Trabajo Fin de Máster
Máster en Ingeniería de Telecomunicación

Entrenamiento y despliegue de un modelo de clasificación de audio

Autor: Antonio José Aragón Molina

Tutor: María del Mar Elena Pérez

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 8 de noviembre de 2023

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Máster
Máster en Ingeniería de Telecomunicación

Entrenamiento y despliegue de un modelo de clasificación de audio

Autor:
Antonio José Aragón Molina

Tutor:
María del Mar Elena Pérez
Profesor Titular

Dpto. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 8 de noviembre de 2023

Trabajo Fin de Máster: Entrenamiento y despliegue de un modelo de clasificación de audio

Autor: Antonio José Aragón Molina
Tutor: María del Mar Elena Pérez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mis padres, por su apoyo incondicional, por su paciencia y por su amor. A mis profesores y profesoras, por su dedicación y su esfuerzo. A mis compañeros de clase, por acompañarme en este camino. A mis compañeros de piso, por su compañía y su apoyo. A mis amigos de siempre, gracias por sacarme siempre una sonrisa. A la música, por hacerme sentir.

Antonio José Aragón Molina
Escuela Técnica Superior de Ingeniería

Sevilla, 8 de noviembre de 2023

Resumen

Este trabajo se centra en la creación y despliegue de un modelo clasificador de audios por emociones. El modelo se ha implementado en Python, utilizando la librería *Transformers* de *HuggingFace* y el modelo preentrenado *Wav2Vec2* de *Facebook AI*. Ha sido desplegado como aplicación web, utilizando *Flask* y *Docker*.

Abstract

This work focuses on the creation and deployment of an audio emotions classifier model. The model has been implemented in Python, using the *Transformers* library from *HuggingFace* and the pre-trained *Wav2Vec2* model from *Facebook AI*. It has been deployed as a web application, using *Flask* and *Docker*.

Índice Abreviado

Índice

1 Introducción

I am going into an unknown future, but I'm still all here, and still while there's life, there's hope.

John Lennon, '70s

Durante los últimos años, la sociedad ha experimentado un auge en el empleo de la Inteligencia Artificial (IA) en diferentes ámbitos. La gran capacidad de especialización en un problema concreto que presentan estos sistemas, junto con la gran cantidad de datos que se generan en la actualidad, han hecho que la IA se haya convertido en una herramienta muy útil en la resolución de problemas complejos.

Estas características han fomentado el empleo de soluciones basadas en IA, obteniendo resultados aceptables para problemas que de otro modo serían irresolubles.

De forma paralela, se observa un crecimiento en el interés por el uso de estas tecnologías. Muchos sistemas empiezan a aparecer en el día a día del usuario promedio, como por ejemplo: [4]

- Sistemas de búsqueda y recomendación
- IA generativa de texto, imágenes o audio
- Sistemas de predicción de eventos
- Asistentes de voz
- Vehículos autónomos
- Compras personalizadas

Sin embargo, la investigación sobre la aceptación de tecnologías que incluyen IA está aún en curso. Algunos estudios sugieren que en ciertos escenarios culturales, la necesidad de contacto humano no puede ser replicada o reemplazada. [7]

Este trabajo busca adentrarse en los límites sobre qué puede hacer un modelo de IA a día de hoy, en concreto en el campo de clasificación de audios. El objetivo es crear un modelo que sea capaz de clasificar audios en diferentes categorías, identificadas por emociones humanas.

Una vez creado el modelo, se aborda el problema de despliegue del modelo en un entorno de producción, para que pueda ser utilizado por usuarios finales. En un caso real de uso, podría diseñarse una interfaz o una API por ejemplo, de modo que la interacción con el modelo se adapte a las requerimientos del proyecto.

Debido a esta clara diferenciación, la memoria ha sido dividida en dos partes, que darán nombre a los capítulos principales: **Modelado** y **Despliegue**. En el capítulo 2 se aborda el problema de modelado, mientras que en el capítulo 3 se aborda el problema de despliegue.

1.1 Motivación

La idea de realizar un modelo capaz de diferenciar emociones humanas a partir de audios surge de la participación en un proyecto anterior. Este proyecto, a grandes rasgos, buscaba crear un modelo que sirviese de ayuda a las personas a detectar emociones específicas para el caso de aplicación.

Por diversas circunstancias, el proyecto no pudo avanzar correctamente y acabó siendo abandonado. Sin embargo, un año más tarde y tras pensar en abordar de nuevo el problema, con un enfoque mucho más abierto, las sensaciones fueron muy positivas

En primer lugar, el avance de la tecnología a lo largo del tiempo ha permitido resolver problemas que antes parecían imposibles. En concreto, ha sido notable la diferencia en cuanto a todo lo relacionado con el modelo, dataset, plataforma de desarrollo, hosting, etc.

En segundo lugar, durante este tiempo, gracias al aprendizaje adquirido en el Máster y a la experiencia laboral, la problemática ha podido ser abordada con más madurez y conocimiento de las tecnologías existentes. Contar con algo de experiencia junto con una visión más amplia de las herramientas disponibles, permite vislumbrar opciones que antes eran invisibles. Además, enfocar un problema conociendo la pila completa ayuda a dividir en bloques la solución y a buscar alternativas para cada uno de ellos en caso de que sea necesario.

Por otro lado, al ser un trabajo con fines de aprendizaje, se eliminaron varias restricciones impuestas durante la realización del proyecto. Para ilustrar cómo limitaban estas restricciones y el resultado obtenido al haberlas modificado, se dedica un apartado para cada una de ellas.

