será analizar las prestaciones de los diferentes modelos planteados, persiguiendo que tengan la mayor precisión posible y siendo capaces de generalizar para datos con los que no ha sido entrenado.

Abstract

In this project we will try to study the effectiveness of deep learning models for the task of emotion classification in speech, comparing it with other types of machine learning solutions. This is an area of special interest within engineering, as it is a potential tool with many applications both in industry and at the scientific level. The objective will be to analyze the performance of the different models proposed, pursuing the highest possible accuracy and being able to generalize to data with which it has not been trained.

Índice

Índice de Tablas

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

En este capítulo introduciremos el proyecto de fin de máster, estableciendo los objetivos que se pretender alcanzar con el mismo, las diferentes herramientas con las que se trabajarán y abordando el estado del arte actual de este campo de estudio, así como el impacto que podría tener una vez implementado en un caso real.

1.1 Introducción

En la última década, los avances en inteligencia artificial, más concretamente en la rama de *deep learning*, han sido vertiginosos, cambiando el paradigma de una gran cantidad de ramas de estudios totalmente diferentes entre ellas, consiguiendo unos resultados notoriamente mejores que los obtenidos con las técnicas anteriormente usadas. Esta tecnología ha revolucionado ámbitos tan diferentes como la medicina con la realización de diagnósticos automáticos, la biología ayudando al conocimiento de las estructuras de las proteínas con *AlphaFold*, a la visión por computador pudiendo distinguir y comprender el entorno que nos rodea de forma automática con una imagen o video, o a la comprensión natural del lenguaje siendo capaz de entender un texto y responder coherentemente al mismo. Todos estos han sido casos disruptivos y sucedidos en un breve espacio de tiempo, lo que nos hace pensar que estamos en la punta del iceberg de los avances que esta tecnología puede traer, y, sobre todo, de cómo puede revolucionar el mundo estos avances llevados a aplicaciones útiles para el ser humano.

Es por ello, que se ha decido realizar una investigación del funcionamiento de estas técnicas que comentamos, aplicados en este caso al reconocimiento de emociones en el habla. El habla es la forma natural y más extendida del ser humano para comunicarse y expresar sus emociones. Dentro del habla, podremos encontrar información tanto lingüística, que contiene el contexto y el lenguaje de lo que estamos diciendo, como la información paralingüística, que nos ofrecería información acerca de las emociones, el género, la edad, etc. En esta aplicación, se tratará de identificar las emociones en el habla de una persona tan solo con el tratamiento de la señal del audio, es decir obtendremos la información paralingüística de la emoción. Esto supone un gran reto, principalmente a la hora de extraer qué características del audio son las más relevantes para expresar emociones, ya que estas pueden ser muy variables según el individuo, ya que se verá afectado por el género, por su estilo de hablar y por su trasfondo cultural.

Poder realizar un análisis de este tipo de forma automatizada supondría un avance importante para los nuevos desarrollos de tecnología, ya que las emociones suponen históricamente uno de los mayores hándicaps para las máquinas, puesto que no están totalmente definidos como los expresamos y son difíciles de parametrizar, y, por ende, difíciles de implementar en un algoritmo informático. Probablemente, las emociones sean una de las características más definitorias y propias de los seres vivos, por lo que dotar a las máquinas de la correcta compresión y expresión de estas, puede ayudar mucho a la introducción de la tecnología en sociedad de forma más natural, simplificando el proceso de adaptación para comunicarnos con ellas.

Las aplicaciones a las que se podrían llevar estos métodos pueden ser varias. Entre ellas destacan sus implementaciones en las interfaces de comunicación entre hombre-máquina, hombre-ordenador u hombre-robot. También en el sector de la psicología puede ser clave a la hora de la detección de la depresión u otros trastornos mentales. También en el sector comercial puede suponer una herramienta útil, aplicado por ejemplo en un *call center*, donde el conocimiento de las emociones de la persona con la que contactes de pueda guiar en tus estrategias de ventas.

1.2 Análisis del audio

Una de las partes claves de este proyecto será el análisis del audio. Para ello, se utilizarán técnicas comúnmente usadas en procesamiento digital de la señal, tratando las ondas sonoras tanto en el dominio del tiempo como en el dominio de la frecuencia, con el objetivo de caracterizar la voz humana de un audio de la mejor forma posible,

de forma que podamos estudiar la influencia de estas características sobre la expresión de emociones. Para ello no solo será importante encontrar estas características, sino conseguir la información de la voz abstrayéndonos de ruidos en la señal u otras informaciones acústicas en el audio cuya fuente no sea una voz humana, para lo que se pretenderá filtrar y enfatizar las frecuencias típicas de la voz. Se dedicará en el presente proyecto un capítulo entero a explicar las diferentes características de la voz utilizadas, y posteriormente en los resultados y conclusiones, se tratará acerca de la influencia de estas características sobre las emociones.

1.3 Aprendizaje automático y aprendizaje profundo

Ya se ha comentado acerca de la importancia que está teniendo las técnicas de aprendizaje automático en los últimos tiempos, hasta el punto en que un gran porcentaje de los nuevos desarrollos tecnológicos vienen acompañados de la palabra inteligencia artificial. Y aunque es cierto que esta llamada inteligencia artificial es aplicada cada vez en más soluciones en el mercado, obteniendo grandes resultados en la mayoría de los casos, conviene aclarar qué son estas técnicas, qué implican, por qué han proliferado tanto en los últimos años y cuáles son sus limitaciones.

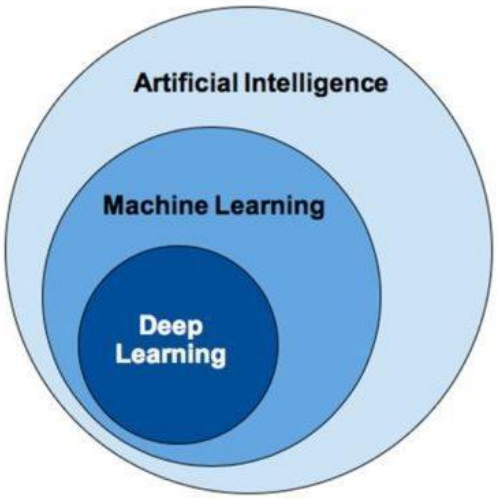


Ilustración 1 Categorización inteligencia artificial [1]

Cuando hablamos de inteligencia artificial, aprendizaje automático y aprendizaje profundo, se tiende a mezclar los conceptos y a usar uno u otro indistintamente, por lo que aclaremos primero qué significa cada uno de estos conceptos:

- Inteligencia artificial: es la rama más general de este campo, en la que se contemplan todas aquellas técnicas en las que mediante algoritmos informáticos se pretende emular un comportamiento lógico.
- Aprendizaje automático: también se engloba dentro de inteligencia artificial, pero como un concepto más específico. En este caso, ese comportamiento lógico no ha desarrollado por un desarrollador, sino que ha sido aprendido automáticamente por un algoritmo a partir de una serie de datos. Este puede ser supervisado, en el caso de que el resultado del modelo esté definido por los datos previamente etiquetados, o no supervisado, en el que el modelo aprende el resultado que debe obtener buscando patrones o relaciones en los datos, que no necesitan estar etiquetados en este caso.
- Aprendizaje profundo: es un tipo de aprendizaje en el que el algoritmo procesa directamente los datos en crudo y no las características (o features) que le proporciona un ser humano. El algoritmo selecciona de forma autónoma aquellas características de los datos que considera más apropiadas.

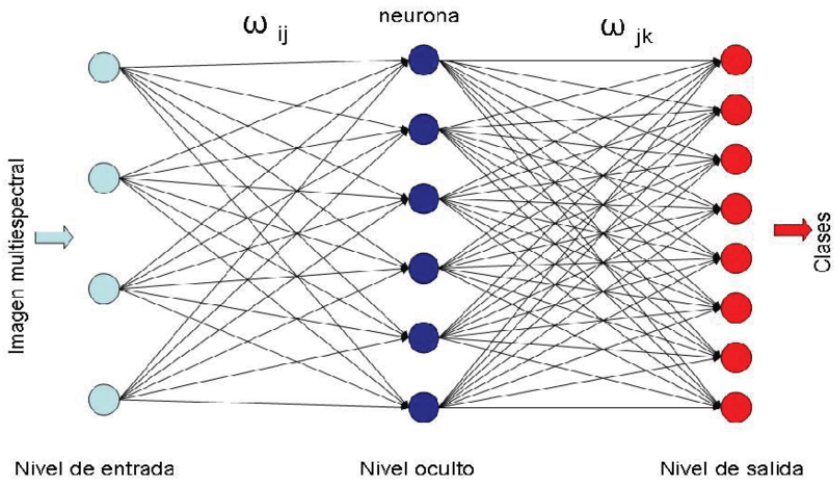


Ilustración 2 Estructura red neuronal

Concretamente, el aprendizaje automático y profundo son de las ramas de la inteligencia artificial que más han avanzado en la última década, cuando en 2012 irrumpió por primera vez un modelo de aprendizaje profundo, llamado *AlexNet*, en la competición de clasificación de imágenes *ImageNet*. Con este modelo, se consiguió reducir el error en la clasificación de imágenes en hasta un 17%, avance considerable respecto a los modelos presentado previamente, y que supuso un antes y un después en la visión por computador. A partir de entonces, se comenzó a implementar este tipo de modelos no sólo en el ámbito de la visión artificial, sino también en audios, texto, problemas de regresión y clasificación, y un largo etcétera.

A pesar de que estos modelos no se inventaron tan recientemente, cierto es que no es hasta este momento que han empezado a reflejar resultados realmente impresionantes. Esto es debido a que estos modelos necesitan realizar gran cantidad de cálculos matemáticos complejos, tanto para su inferencia como sobre todo para su entrenamiento, por lo que no fueron tan útiles hasta que no llegaron mejoras en el hardware de las computadoras, mejorando la velocidad y capacidad de la memoria, así como el procesamiento de las CPU, y posteriormente las GPU. Otro de los puntos clave que ayudó al buen rendimiento de estos modelos fue el uso extendido de internet por todos los usuarios, lo que permitió recopilar cada vez mayor cantidad de datos, los cuales posteriormente han sido usados para entrenar grandes modelos del lenguaje. A su vez, estos dos puntos suponen las principales limitaciones de estos algoritmos, ya que, para implementar y entrenar un modelo de estas características, necesitaremos en primer lugar muchos datos, que dependiendo del proyecto pueden llegar a ser difíciles de recopilar, y también un computador potente capaz de realizar tal procesamiento, aunque hoy día se puede tener acceso a computación en la nube a precios asequibles o incluso gratuitos.

1.4 Objetivos

El objetivo del presente proyecto será implementar diferentes modelos de aprendizaje automático, y entrenarlos para clasificar grabaciones de personas hablando según la emoción con la que está hablando. Para ello, en primer lugar, se buscarán y analizarán diferentes *datasets* públicos de grabaciones, etiquetados con las emociones que representan. Será importante tener en cuenta la calidad de estos *datasets* y la distribución de las diferentes clases. Además, se buscará que los datos de entrenamiento sean lo más amplios posibles en cuanto a sexo, edad, etnia, etc. Esto es debido a que sabemos que los modelos de IA tienden a tener sesgos si los datos usados no abarcan la mayor cantidad de posibilidades, lo que puede generar que funcione mejor para hombres que para mujeres, por poner un ejemplo.

Una vez analizados y escogidos los datos que vamos a usar en nuestro proyecto, procedemos a implementar y entrenar diferentes modelos de reconocimiento automático, y evaluar su desempeño para datos que no ha visto. Para ello, se implementarán modelos tanto de aprendizaje automático como de aprendizaje profundo, para poder

comparar los diferentes tipos de modelos.

Los puntos que se tendrán en cuenta para evaluar el funcionamiento de dichos modelos son los siguientes:

- Precisión: capacidad de determinar las emociones con el menos error posible.
- Coste computacional: capacidad del modelo para realizar inferencia sobre nuevos datos en el menor tiempo posible dentro de un mismo hardware.
- Generalización: capacidad de clasificar correctamente las emociones de grabaciones de audio que el modelo no haya visto durante el entrenamiento.

La elección de estos puntos se realiza buscando el objetivo de que dichos modelos puedan ser implementados en aplicaciones reales, donde puedan ser capaces de obtener los resultados con el menor error posible, se pueda ejecutar en computadores convencionales con una capacidad de procesamiento en tiempo real, y además que sea funcional para cualquier tipo de grabación que reciba.

1.5 Herramientas

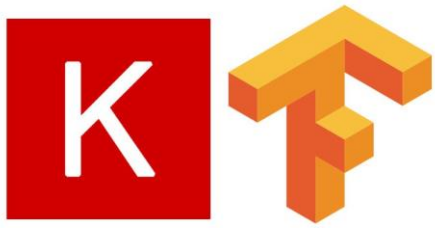
Se presenta a continuación cuales son las herramientas de las que se harán uso para el objetivo propuesto.

Entorno de programación

Uno de los puntos importantes es el entorno en el que se programarán estos modelos de reconocimiento automático. Para ello, se ha escogido instalar un entorno Python dentro de anaconda.

Anaconda es una distribución de Python, que facilita la instalación de los diferentes paquetes que se usarán en el proyecto. Además, permite crear diferentes entornos de forma sencilla, lo que facilita si deseamos tener entornos con versiones diferentes de ciertos paquetes, evitando conflictos con el resto de paquetes y entornos. Además, se ha escogido el lenguaje de programación Python, ya que es un lenguaje sencillo y ampliamente extendido en el ámbito de la ciencia de datos y de la inteligencia artificial, por lo que ya tiene una enorme cantidad de librerías y herramientas implementadas para Python que facilitan el desarrollo de este tipo de proyectos.

Una de las librerías más importantes para este proyecto será *tensorflow*. *Tensorflow* es un paquete de herramientas para desarrollo de inteligencia artificial, de código abierto y desarrollado por Google. Por encima de *tensorflow*, Google también ofrece una librería con una abstracción mayor, que facilita el uso de *tensorflow*, llamada *Keras*, de la cual también se hará uso.



Finalmente, se utilizará visual *studio code* como entorno en el que programar y probar nuestro código, en conjunto con *jupyter* notebook, también integrado dentro del mismo *visual studio* y el cual ofrece la posibilidad de ejecutar y probar código de forma más rápida.

1.6 Estado del arte

La capacidad por parte de las máquinas para poder reconocer las emociones de los humanos es un campo de un alto interés, tanto en el ámbito empresarial como en el ámbito de investigación. Esto es debido a que la capacidad de las máquinas interpretar emociones, e incluso de representarlas, siempre ha supuesto un punto diferencial entre las capacidades de una máquina y un humano, ya que es un concepto abstracto y difícil de interpretar de forma automática con las herramientas que se tenían hasta ahora. Por esto mismo, ha sido un campo de estudio ampliamente investigado a lo largo de la historia.

Uno de los principales problemas que se han encontrado a la hora de realizar estudios al respecto, es el acceso a datos de calidad correctamente etiquetados, ya que es una tarea que no resulta trivial y es difícil de automatizar y escalar la obtención de este tipo de datos. Vamos a continuación una tabla de los *datasets* más citados, ofrecidos por [3].

Most cited SER databases. #em = number of emotions, AV = audio-visual data, Nat. = natural versus actors recording, #ms #fs = number of male and female subjects, #ut = number of utterances, set = number of sentences.

Database	#em	language	AV	Nat.	#ms	#fs	#ut	#sent
DES [36,37]	5	Danish	n	actors	2	2	13	
EMODB [18]	7	German	n	actors	5	5	10	800
eNTERFACE [89]	6	English	y	natural	34	8	-	-
IEMOCAP [19]	10	English	y	actors	5	5	-	-
SAVEE [47]	7	English	y	actors	4	-	120	480
Thai DB [128]	6	Thai	y	actors	3	3	972	5832
INTERISP [87]		Spanish	n	actors	1	1	184	6040
TESS [34]	7	English	n	actors	-	2	200	2800
RAVDESS [79]	8	English	y	actors	12	12	2	1440
JL-Corpus [57]	10	NZ English	n	actors	4	0	-	-
MSP-PODCAST [80]	8	English	n	natural	-	-	-	18000

Ilustración 3 Bases de datos de audios con emoción etiquetadas más citados [3]

Diferentes tipos de modelos han sido entrenados con estos *datasets*, ofreciendo diferentes resultados para cada uno de ellos. Originalmente, hasta hace una década, los modelos que mejores resultados obtuvieron en el análisis de emociones en audios fueron modelos de aprendizaje automático, como por ejemplo los modelos *Support Vector Machine*, o SVM, pero posteriormente, se comenzaron a usar modelos con redes neuronales, ya que en primer lugar facilitaban la extracción de características, y además llegaban a obtener mejores resultados. Si analizamos las publicaciones existentes, las gran mayoría se corresponde con este tipo de modelos ya mencionados, aunque es cierto que otros enfoques se han realizado los últimos años implementando modelos con *transformers* y *autoencoders*, arrojando también buenos resultados. Se copia a continuación una comparación de los modelos que mejor precisión han obtenido para uno de los *datasets* más usados, el *dataset* RAVDESS. La comparación ha sido realizada por la web *paperswithcode.com*, la cual se encarga de analizar artículos relacionados con *machine learning* y evaluar los resultados que obtienen, para así poder realizar un estudio del estado del arte del mismo.

Rank	Model	Accuracy	F1 Score	Precision	Recall	F1	Extra Training Data	Paper	Code	Result	Year	Tags
1	**VQ-MAE-S-12** (Frame)	84.1				0.844	×	A vector quantized masked autoencoder for speech emotion recognition			2023	
2	**CNN-X** (Shallow CNN)	82.99%	0.82	0.82	0.82		×	Shallow over Deep Neural Networks: A empirical analysis for human emotion classification using audio data			2020	
3	**xlsr-Wav2Vec2.0** (FineTuning)	81.82%					✓	A proposal for Multimodal Emotion Recognition using aural transformers and Action Units on RAVDESS dataset			2021	
4	**CNN-14** (Fine-Tuning)	76.58%					✓	Multimodal Emotion Recognition on RAVDESS Dataset Using Transfer Learning			2021	
5	**AlexNet** (FineTuning)	61.67%					✓	Multimodal Emotion Recognition on RAVDESS Dataset Using Transfer Learning			2021	

Como observamos, aunque métodos más modernos han irrumpido con buenos resultados, como los comentados *transformers* y *autoencoders*, las redes neuronales convolucionales son las que mayor protagonismo han tenido en los últimos 5 años, por su eficiencia y precisión.

2 DATASETS

Uno de los elementos más importantes en el desarrollo de un proyecto de aprendizaje automático son los datos, dado que serán los que usaremos para el entrenamiento de nuestro modelo y, también, para testar y evaluar su funcionamiento. Para ello, se han escogido 4 *datasets* diferentes, con pequeños fragmentos de audios de actores interpretando con una emoción en concreto, que será la emoción con la que etiquetaremos posteriormente dicho audio. Dado que en este proyecto se pretender evaluar el reconocimiento automático de emociones en un entorno real, y no el rendimiento de un modelo para un *dataset* en concreto, se ha decidido por entrenar y obtener los resultados para cuatro *datasets* en conjunto, de forma que el modelo pueda generalizar lo máximo posible, al contar con mayor cantidad de actores diferentes.

El idioma de los audios de todos los *datasets* será el inglés. Esto último es importante a tener en cuenta, ya que, aunque no se esté usando las transcripciones del audio para la clasificación, la sonoridad y entonación puede cambiar mucho entre idiomas, por lo que probablemente el modelo con una precisión alta en inglés, no tiene por qué tener la misma precisión en español. Dado que este tema se escapa del objetivo de este proyecto, se deja esta interesante investigación para futuros trabajos.

Por último, en mucho de los *datasets* que presentamos a continuación, se hace una diferenciación entre los audios dependiendo si son de un hombre o de una mujer, pero en este caso se tratará de identificar la emoción predominante en el audio si atender a si el audio proviene de una voz femenina o de una voz masculina.

2.1 RAVDESS

Siglas de *Ryerson Audio-Visual Database of Emotional Speech and Song Dataset*, es un *dataset* compuesto por 7356 grabaciones de un total de 24 actores (12 hombres y 12 mujeres), representando emociones de felicidad, tristeza, enfado, tranquilidad, miedo, sorpresa, disgusto y neutral. Este es un *dataset* ampliamente utilizado en la investigación para el tratamiento de audios, especialmente para el reconocimiento de emociones en el habla, por lo que es muy común usar la precisión en este *dataset* para comparar los diferentes modelos del campo de estudio. En este trabajo, nos quedaremos realmente con la parte del *dataset* del habla, descartando los audios de canciones, por lo que tendremos un total de 1440 audios. Estos audios serán en inglés norteamericano, con un acento neutro, y en diferentes niveles de intensidad (normal y fuerte). La distribución de los datos según las clases queda de la siguiente forma:

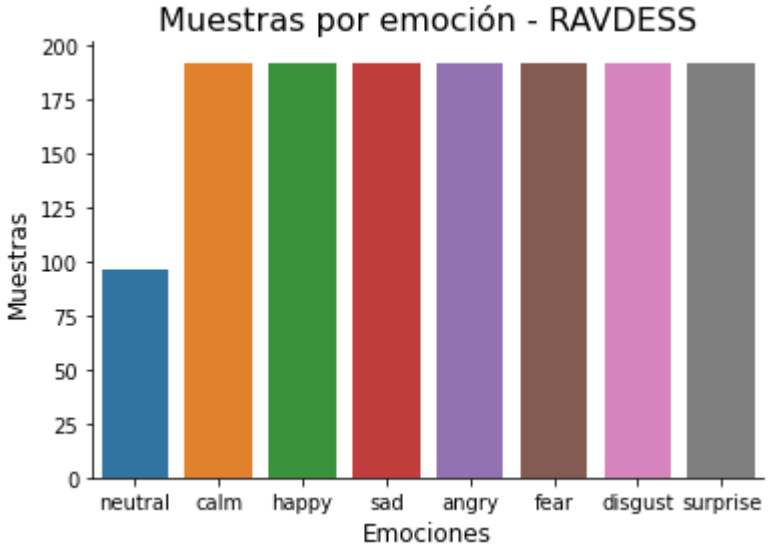


Ilustración 4 Distribución de clases RAVDESS

2.2 Crema D

Crowd Sourced Emotional Multimodal Actors Dataset, Crema D por sus siglas en inglés, es un *dataset* muy interesante ya que contiene audios de una gran cantidad de actores, 91 en total, siendo 48 hombres y 41 mujeres. La procedencia de los actores será diferente (africanos, americanos, hispánicos, asiáticos, caucásicos y no especificados). Esto es especialmente importante para la generalización del modelo que se comentaba en la introducción, ya que al contener una mayor cantidad de actores es más complicado que se produzca un sobreajuste, favoreciendo que el modelo funcione bien con audios de otros actores que nunca haya visto.