Dataset

En el anterior proyecto, la sensibilidad de los datos era especialmente elevada. Esto generaba diversos problemas, ya que en ningún caso podrían salir de las instalaciones de la empresa.

Por otro lado, otro obstáculo fue la creación del dataset, ya que los datos debían ser recogidos y etiquetados por la propia empresa. Esto puede parecer una liberación de carga de trabajo en un principio. Sin embargo, al no disponer de datos de entrada, es difícil avanzar en el desarrollo del modelo.

En este proyecto, al no estar diseñando la solución para un dataset concreto, se puede realizar una búsqueda más general en bancos de datos públicos y elegir alguno similar al objetivo. De este modo se podría haber avanzado en el desarrollo del modelo, mientras se recogían los datos por parte de la empresa. El aprendizaje obtenido acerca de este asunto es que es posible avanzar en el desarrollo del modelo, aprendiendo las técnicas y requisitos necesarios para el problema, con un dataset semejante, sin necesidad de esperar a tener los datos finales.

Enfoque general

Durante el proyecto, fue difícil encontrar un método para abordar el problema. La búsqueda de proyectos que resolviesen problemas similares no fue muy fructífera, debido a la particular naturaleza del problema.

De forma similar a lo comentado en el párrafo anterior, un mejor acercamiento hubiese sido encontrar un problema similar y adaptarlo poco a poco al objetivo. La mejora en el estado del arte y una mayor madurez en el conocimiento de la materia han permitido realizar búsquedas de soluciones que pueden ser adaptadas al problema.

Además, la aparición de herramientas de mayor alto nivel permiten un acercamiento más paulatino al problema, pudiendo introducirnos en detalles más concretos más adelante, también con los datos finales. El enfoque de lo general a lo particular, junto con la división por bloques, permite avanzar en el desarrollo de cualquier proyecto, pero de nuevo, la experiencia ayuda en estas tareas.

Hardware

Uno de los principales problemas que presenta el desarrollo de modelos de IA es la necesidad de una alta capacidad de cómputo. Otros inconvenientes, como la necesidad de una gran cantidad de datos pueden ser apaciguados empleando técnicas de Data Augmentation, pero para entrenar un modelo complejo es necesaria una gran capacidad de cómputo.

No todos los problemas requieren de soluciones extremadamente complejas, de hecho en el mundo del Machine Learning (ML) a veces las soluciones más simples ofrecen mejores resultados. Sin embargo, como se verá más adelante, la complejidad del problema requiere de un modelo complejo, lo que nos obliga a disponer de hardware con gran capacidad de computación en paralelo si queremos obtener resultados en un tiempo razonable.

Hoy en día existen diversas alternativas para entrenar grandes modelos aunque no dispongas de un gran equipo (el crecimiento en el número de alternativas viene motivado a su vez por el crecimiento en el interés en la IA). Continuando con las reflexiones anteriores, en este caso la solución ha sido emplear una plataforma de entrenamiento en la nube, brindando la posibilidad de entrenar modelos complejos sin necesidad de disponer de un equipo dedicado o desgastar en exceso un equipo personal, todo ello por un coste reducido.

Volviendo al proyecto original, es necesario recordar que esta solución no podría ser aplicada debido a que los datos no podrían salir de las instalaciones de la empresa. De todos modos, puede servir para iniciar en el proceso de investigación y creación de modelos alternativos, con datasets alternativos.

Entorno de despliegue

Otro punto problemático residía en torno al entorno de despliegue. Ante un gran problema, difícil de dividir a simple vista, el no disponer de una visualización del resultado final genera malestar y dudas que no favorecen al desarrollo del proyecto. Para que no ocurriera esto, se ha optado por buscar como resultado gráfico final lo mínimo necesario por una persona para poder utilizar el modelo.

Más allá de la interfaz, el paradigma de despliegue de cualquier software requiere de un entorno robusto que asegure su funcionamiento. El aprendizaje recibido durante el máster sobre contenedores, ha permitido utilizarlos para encapsular el modelo como una aplicación web y realizar el despliegue en una plataforma de hosting, asegurando su funcionamiento.

La conclusión en cuanto a este apartado es que, para poder pensar en un producto final, con una interfaz gráfica perfilada y plena funcionalidad, es necesario primero probar una versión más simple, con funcionalidad reducida, pero que permita validar el modelo y el despliegue. Teniendo esto en mente, es posible avanzar en el desarrollo del modelo, a la vez que se imaginan las distintas posibilidades para el caso de uso concreto.

1.2 Planificación

Una vez definido el contexto en el que surge esta idea, vamos a definir cómo se va a desarrollar el proyecto. Debemos definir el objetivo y el alcance del proyecto, para no perder la visión del mismo y poder evaluar el resultado final.

Al mismo tiempo, se incluye un listado de requisitos, que ayudarán a dar soporte a la definición del objetivo.

Además, se incluye la planificación temporal inicial del proyecto, en la que destacan varios hitos importantes.

1.2.1 Objetivo y alcance

El **objetivo** principal de este proyecto consiste en la creación y el despliegue de un modelo clasificador de audios según emociones humanas.

Al no ser un proyecto destinado para un caso de uso concreto (además de muchas otras limitaciones que presenta), el alcance del proyecto es un poco más abierto. Definimos el **alcance** del proyecto como la creación de una primera versión de una aplicación que implemente la mínima funcionalidad. Debe poder permitir utilizar el modelo de un modo sencillo, sin necesidad de tener conocimientos técnicos, debido a que en un caso real de uso, el usuario final no tendría por qué tener conocimientos técnicos.

1.2.2 Requisitos

Para acompañar al objetivo y el alcance, se ha diseñado una tabla de requisitos, separados por categorías:

Requisitos funcionales

- **F.1:** El sistema debe ser capaz de clasificar audios en diferentes categorías.
- **F.2:** El sistema debe ser capaz de recibir audios en formato WAV.
- **F.3:** Debe ser accesible desde múltiples dispositivos.
- **F.4:** El modelo permitirá ser actualizado periódicamente con nuevos datos de entrenamiento.

Requisitos operacionales

- **O.1:** La respuesta del sistema debe ser inferior a 5 segundos.
- **O.2:** El sistema debe ser robusto en entornos de grabación de alto ruido.
- **O.3:** El sistema debe ser accesible desde cualquier lugar y en cualquier instante.
- **O.4:** El coste computacional de la inferencia del sistema debe ser reducido.

Requisitos de diseño

- **D.1:** La implementación del sistema se realizará en Python.
- **D.2:** El despliegue del sistema será desplegado mediante contenedores.
- **D.3:** El entorno de despliegue será accesible remotamente.
- **D.4:** El sistema se ejecutará en un equipo con 4vcpu y 8GB de RAM.

Requisitos de seguridad

- **S.1:** Los audios no podrán abandonar nunca las instalaciones.
- **S.2:** La implementación web debe realizarse sobre un servidor seguro.
- **S.3:** El sistema debe ser robusto ante ataques de denegación de servicio.

Muchos de estos requisitos son utilizados a modo de muestra, pero no tienen mayor relevancia en este proyecto, ya que limitarían de forma innecesaria el desarrollo del mismo. Sin embargo, estos requisitos han servido para imaginar cómo podría ser el sistema en un caso de uso real, además de entender problemas que podrían surgir en caso de que el sistema se desplegase en un entorno real. Estos requisitos nos ayudan a dar forma a la solución final, entendiendo cómo podríamos mejorar su funcionalidad, además de su robustez.

Es interesante comentar que, aunque no es alcance para este proyecto, los requisitos de seguridad pueden llegar a ser los más críticos en un caso de uso real. En este caso, se ha optado por no profundizar en este aspecto, pero desde luego no es un detalle que deba pasarse por alto.

1.3 Diseño de la solución

Previamente, se ha explicado de qué trata el problema y de dónde surge la idea. En esta sección, intentamos responder a la pregunta de cómo se ha abordado el problema.

Siguiendo el famoso dicho, divide y vencerás, se ha optado por dividir el problema en dos bloques principales: **Creación del modelo** y **Despliegue del modelo**. De este modo, podemos centrarnos en cada uno de los bloques por separado, sin perder la visión general del proyecto.

Los dos bloques son lo suficientemente importantes y están separados entre sí lo suficiente como para que puedan ser abordados por separado y no interfieran entre sí.

En un proyecto real, esta división sería apropiada para separar dos grupos de trabajo. Se podría designar cada bloque a un grupo de trabajo o departamento interno, de modo que cada uno de ellos se encargue únicamente de su parte. Conseguiríamos así una mayor especialización en cada uno de los bloques, además de una mayor eficiencia en el desarrollo del proyecto.

Como en este caso solo hay una persona trabajando en el desarrollo de la solución, la paralelización no influye directamente en el tiempo de desarrollo. Sin embargo, pensarlo de este modo ayuda a:

- Dividir la complejidad del problema a la mitad en primera instancia.
- Focalizar el objetivo de cada bloque.
- Independizar los bloques en caso de que uno de ellos no pueda ser completado.

1.3.1 Creación del modelo

El primer bloque, recogido en el Capítulo 2, se centra en la creación de un modelo capaz de clasificar audios según emociones humanas. El objetivo de este bloque será crear un modelo que pueda clasificar los audios con cierta precisión entre un conjunto de emociones previamente definidas.

Algunos aspectos que se tendrán en cuenta en este bloque son:

- Elección del dataset de entrenamiento
- Estudio del estado del arte
- Elección de la arquitectura del modelo
- Entrenamiento del modelo
- Evaluación del modelo
- Alojamiento del modelo

1.3.2 Despliegue del modelo

El segundo bloque, detallado en el Capítulo 3, se centra en la creación de una aplicación que permita utilizar el modelo creado en el bloque anterior. El objetivo de este bloque será utilizar el modelo de un modo determinado, ponerlo a disposición del usuario final, y que pueda ser desplegado en cualquier entorno.

Algunos aspectos relevantes en cuanto a este bloque son:

- Diseño de la interfaz
- Despliegue de la aplicación
- Hosting de la aplicación
- Acceso a la aplicación

2 Diseño del modelo

I don't know where I'm going from here, but I promise it won't be boring.

David Bowie

Este capítulo se centra en explicar el procedimiento seguido para la creación del modelo clasificador de audios.

Es importante destacar la gran cantidad de documentación existente además del gran apoyo que la comunidad ofrece en este campo en concreto. Puede parecer un problema difícil de resolver, pero gracias a toda la información disponible y a la gran cantidad de herramientas que existen, ha sido posible crear un modelo capaz de ofrecer una funcionalidad básica.

En concreto, se destacan los sitios web Kaggle y Hugging Face como las principales fuentes de información y herramientas utilizadas. Son dos plataformas directamente enfocadas al entrenamiento de modelos de *Machine Learning* y *Deep Learning*. Ambas fomentan la colaboración entre usuarios y ofrecen una gran cantidad de recursos para la creación de modelos.

En concreto, se ha utilizado un dataset de *Kaggle* para el entrenamiento del modelo y librerías específicos de Hugging Face para la creación del modelo en Python.