El total de audios serán 7442, con emociones clasificadas en enfado, disgusto, miedo, felicidad, neutral y tristeza. Cada actor dirá una selección de 12 frases. Además, estos audios tendrán diferentes intensidades en las emociones, que serán baja, media, alta o no especificada). La distribución de los datos será de la siguiente forma:

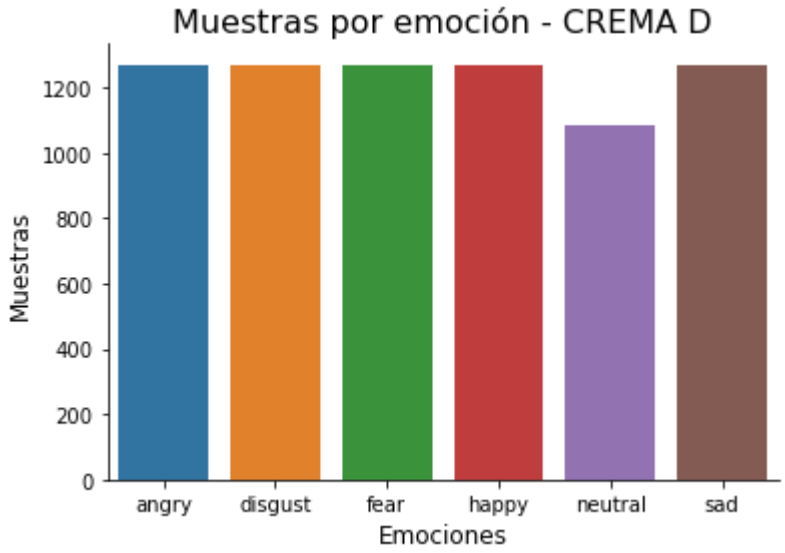


Ilustración 5 Distribución de clases Crema D

Como veremos posteriormente en el trabajo, los audios con intensidades en la emoción medias y sobre todo bajas, suponen un problema a la hora de entrenar el modelo, ya que no son tan diferentes unos de otros, y dificulta bastante que el modelo aprenda las características más importantes de las emociones, por lo que finalmente tan solo usaremos los audios etiquetados como HI (*High Intensity*), quedando la distribución de clases de la siguiente forma:

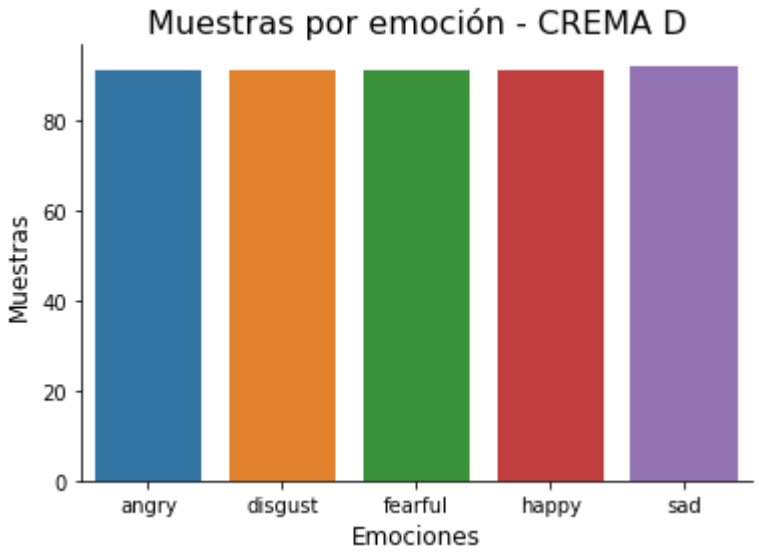


Ilustración 6 Distribución de clases Crema D High Intensity

2.3 SAVEE,

Siglas de *Surrey Audio-Visual Expressed Emotion*, realizado por un grupo de investigación de la universidad de Surrey, es un *dataset* caracterizado por tener audios de una muy buena calidad, pero solo de actores masculinos, por lo que queda desbalanceado en ese sentido. Los 4 actores de los que está compuesto este *dataset* serán ingleses nativos. Las clases de los audios serán enfadado, disgustado, miedo, felicidad, tristeza, sorpresa y neutral. Esto nos da un total de 480 muestras de audios, quedando la distribución de clases de la siguiente forma:

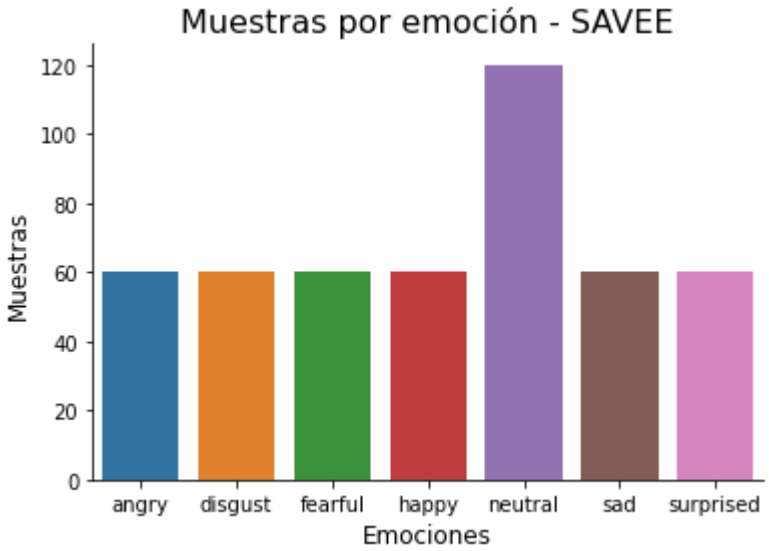


Ilustración 7 Distribución de clases de SAVEE

2.4 TESS

Toronto emotional speech set, es un *dataset* realizado por la universidad de Toronto, es un *dataset* caracterizado por su alta calidad de audio, y por estar compuesto solo por actrices. Esto último es negativo para nuestro objetivo de que nuestros modelos generalicen lo máximo posible, pero contrarresta el *datset* de SAVEE solo compuesto por voces masculinas, siendo este precisamente el objetivo perseguido por los creadores de este *dataset*, el ofrecer más voces masculinas para eliminar el sesgo de los modelos a voces masculinas.

Concretamente estará compuesto por 2800 audios, de dos actrices diferentes. Las categorías de las emociones clasificadas serán enfado, disgusto, miedo, felicidad, sorpresa, tristeza y neutral. La distribución de las clases quedará de la siguiente forma:

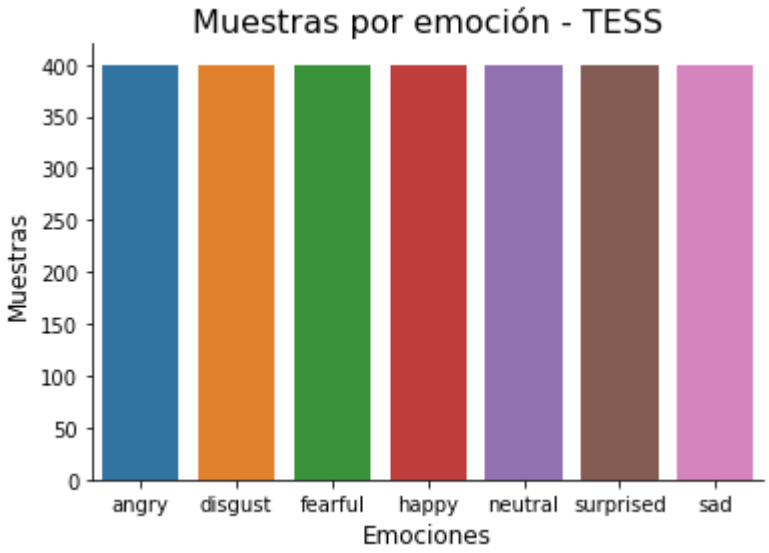


Ilustración 8 Distribución de clases de TESS

2.5 Distribución total de los datos

Finalmente, como hemos comentado al principio de este capítulo, se juntarán los datos de los 4 *datasets*, de forma que formemos un *dataset* más grande y con mayor variedad de actores. Para ello, se han procesado todos los *datasets* anteriores en Python, obteniendo la emoción característica de cada audio y almacenando dichos datos en un *dataframe* de Pandas. Una vez procesados todos los audios, se han concatenado en un *dataframe* general. Posteriormente, se han guardado estos datos en una base de datos SQL, para una mejor usabilidad posteriormente. La distribución de clases total queda de la siguiente forma, con un total de 12161 muestras:

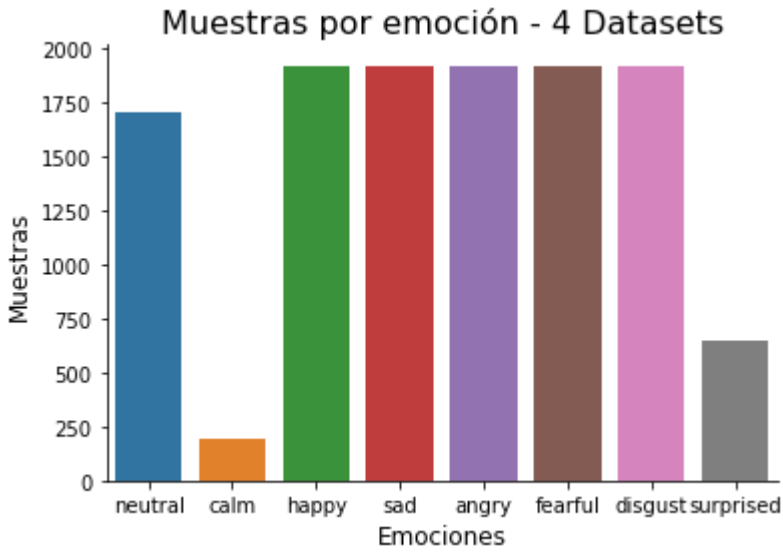


Ilustración 9 Distribución total de clases 4 *datasets*

Como ya hemos dicho en la explicación del *dataset* SAVEE, se han eliminado gran parte de las muestras del *dataset* de SAVEE por no ser suficientemente diferentes entre las emociones, por lo que tras eliminar estas muestras nuestro *dataset* queda de la siguiente forma, con un total de 5175 muestras:

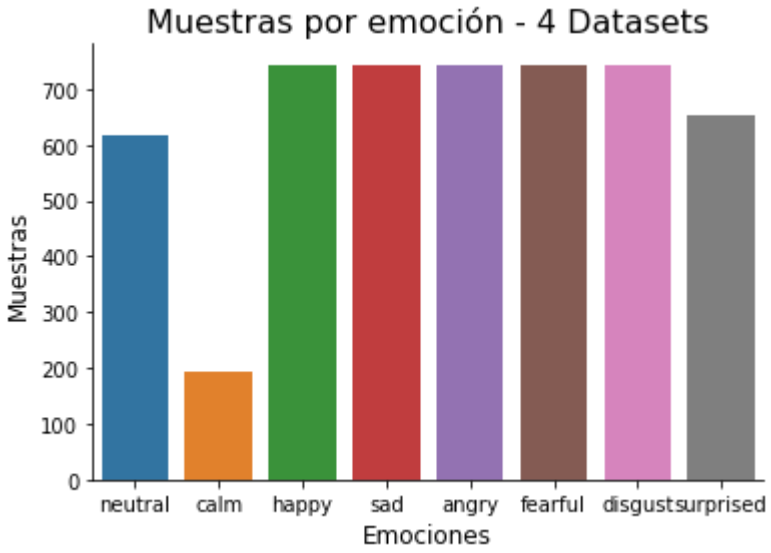


Ilustración 10 Distribución de clases, solo con HI de SAVEE

Finalmente, dados que la categoría de audios tranquilos tiene sustancialmente menos muestras que las otras, las eliminamos también para que nuestro *dataset* quede balanceado, quedan por lo tanto la distribución de la siguiente forma, con un total de 4984 muestras:

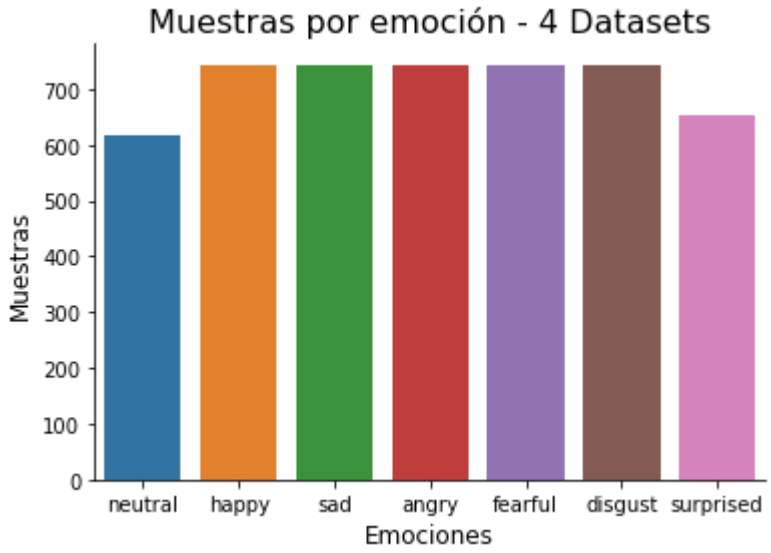


Ilustración 11 Distribución total de clases eliminando categoría *calm*

3 ANÁLISIS DEL AUDIO

Como sabemos, la voz se genera a partir de nuestras cuerdas vocales, se propagan como ondas mecánicas a través de oscilaciones en la presión del aire, y finalmente se reproducen como una onda mecánica en las membranas de nuestro oído. Para el procesamiento digital de estas señales acústicas, será necesario captar estas señales por medio de un micrófono y digitalizarlas para poderlas almacenar, procesar y reproducir a través de un ordenador. Para ello existen múltiples formatos de archivos de audio con los que podemos representar digitalmente los sonidos, y en nuestro caso trataremos con archivo .wav, ampliamente utilizados, que aunque son archivos pesados se caracterizan por conservar el audio con una calidad alta.

Una vez procesado este audio digitalmente, podemos representarlo en el dominio del tiempo como un *waveform*, que será la representación de las diferentes amplitudes de onda del audio muestreados a lo largo del tiempo. Un ejemplo es el siguiente:

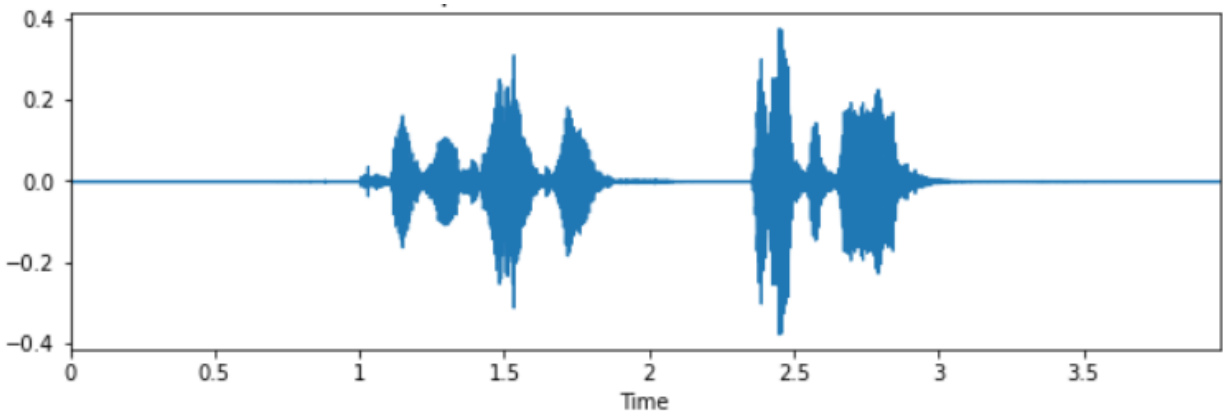
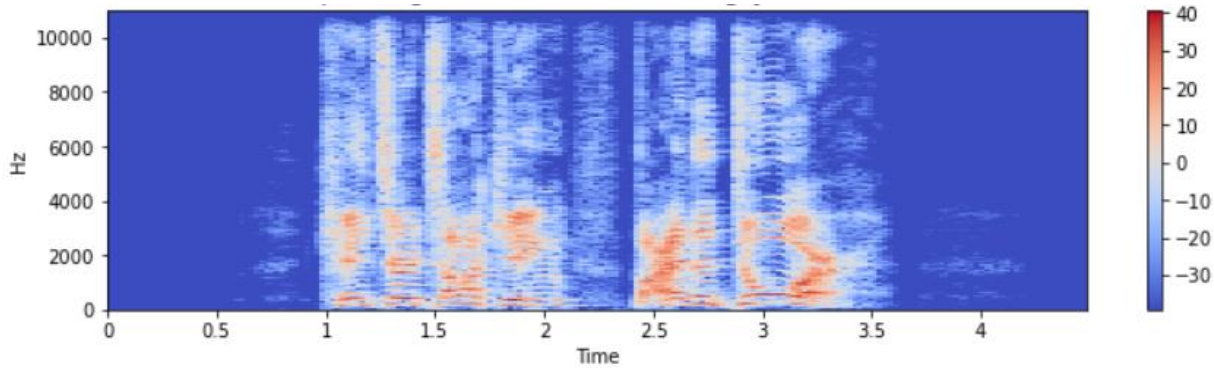


Ilustración 12 *Waveform* de un audio

Pero el audio no solo se puede representar en el dominio del tiempo, sino que también puede suele ser representado en el dominio de la frecuencia, para lo que se usa la transformada de Fourier. Como veremos posteriormente, el análisis del audio en frecuencia será ampliamente utilizado la búsqueda de características de en la voz.

Para representar el audio tanto en el dominio de la frecuencia como en el dominio del tiempo, normalmente se usan espectrogramas, donde podemos ver incluidos ambos aspectos del audio. Estos, son calculados aplicando la *STFT (short time Fourier transform)*, que básicamente se trata de aplicar la transformada de Fourier en pequeños periodos de tiempo dentro de un audio completo. Esta representación la podremos usar posteriormente, ya que aunque no es una característica del audio en sí, si es una representación del audio en la que un modelo de *Deep learning* puede extraer características. Dicha representación queda de la siguiente forma:



3.1 Extracción de características (*features*)

La extracción de características es uno de los principales de procesamientos en la tarea de clasificación

automática, ya que estos serán los datos a partir de los cuales posteriormente trataremos de diferenciar una clase de otra. Por lo tanto, juegan un papel decisivo, ya que se necesita que estas características sean los suficientemente únicas y diferenciales en cada una de las clases que pretendemos reconocer, ya que, si no, será imposible de clasificar para el modelo de reconocimiento que posteriormente desarrollemos. A continuación, explicamos una serie de características que posteriormente usaremos para la tarea de clasificación.

3.1.1 MFCC

Los coeficientes MFCC (*Mel-Frequency Cepstral Coefficient*) son una representación espectral del audio ampliamente utilizada en el campo de estudio del reconocimiento del lenguaje, siendo unos de los principales extractores de características para esta tarea.

Como hemos dicho, este algoritmo está especialmente pensado para detectar las características propias de la voz humana dentro de un audio. Para esta tarea, el algoritmo aplica dos conceptos clave, la escala de frecuencias de *mel*, con la que adaptamos el espectro del audio a las características de la voz humana, y la separación de frecuencias *cepstrum*, con el separamos la señal de voz en dos bandas de frecuencias altas y bajas, correspondiéndose con los graves y agudos de la voz.

El procedimiento concreto de extracción de MFCC es el siguiente:

1. Pre-énfasis: se aplica un filtro para enfatizar las frecuencias atenuadas.

$$y(n) = x(n) - ax(n-1)$$

2. Enventanado: se divide el audio completo en *frames* más cortos, con un solapamiento entre frame y frame. Normalmente, se hará un ventanado de tipo *hanning* o haming, donde cada frame de audio posteriormente se le aplicará alguna de las dos ventanas mencionadas anteriormente.