2.1 Dataset

Como ha sido comentado en el Capítulo 1, un problema que se enfrentó en el primer acercamiento a la problemática que este proyecto pretende resolver fue la falta de una base de datos que contuviera audios con las emociones específicas que se querían clasificar. Al no haber impuesto restricciones, se ha optado por elegir un dataset ya existente.

La búsqueda de base de datos se ha realizado en *Kaggle*. La popularidad de esta plataforma no es injustificada, ya que cuenta con una gran cantidad de datasets de todo tipo. En concreto, se han realizado búsquedas de datasets relacionados con audios clasificados por emociones.

En *Kaggle* existen datasets oficiales creados por grandes organizaciones, y otros creados por usuarios de la plataforma. La mayoría de los datasets son de libre acceso, por lo que tenemos la posibilidad de descargarlos y utilizarlos para nuestros propios proyectos.

En este caso, se ha optado por utilizar un dataset llamado *Audio Emotions* compartido por el usuario *Uldis Valainis*. [10] Este dataset es una recopilación de varios datasets similares creados por organizaciones diferentes, que contienen audios grabados por actores interpretando diferentes emociones.

La elección de este dataset se ha realizado por la gran cantidad de audios que contiene, y por tener un número variado de emociones. Estas emociones son: *neutral*, *happy*, *sad*, *angry*, *fearful*, *disgust*, *surprised*.

El dataset contiene un total de 12.798 audios, con una duración de 3 segundos cada uno.

Echando un vistazo a la Figura 2.1 y a la Figura 2.2, podemos observar una clara falta de balance en la clase *surprised*, que contiene un número muy inferior de audios que el resto de clases. Como no existen restricciones en cuanto a las clases que se quieren clasificar, se ha optado por eliminar esta clase del dataset, y quedarnos con las 6 clases restantes. De este modo se consigue un dataset más equilibrado, lo cual influirá positivamente en el entrenamiento del modelo.

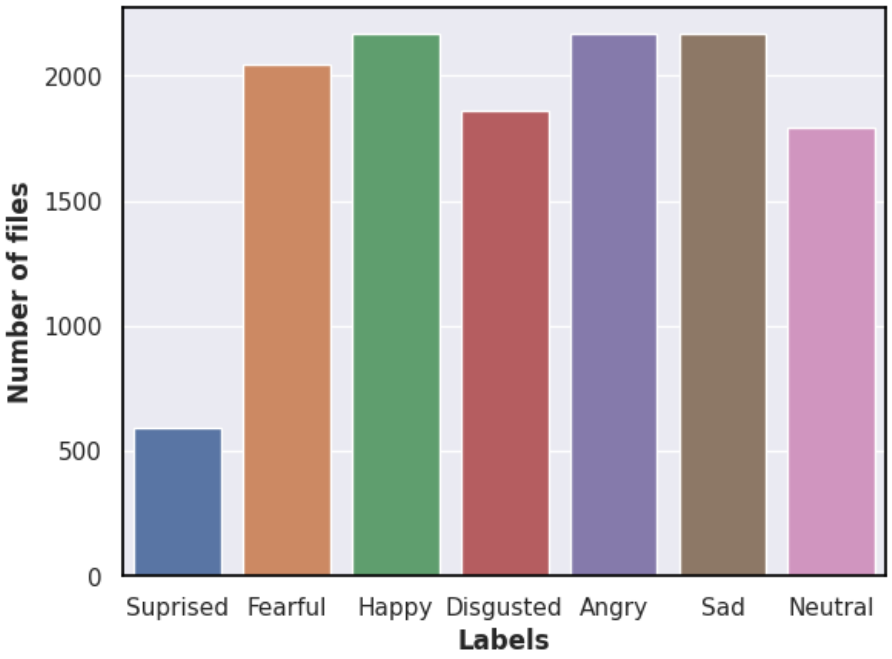

Figura 2.1 Número de audios del dataset.

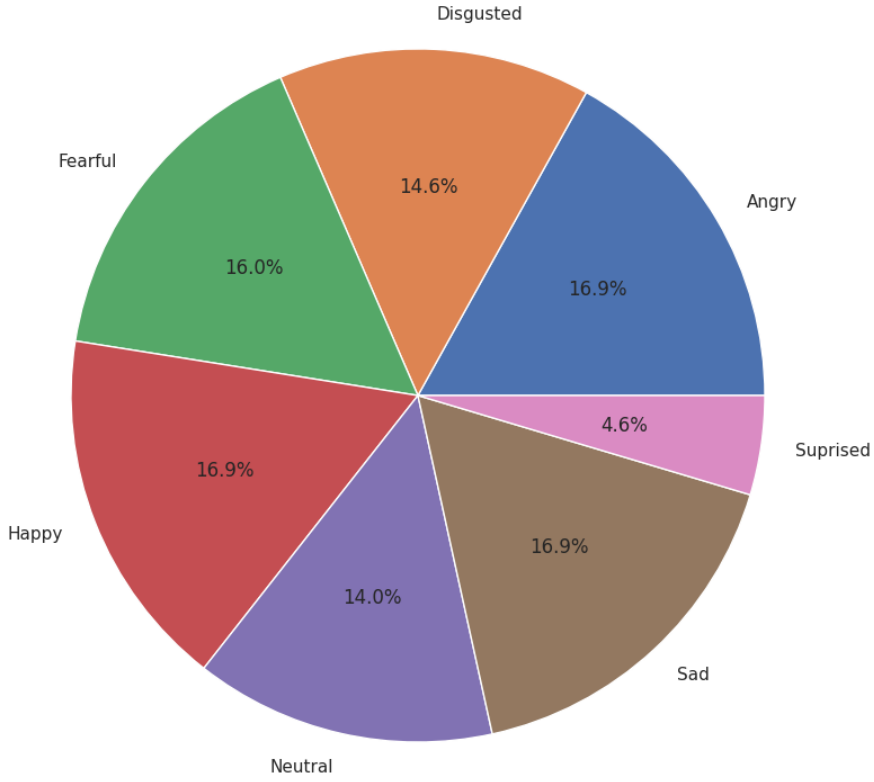

Figura 2.2 Porcentaje de audios del dataset.

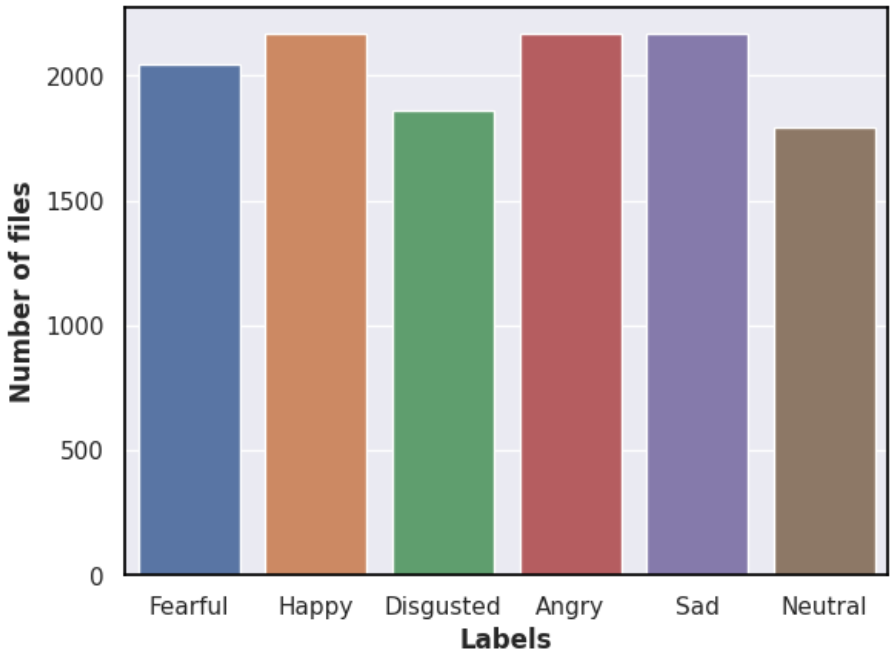

Figura 2.3 Número de audios del dataset reducido.

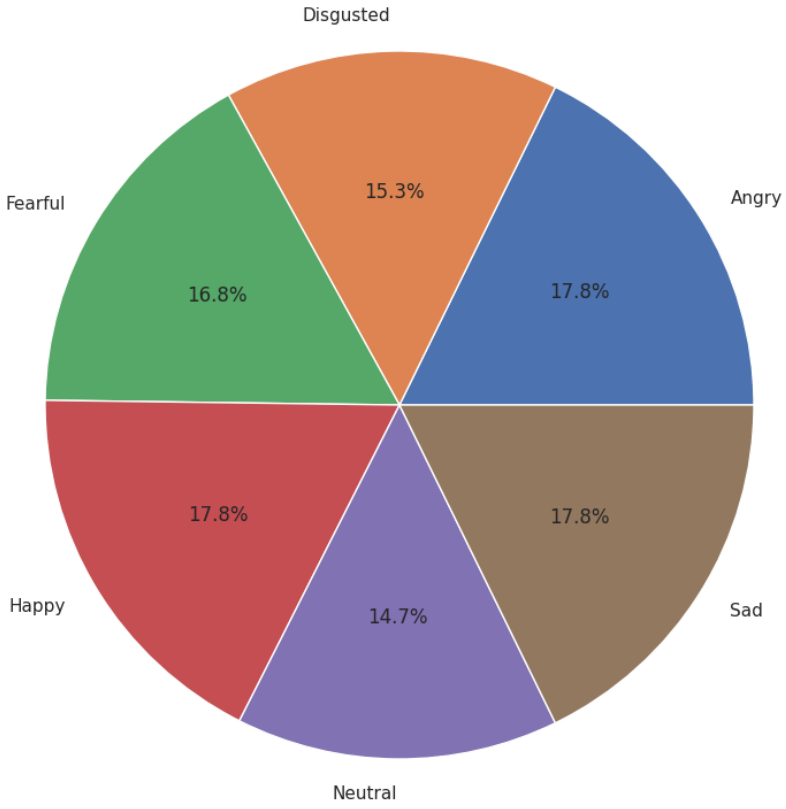

Figura 2.4 Porcentaje de audios del dataset reducido.

Esta versión del dataset cuenta con un total de 12.206 audios, con una duración de 3 segundos cada uno.

2.2 Modelo

Una vez se ha seleccionado el dataset, se ha procedido a la creación del modelo.

2.2.1 Estado del arte

Para poder enfocar un problema del que no se tiene conocimiento previo, es necesario realizar una investigación previa sobre el estado del arte, primero, para determinar si es posible resolver el problema, y segundo, para conocer las herramientas que existen para resolverlo. En este caso, se ha realizado una investigación sobre las herramientas que existen para la clasificación de audios.

La solución parece estar ubicada en el campo de la *Inteligencia Artificial*, y más concretamente en el campo del *Deep Learning* debido a la dificultad de encontrar patrones en los audios. Es importante saber clasificar nuestro problema dentro de un campo de la *Inteligencia Artificial*, ya que existen diferentes herramientas para resolver problemas de diferentes campos.