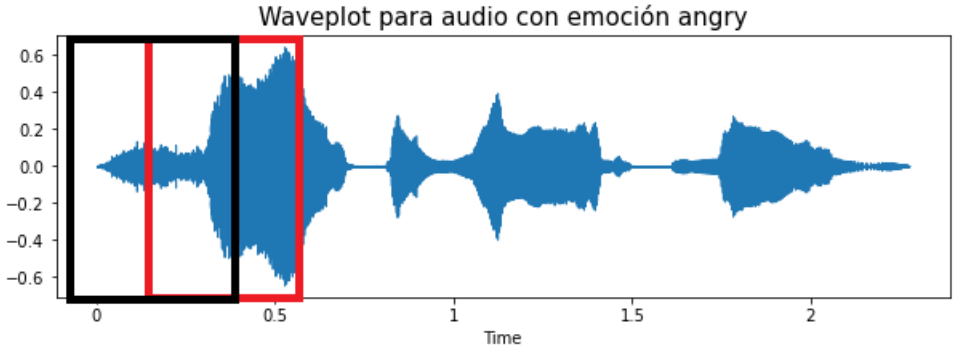


Ilustración 13 Enventanado de un audio

$$y(n) = x(n) * h(n)$$

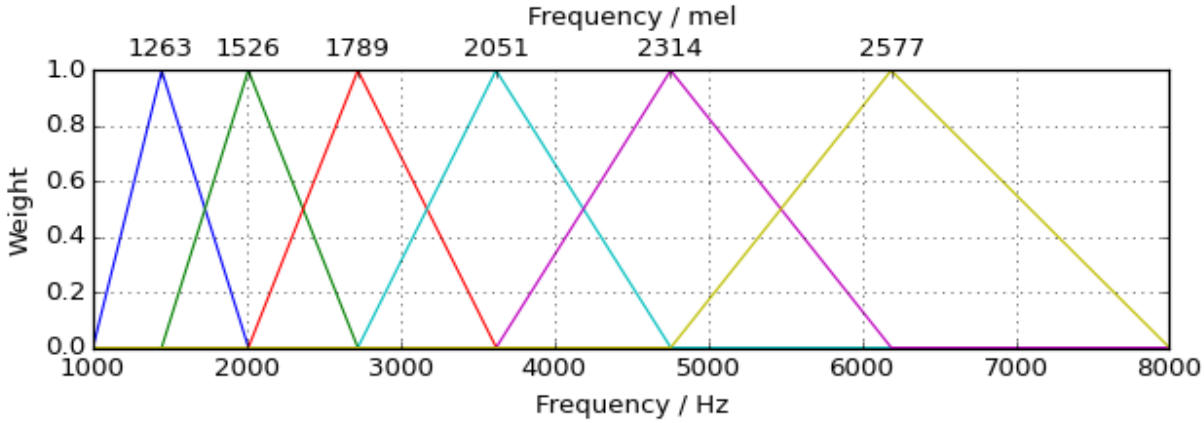
3. FFT: se aplica la transformada rápida de Fourier a cada *frame* de audio para pasar al dominio de la frecuencia.

$X_m(k) = \sum_{n=0}^{N-1} x_m(n) * e^{-j\frac{2\pi}{N}kn}$, siendo k el índice de la frecuencia y N el tamaño de ventana.

4. Banco de filtros de *mel*: definiendo la frecuencia de *mel* como

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

Aplicaremos una serie de filtros, definidos a partir de las frecuencias mel como se indica en la siguiente figura, siendo el número de filtros un hiperparámetro a ajustar mientras configuramos nuestro extractor de *features*: un valor típico pueden ser 20. Normalmente, los primeros filtros correspondiente a las frecuencias más bajas, se aplican equiespaciados, y los filtros correspondientes a las frecuencias más altas estarán logarítmicamente separados, como se puede apreciar en la imagen a continuación:



5. Energía: se calcula la energía de la salida de cada uno de los filtros previamente calculados:

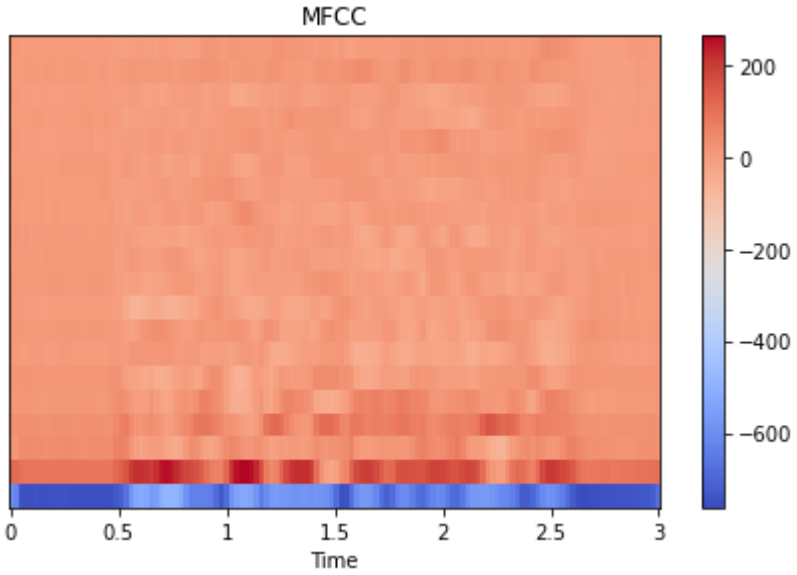
$$E_m = \sum_{k=0}^{N-1} |X(k)|^2 H_m(k) \quad 1 \leq m \leq F$$

Se calcula posteriormente el logaritmo en base 10 de la energía.

6. DCT: finalmente se calcula la transformada discreta del coseno de la siguiente forma:

$$C_{MFCC}(m) = \sum_{k=0}^{N-1} \log(E_k) \cos\left(m(mk - \frac{1}{2})\frac{\pi}{N}\right.$$

La representación gráfica del MFCC quedará de la siguiente forma:



3.1.2 Pitch

El pitch es cómo llamamos a la frecuencia fundamental del audio, el cual es una característica muy relevante para el procesamiento del audio y reconocimiento de la voz. El cálculo de esta frecuencia fundamental no es algo trivial, y para ello en este proyecto se aplicarán dos métodos diferentes: la función de *auto-correlación* y el *cepstrum*.

ACF

La función de autocorrelación es muy útil a la hora de detectar patrones repetitivos en la voz, y por lo tanto nos puede servir para determinar la frecuencia fundamental de un audio. Para ello, lo que hace es medir la similitud de la señal consigo misma en diferentes retardos, por medio de la siguiente ecuación:

$$R^i = \sum_w^N |X(w)|^2 * e^{jwn}$$

La cual representa la DFT inversa de la densidad espectral de potencia.

Posteriormente se calcula el pitch de la siguiente forma a partir del resultado de la autocorrelación:

$$R_p^i = \frac{f}{idx(\max(R^i[f*0.02:f*0.2])) + f*0.02 - 1}$$

Cepstrum

Este método consiste en aplicar la transformada de Fourier a un espectro logarítmico, pudiendo así analizar el contenido en frecuencia de la señal facilitando la obtención del pitch de la señal. Se calculará el *cepstrum* de la siguiente forma:

$$C^i = \sum_w^N \log|X(w)| * e^{jwn}$$

A partir del cual podremos calcular el pitch de la siguiente forma:

$$C_p^i = \frac{f}{idx(\max(C^i[f*0.02:f*0.02])) + f*0.02 - 1}$$

3.1.3 Intensidad

La intensidad de sonido se define como la potencia acústica transferida por una onda sonora por unidad de área normal a la dirección de propagación. Este representa uno de los mayores atributos del audio, lo que puede ayudar mucho a identificar diferencias entre las distintas emociones. Calcularemos la intensidad del audio de la siguiente manera:

$$I_i = \frac{1}{N}\sum_n^N Ham^2(x^i(n))$$

Siendo Ham la función de ventana hamming, aplicada sobre un *frame* concreto de audio, al igual que se hiciera para el cálculo de los coeficientes MFCC.

3.1.4 ZCR

El *Zero-crossing rate* (ZRC), es otra de las características ampliamente utilizadas para el reconocimiento del habla y de música, arrojando buenos resultados en diferentes aplicaciones de este entorno. Esta característica representa la cantidad de veces que la señal cambia de positivo a negativo o viceversa, es decir su paso por cero.

3.1.5 LSP

Line Spectral Pairs, es una representación de los *Linear Prediction Coefficients LPC*, muy utilizada para la codificación y síntesis de voz, principalmente por su estabilidad, su facilidad de interpolación de parámetro de la voz y por ser adecuadas para la cuantización. Este método convertirá los coeficientes LPC en pares de frecuencia.

El sistema de predicción lineal de orden p se representa en el dominio de la Z de la siguiente forma, siendo *ak* los coeficientes LPC:

$$x(n) = \sum_{k=1}^{p} a_k x(n-k) \rightarrow x(z) = \sum_{k=1}^{p} a_k x(z) z^{-k}$$

Y la predicción del error A(z) como:

$$A(z) = 1 - \sum_{k=1}^{p} a_k * z^{-k}$$

Los polinomios P(z) y Q(Z) se vienen dados por:

$$P(z) = A(z) + z^{p+1}A(z^{-1}), \;\; Q(z) = A(z) - z^{-(p+1)}A(z^{-1})$$

Los parámetros LSP se expresan como los ceros de P(z) y de Q(z).

3.1.6 RMS

Root mean square, o raíz cuadrática media en español, es una medida de la energía de una cantidad variable, comúnmente utilizada en teoría de señales, que puede ser una característica útil para la tarea que pretendemos desarrollar. Se calcula de la siguiente forma:

$$x_{RMS} = \sqrt{\frac{1}{N}\sum_{n}^{N} x^2(n)}$$

4 MODELOS DE APRENDIZAJE AUTOMÁTICO

En el presente capítulo, presentaremos diferentes modelos de reconocimiento automático que tenemos a nuestra disposición para el objetivo de este proyecto. Se pretende presentar una variedad de modelos de aprendizaje automático amplio, de forma que podamos evaluar en resultados de posteriores experimentos las ventajas y desventajas de cada uno.

4.1 SVM

El *support vector machine*, es un algoritmo muy utilizado en problemas de clasificación. Este algoritmo tiene como objetivo encontrar un hiperplano que separe dos clases diferentes según sus características. Para ellos, tratará de que el margen entre el hiperplano y las dos clases que el más grande posible. Para una clasificación binario podríamos poner el siguiente ejemplo:

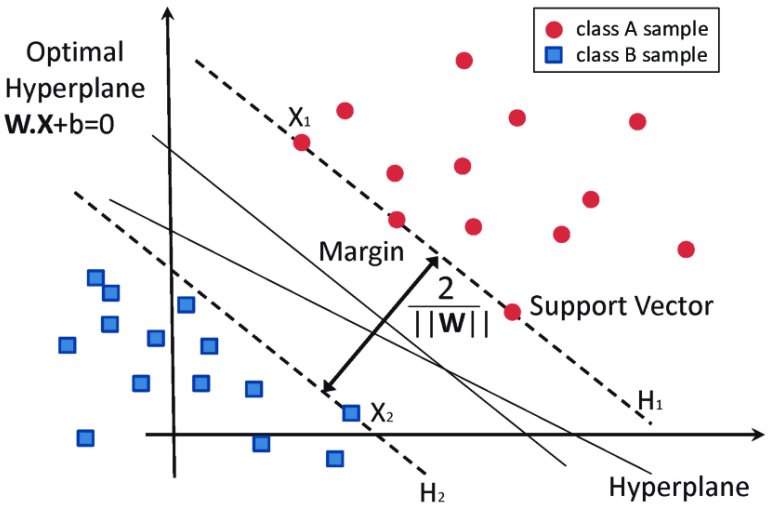


Ilustración 14 Clasificación binaria de un SVM [4]

En la gráfica anterior, podemos observar las muestras de dos clases diferentes representadas por dos características distintas. En un caso ideal como en el de la imagen, sería sencillo separar las dos clases mediante una función lineal, pero normalmente los datos están más mezclados. Por ellos, el SVM usa las denominadas funciones *kernel*, que se encargan de transformar las características, asignando los datos a un espacio dimensional diferente que facilite la posterior clasificación de los mismo.

Las funciones *Kernel* más utilizadas son:

- Función base radial

$$K(x_1, x2_2) = \exp\left(-\frac{||x_1 - x_2||^2}{2\sigma^2}\right), donde\ \sigma\ representa\ la\ anchura\ del\ kernel$$

- Función lineal

$$K(x_1, x_2) = x_1^T x_2$$

- Función polinómica

$$K(x_1, x_2) = (x_1^T x_2 + 1)^p, donde\ p\ representa\ el\ orden\ del\ polinomio$$

- Función sigmoide

$$K(x_1, x_2) = \tanh(\beta_0 x_1^T x_2 + \beta_1)$$

De esta forma, dados unos datos de entrenamiento al algoritmo SVM, este realizará un proceso de optimización, en el que maximizará la función de costo correspondiente al margen entre las clases.

Aunque este método está originalmente ideado para clasificación binaria, es fácilmente implementable en problemas multiclase. Para ello, suele implementar un clasificador por cada una de las clases, de forma que cada uno forme una clasificación binaria de la clase en cuestión frente al resto.

4.2 Red Neuronal Artificial

4.2.1 Estructura

Las redes neuronales artificiales son un modelo informático creado en 1943 por Warren McCulloch y Walter Pitts, dando comienzo a la investigación en inteligencia artificial, que posteriormente con el aumento del poder computacional se volvieron muy populares. La unidad básica de una red neuronal es lo que llamamos perceptrón. Básicamente un perceptrón es una función de activación que dado un input de entrada te ofrece una salida. Este input de entrada es ponderado por unos pesos más un sesgo, los cuales serán los parámetros que tendrán que ser aprendidos por la red para ajustarse. La notación matemática de esto sería la siguiente:

$$z = \sum_i w_i x_i + b$$

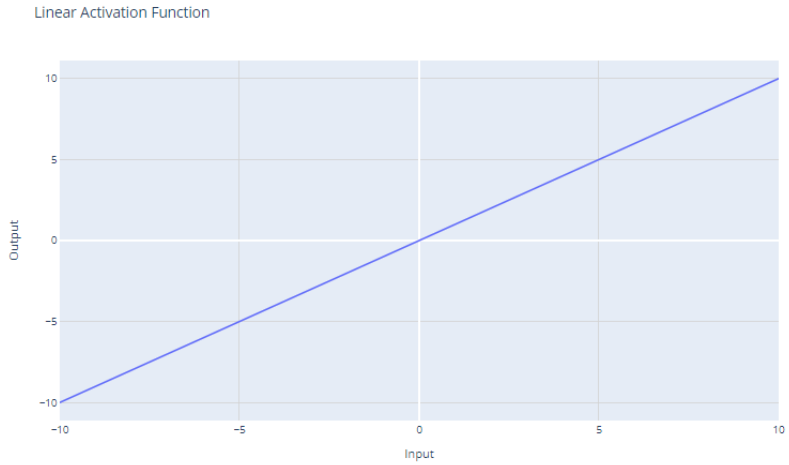
Posteriormente a este resultado se le aplicará una función de activación:

$$y = f(z)$$

Estas funciones de activación servirán para introducir no linealidad en nuestro sistema. Entre las más populares se encuentran las siguientes:

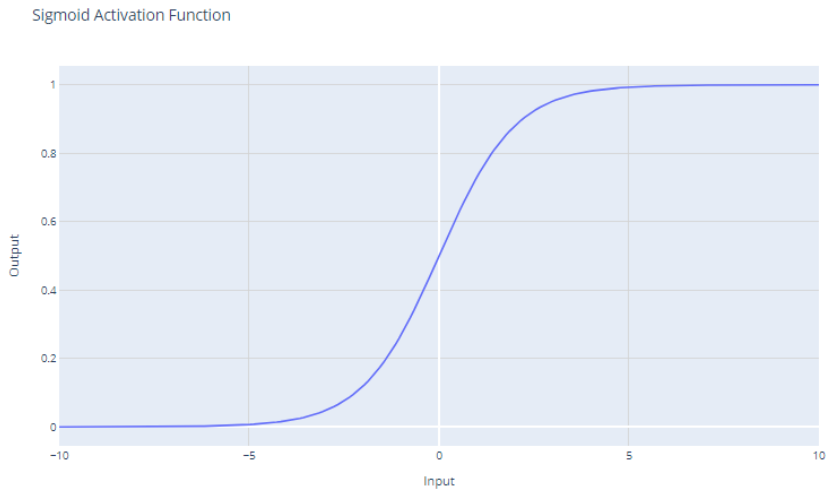
- Linear: se basa en una función identidad.

$$f(x) = x$$



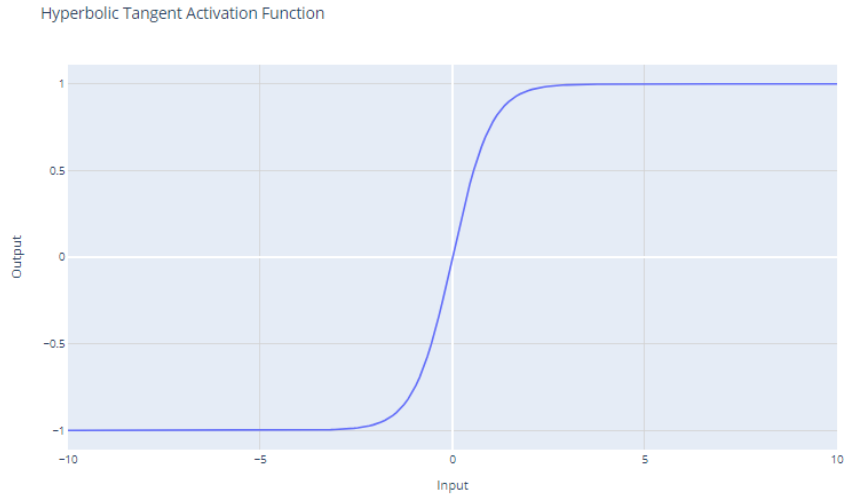
- Sigmoide: convierte la salida en valores entre 0 y 1.

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



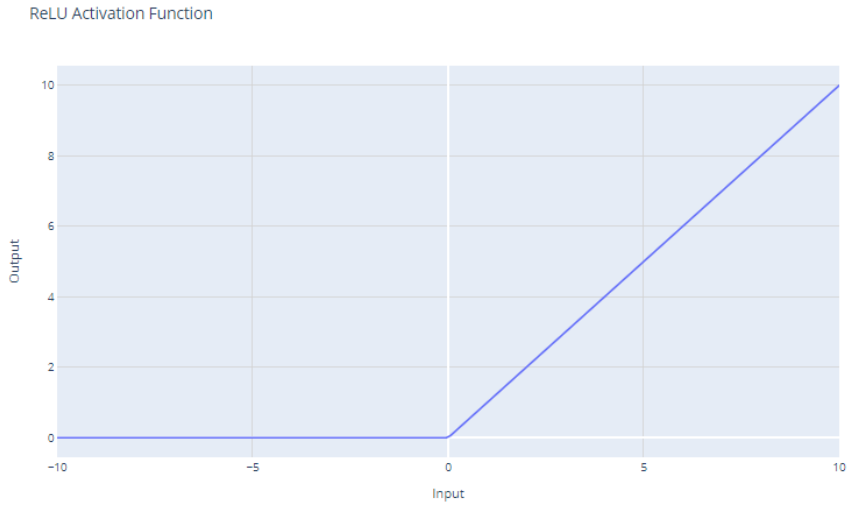
- Tanh: convierte la salida en valores entre -1 y 1.

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



- Relu: función que vale 0 para valores negativos, y f(x) = x para valores positivos.

$$f(x) = \max(0, x)$$



- Softmax: devuelve el valor máximo de las entradas recibidas.

Este perceptrón, por sí solo, ya podrá ser aplicado en problemas de clasificación binaria y de regresión, y en muchos casos puede ser suficiente para obtener buenos resultados, pero es cuando unimos unas neuronas a otras, en serie y en paralelo, cuando aumentamos las capacidades de este modelo. Llamaremos capas a las neuronas que estén conectadas en paralelo, de forma que tendremos una capa de entrada, que serán las neuronas a las que conectemos nuestros datos de entrada directamente, una capa de salida que será el resultado de la inferencia de la red, y unas capas ocultas que serán las capas intermedias. Según el número de las capas ocultas, tendremos una red más o menos profunda. El esquema de todo lo explicado se corresponde con el siguiente:

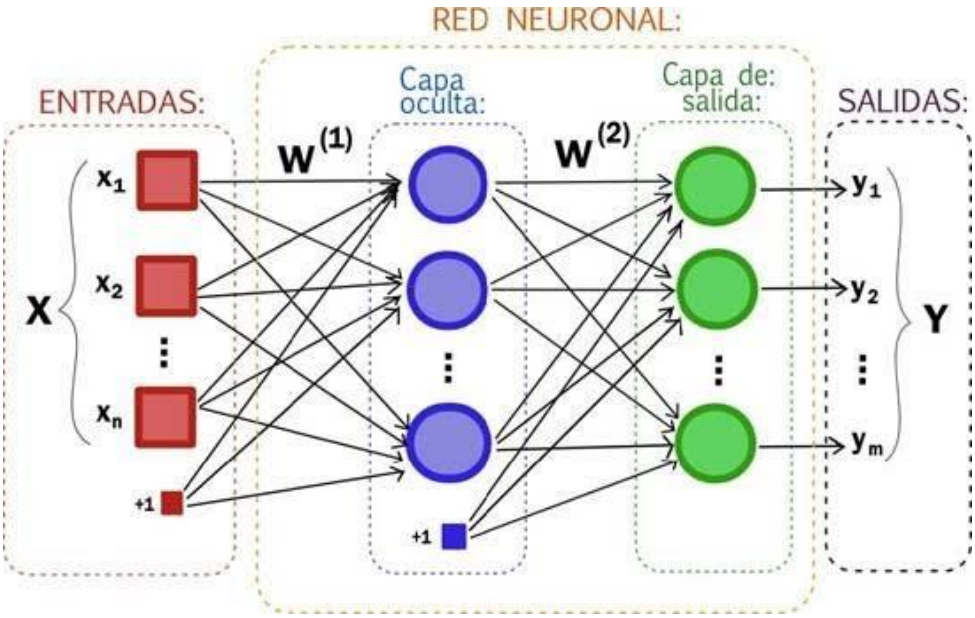


Ilustración 15 Esquema de capas de una red neuronal

Matemáticamente, la operación realizada por cada neurona del esquema anterior será la siguiente:

$$y = w_1 x_1 + w_2 x_2 + \cdots + w_n w_n + b$$

Por temas de rendimiento a nivel de cálculo computacional, estas operaciones se vectorizarán quedando de la siguiente forma:

$$y = W^T X + b$$

Siendo W el vector de pesos de dicha neurona, y X el vector de las distintas salidas de la capa anterior. Si formamos con W una matriz con todos los vectores de pesos de todas las neuronas de dicha capa, y con b el vector de todos los *bias*, se puede hacer la operación de toda la capa de una vez de forma matricial, lo que simplifica mucho el cálculo. Al ser operaciones matriciales, se mejora mucho el rendimiento sobre todo cuando el cálculo lo realizamos con la GPU en vez de la CPU, ya que este hardware está diseñado para ser eficiente en este tipo de operaciones, heredado de las operaciones necesarias para las imágenes.