Trabajos relacionados muestran un gran desempeño de la arquitectura conocida como **Transformers** en la clasificación de audios, ya que son capaces de capturar patrones en los audios que otras arquitecturas no son capaces de capturar. Estos *Transformers* son una arquitectura de *Deep Learning* que se ha popularizado en los últimos años, y que ha demostrado un gran desempeño en la clasificación de textos y audios. Por eso, son muy utilizados para tareas relacionadas con el campo del *Natural Language Processing* (NLP), el campo del *Speech Recognition* (SR), el campo del *Speech Synthesis* (SS), y el campo del *Emotion Recognition* (ER).

Desde que en 2017 salió a la luz el artículo *Attention is all you need* [11], en el que se presentaba la arquitectura *Transformer*, se han realizado numerosos trabajos que han mostrado prometedores resultados en la clasificación de textos y audios. [2] [9]

Sin embargo, su uso estaba reservado a grandes empresas con grandes recursos, ya que el entrenamiento de estos modelos requiere de una gran cantidad de datos y de una gran capacidad de cómputo.

Además de esto, la gran complejidad que presentaban hacía que su uso fuera muy complicado para usuarios con pocos conocimientos en el campo del *Deep Learning*. La aparición de librerías de un mayor nivel de abstracción ha acercado el uso de estas arquitecturas a usuarios con menos conocimientos. Una de las librerías más utilizadas es *Transformers*, desarrollada y mantenida por *Hugging Face*, que ofrece una gran cantidad de herramientas para la carga de modelos pre-entrenados, realizar ajustes finos de estos modelos, procesar datos de entrada, etc. Está pensado para poder ser usado por todo tipo de usuarios, desde usuarios con pocos conocimientos que quieran usar modelos pre-entrenados, usuarios con conocimientos avanzados que quieran realizar un ajuste fino de los modelos pre-entrenados, hasta investigadores que quieran crear sus propios modelos. [5]

2.2.2 Elección del modelo

Una vez se ha realizado una investigación sobre el estado del arte, se ha decidido utilizar un modelo pre-entrenado de los disponibles en la librería *Transformers*. Esta elección no es sencilla debido a la gran cantidad de opciones disponibles y la aparente similitud entre ellas. Para poder elegir el modelo más adecuado, se ha realizado una búsqueda sobre cuáles son los modelos más utilizados en la clasificación de audios.

Tras investigar sobre los modelos más utilizados, se ha decidido utilizar el modelo *Wav2Vec2* [1] Este modelo ha sido creado por *Facebook AI* específicamente para ser utilizado en tareas de los campos de *Speech Recognition* (SR) y *Audio Classification*. Además, es un modelo popular entre usuarios de la librería *Transformers*, incluso cuenta con ejemplos de uso en la documentación oficial de la librería.

Por estos motivos, se ha considerado más que apropiado para la resolución del problema que se plantea en este proyecto.

También ha sido empleado otro modelo bastante más grande, el modelo *XLSR-Wav2Vec2* [3]. Es una mejora del modelo "Wav2Vec2", que ha sido entrenado con una gran cantidad de datos de diferentes idiomas. Este modelo ha sido utilizado para la clasificación de audios en diferentes idiomas, y se ha considerado interesante probarlo para la clasificación de audios en español. La elección de este modelo se ha realizado con la finalidad de mejorar los resultados obtenidos con el modelo "Wav2Vec2", ya que es un modelo más versátil para la clasificación de audios en diferentes idiomas. [6] Sin embargo, al ser más complejo requeriría una mayor capacidad de cómputo para su entrenamiento, además de un mejor ajuste de los parámetros. Al no

obtener mejores resultados que el modelo "Wav2Vec2", siendo más exigente computacionalmente, se ha decidido no utilizarlo en la solución final.

2.3 Entrenamiento

Una vez se ha seleccionado el modelo, se ha procedido a realizar el entrenamiento del mismo.

2.3.1 Preparación del entrenamiento

Antes de comenzar el entrenamiento del modelo, es necesario realizar una serie de preparaciones previas. La siguiente elección consiste en decidir qué librería de bajo nivel se va a utilizar para el entrenamiento del modelo. *Transformers* ofrece una documentación muy extensa con multitud de ejemplos en muchos campos, y dentro de estos ejemplos, podemos elegir entre *PyTorch* y *TensorFlow*, dos librerías de bajo nivel muy populares en el campo del *Deep Learning*.

En principio no debe existir diferencia entre utilizar una u otra, ya que ambas librerías ofrecen las mismas funcionalidades. Dependerá de la experiencia del usuario con una u otra, o de la preferencia del usuario. En este caso, se ha optado por utilizar *PyTorch*, ya que es una librería con la que se tiene algo más de experiencia, y además, es la librería que se utiliza en la documentación oficial de *Transformers*. Para un caso de uso en el que se requiera una mayor optimización de la solución, convendría estudiar más en profundidad las diferencias entre ambas librerías, y elegir la que mejor se adapte a las necesidades del usuario.

Una vez se ha elegido la librería de bajo nivel, se ha procedido a la preparación de los datos de entrada. Los datos son en primer lugar descargos en una carpeta local. Esta tarea es sencilla ya que *Kaggle* permite cargar automáticamente los datos desde un script de Python.

Los datos no pueden ser introducidos directamente en la red neuronal, ya que esta espera recibir los datos en un formato específico. Para esto, el modelo ofrece una herramienta que se encarga de procesar los datos de entrada y convertirlos en el formato que espera recibir la red neuronal.

El preprocesado de los datos, se basa en los siguientes pasos:

1. Descarga de los datos desde *Kaggle*.
2. Descarte de la clase *surprised*.
3. Creación de un dataframe con los datos descargados.
4. Conversión a objeto interpretable por la librería de Hugging Face.
5. Mezcla de los datos y división en conjunto de entrenamiento y conjunto de test.
6. Codificación de las etiquetas.
7. Preparación del Feature Extractor de nuestro modelo.
8. Procesado de los datos de entrada (frecuencia de muestreo y duración).

Código 2.1 Preprocesado del dataset.

```
from glob import glob
import pandas as pd
import datasets

# download dataset
import opendatasets as od
od.download("https://www.kaggle.com/datasets/uldisvalainis/audio-emotions")

df=pd.DataFrame(glob('audio-emotions/Emotions/*/*.wav'),columns=['paths'])

df['labels']=df.paths.apply(lambda x:x.split('/')[-2])

# drop rows where label is 'Surprised'
df=df[df.labels!='Suprised']
```

```
hf_data=datasets.Dataset.from_pandas(df)
hf_data=hf_data.class_encode_column("labels")#convert label to HF label class

hf_data=hf_data.train_test_split(train_size=0.8,seed=0)

# Create a dictionary that maps a label name to an integer and vice versa. The
    mapping will help the model recover the label name from the label number:
labels = hf_data['train'].features['labels'].names
label2id, id2label = dict(), dict()
for i, label in enumerate(labels):
    label2id[label] = str(i)
    id2label[str(i)] = label

#feature extractor
from transformers import AutoFeatureExtractor
model='facebook/wav2vec2-base-960h'
feature_extractor = AutoFeatureExtractor.from_pretrained(model)

#process the data such as fix sample rate
hf_data = hf_data.cast_column("paths", datasets.Audio(sampling_rate=16_000))

def preprocess_function(examples):
    audio_arrays = [x["array"] for x in examples["paths"]]
    inputs = feature_extractor(
        audio_arrays, sampling_rate=feature_extractor.sampling_rate, max_length
            =16000*2, truncation=True
    )
    return inputs


encoded_dataset = hf_data.map(preprocess_function, remove_columns=["paths"],
    batched=True)
```

2.3.2 Entrenamiento del modelo

Una vez se ha realizado el preprocesado de los datos, como se muestra en el Código 2.1, podemos entrenar el modelo.

Debemos definir ahora la métrica que vamos a utilizar para evaluar el desempeño del modelo. De este modo podremos calcular lo bien o mal que se comporta el modelo durante el entrenamiento, y así ir ajustando los pesos.

En este caso, se ha optado por utilizar la métrica *accuracy*, que define la proporción de predicciones correctas realizadas por el modelo. Para esta tarea, de nuevo, existe una librería de *Hugging Face* que facilita la integración de esta métrica en el entrenamiento del modelo.

En este punto, podemos comenzar el entrenamiento del modelo. La librería *Transformers* permite guardar checkpoints del modelo durante el entrenamiento, de modo que podemos parar el entrenamiento en cualquier momento y continuar desde el último checkpoint guardado. Una vez completado el entrenamiento, subiremos la mejor versión del modelo a la plataforma *Hugging Face* para poder utilizarlo en producción. De este modo nos aseguramos de que el modelo va a estar disponible en cualquier momento, y podemos compartirlo con otros usuarios, además de las ventajas que ofrecen los sistemas de control de versiones.

Los requisitos de seguridad de un proyecto real podrían requerir que el modelo no estuviera disponible públicamente, por lo que se debería buscar una alternativa para compartir el modelo con otros usuarios. Como no es objetivo de este proyecto profundizar en este aspecto, se ha optado por aprovechar las ventajas de utilizar los servicios gratuitos que ofrece *Hugging Face*.

El modelo desarrollado se encuentra disponible en la plataforma *Hugging Face* guardado con el nombre de *antonjaragon/emotions_6_classes_small* donde se muestran los resultados, y podemos probar el modelo

con audios de prueba. Por otro lado, aunque no ha sido utilizado en este trabajo, también existe el modelo entrenado con el modelo *XLSR-Wav2Vec2*, guardado con el nombre de *antonjaragon/emotions_6_classes*.

En el **repositorio** se puede acceder a información más detallada sobre el entrenamiento del modelo, y se puede probar el modelo con audios de prueba.

Código 2.2 Entrenamiento del modelo.

```
import evaluate
accuracy = evaluate.load("accuracy")

def compute_metrics(eval_preds):
    # metric = datasets.load_metric("accuracy")
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=1)
    return accuracy.compute(predictions=predictions, references=labels)

from transformers import AutoModelForAudioClassification, TrainingArguments,
    Trainer

num_labels = len(id2label)
model = AutoModelForAudioClassification.from_pretrained(model, num_labels=
    num_labels, label2id=label2id, id2label=id2label)

training_args = TrainingArguments(
    output_dir="./emotions",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    fp16=True,
    learning_rate=3e-5,
    per_device_train_batch_size=32,
    gradient_accumulation_steps=4,
    per_device_eval_batch_size=32,
    num_train_epochs=10,
    warmup_ratio=0.1,
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    push_to_hub=True,

)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=encoded_dataset['train'],
    eval_dataset=encoded_dataset["test"],
    tokenizer=feature_extractor,
    compute_metrics =compute_metrics,

)

trainer.train()
```

Las posibilidades de la librería *Transformers* son muy amplias, y permiten realizar ajustes finos del modelo, como por ejemplo, la posibilidad de utilizar diferentes funciones de pérdidas, división de los datos en batches, diferentes métricas, etc. Esto permite que el usuario pueda ajustar el modelo a sus necesidades, y pueda realizar un entrenamiento más eficiente.