4.2.2 Entrenamiento

Uno de los puntos más importantes de las redes neuronales es su entrenamiento, que es lo que comúnmente se denomina aprendizaje. Este entrenamiento será el proceso en el que la red neuronal ajustará sus parámetros para que sea capaz de predecir de forma precisa dados unos datos de entrada, es decir, ajustará los pesos de las neuronas para que al inferir en la red obtengamos una salida esperada.

Este entrenamiento se basa en la optimización de una función de coste. Esta función de coste representa el error que obtenemos entre la salida que ofrece la red con respecto la salida que esperamos según los datos que tenemos. Esta señal de error es propagada desde la capa de salida hacia atrás, fraccionando el error en las diferentes neuronas en un proceso llamada *backpropagation*. A partir de este proceso se comienza con el descenso del gradiente, en el que, por medio de la derivada de esta función de coste, actualizaremos los pasos avanzando hacia un mínimo en dicha función. El tamaño de dichos pasos será el parámetro de *learning rate*. Este parámetro de *learning rate* será muy importante, ya que, en este proceso, podemos caer en mínimos locales, en los que, si el paso es demasiado pequeño, podemos no salir de él. Este proceso se repetirá iterativamente, hasta conseguir los valores de error mínimos posibles. Habrá que tener cuidado ya que si iteramos demasiadas

veces, o nuestra red es muy extensa, podemos tener un sobreajuste de nuestro modelo a los datos utilizados, lo que puede llevarnos a que el modelo no generalice lo suficiente. Por ello, se suele separar los datos en diferentes lotes:

- Lote de entrenamiento: suele ser el lote más grande y serán los datos con los que inferiremos en la red en el proceso de entrenamiento, y posteriormente a partir de su función de coste actualizaremos sus parámetros.
- Lote de validación: cada vez que se produzca una iteración completa de entrenamiento, es decir, sobre todos los datos de entrenamiento, procederemos a inferir el resultado de la red con este lote de datos, y calcular su función de coste, pero en este caso no actualizaremos los pesos, sino que tomaremos este valor como información acerca de cómo está actuando la red frente a parámetros no usados en el entrenamiento.
- Lote de test: finalmente, se suele dejar un porcentaje de los datos para un proceso final de test, en el que calcularemos el error de nuestra tras todo el proceso de entrenamiento. Esto se suele hacer porque durante el entrenamiento, aunque implícitamente no se usan los datos de validación, sí que nos dan información a nosotros a la hora de ajustar los diferentes hiperparámetros, por lo que si han llegado a estar involucrados en el entrenamiento de alguna forma. Por ese motivo, el resultado de test es el más representativo para evaluar el funcionamiento del modelo.

Como vemos, esta optimización se basa en los datos conocidos que usamos en el entrenamiento, por lo que será importante que estos datos cubran el mayor espectro posible de posibilidades para que aprenda correctamente.

A la hora de diseñar una red neuronal, habrá una serie de hiperparámetros que podremos ir ajustando según conveniencia para el caso en concreto. Algunos de los parámetros más típicos ya se han comentado, pero exponemos a continuación los más comunes que se irán ajustando en este proyecto:

- Número de épocas: número de veces que iteramos el proceso de entrenamiento.
- Tamaño de lotes: en cada iteración de entrenamiento se procesarán todos los datos de entrenamiento, pero estos podrán se agrupados en lotes para un mejor rendimiento computacional. Este parámetro representa el número de datos en los que se agrupan.
- Learning rate: el tamaño del paso con el que actualizamos los pesos cada iteración del entrenamiento.
- Función de activación: la función de activación que usan las neuronas de la red, cuya elección es importante según los datos con los que se trabaje.
- Optimizador: dentro de la tarea del descenso del gradiente, el optimizador será el algoritmo encargado de avanzar cada época en busca del mínimo de la función de pérdida. Para esto hay múltiples estrategias, cada una con sus ventajas y desventajas, en este proyecto se trabajará principalmente con tres: *RMSprop, SGD y Adam.* Se muestra a continuación la gráfica del entrenamiento de la misma red, implementando en cada una de ellas uno de estos tres optimizadores:

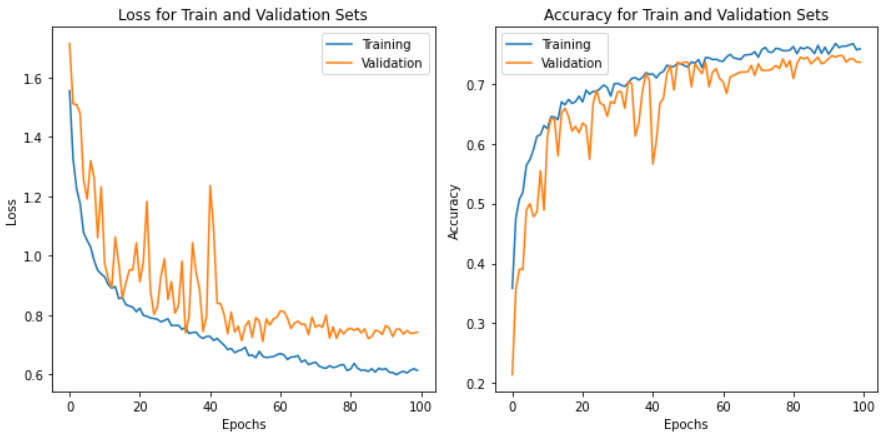


Ilustración 16 Optimizador RMSProp

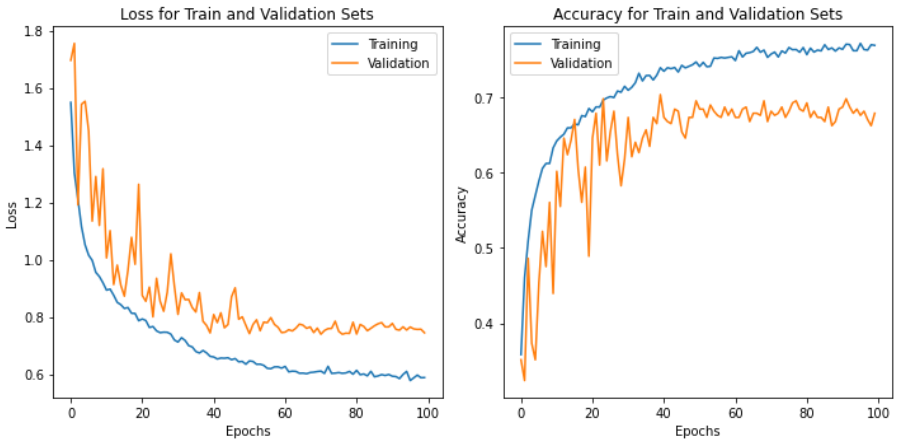


Ilustración 17 Optimizador Adam

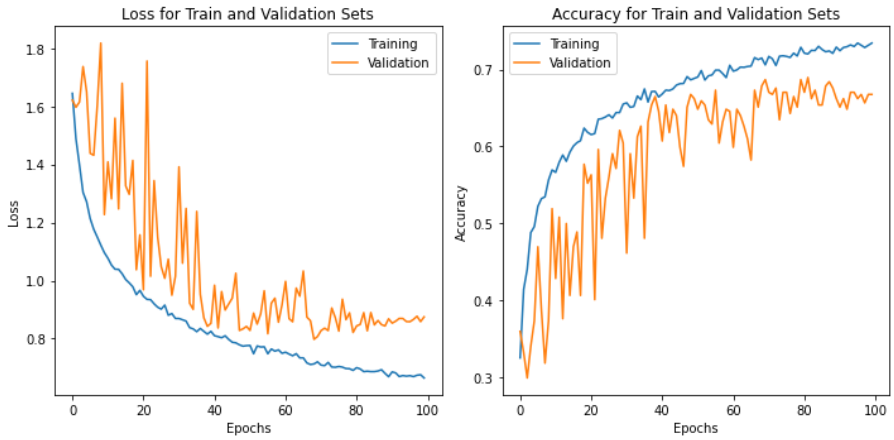


Ilustración 18 Optimizador SGD

Como vemos, el optimizador RMS va más directo en su búsqueda del mínimo, pero al estar basado simplemente en la derivada del error cuadrático medio, es propenso a caer en mínimos locales. Por otro lado, tenemos el optimizador Adam, el cual va ajustando su *learning rate* según la media móvil de los valores más recientes del gradiente, lo cual hace que lleguemos a un mínimo de forma más rápida y precisa, pero tenemos más variabilidad en los datos de validación y es más propenso a sobreajustar. Finalmente, vemos el optimizador *SGD*, el cual busca el mínimo en función de un pequeño lote de los datos, lo que introduce mucha más variabilidad en el avance del valor de pérdida en cada época haciendo que sea muy importante una correcta elección del learning rate.

Dentro de las redes neuronales artificiales, existen diferentes tipos de estructuras específicas que engloban una subcategoría de las mismas. Las redes neuronales densamente conectadas, son como las que se acaban de explicar, donde tendremos múltiples capas con múltiples neuronas, las cuales estarán conectadas con todas las neuronas de sus capas anteriores y posteriores. Además de estas, en este proyecto se implementarán otros dos tipos de redes, la red neuronal de convolución y las redes LSTM, las cuales explicamos a continuación.

4.3 Red Neuronal Convolucional

Las redes neuronales convolucionales serán un tipo específico de redes neuronales, basadas en la operación de convolución de los datos sobre filtros. Este método proviene originalmente del procesado de imágenes por computador, en las que se aplicaba una operación de convolución a la imagen sobre un filtro en concreto, el cual estaba específicamente diseñado para una tarea en concreto, como puede ser la búsqueda de bordes o esquinas. Esta es una técnica ampliamente utilizada, pero que requiere del diseño del filtro especifico y su posterior procesamiento para interpretar los resultados. Esta operación de convolución es ejecutada sobre todos los datos, obteniendo así unos nuevos datos filtrados.

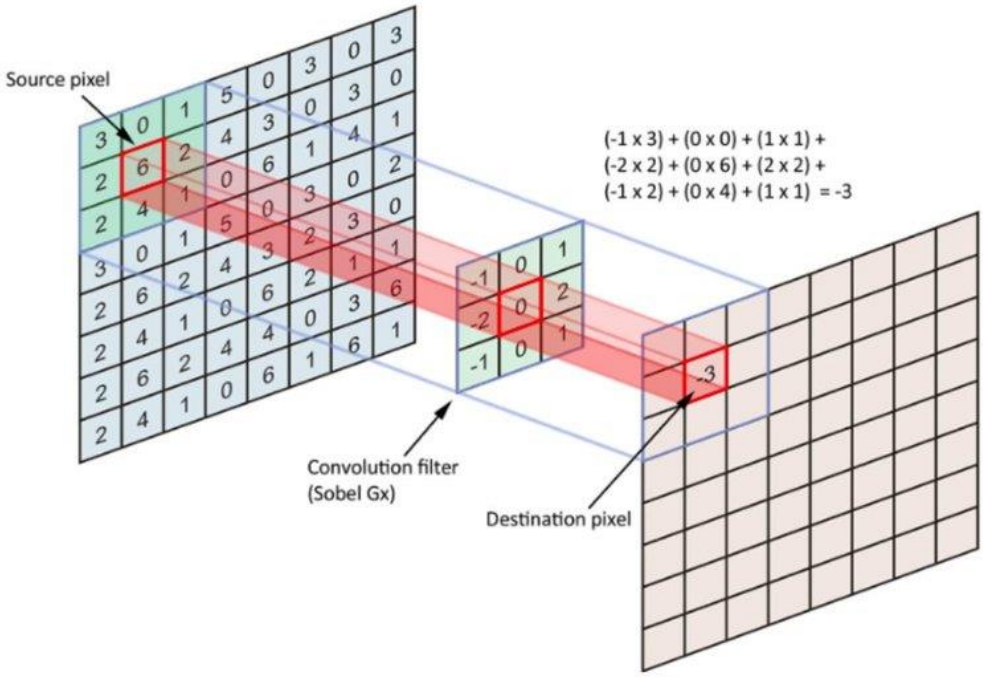


Ilustración 19 Ejemplo de filtro [5]

Este método, llevado a una red neuronal, resultará en una capa convolucional compuesta por un número determinado de filtros, y estos de un tamaño específico, normalmente de forma cuadrada. Estos filtros se irán actualizando según para optimizar la función de coste en cada iteración del entrenamiento, de forma que serán filtros diseñados automáticamente para que sean capaces de predecir el resultado para el que hayamos entrenado nuestra red, siendo así estos filtros unos detectores de características automáticos. Este tipo de red neuronal ha supuesto un cambio de paradigma, no solo en la visión por computador, sino también su aplicación a otro tipo de datos como puede ser el audio.

4.4 Red LSTM

Otro de los tipos de redes neuronales que también se implementarán en este proyecto serán las redes LSTM, o *Long Short-Term Memory* por su nombre en inglés. Este tipo de red, es a su vez un tipo de red que llamamos redes neuronales recurrentes. Las redes neuronales recurrentes están pensadas para ser útiles en el procesado de datos secuenciales, como pueden ser cadenas de texto escrito o valores de mercado. En nuestro caso también es interesante, ya que una señal de audio es secuencial en el tiempo.

Antes de explicar de qué se compone una red LSTM, debemos explicar qué es una red neuronal recurrente, que es la base de la teoría de donde parte las redes LSTM, y cuáles son sus limitaciones.

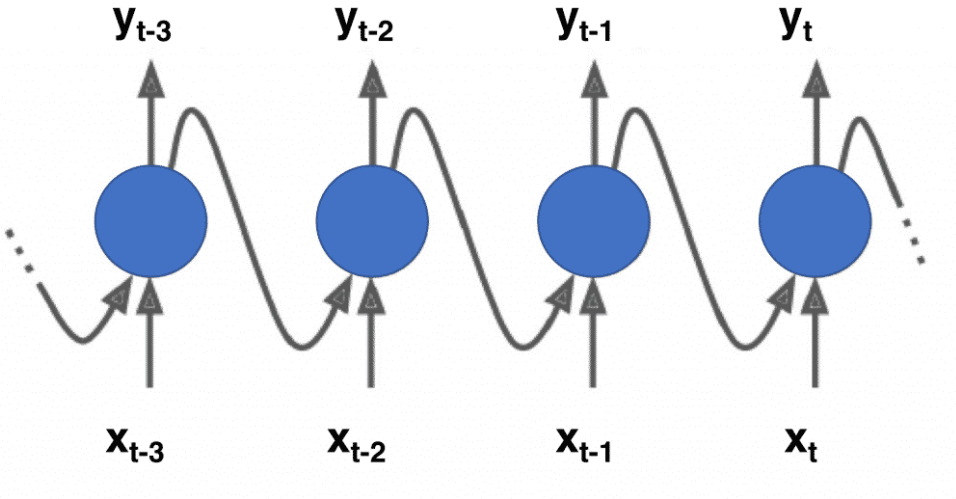


Ilustración 20 Red neuronal recurrente [6]

En el esquema anterior, vemos una red neuronal recurrente desplegada en 4 tiempos. Si nos fijamos, esta neurona no solo tiene conexiones de entrada y de salida, sino que también tiene una conexión añadida para el siguiente valor de la secuencia. De esta forma, cada vez que la neurona infiera un dato, no sólo tendrá información del valor en ese instante, sino que tendrá información de todos los valores anteriores. Esta neurona se expresa matemáticamente de la siguiente forma:

$$y_t = f(Wx_1 + Uy_{t-1} + b)$$

Por este motivo, este tipo de redes son buenas para datos secuenciales, ya que permite analizar el dato dentro del contexto de la secuencia anterior. Aunque estas redes dan buenos resultados en muchas aplicaciones, tienen un problema cuando queremos que la red *recuerde* datos muy antiguos, ya que como vemos en la estructura de la neurona, tendrá más importancia los datos más recientes mientras que los anteriores se irán desvaneciendo.

Para solucionar este problema se desarrollaron las neuronas LSTM, que permiten almacenar información por un largo periodo de tiempo, como su propio nombre indica. Su estructura será la siguiente:

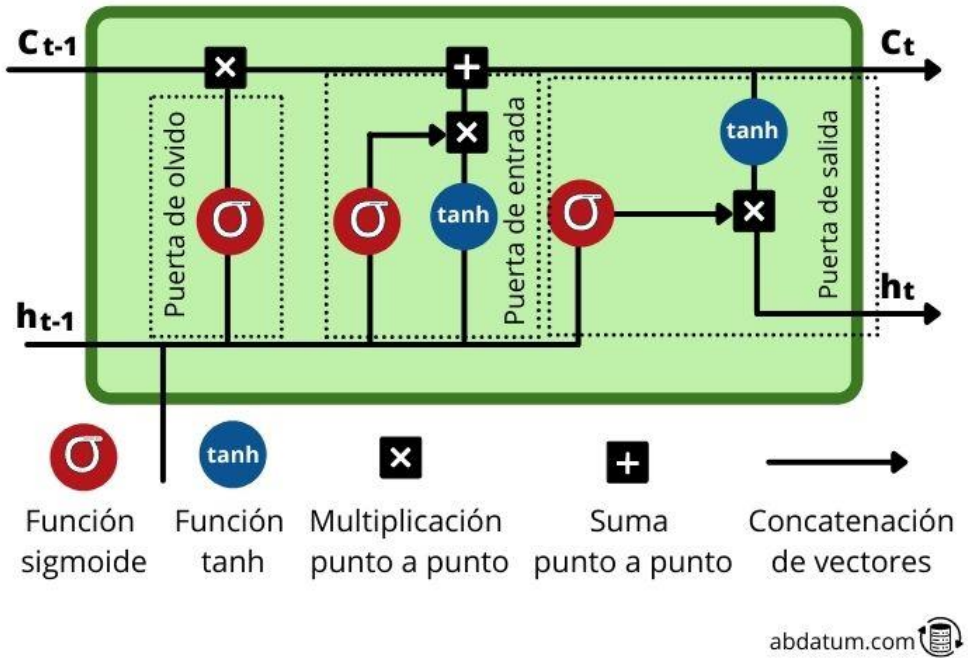


Ilustración 21 Estructura neurona LSTM [7]

Como vemos, esta neurona es una evolución de la neurona recurrente simple. En esta estructura, se añaden algunas etapas diferenciales. Si nos fijamos, aparte de propagar las salidas anteriores como se hacía en la simple, ahora propagamos también un dato de *memoria*. Si atendemos ahora a las etapas, vemos que en primer lugar tenemos la puerta de olvido, donde se decide qué se va a eliminar de los datos de memoria. Posteriormente,

tenemos la puerta de entrada, donde se decide qué nuevos datos vamos a incorporar a nuestra memoria. Finalmente, en la puerta de salida, mediante una función sigmoide, se decide cuál va a ser la salida de la neurona en ese instante. Además, se aplica una función tanh, que es ampliamente utilizada para normalizar los valores entre 1 y -1, y evitar así valores demasiado grandes que dificulten el entrenamiento al aumentar en exceso el gradiente.

5 BASE DE PRUEBAS

Antes de comenzar con los experimentos, se ha desarrollado una base de pruebas con la poder probar los diferentes procesos de forma eficiente y ordenada. Los proyectos de *machine learning* pueden llegar a ser proyectos complejos, en los que una idea inicial se va transformando según se realizan experimentos y se prueban diferentes métodos de procesamiento. Por este motivo, se ha considerado importante mantener un orden en los datos y los diferentes procesamientos realizados a los mismos, y por otro lado a los resultados obtenidos en los diferentes procesamientos de los diferentes modelos desarrollados. Para llevar a cabo esto se ha realizado una librería Python y se ha creado una base de datos específica para el objetivo perseguido.

Ilustración 22Herramientas utilizadas

5.1 Base de datos

Se ha creado una base de datos en Microsoft SQL Server, en la que se guardarán por un lado los datos con los que estamos trabajando (audios, categorización, *dataset* de procedencia, etc.), y los modelos junto con los resultados obtenidos en los diferentes entrenamientos del modelo. Se tendrá por lo tanto las tres siguientes tablas principales:

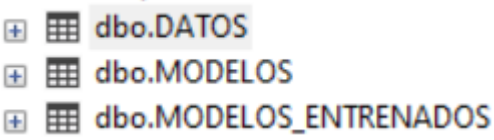


Ilustración 23 Tablas principales base de datos

En la tabla DATOS es donde tendremos guardados la información de los *datasets* utilizados y ya presentados previamente. Estos datos serán posteriormente procesados de diversas formas, para lo cual se crearán otras tablas enlazadas a esta misma, donde se definan estas propiedades procesadas para cada audio de la tabla datos. Esto último será especialmente útil para poder repetir pruebas sobre determinadas propiedades sin tener que volver a calcularlas, con el coste computacional que conlleva determinados cálculos. Las columnas de esta tabla serán:

- ID (int): identificador único del audio.
- PATH (varchar(200)): ruta del archivo de audio.
- EMOTION (varchar(50)): categorización de la emoción del audio.
- DATABASE (varchar(50): dataset del que procede el audio.

Por otro lado, tendremos la tabla de MODELOS. En esta tabla es donde guardaremos los diferentes modelos de *machine learning* implementados. Será de especial utilidad para, por un lado, tener un registro de los diferentes modelos con los que se ha llegado a experimentar, como por otro lado para volver a formar el mismo modelo en base a la descripción guardada del mismo. Para esta descripción se ha usado un formato *json*, que es un formato ampliamente extendido en programación para el almacenamiento y el intercambio de datos. La librería

que posteriormente explicaremos en mayor profundidad se ha preparado para interpretar y formar los modelos a partir de esta descripción *json*. Las columnas de la tabla serán:

- ID (varchar(50)): Identificador único del modelo.
- DESCRIPCION (varchar(50)): Descripción para dar un poco más de información acerca del modelo.
- ESTRUCTURA (text): json con la descripción de la estructura del modelo.

Ejemplo de modelo descrito en formato json:

```
[
      {
        "name": "CapaInput",
        "type": "Input",
        "shape": [370, 1]
      },
      {
        "name": "convolucional_1",
        "type": "Conv1D",
        "filters": 64,
        "kernel_size": 3,
        "activation": "relu"
      },
      {
        "name": "BatchNorm1",
        "type": "BatchNormalization"
      },
      {
        "name": "dropout_1",
        "type": "Dropout",
        "percentage": 0.8
      },
      {
        "name": "convolucional_2",
        "type": "Conv1D",
        "filters": 32,
        "kernel_size": 3,
        "activation": "relu"
      },
      {
        "name": "BatchNorm2",
        "type": "BatchNormalization"
      },
      {
```

```
        "name": "dropout_2",
        "type": "Dropout",
        "percentage": 0.8
      },
      {
        "name": "flatten_1",
        "type": "Flatten"
      },
      {
        "name": "dense_1",
        "type": "Dense",
        "units": 32,
        "activation": "relu"
      },
      {
        "name": "dense_1",
        "type": "Dense",
        "units": 6,
        "activation": "softmax"
      }
]
```

Finalmente, tendremos la tabla MODELOS_ENTRENADOS. En se guardarán todos los entrenamientos realizados a estos modelos, las características del entrenamiento, sus pesos, métricas para evaluar su precisión… y todo a nivel de época, es decir a cada época entrenada será guardada y recuperada posteriormente. Las columnas de dichas tablas serán:

- ID (int): identificador único del registro.
- MODELO (varchar(50)): identificador del modelo.
- OPTIMIZADOR (varcahr(50)): optimizador implementado en el entrenamiento.
- LOSS (varchar(50)): función de coste implementada.
- TRAIN_ACC (flaot): precisión en el dataset de entrenamiento.
- TRAIN_LOSS (float): valor del error en el entrenamiento.
- VAL_ACC (float): precisión en el dataset de validación.
- VAL_LOSS (float): valor del error en validación.
- BATCH_SIZE (int): tamaño del entrenamiento por lotes.
- EPOCH (int): número de época del registro.
- EPOCH_TIME (int): tiempo en segundos que ha tardado en procesar diche época.
- WEGHTS (text): ruta al archivo de pesos del modelo.
- DESCRIPCIÓN (varchar(100)): descripción acerca del proceso de entrenamiento en concreto.

5.2 Librería SER

Junto con la base de datos, se ha desarrollado una librería que sirva como apoyo para los diferentes experimentos, de forma que estos puedan realizarse de forma más rápida y ordenada. La librería está desarrollada en Python, constará de 5 clases fundamentales: Conexión, Datos, Procesado, Model y Stadistics.

Conexión

En esta clase de realizará la conexión con la base de datos, de forma que a través de ella podremos cargar los datos, guardar nuestros modelos y los resultados de los experimentos, recuperar experimentos previos, etc.

Datos

Es la encargada de cargar los datos. Por lo general, la librería extraerá los datos de la propia base de datos, haciendo uso de la clase Conexión como interfaz, pero también contendrá diferentes métodos para el tratado de los archivos de audio y su categorización.

Procesado

Será la clase encargada de procesar los datos, implementando diferentes extractores de features que nos serán útiles posteriormente, como pueden ser MFCC, ZCR, Cepstrum… así como otros procesamientos útiles como puede ser el ventanado de los audios.

Model

Con esta clase se construirá el modelo que posteriormente entrenaremos y testearemos con los datos. Para este proyecto, el modelo se creará usando la librería de *tensorflow*, que es una biblioteca de código abierto desarrollada por Google, y especializada en aprendizaje automático. Como ya se ha comentado, el modelo se vendrá descrito en formato json, por lo que esta clase deberá interpretar este formato para crear el modelo. Además de interpretarlo, también hará uso de la clase Conexión para cargar modelos de la base de datos y para guardarlos.

Statistics

En esta clase tendremos diferentes métodos útiles para obtener estadísticas de los modelos entrenados, tanto de las métricas del proceso de entrenamientos, como de la eficacia y rendimiento de las predicciones del modelo entrenado.

6 EXPERIMENTOS

En este capítulo, se presentarán diferentes experimentos realizados con distintos enfoques del problema. El objetivo será abordar la tarea del reconocimiento de emociones en el lenguaje, no solo con algoritmos de deep learning, sino también con otro tipo de algoritmos de *machine learning*. De esta manera, podremos evaluar posteriormente si tenemos una mejoría en los resultados al aplicar este tipo de algoritmos, junto con las ventajas y desventajas de cada una.

Las características del computador en el que se han implementado los experimentos son las siguientes:

Sistema operativo: Windows 11 Home

Procesador: AMD Ryzen 7 4800HS 2.9Ghz

RAM: 16Gb

GPU: Geforce GTX 1650Ti

6.1 Datos empleados

Realizaremos todos los experimentos con el mismo conjunto de datos, para así poder comparar los distintos modelos a partir de la misma base. Para ellos se escogerán los siguientes *datasets* ya presentados en el capítulo 2:

- Ravdess
- Tess
- Savee
- CremaD (solo los audios High Intensity)

Como vemos, del dataset CremaD se han escogido solo los audios categorizados como HI, que son los que más intensidad tienen en la emoción que expresan. Esto es debido a que el resto de audios, al no estar tan intensificada la emoción, suaviza demasiado las características propias de las mismas, lo que en experimentos previos ha provocado que en el entrenamiento le cueste converger en soluciones con nivel de precisión alto.

Además, aunque en algunos de los *datasets* teníamos algunas emociones añadidas, como tranquilo y sorprendido, se ha decidido eliminar estas muestras, ya que se tenían muy pocos audios de estas categorías en comparación al resto. De esta forma se clasificarán los audios entre disgusto, miedo, felicidad, tristeza, neutral y enfadado.

Todo esto nos deja un total de 4787 muestras de audio. Estos datos se dividirán posteriormente un 88% para el lote de entrenamiento, un 8,4% para el lote de validación y un 3,6% para el lote de test.

6.2 Ajuste de hiperparámetros

Los modelos presentados a continuación, serán los modelos que mejores prestaciones han dado dentro de su tipo de arquitectura. Para ello, se he seguido una metodología para ajustar las dimensiones y los diferentes hiperparámteros, tanto del modelo como del proceso de entrenamiento. El objetivo perseguido en este ajuste ha sido del de conseguir la mejor precisión posible en el dataset de validación, sin que la precisión de este diste demasiado de la precisión del lote de entrenamiento, evitando así un sobreajuste del modelo. Para ayudar a este proceso se ha utilizado la librería de tensorboard, con la cual se puede hacer un seguimiento del entrenamiento del modelo en tiempo real, y así poder realizar los ajustes antes de finalizar el entrenamiento al completo.

Son muchos los hiperparámetros que se pueden ajustar dentro de una red neuronal, como la función de activación, el tamaño de lote de entrenamiento, regularización, dropout, etc. Algunos de ellos han sido ajustados experimentalmente por prueba y error, pero existen unas reglas básicas que nos ayudan a ajustar correctamente algunos de ellos:

- Simplificación del modelo: si vemos que el modelo sobreajusta demasiado a los datos de entrenamiento, lo primero que se debe hacer es simplificar el modelo, ya sea reduciendo el número de neuronas o filtros, o reduciendo el número de capas ocultas. En el caso de que el modelo no sea capaz de aumentar su precisión, se realizará el proceso contrario, añadiendo más parámetros al modelo.
- Tasa *dropout*: en los siguientes experimentos, se puede ver como comúnmente se ha usado un tipo de capa llamada *dropout,* la cual se encarga de eliminar un porcentaje de las conexiones entre las capas durante el proceso de entrenamiento. Este se hace para evitar un sesgo del modelo hacia los datos de entrenamiento, mejorando normalmente los resultados en validación y en test. Al aumentar el porcentaje de *dropout*, dificultaremos el entrenamiento, por lo que normalmente será necesario entrenarlo más épocas, teniendo la precaución de que no poner un porcentaje demasiado alto que pueda hacer que nuestro modelo no sea capaz de mejorar.
- Optimizador: como ya se ha comentado anteriormente, se han empleado optimizadores *RMSProp, SGD* y *Adam,* probando cada uno de ellos en función de si el modelo caía en mínimos locales, o por el contrario tenía mucha variabilidad en los valores de pérdida.

6.3 SVM *features* globales

En primer lugar, se realizan experimentos con el algoritmo de *SVM linear¸* anteriormente explicado. Para este algoritmo, es necesario realizar previamente una extracción de features, o características, que serán a partir de las cuales los vectores de soporte tratarán de identificar cada una de las diferentes categorías. [8]

6.3.1 Extracción de features

Preprocesamiento

En primer lugar, se hace un preprocesamiento del audio, en el que dividimos el audio en pequeños fragmentos de igual tiempo, con un solapamiento entre cada uno de los fragmentos, de forma que el final de un fragmento está presente en el inicio del siguiente fragmento. Posteriormente, se le aplica a cada fragmento la convolución con la ventana Hanning. Este preprocesamiento es similar al que se aplica para calcular los coeficientes MFCC, ya explicados con anterioridad.

***Features* locales**

A cada fragmento de audio le calculamos una serie de características, las cuales llamamos features locales al tratarse de las características del audio en cada instante de tiempo. Las características serán las siguiente:

- ACF Pitch
- Cepstrum pitch
- Instensidad
- MFCC
- ZCR
- LSP

El siguiente gráfico resume el proceso para extraer estas características:

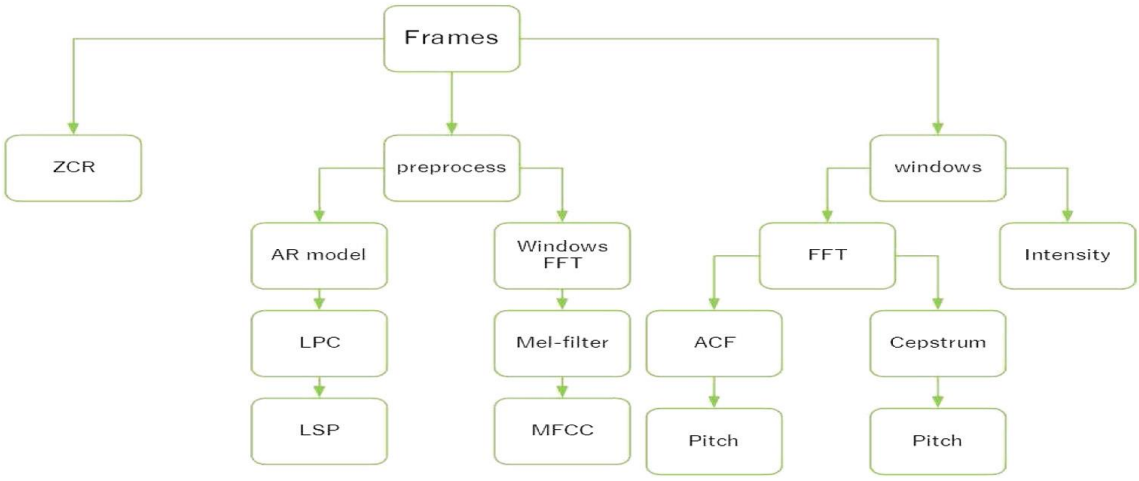


Ilustración 24 Extracción de features locales [8]

***Features* globales**

Una vez calculadas estas características locales, se realizan una serie de cálculos sobre las mismas, por cada uno independientemente, de formas que se caracterice el audio completo, y no solo los pequeños fragmentos. Serán los siguientes:

- Media: media de la característica entre todos los fragmentos.
- Máximo: máximo de todos los fragmentos.
- Mínimo: mínimo de todos los fragmentos.
- Rango: máximo menos mínimo.
- Offset regresión lineal: el valor b de la siguiente recta de ajuste a los datos: y=b+mx.
- Pendiente de la regresión lineal: el valor m de la siguiente recta de ajuste a los datos: y=b+mx,
- Rango intercuartílico: en la distribución de los datos, será el rano entre el primer y el tercer cuartil
- Oblicuidad: medida de la simetría de la distribución respecto su media. Se calcula de la siguiente forma $Oblicuidad = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{x_i-\bar{x}}{\sigma}\right)^3$
- Kurtosis: caracteriza la forma de la distribución.

Finalmente, de cara a homogenizar los datos y facilitar el posterior entrenamiento, se normalizan los resultados entre 0 y 1.

6.3.2 Elección de hiperparámetros

Para escoger el ventanado con mayor rendimiento, con una frecuencia de muestreo de 22050 Hz, se ha hecho un barrido por los siguientes tamaños de Ventana y saltos de Ventana, para posteriormente comprobar su eficacia con el modelo SVM:

- Tamaño 20 y salto 10
- Tamaño 40 y salto 10
- Tamaño 40 y salto 20
- Tamaño 60 y salto 10
- Tamaño 60 y salto 20
- Tamaño 60 y salto 30
- Tamaño 80 y salto 10
- Tamaño 80 y salto 20

- Tamaño 80 y salto 30
- Tamaño 80 y salto 40
- Tamaño 100 y salto 10
- Tamaño 100 y salto 20
- Tamaño 100 y salto 30
- Tamaño 100 y salto 40
- Tamaño 100 y salto 50

6.3.3 Resultados

Tras la implementación y ajuste del modelo SVM, se han obtenido los siguientes resultados para cada uno de los enventanados propuestos:

Tamaño (ms)	Solapamiento (ms)	Accuracy (%)
20	10	81.77
40	10	82.23
40	20	82.69
60	10	80.85
60	20	82.58
60	30	81.19
80	10	81.19
80	20	80.96
80	30	81.08
80	40	79.35
100	10	81.08
100	20	82.12
100	30	80.39
100	40	80.39
100	50	81.54

Como vemos, llega a haber una diferencia notable entre los diferentes enventanados, por lo que escogeremos el tamaño de ventana y solapamiento con el que más precisión se ha obtenido, que si observamos la tabla será tamaño de ventana 40ms con un solapamiento de 20ms, obteniendo una precisión de 82.69%.

Presentamos a continuación la matriz de confusión para el mencionado modelo, la cual nos sirve para visualizar los aciertos y fallos de cada una de las categorías:

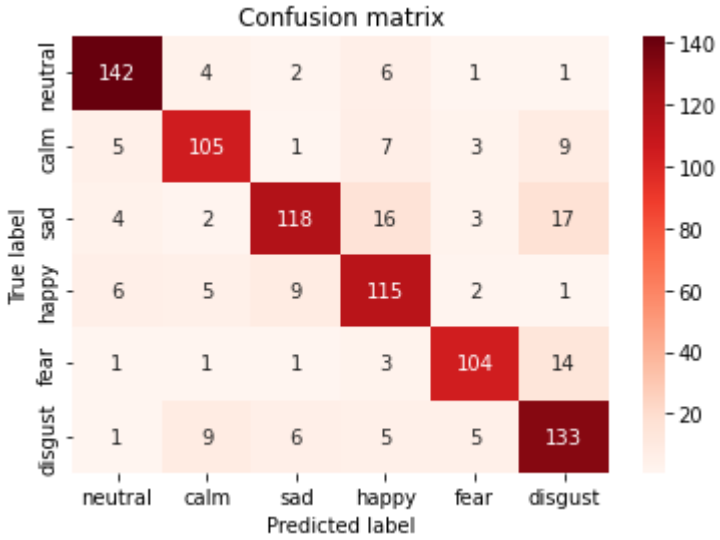


Ilustración 25 Matriz de confusión con modelo SVM

6.4 Red Neuronal Densamente conectada

En esta ocasión, se realiza el entrenamiento de una red neuronal densamente conectada, con exactamente el mismo tratamiento de los datos que se realizó en el apartado 6.2, es decir tomaremos los features globales de cada audio para tratar de identificarlos. Como hemos visto, este tipo de red neuronal puede ser buena en clasificación cuando la separación entre clases no es estrictamente lineal. El modelo propuesto es el siguiente:

Capa	Tipo	Forma de Salida	Parámetros
dense_6	Dense	(None, 256)	94,976
dropout_4	Dropout	(None, 256)	0
dense_7	Dense	(None, 128)	32,896
dropout_5	Dropout	(None, 128)	0
dense_8	Dense	(None, 64)	8,256
dropout_6	Dropout	(None, 64)	0
dense_9	Dense	(None, 32)	2,080
dropout_7	Dropout	(None, 32)	0
dense_10	Dense	(None, 6)	198

Tabla 1 Red densamente conectada para features globales

Como vemos, tenemos 5 capas densamente conectadas con capas *dropout* intercaladas. Estas capas *dropout* se encargarán de eliminar de forma aleatoria un porcentaje de las conexiones entre las dos capas en la que se sitúa.

Esto es útil para evitar el sobreajuste, evitando además que ciertas conexiones tengan más peso en la clasificación que otras. Este porcentaje es un hiperparámetro ajustable, en este caso se ha ajustado a 20% para todas las capas *dropout*. La función de activación de las capas *Dense* serán funciones sigmoidales para todas excepto la última capa, que será una función *softmax* que determine cuál de las 6 clases es más probable para ese audio.

Se entrena dicho modelo 60 épocas, con un tamaño de lote de 16, la función de pérdida 'categorical_crossentropy' y el optimizador 'adam'. Vemos a continuación el gráfico de la evolución por época tanto del valor de pérdida como de la precisión.

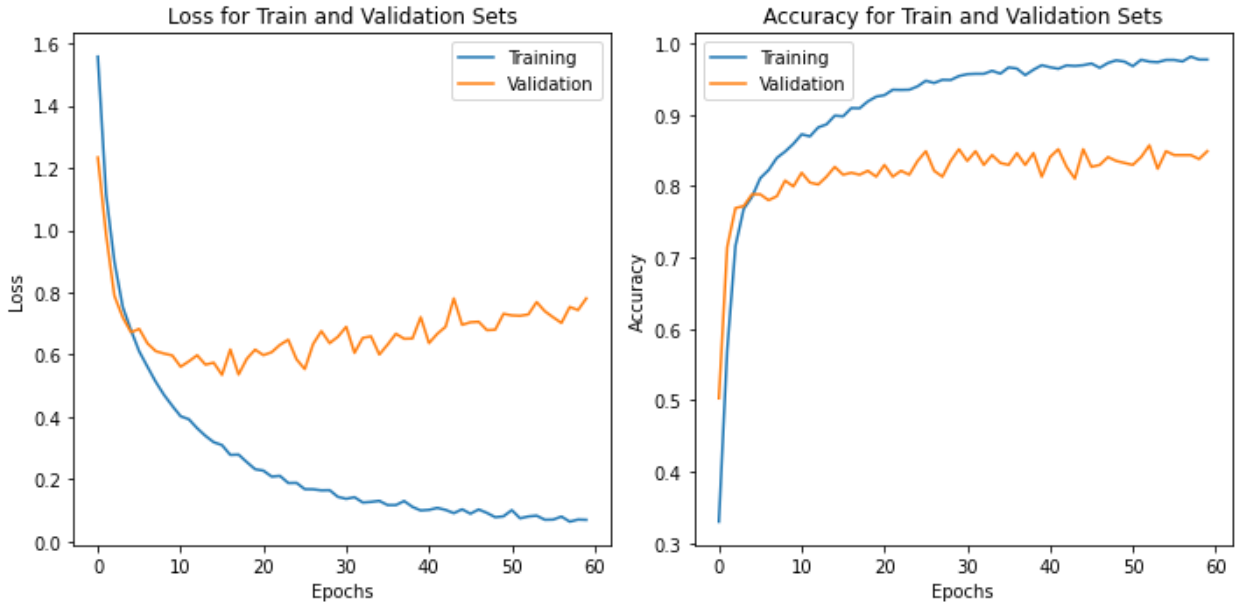


Ilustración 26 Entrenamiento de red neuronal densamente conectada

Estos gráficos los usaremos para comparar los entrenamientos de las distintas redes neuronales, ya que sirven para visualizar la evolución del entrenamiento, comprobando si el modelo ha llegado a converger en unos valores de pérdida los cuales ya no puede mejorar, y viendo también si el modelo llega a sobreajustarse a los datos de entrenamiento, sin mejorar en los datos de validación. En este caso, ya vemos como a partir de la época 20 el modelo no consigue mejorar. Mostramos a continuación la matriz de confusión para los datos de test:

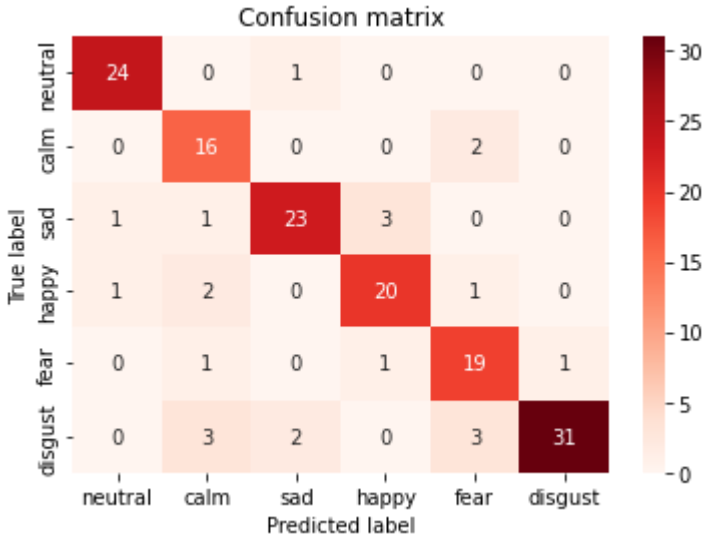


Ilustración 27 Matriz de confusión de la red densamente conectada.

De esta forma, los resultados obtenidos son los siguientes:

- Precisión en entrenamiento: 92.72%
- Precisión en validación: 84.89%
- Precisión en test: 85.25%

Como vemos, ha habido una leve mejora en test si comparamos con el modelo SVM Lineal implementado anteriormente.

6.5 Red Neuronal Convolucional 1D

Sabemos que normalmente las redes convolucionales sirven como extractores de características, obteniéndolas a partir de los datos en bruto como sucede en el caso de las imágenes. Esto será algo con lo que experimentaremos posteriormente, pero se ha querido realizar un experimento con redes convolucionales con las características extraídas para los modelos anteriores. Esto es porque este tipo de modelos son buenos buscando patrones, y en ocasiones al realizar un cálculo previo de características puede llegar a tener una información más rica de la que nutrirse.

Para ellos se entrenará el siguiente modelo convolucional de 1 dimensión, para las características globales calculadas de los audios, un total de 60 épocas, con un tamaño de lote de 16, función de pérdida *categorical crossentropy*, y optimizador *Adam*. La función de activación para las capas convolucionales será *relu*, y el porcentaje de dropout de un 80% en cada capa.

Capa	Tipo	Forma de Salida	Parámetros
conv1d	Conv1D	(None, 368, 64)	256
batch_normalization	BatchNormalization	(None, 368, 64)	256
dropout	Dropout	(None, 368, 64)	0
conv1d_1	Conv1D	(None, 366, 32)	6,176
batch_normalization_1	BatchNormalization	(None, 366, 32)	128
dropout_1	Dropout	(None, 366, 32)	0
flatten	Flatten	(None, 11,712)	0
dense	Dense	(None, 32)	374,816
dense_1	Dense	(None, 6)	198

Tabla 2 Red convolucional 1D

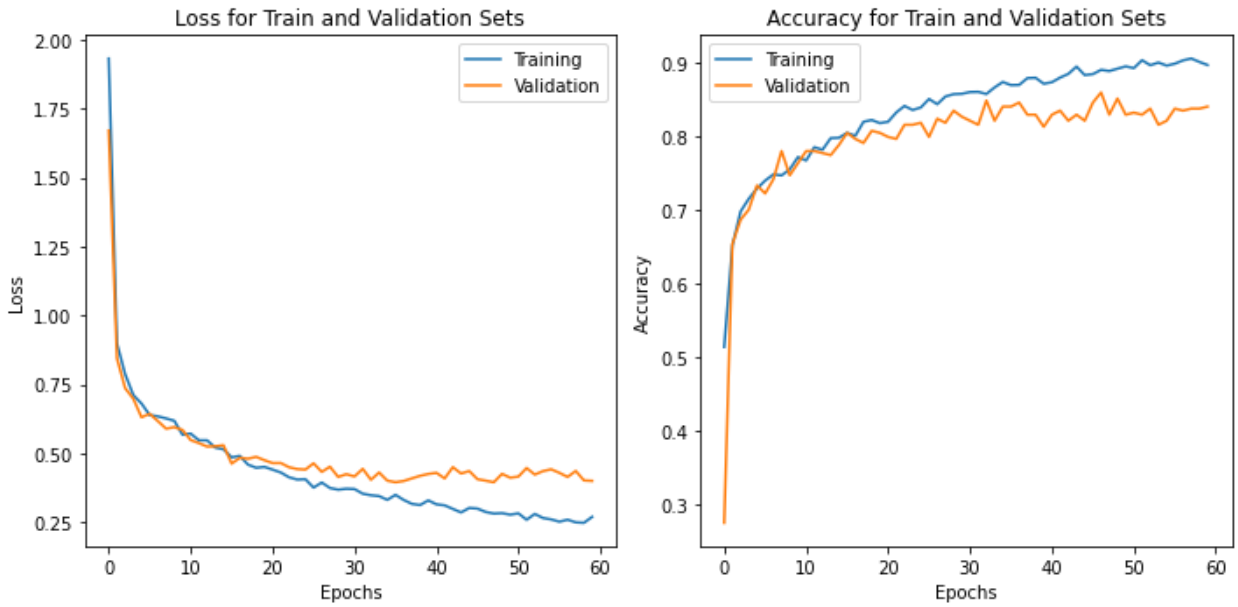


Ilustración 28 Entrenamiento de red neuronal convolucoinal 1D

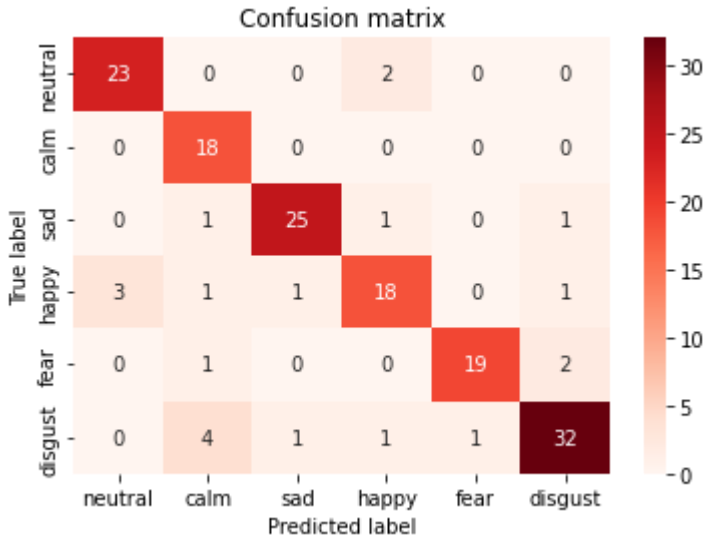


Ilustración 29 Matriz de confusión de red neuronal convolucional 1D

A la vista de los resultados, podemos determinar que son bastante positivos, ya que en primer lugar se produce un menor sobreajuste a los datos de entrenamiento que en el caso de la red densamente conectada, teniendo un 89.72% de precisión en entrenamiento y un 84.07% de precisión en validación. Lo positivo llega en el análisis del lote de test, teniendo un 86.53% de precisión, el valor más alto hasta el momento.

6.6 Red Neuronal Convolucional 2D

Hasta ahora, se han realizado experimentos con *features* calculados previamente, de forma que los modelos han tenido que clasificar los audios en función de estas características. Aunque como vemos, se llegan a tener unos buenos resultados para estos modelos propuestos, estos son en base a una serie de características escogidas por nosotros y que han de ser calculadas antes de la inferencia, siendo muchas de ellas muy costosas computacionalmente. Por ello, se presentan ahora una serie de modelos que sirven en sí mismos como extractores de *features*.

Para esta tarea se ha implementado una red neuronal convolucional, que como ya se ha explicado, estas están compuestas por una serie de filtros que serán aprendidos durante el entrenamiento, y que serán los encargados

de buscar las características adecuadas para la tarea de clasificación. Posteriormente a las capas de convolución se aplicarán una serie de capas densamente conectadas, que harán la tarea de clasificación según la información obtenida por los filtros anteriormente aplicados.

6.6.1 Preparación de los datos

Esta técnica ha sido comúnmente empleada en aplicaciones de visión por ordenador, por la facilidad que tienen estos modelos de encontrar características en imágenes, que normalmente son matrices de 2 o 2 dimensiones (según si es en color o no). Para el caso del audio, la aplicación de este tipo de redes no es tan trivial, ya que necesitamos adaptar los datos para un formato parecido al de las imágenes. En este caso, se ha optado por el cálculo del MFCC de cada audio, ya que por un lado es una representación del audio que ha demostrado tener información relevante para caracterizar al mismo, y por otro lado porque este puede ser representado como una imagen, en la que en el eje y tenemos los valores de los diferentes coeficientes, con una escala de color, y en el eje x tenemos los diferentes instantes de tiempo. Vemos a continuación de un MFCC calculado representado como imagen:

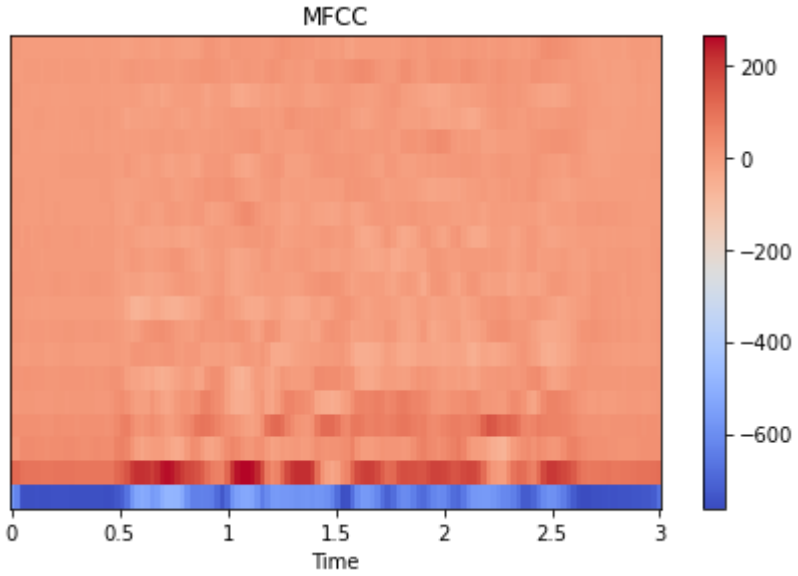


Ilustración 30 Representación del MFCC

El número de coeficientes MFCC que calcularemos para cada audio serán 20. Además, para facilitar la tarea a la red, los coeficientes MFCC se normalizarán entre 0 y 1.

6.6.2 Modelo y resultados

El modelo concreto que implementaremos será el siguiente:

Capa	Tipo	Forma de Salida	Parámetros
conv2d_4	Conv2D	(None, 20, 130, 128)	3,328
max_pooling2d_2	MaxPooling2D	(None, 10, 65, 128)	0
dropout_2	Dropout	(None, 10, 65, 128)	0
conv2d_5	Conv2D	(None, 10, 65, 64)	204,864
max_pooling2d_3	MaxPooling2D	(None, 5, 33, 64)	0
dropout_3	Dropout	(None, 5, 33, 64)	0

Capa	Tipo	Forma de Salida	Parámetros
conv2d_6	Conv2D	(None, 5, 33, 32)	18,464
conv2d_7	Conv2D	(None, 5, 33, 16)	4,624
dropout_4	Dropout	(None, 5, 33, 16)	0
flatten_1	Flatten	(None, 2,640)	0
dense_2	Dense	(None, 32)	84,512
dense_3	Dense	(None, 6)	198

Tabla 3 Modelo de red convolucional 2D

Como vemos, tenemos 4 capas convolucionales de 2 dimensiones, con una función de activación *relu* y un tamaño de *kernel* de 5 para las dos primeras y de 3 para las dos últimas. Además, al igual que se ha hecho en modelos anteriores se han añadido capas *dropout* para evitar el sobreajuste, en este caso con un porcentaje de *dropout* del 40%, escogidos experimentalmente. Por otro lado, vemos una capa que no habíamos usado hasta ahora, que son las llamadas *max pooling*. Esta capa, aplicará un *kernel* que recorrerá toda la imagen quedándose con el valor máximo de la región, lo que hará que reduzca los datos para la siguiente capa. Finalmente, tendremos una capa densamente conectada con activación *relu* y una última capa dense con activación *softmax* para la clasificación de las emociones.

Se entrena dicho modelo 100 épocas, con un tamaño de lotes de 16, función de pérdida 'categorical_crossentropy', optimizador RMSProp.

Se obtienen los siguientes resultados:

- Precisión en el lote de entrenamiento: 91.58%
- Precisión en el lote de validación: 78.85%
- Precisión en el lote de test: 82.05%

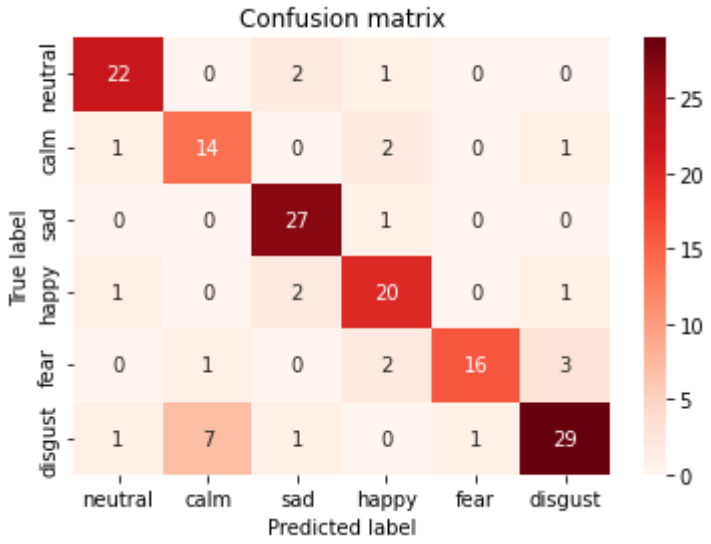


Ilustración 31 Matriz de confusión para red neuronal convolucional 2D

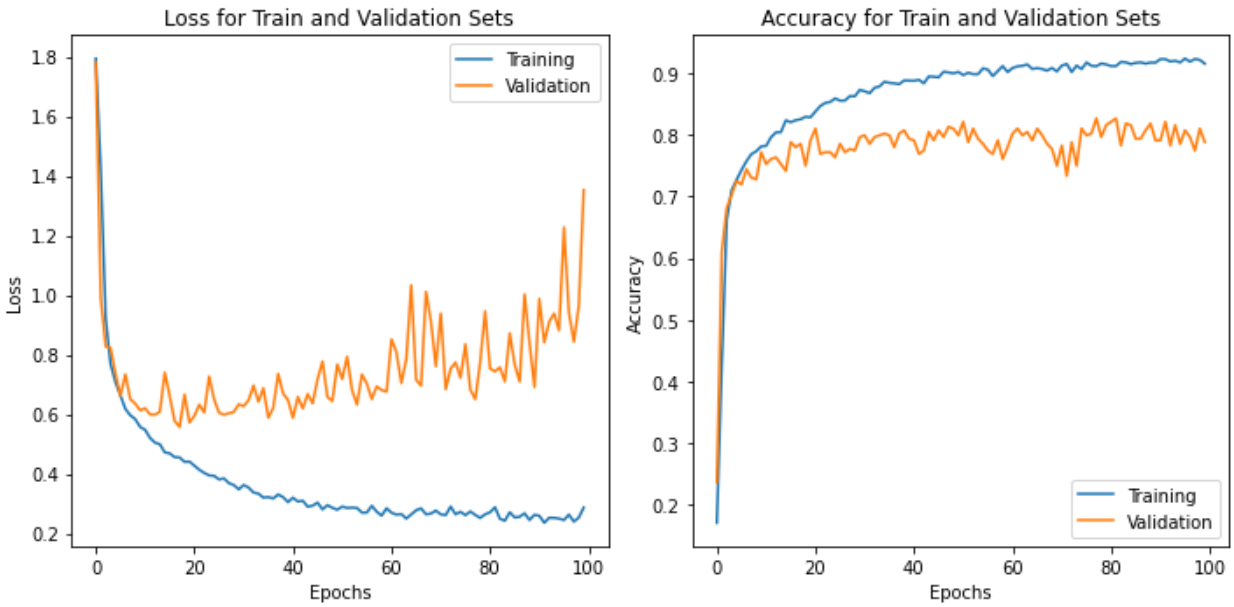


Ilustración 32 Entrenamiento de red neuronal convolucional de 2D

Llegamos a tener una precisión de 82.05% en test, que queda por debajo de lo conseguido con la red neuronal para *features* globales, en el que se obtuvo un 85.25%. Además, atendiendo a las gráficas del entrenamiento, vemos como el sobreajuste para este modelo es mucho mayor. Este se ha tratado de reducir un poco incrementando el porcentaje de *dropout* de 20% al 40%, lo cual ha supuesto una ligera mejora, pero no suficiente.

6.7 LSTM

Como ya se ha introducido, los modelos LSTM son modelos especialmente buenos cuando estamos ante datos secuenciales. Dado que estamos trabajando con datos de audio, que son secuenciales en el tiempo, se ha visto interesante tratar de plantear un modelo de este tipo, para ver si es capaz de mejorar lo ya implementado. Para ello, se ha tenido que preparar en primer lugar los datos para adaptarlo al formato de la red que vamos a implementar.

En primer lugar, preparamos los audios para que tengan todos la misma duración, ya que todas las muestras deberán tener el mismo tamaño de secuencias para el entrenamiento. Cortamos los audios para una duración de 8s, rellenando con 0 en caso de no llegar a dicha duración. Luego, realizamos un enventanado de cada audio, con un tamaño de ventana de 2048 y un salto entre ventanas de 512, siendo casa ventana una secuencia del audio. Para cada secuencia calculamos posteriormente las características *ZRC, RMS y MFCC.* Así ya tenemos los audios con las características secuenciadas en el tiempo preparadas para la red LSTM.

Se entrenará con estos datos la siguiente arquitectura de red, con dos capas LSTM y una última capa *Dense* con activación softmax para realizar la clasificación de los audios. El entrenamiento realizado es de 200 épocas, con tamaño de lote de 6, función de pérdida *categorical crossentropy* y optimizador *RMSProp.*

Capa	Tipo	Forma de Salida	Parámetros
lstm	LSTM	(None, 352, 64)	20,480
lstm_1	LSTM	(None, 64)	33,024
dense	Dense	(None, 6)	390

Tabla 4 Modelo LSTM

En este caso, tenemos un entrenamiento más costoso a nivel de cálculo computacional, ya que este tipo de redes son más complejas. Los resultados obtenidos son los siguientes:

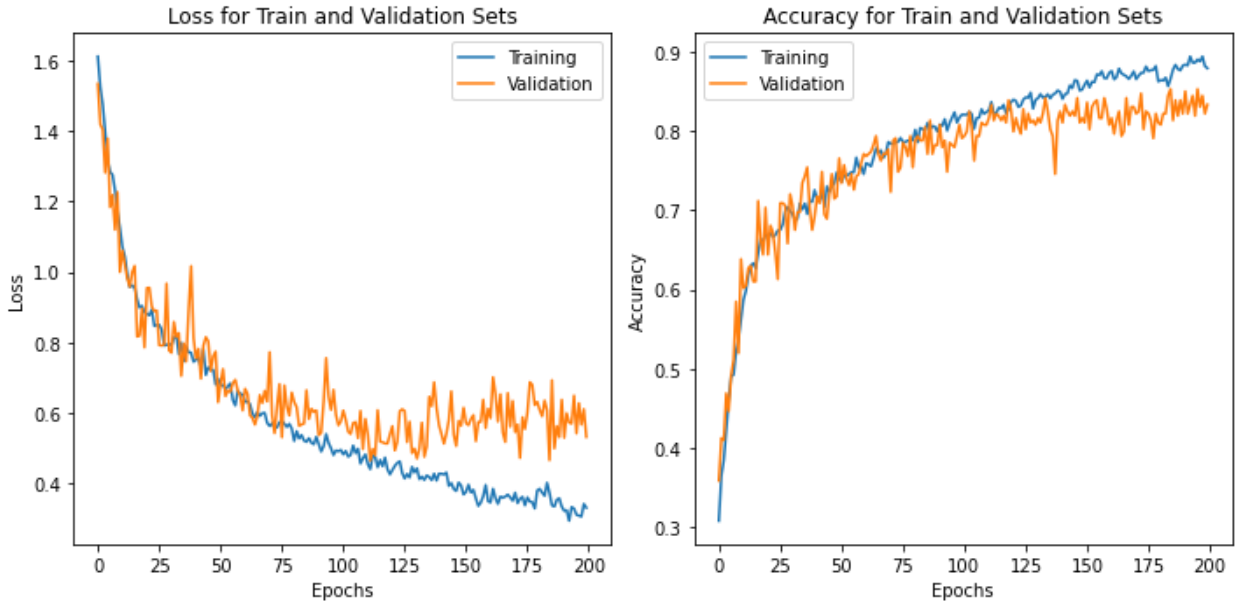


Ilustración 33 Entrenamiento LSTM

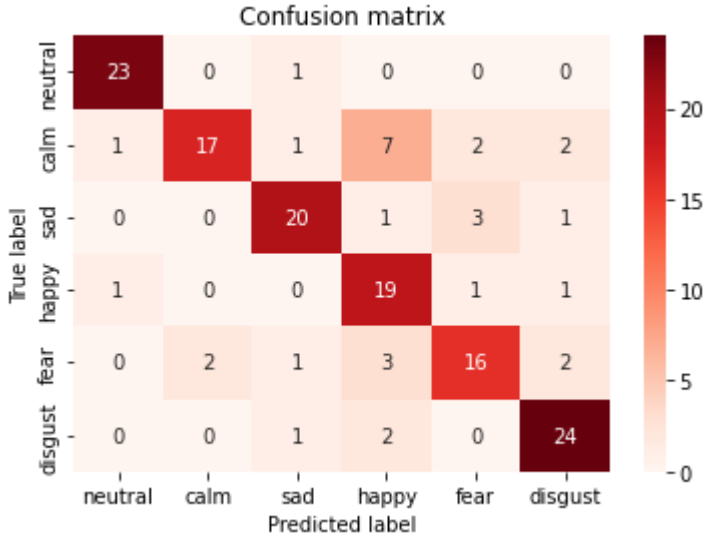


Ilustración 34 Matriz de confusión modelo LSTM

Si atendemos a la precisión de entrenamiento y de validación, 87.94% y 83.93%, podemos entrever que no tenemos tanto sobreajuste como si lo llegamos a tener en las redes de tipo convolucional. Por otro lado, en cuanto a la precisión en test, si que nos quedamos un poco por debajo de otras redes mostradas anteriormente, obteniendo un 78.28%.

6.8 Convolucional 1D y LSTM

Finalmente, se trata de crear un modelo juntando los conceptos implementados en los modelos anteriores. Este modelo estará compuesto por unas primeras capas convolucionales de una dimensión, que serán las encargadas de extraer las *features* locales de nuestro audio, de forma automática, a diferencia de como lo hacíamos en el primer experimento realizado. Posteriormente, tendremos una capa LSTM, que será la encargada de, a través de las *features* locales capturadas por las capas convolucionales, obtener las *features* globales a través de la cuales

determinar el sentimiento de nuestros audios. Finalmente, tendremos unas capas densamente conectadas que harán la clasificación.

La forma en la que se pasarán los datos al modelo será similar a como se hizo anteriormente en los experimentos con redes convolucionales, donde se realiza un cálculo de 20 coeficientes mfcc del audio enventanado en el tiempo. En el experimento anterior, estos mfcc se pasaban a una red convolucional 2D, de forma que se trataba como una imagen, en este caso se hará diferente procesándolos con capas convolucionales 1D. Así, los datos de entrada tendrán el formato (130, 20, 1), donde la primera dimensión será la parte temporal de nuestro audio, a cada cual se les extraerán las características con las capas convolucionales que alimentarán a las capas LSTM.

6.8.1 Modelo y resultados

El modelo concreto entrenado es el siguiente:

Capa	Tipo	Forma de Salida	Parámetros
Input_MELSPECT	InputLayer	(None, 517, 128, 1)	0
Conv_1_MELSPECT	TimeDistributed (Conv2D)	(None, 517, 128, 32)	128
BatchNorm_1_MELSPECT	TimeDistributed (BatchNorm)	(None, 517, 128, 32)	128
Activ_1_MELSPECT	TimeDistributed (Activation)	(None, 517, 128, 32)	0
MaxPool_1_MELSPECT	TimeDistributed (MaxPool)	(None, 517, 64, 32)	0
Drop_1_MELSPECT	TimeDistributed (Dropout)	(None, 517, 64, 32)	0
Conv_2_MELSPECT	TimeDistributed (Conv2D)	(None, 517, 64, 32)	3,104
BatchNorm_2_MELSPECT	TimeDistributed (BatchNorm)	(None, 517, 64, 32)	128
Activ_2_MELSPECT	TimeDistributed (Activation)	(None, 517, 64, 32)	0
MaxPool_2_MELSPECT	TimeDistributed (MaxPool)	(None, 517, 32, 32)	0
Drop_2_MELSPECT	TimeDistributed (Dropout)	(None, 517, 32, 32)	0
Conv_3_MELSPECT	TimeDistributed (Conv2D)	(None, 517, 32, 64)	6,208
BatchNorm_3_MELSPECT	TimeDistributed (BatchNorm)	(None, 517, 32, 64)	256
Activ_3_MELSPECT	TimeDistributed (Activation)	(None, 517, 32, 64)	0
MaxPool_3_MELSPECT	TimeDistributed (MaxPool)	(None, 517, 16, 64)	0
Drop_3_MELSPECT	TimeDistributed (Dropout)	(None, 517, 16, 64)	0
Conv_4_MELSPECT	TimeDistributed (Conv2D)	(None, 517, 16, 128)	73,856
BatchNorm_4_MELSPECT	TimeDistributed (BatchNorm)	(None, 517, 16, 128)	512
Activ_4_MELSPECT	TimeDistributed (Activation)	(None, 517, 16, 128)	0

Capa	Tipo	Forma de Salida	Parámetros
MaxPool_4_MELSPECT	TimeDistributed (MaxPool)	(None, 517, 8, 128)	0
Drop_4_MELSPECT	TimeDistributed (Dropout)	(None, 517, 8, 128)	0
Reshape_MELSPECT	Reshape	(None, 517, 1024)	0
LSTM_1_MELSPECT	LSTM	(None, 517, 64)	278,784
LSTM_2_MELSPECT	LSTM	(None, 64)	33,024
Output_MELSPECT	Dense	(None, 6)	390

Tabla 5 Modelo Convolucional 1D + LSTM

Entrenamos dicho modelo un total de 45 épocas, con un tamaño de lote de 16 y el optimizador Adam. Los resultados del entrenamiento son los siguientes:

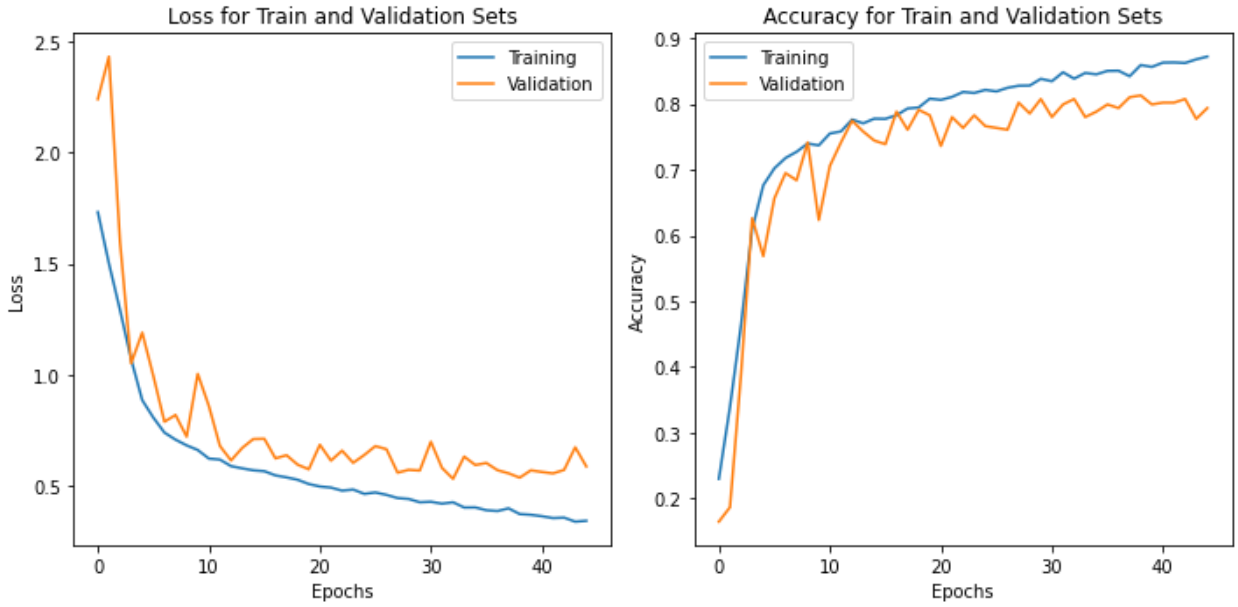


Ilustración 35 Entrenamiento CNN1D+LSTM

Obtenemos unos valores de precisión en entrenamiento de 87.22% y de validación del 79.40% Como vemos, a partir de la época 40 nuestro modelo ya no consigue mejorar su valor de pérdida en validación, por lo que nos quedamos con el modelo hasta dicho punto. El resultado del test es el siguiente:

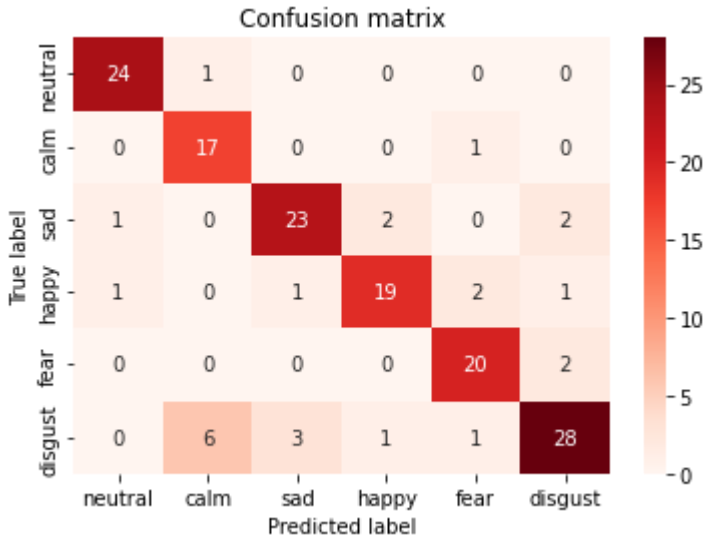


Ilustración 36 Matriz de confusión CNN 1D + LSTM

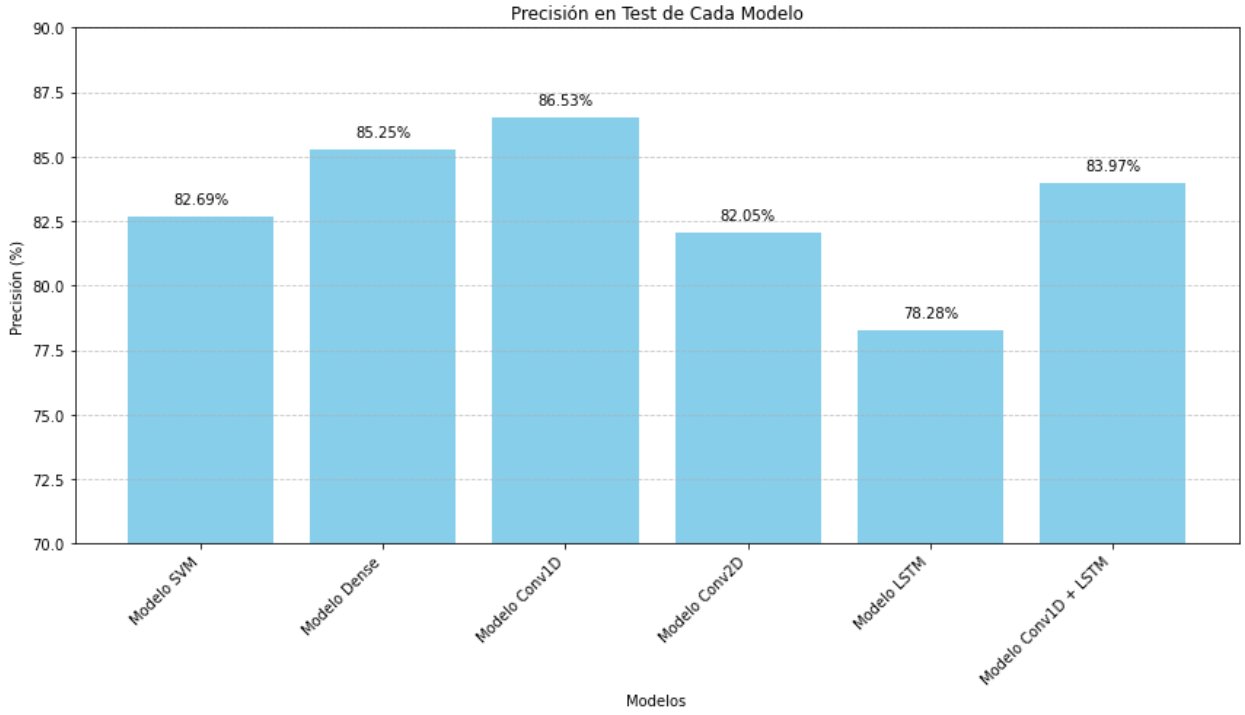
La precisión en test es del 83.97%, siendo así el modelo con mejores prestaciones aparte de los modelos entrenamos a partir de la extracción de características explícita.

7 CONCLUSIONES

El objetivo del presente capítulo será por un lado evaluar los modelos implementados, comparando las prestaciones de cada uno de ellos. Por otro lado, se estudiará de que forma se podría mejorar los resultados obtenidos, así como futuros trabajos de investigación que pueden ser interesantes para avanzar en esta línea de investigación.

7.1 Estudio de los resultados

Se muestra a continuación un gráfico comparativo de los valores de precisión obtenidos para cada unos de los modelos implementados, todo ello para los datos de test, que son los que nos dan una visión más cercana al rendimiento de los modelos para datos que nunca ha visto ni han sido usados para ajustar sus parámetros.



Analizamos en primer lugar los 3 primeros modelos, que son los modelos entrenados a partir del conjunto de características globales previamente calculadas. Las prestaciones de estos modelos han sido buenas, con unos porcentajes de precisión entre 82.62% y 86.53%. Por un lado tenemos el modelo SVM, con el que se han obtenido resultados por encima de lo esperado para ser uno de los modelos más sencillos que se han implementado. Posteriormente, se han entrenado modelos de redes neuronales densamente conectadas y con capas convolucionales, los cuales han mejorado los resultados entre un 3%-4% comparándolo con el modelo SVM, siendo el modelo convolucional el mejor de los dos estando un 1% por encima. En el caso de las capas dense, la mejora probablemente se debe a que es capaz de adaptarse mejor a un entorno no lineal como es el caso, pudiendo delimitar con mejor precisión las regiones de los conjuntos de características que pertenecen a una clase y a otra. Con respecto al modelo convoluciones, se ve beneficiado también por esto mismo, añadiendo además la capacidad que tienen las capas convolucionales de encontrar relaciones y patrones dentro de las características. De esta forma, tal como era esperado, se obtiene una mejora notoria al introducir técnicas de redes neuronales y *deep learning* para nuestra tarea de clasificación. Además, es importante destacar la relevancia del conjunto de características escogidas, ya que han demostrado ser capaces de diferenciar las emociones del habla con gran eficacia.

Posteriormente, se trata de analizar una serie de algoritmos para los cuales no se ha extraído una serie de características específicas, pensadas especialmente para esta tarea de clasificación, sino que se ha tratado de que

el propio modelo sea el encargado de encontrar estas características, para comprobar si este es capaz de encontrar patrones en el audio mejores de los obtenidos. Para ello, se han propuesto 3 modelos, siendo el último de ellos un híbrido de los dos primeros. En primer lugar, se tiene un modelo convolucional 2D, con el que tan solo a partir de los coeficientes *mfcc* del audio, forma en la que se le pasarán los datos al modelo, se pretende encontrar a partir de los filtros aprendidos por las capas convolucionales, una serie de patrones que sean capaces de diferenciar las diferentes emociones propuestas. Es resultado obtenido es similar al obtenido con el modelo SVM, por lo que podemos determinar que es capaz de encontrar patrones igual de buenos que los calculados por nosotros. Luego, se ha entrenado un modelo LSTM, con el que, a partir de unos ciertos datos temporales de nuestros audios, se pretende que el modelo sea capaz de diferenciar las diferentes clases a partir del avance de estas características a lo largo del tiempo, haciendo un proceso similar al empleado en el cálculo de las características globales, pero de forma automática. Los resultados ofrecen la peor precisión de los 6 modelos entrenados, pero no queda lejos y nos da la intuición para el siguiente modelo entrenado. Este último modelo, será un conjunto de capas convolucionales 1D, junto unas capas LSTM, de forma que se busca emular de forma automática el cálculo de las características locales y globales, y clasificar los audios en función de estas. Como vemos, este último modelo mejora la precisión dentro de este subconjunto de 3 modelos entrenados, demostrando la eficacia que puede llegar a tener este planteamiento.

Finalmente, analizando los dos conjuntos de modelos, vemos como los modelos basados en el conjunto de características previamente calculados quedan ligeramente por encima de los modelos que hacen de extractores de características por si mismos. Una de las principales causas de esto, es debido a la falta de datos etiquetados para entrenar el modelo, dado que las poco más de 4000 muestras pueden ser insuficientes cuando se trata de entrenar un modelo capaz de caracterizar un audio para una tarea tan poco trivial como clasificar sus emociones. Durante el diseño y entrenamiento de estos 3 modelos, ha supuesto una dificultad añadida tratar de que estos no sobreajustasen a los datos de entrenamiento, por lo que se ha tenido que recurrir a técnicas como simplificar los modelos, realizar menos épocas de entrenamiento y añadir porcentajes de *dropout* entre las capas, cosa que se podría haber evitado si se tuviera una mayor cantidad de datos diferentes, que además hubieran ayudado a que los modelos generalicen mejor.

7.2 Futuros trabajos

Como ya hemos visto, la tarea de clasificación de emociones solo a partir del habla, es una tarea de especial interés en múltiples disciplinas. Primero a nivel científico, en el estudio sociológico del ser humano, ya que puede ayudarnos a comprender como comunicamos y expresamos nuestras emociones, cosa que no solo puede ayudarnos a entender el funcionamiento de nuestra especie, sino que también puede ser útil en el campo de la psicología para realizar un diagnóstico más preciso acerca del estado mental de una persona. Por otro lado, también es muy relevante dentro del sector empresarial, ya que por un lado, puede ayudar a mejorar la atención al cliente sustancialmente, siendo esta una herramienta de apoyo para comerciales y equipos de soporte, y por otro lado, es una herramienta muy útil a la hora de desarrollar productos que sean capaces de comprender mejor a los seres humanos, e interactuar con ellos con mayor naturalidad, tarea realmente importante en el campo de la robótica y la informática. Es por ello que es interesante continuar con este estudio con las siguientes líneas de investigación propuestas:

- Entrenamiento con mayor cantidad de datos etiquetados. Si atendemos a los avances de los últimos años en la inteligencia artificial, uno de los puntos más importantes ha sido el de entrenar los modelos con cada vez más datos, lo que permite a su vez crear modelos más complejos con más parámetros, capaces de generalizar cada vez mejor. Es por ello que, se considera una línea de investigación, con potencial mejora en los resultados, el tratar de obtener más muestras bien etiquetadas y entrenar con ellas los modelos.
- Entrenamiento de modelo *transformer*. En los últimos años, este tipo de modelo ha supuesto un cambio de paradigma completo en todos los campos de investigación en torno a la inteligencia artificial, batiendo a modelos previos basados en principios básicos de *deep learning*, como los entrenados en este proyecto. Este tipo de modelo se ha quedado lejos del alcance de este proyecto, ya que son modelos especialmente complejos, que necesitan muchos datos, y que además son muy costosos computacionalmente, pero se considera que este tipo de modelos pueden suponer un avance también en este campo de estudio, ya que ha demostrado ser bueno no solo en tareas de texto, sino también en tareas de audio como *speech recognition* y clasificación de audios.

- Implementar modelos multimodales. En este proyecto, nos hemos centrado en el análisis solamente del audio, pero como sabemos, el ser humano no sólo expresa sus emociones con el habla. Es por ello que se propone estudiar cómo podría un modelo multimodal, compuesto por audio, texto e imagen, capaz de determinar las emociones que expresa un ser humanos, de forma que ya no solo se contaría con las características del sonido, sino también con el significado del mensaje y con las expresiones faciales y gesticulación del individuo.

8 ANEXO

8.1 Código python clase Conexión

```
class Conexion():
    def __init__(self, base_datos:str, servidor:str='localhost') -> None:
        self.cadena_conexion =
f'mssql+pyodbc://{servidor}\\SQLEXPRESS/{base_datos}?trusted_connection=yes&driv
er=ODBC+Driver+18+for+SQL+Server&TrustServerCertificate=yes'
        self.engine = create_engine(self.cadena_conexion)

    def loadData(self, tabla:str, columnas:str, condicion:str=''):
        consulta_sql = f'SELECT {columnas} FROM {tabla} {condicion}'
        return pd.read_sql_query(consulta_sql, self.engine)

    def saveModel(self, id:str, model:list, descripcion=''):
        model_json = json.dumps(model)
        with self.engine.connect() as conn:
            conn.execute(
            text("INSERT INTO MODELOS (ID, DESCRIPCION, ESTRUCTURA) VALUES
(:nombre, :descripcion, :estructura)"),
            [{"nombre":id, "descripcion":descripcion, "estructura": model_json}]
            )

    def saveEpoch(self,
                  id:str,
                  optimizador:str,
                  loss:str,
                  epoch:int,
                  batch_size:int,
                  train_acc:float=0.0,
                  train_loss:float=0.0,
                  val_acc:float=0.0,
                  val_loss:float=0.0,
                  epoch_time:float=0.0,
                  weights:str='',
                  descripcion:str='') -> None:
        with self.engine.connect() as conn:
            conn.execute(
            text('''INSERT INTO MODELOS_ENTRENADOS (MODELO, OPTIMIZADOR, LOSS,
TRAIN_ACC, TRAIN_LOSS, VAL_ACC, VAL_LOSS, BATCH_SIZE, EPOCH, EPOCH_TIME,
WEIGHTS, DESCRIPCION)
                  VALUES (:modelo, :optimizador, :loss, :train_acc, :train_loss,
:val_acc, :val_loss, :batch_size, :epoch, :epoch_time, :weights,
:descripcion)'''),
            [{"modelo":id,
              "optimizador":optimizador,
              "loss":loss,
```

```
            "train_acc":train_acc,
            "train_loss":train_loss,
            "val_acc":val_acc,
            "val_loss":val_loss,
            "batch_size":batch_size,
            "epoch":epoch,
            "epoch_time":epoch_time,
            "weights":weights,
            "descripcion":descripcion}]
        )
```

8.2 Código python clase Datos

```
class Datos():
    def __init__(self, conexion:Conexion) -> None:
        data_path = os.path.join('Datasets')
        self.ravdess_path = os.path.join(data_path, 'ravdess-dataset',
'audio_speech_actors_01-24')
        self.cremaD_path = os.path.join(data_path, 'Crema D')
        self.savee_path = os.path.join(data_path, 'SAVEE')
        self.tess_path = os.path.join(data_path, 'TESS Toronto emotional speech
set data')

        self.conexion = conexion

    def loadDataDB(self, condicion:str='', return_path:bool=False):
        # Cargando los datos desde base de datos
        df = self.conexion.loadData('DATOS','path,emotion', condicion)
        assert len(df.path) == len(df.emotion), "Data length is inconsistent"

        audios = []
        labels = []

        i=0
        for index, row in df.iterrows():
            print('loading data: ', i,'/',len(df.path), end="\r")

            if return_path:
                audios.append(row['path'])
            else:
                X, self.sampleRate = librosa.load(row['path'], duration=3,
offset=0)
                audios.append(X)

            labels.append(row['emotion'])

            i+=1
```

```
        print('\n')
        print('data loaded correctly!')

        return audios, labels

    def loadRavdess(self) -> pd:
        # Cargando datos ravdess a un dataframe
        emotions_dict = {'01':'neutral', '02':'calm', '03':'happy', '04':'sad',
'05':'angry', '06':'fearful', '07':'disgust', '08':'surprised'}
        file_path = []
        emotion = []

        for actor_dir in os.listdir(self.ravdess_path):
            for audio_file in os.listdir(os.path.join(self.ravdess_path,
actor_dir)):
                file_path.append(os.path.join(self.ravdess_path, actor_dir,
audio_file))

                audio_parts = audio_file.split('-')
                emotion.append(emotions_dict[audio_parts[2]])

        ravdess_df = pd.DataFrame()
        ravdess_df['path'] = file_path
        ravdess_df['emotion'] = emotion
        ravdess_df['database'] = 'ravdess'

        return ravdess_df

    def loadCremaD(self) -> pd:
        # Cargando datos crema d a un dataframe
        emotion_dict = {'ANG':'angry', 'DIS':'disgust', 'FEA':'fearful',
'HAP':'happy', 'NEU':'neutral', 'SAD':'sad'}
        file_path = []
        emotion = []

        for file in os.listdir(self.cremaD_path):
            file_path.append(os.path.join(self.cremaD_path, file))

            parts = file.split('_')
            emotion.append(emotion_dict[parts[2]])

        cremaD_df = pd.DataFrame()
        cremaD_df['path'] = file_path
        cremaD_df['emotion'] = emotion
        cremaD_df['database'] = 'cremaD'

        return cremaD_df

    def loadSavee(self) -> pd:
        # Cargando datos savee a un dataframe
```

```
        emotion_dict = {'n':'neutral', 'h':'happy', 'sa':'sad', 'a':'angry',
'f':'fearful', 'd':'disgust', 'su':'surprised'}
        file_path = []
        emotion = []

        for file in os.listdir(self.savee_path):
            file_path.append(os.path.join(self.savee_path, file))
            if file[4].isdigit():
                emotion.append(emotion_dict[file[3]])
            else:
                emotion.append(emotion_dict[file[3:5]])

        savee_df = pd.DataFrame()
        savee_df['path'] = file_path
        savee_df['emotion'] = emotion
        savee_df['database'] = 'savee'

        return savee_df

    def loadTess(self) -> pd:
        # Cargando datos tess a un dataframe
        emotion_dict = {'neutral':'neutral', 'happy':'happy', 'sad':'sad',
'angry':'angry', 'fear':'fearful', 'disgust':'disgust', 'ps':'surprised'}
        file_path = []
        emotion = []

        for actors in os.listdir(self.tess_path):
            for file in os.listdir(os.path.join(self.tess_path, actors)):
                file_path.append(os.path.join(self.tess_path, actors, file))

                parts = file.split('_')
                part_2 = parts[2].split('.')
                emotion.append(emotion_dict[part_2[0]])

        tess_df = pd.DataFrame()
        tess_df['path'] = file_path
        tess_df['emotion'] = emotion
        tess_df['database'] = 'tess'

    def loadDatasets(self, datasets:list) -> pd:
        dataset = pd.DataFrame()
        for dataset in datasets:
            if dataset == 'ravdess':
                pd.concat([dataset, self.loadRavdess()], axis=0)
            if dataset == 'cremaD':
                pd.concat([dataset, self.loadCremaD], axis=0)
            if dataset == 'savee':
                pd.concat([dataset, self.loadSavee()], axis=0)
            if dataset == 'tess':
```

```
                pd.concat([dataset, self.loadSavee()], axis=0)

        return dataset

    def audioFraming(self, audio, frame_len, hop_len):
        '''
        Divide un audio dado en ventanas de igual tamaño con solapamiento
        '''
        return librosa.util.frame(audio, frame_length=frame_len,
hop_length=hop_len)

    def audioFrameWindowing(self, audio):
        '''
        Dado una señal de audio dividida en frames, multiplica cada frame por
una ventana hamming
        '''
        window = signal.windows.hamming(len(audio[0]))
        return audio * window

    def ACF_pitch(self, windowed_frames, sampleRate):
        '''
        Determina el pitch de la señal según el cálculo de la autocorrelación
        '''
        xs = fftfreq(len(windowed_frames[0]), 1/sampleRate)

        r = np.abs(ifft(np.abs(fft(windowed_frames))**2))
        pitch = sampleRate / (xs[np.argmax(r[:,8:88],1)+7])

        return np.array(pitch)

    def Cepstrum_pitch(self, frames, sampleRate):
        '''
        Determina el pitch de la señal según el cálculo del Cepsturm de la señal
        '''
        xs = fftfreq(len(frames[0]), 1/sampleRate)

        cepstrum = np.abs(ifft(np.log(np.absolute(fft(frames)))))
        f = sampleRate
        pitch = sampleRate / (xs[np.argmax(cepstrum[:,8:88],1)+7])

        return np.array(pitch)

    def MFCC(self, frames, sampleRate):
        '''
        Calcula el MFCC de cada frame dado
        '''
        mfcc = librosa.feature.mfcc(y=frames, sr=sampleRate, n_mfcc=12,
n_fft=frames.shape[1])

        return np.array(mfcc)
```

```
    def intensity(self, windowed_frames):
        '''
        Calcula la intensidad de cada frame de audio
        '''
        intensity = np.sum(windowed_frames**2, axis=1)/windowed_frames.shape[1]

        return np.array(intensity)

    def LSP(self, frames):

        lsp = []
        for i in range(frames.shape[0]):

            a = librosa.lpc(frames[i], order=8)

            a = np.array(a)
            if a[0] != 1:
                a/=a[0]

            p  = len(a)-1
            a1 = np.concatenate((a, np.array([0])))
            a2 = a1[-1::-1]
            P1 = a1 - a2
            Q1 = a1 + a2

            if p%2:
                P, r = deconvolve(P1,[1, 0 ,-1])
                Q = Q1
            Else:
                P, r = deconvolve(P1, [1, -1])
                Q, r = deconvolve(Q1, [1,  1])

            rP  = np.roots(P)
            rQ  = np.roots(Q)

            aP  = np.angle(rP[1::2])
            aQ  = np.angle(rQ[1::2])

            lsp.append(sorted(np.concatenate((-aP,-aQ))))

        return np.array(lsp)

    def ZCR(self, frames):
        '''
        Calcula el zero crossing rate de cada frame de audio
        '''
        zcr = librosa.feature.zero_crossing_rate(y=frames)
```

```
        return np.array(zcr)

    def computeGlobalFeatures(self, data):
        '''
        Calcula las características globales de cada feature local
        '''
        if (len(data.shape) > 1):
            offset_list = np.zeros(data[0,:,:].shape)
            slope_list = np.zeros(data[0,:,:].shape)
            iqr_list = np.zeros(data[0,:,:].shape)
            for i in range(data.shape[1]):
                for j in range(data.shape[2]):
                    kernel_size = 3
                    kernel = np.ones(kernel_size) / kernel_size
                    data_conv = np.convolve(data[:,i,j], kernel, mode='same')
                    model = LinearRegression().fit(data_conv.reshape((-1, 1)),
range(len(data)))
                    offset = model.intercept_
                    slope = model.coef_[0]
                    q3, q1 = np.percentile(data[:,i,j], [75 ,25])
                    iqr = q3 - q1

                    offset_list[i,j] = offset
                    slope_list[i,j] = slope
                    iqr_list[i,j] = iqr

            offset = offset_list
            slope = offset_list
            iqr = offset_list

        else:
            kernel_size = 3
            kernel = np.ones(kernel_size) / kernel_size
            data_conv = np.convolve(data, kernel, mode='same')
            model = LinearRegression().fit(data_conv.reshape((-1, 1)),
range(len(data)))
            offset = model.intercept_
            slope = model.coef_[0]
            q3, q1 = np.percentile(data, [75 ,25])
            iqr = q3 - q1

        media = np.mean(data, axis=0)
        maximo = np.max(data, axis=0)
        minimo = np.min(data, axis=0)
        rango = maximo - minimo
        std = np.std(data, axis=0)
        skewness = skew(data, axis=0)
        kurt = kurtosis(data, axis=0)
```

```
        return [media, maximo, minimo, rango, offset, slope, iqr, std, skewness,
kurt]

    def padding(self, audios, duration, sampling_rate=22050):
        max_len = duration * sampling_rate
        audios_padded = []
        for audio in audios:
            len_audio = len(audio)
            if len_audio < max_len:
                audio_padded = np.zeros(max_len)
                audio_padded[:len_audio] = audio
            else:
                audio_padded = audio[:max_len]
            audios_padded.append(audio_padded)

        return audios_padded

class Procesado():
    pass
```

8.3 Código python clase Modelo

```
class Model():

    def __init__(self, model:list, id:str, conexion:Conexion=None) -> None:
        self.estructura = model
        self.sequentialModel(model)
        self.id = id
        self.conexion = conexion

    def sequentialModel(self, model:list):
        self.model = Sequential()

        for capa in model:
            tipo = capa['type']
            if tipo == 'Input':
                self.model.add(self.__InputLayer(capa))
            elif tipo == 'Conv2D':
                self.model.add(self.__Conv2D(capa))
            elif tipo == 'Conv1D':
                self.model.add(self.__Conv1D(capa))
            elif tipo == 'BatchNormalization':
                self.model.add(self.__BatchNormalization(capa))
            elif tipo == 'MaxPooling2D':
                self.model.add(self.__MaxPooling2D(capa))
            elif tipo == 'Dense':
                self.model.add(self.__Dense(capa))
            elif tipo == 'Flatten':
```

```
                self.model.add(self.__Flatten(capa))
            elif tipo == 'Dropout':
                self.model.add(self.__Dropout(capa))
            elif tipo == 'LSTM':
                self.model.add(self.__LSTM(capa))
            elif tipo == 'MaxPooling1D':
                self.model.add(self.__MaxPooling1D(capa))

    def compile(self, optimizer, loss, metrics:list=[]):
        self.optimizer = optimizer
        self.loss = loss
        self.metrics = metrics

        self.model.compile(optimizer, loss, metrics)

    def saveModel(self, descripcion:str=''):
        self.conexion.saveModel(self.id, self.estructura, descripcion)

    def train(self, x_train, y_train, x_val, y_val, epochs:int,
batch_size:int=1, callbaks:list=[], weights_path=''):
        self.batch_size = batch_size

        if weights_path != '':
            checkpoint_path = os.path.join(weights_path, "{epoch:04d}.ckpt")
            checkpoint_dir = os.path.dirname(checkpoint_path)
            cp_callback = ModelCheckpoint(
                checkpoint_path, verbose=1, save_weights_only=True,
                save_freq='epoch')
            callbaks.append(cp_callback)

        callbaks.append(CallbackToDB(self.conexion, self.id, self.optimizer,
self.loss, self.batch_size, weights_path))
        self.history=self.model.fit(x_train, y_train, batch_size=batch_size,
epochs=epochs, validation_data=(x_val, y_val), callbacks=callbaks)

    def __InputLayer(self, parametros:dict):
        return InputLayer(input_shape=parametros['shape'])

    def __Conv2D(self, parametros:dict):
        # Definición de parámteros disponibles
        activation = None
        kernel_regularizer = None
        strides = (1,1)
        padding = 'valid'

        # Asignación de valores de parámetros
        for parametro in parametros:
            if parametro == 'filters':
                filters = parametros['filters']
            elif parametro == 'kernel_size':
```

```
                kernel_size = parametros['kernel_size']
            elif parametro == 'activation':
                activation = parametros['activation']
            elif parametro == 'kernel_regularizer':
                kernel_regularizer = parametros['kernel_regularizer']
            elif parametro == 'strides':
                strides = parametros['strides']
            elif parametro == 'padding':
                padding = parametros['padding']

        return Conv2D(filters, kernel_size, strides=strides, padding=padding,
                      activation=activation,
kernel_regularizer=kernel_regularizer)

    def __Conv1D(self, parametros:dict):
        # Definición de parámteros disponibles
        activation = None
        kernel_regularizer = None
        strides = 1
        padding = 'valid'

        # Asignación de valores de parámetros
        for parametro in parametros:
            if parametro == 'filters':
                filters = parametros['filters']
            elif parametro == 'kernel_size':
                kernel_size = parametros['kernel_size']
            elif parametro == 'activation':
                activation = parametros['activation']
            elif parametro == 'kernel_regularizer':
                kernel_regularizer = parametros['kernel_regularizer']
            elif parametro == 'strides':
                strides = parametros['strides']
            elif parametro == 'padding':
                padding = parametros['padding']

        return Conv1D(filters, kernel_size, strides=strides, padding=padding,
                      activation=activation,
kernel_regularizer=kernel_regularizer)

    def __BatchNormalization(self, parametros:dict):
        return BatchNormalization()

    def __MaxPooling2D(self, parametros:dict):
        # Definición de parámteros disponibles
        pool_size = (2,2)
        strides = None
        padding = 'valid'
```

```
        # Asignación de valores de parámetros
        for parametro in parametros:
            if parametro == 'strides':
                strides = parametros['strides']
            elif parametro == 'padding':
                padding = parametros['padding']
            elif parametro == parametros['pool_size']:
                pool_size = parametros['pool_size']

        return MaxPooling2D(pool_size=pool_size, strides=strides,
padding=padding)

    def __Dense(self, parametros:dict):
        # Definición de parámetros
        units = None
        activation = None
        kernel_regularizer = None

        # Asignación de valores de parámetros
        for parametro in parametros:
            if parametro == 'units':
                units = parametros['units']
            elif parametro == 'activation':
                activation = parametros['activation']
            elif parametro == 'kernel_regularizer':
                kernel_regularizer = parametros['kernel_regularizer']

        return Dense(units, activation=activation,
kernel_regularizer=kernel_regularizer)

    def __Flatten(self, parametros:dict):
        return Flatten()

    def __Dropout(self, parametros:dict):
        return Dropout(parametros['percentage'])

    def __LSTM(self, parametros:dict):
        # Definición de parámetros disponibles
        activation = 'tanh'
        return_sequences = False
        for parametro in parametros:
            if parametro == 'units':
                units = parametros['units']
            elif parametro == 'activation':
                activation = parametros['activation']
            elif parametro == 'return_sequences':
                return_sequences = parametros['return_sequences']

        return LSTM(units, activation=activation,
return_sequences=return_sequences)
```

```
    def __MaxPooling1D(self, parametros:dict):
        # Definición de parámteros disponibles
        pool_size = 2
        strides = None
        padding = 'valid'

        # Asignación de valores de parámetros
        for parametro in parametros:
            if parametro == 'strides':
                strides = parametros['strides']
            elif parametro == 'padding':
                padding = parametros['padding']
            elif parametro == parametros['pool_size']:
                pool_size = parametros['pool_size']

        return MaxPooling1D(pool_size=pool_size, strides=strides,
padding=padding)
```

8.4 Código python clase Statistics

```
class Statistics():
    def __init__(self, model:keras.callbacks.History) -> None:
        self.Model = model

    def plotHistory(self) -> None:
        fig, axes = plt.subplots(1, 2, figsize=(10, 5))

        axes[0].plot(self.Model.history['loss'])
        axes[0].plot(self.Model.history['val_loss'])
        axes[0].set_title('Loss for Train and Validation Sets')
        axes[0].set_ylabel('Loss')
        axes[0].set_xlabel('Epochs')
        axes[0].legend(['Training', 'Validation'])

        axes[1].plot(self.Model.history['categorical_accuracy'])
        axes[1].plot(self.Model.history['val_categorical_accuracy'])
        axes[1].set_title('Accuracy for Train and Validation Sets')
        axes[1].set_ylabel('Accuracy')
        axes[1].set_xlabel('Epochs')
        axes[1].legend(['Training', 'Validation'])

        fig.tight_layout()

        plt.show()

    def printConfussionMatrix(self, x, y):
        y_pred = np.argmax(self.Model.predict(x), axis=1)
```

```
        labels = ['neutral', 'calm', 'sad', 'happy', 'fear', 'disgust']
        cm = confusion_matrix(np.argmax(y, axis=1), y_pred, labels=range(6))

        sns.heatmap(cm, annot=True, fmt='d', cmap='Reds', xticklabels=labels,
yticklabels=labels)
        plt.xlabel('Predicted label')
        plt.ylabel('True label')
        plt.title('Confusion matrix')
        plt.show()

class CallbackToDB(Callback):

    def __init__(self, conexion:Conexion, id:int, optmizer:str, loss:str,
batch_size:int, weights_path:str) -> None:
        self.con = conexion
        self.id = id
        self.optimizer = optmizer
        self.loss = loss
        self.batch_size = batch_size
        self.weights_path = weights_path

    def on_epoch_end(self, epoch, logs=None) -> None:
        if self.weights_path != '':
            weights = self.weights_path+f'{epoch}.ckpt'
        else:
            weights = ''
        self.con.saveEpoch(
            self.id,
            self.optimizer,
            self.loss,
            epoch,
            self.batch_size,
            logs.get('categorical_accuracy'),
            logs.get('loss'),
            logs.get('val_categorical_accuracy'),
            logs.get('val_loss'),
            weights=weights
            )
```

9 REFERENCIAS

[1] Thakkar, S. (2020, 24 de febrero). AI vs ML vs DL: What's the difference? *Siddhi Thakkar*. https://siddhithakkar.com/2020/02/24/ai-vs-ml-vs-dl-whats-the-difference/

[2] Aguilar, F. J. (n.d.). Figura 3 [Figura]. En *Estructura de una red neuronal de tres niveles*. Recuperado de https://www.researchgate.net/figure/Figura-3-Estructura-de-una-red-neuronal-de-tres-niveles_fig1_228543266

[3] de Lope, Javier, & Graña, Manuel (2023) An ongoing review of speech emotion recognition. Neurocomputing 528 1-11.

[4] Figura 2 [Figura] Classification of data by support vector machine (SVM). Recuperado de https://www.researchgate.net/figure/Classification-of-data-by-support-vector-machine-SVM_fig8_304611323

[5] Radja Hachilif. Figura 1 [Figura]. 2D convolution with filter size 3x3. Recuperado de https://www.researchgate.net/figure/2D-convolution-with-filter-size-3x3-49_fig4_334974839

[6] Torres, Jordi (2020). Python Deep Learning. Marcombo.

[7] Cañadas, R. (2021, noviembre 22). Redes neuronales recurrentes. *Abdatum*. Recuperado de https://abdatum.com/tecnologia/redes-neuronales-recurrentes

[8] Ala Saleh Alluhaidan, Oumaima Saidani, Rashid Jahangir, Muhammad Asif Nauman & Omnia Saidani Neffati (2023) Speech Emotion Recognition through Hybrid Features and Convolutional Neural Network. Applied Science, 13, 4750. https://doi.org/10.3390/app13084750

[9] ProjectPro. (2023). Speech Emotion Recognition Project using Machine Learning. *ProjectPro*. Recuperado de https://www.projectpro.io/article/speech-emotion-recognition-project-using-machine-learning/573

[10] Burnwal, S. (n.d.). Speech Emotion Recognition. *Kaggle*. Recuperado de https://www.kaggle.com/code/shivamburnwal/speech-emotion-recognition

[10] Fayek, H. M. (2016, abril 21). Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. *Haytham Fayek*. Recuperado de https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

[11] Raghu Kogila1, Manchala Sadanandam1, Hanumanthu Bhukya (2023) Deep Learning Algorithms for Speech Emotion Recognition with Hybrid Spectral Features. SN Computer Science 5:17 https://doi.org/10.1007/s42979-023-02358-z

[12] Yuanbo Gao1, Baobin Li1, Ning Wang, & Tingshao Zhu (2017) Speech Emotion Recognition Using Local and Global Features. LNAI 10654, pp. 3–13. https://doi.org/10.1007/978-3-319-70772-3_1

[13] Samson Akinpelu & Serestina Viriri (2022) Robust Feature Selection-Based Speech Emotion Classification Using Deep Transfer Learning. Applied Science 12, 8265. https://doi.org/10.3390/app12168265

[14] R. Leelavathi1, S. Aruna Deepthi, V. Aruna (2022) Speech Emotion Recognition using LSTM.

IRJET Volumen 09.