En este caso, se ha optado por utilizar las opciones por defecto, e intentar mejorar resultados modificando algunos parámetros como el ratio de aprendizaje o el número de épocas de entrenamiento. Al ser un procesamiento muy costoso, no se ha podido profundizar mucho en este aspecto, pero se ha conseguido un modelo que ofrece una funcionalidad básica.

Los resultados obtenidos por el mejor modelo tras el entrenamiento son los siguientes:

- **Accuracy**: 0.7920
- **Loss**: 0.9106

2.4 Saturn Cloud

Para el entrenamiento del modelo, se ha hecho uso de los recursos gratuitos que ofrece la plataforma *Saturn Cloud*.

El proceso de entrenamiento de un modelo de *Deep Learning* implica un elevado coste computacional. Las primeras pruebas se realizaron en un portátil personal con una tarjeta gráfica integrada.

Realizar entrenamientos pesados de varias horas de duración no es viable para un ordenador personal, ya que el desgaste sería muy elevado. Por ello, se han buscado alternativas a probar en este proyecto en concreto. Utilizar servicios de terceros para el entrenamiento de modelos puede no ser viable en muchos casos, debido a que necesitamos cargar los datos en la nube. Esto puede ser un problema si los datos son sensibles, ya que no podemos garantizar la seguridad de los mismos. Sin embargo, al estar utilizando un dataset público, no es un problema cargar los datos en el servidor de entrenamiento.

Además, el coste de utilizar estos servicios puede ser muy elevado, ya que el entrenamiento de un modelo puede durar varias horas, y el coste se calcula en función del tiempo de uso de los recursos. Esta solución no sería la mejor para muchos escenarios, pero en este caso, se ha optado por utilizar los recursos gratuitos que ofrece la plataforma *Saturn Cloud*. La elección se debe a que es una de las pocas plataformas que ofrecen una instancia con GPU en el segmento gratuito. Para nuevos usuarios, contamos con 150 horas de uso de una instancia con GPU, que es más que suficiente para realizar varios entrenamientos del modelo.

Este tipo de plataformas están directamente enfocadas al entrenamiento de modelos de *Deep Learning*, por lo que ofrecen una gran cantidad de herramientas para facilitar el entrenamiento de modelos. Existe multitud de imágenes pre-configuradas con las herramientas más utilizadas, y además, ofrecen la posibilidad de crear imágenes personalizadas con las herramientas que necesitemos. En concreto, se han utilizado imágenes *Debian* con *PyTorch* pre-instalado.

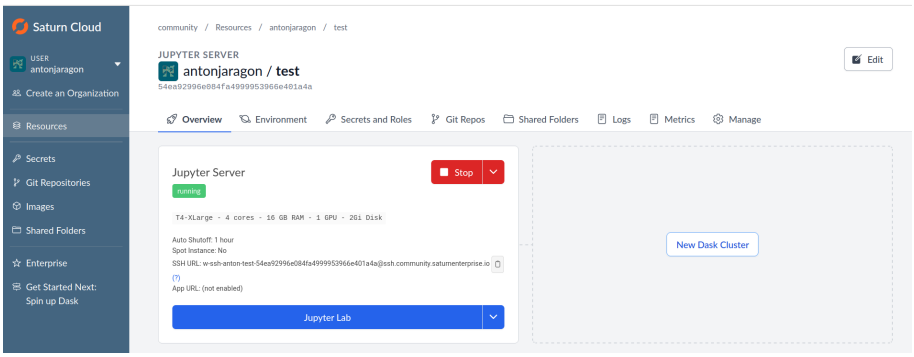

Figura 2.5 Ejemplo de instancia activa con GPU en Saturn Cloud.

2.5 Evaluación del modelo

Conocemos el accuracy del modelo tras el entrenamiento, pero realmente no tenemos suficiente información para saber si el modelo es capaz de clasificar correctamente los audios. Tampoco sabemos exactamente dónde podrían estar las debilidades del modelo, por lo que estaríamos a ciegas para intentar mejorar su desempeño.

Debemos extraer más información del modelo para poder evaluarlo correctamente. Esta información ha sido obtenida mediante el cálculo de métricas de evaluación del modelo, y mediante la visualización de la matriz de confusión.

2.5.1 Dataset de evaluación

Para poder evaluar el modelo, se ha utilizado un dataset diferente al utilizado para el entrenamiento. El dataset elegido es el dataset *AESDD* [12][13], que contiene audios grabados por actores interpretando diferentes emociones.

El idioma de los audios es el griego, cuenta con un total de 7.000 audios, y una duración de 3 a 10 segundos cada uno. Están clasificados en las siguientes emociones: ***anger***, ***disgust***, ***fear***, ***happiness*** y ***sadness***.

La elección de este dataset se debe principalmente a la similitud que presenta con el dataset utilizado para el entrenamiento. Esto facilita la comparación de los resultados obtenidos con el modelo entrenado, y nos permite evaluar el desempeño del modelo en un escenario más realista.

Además, al ser un dataset de un idioma diferente, supone un reto mayor para el modelo, ya que no ha sido entrenado con audios en griego. El modelo no es capaz de transcribir los audios, pero las emociones no son expresadas del mismo modo entre diferentes culturas. Esta prueba nos ofrece una visión más realista de cómo se comportaría el modelo en un escenario real.

2.5.2 Métricas de evaluación

Para calcular las métricas, se han realizado predicciones con el modelo sobre el dataset de evaluación. Estas predicciones se han comparado con las etiquetas reales de los audios, y se han calculado las métricas de evaluación.

Accuracy

El *accuracy* se define como la proporción de predicciones correctas realizadas por el modelo. Es la métrica más utilizada para evaluar el desempeño de un modelo, ya que nos ofrece una visión general del desempeño del modelo. También fue la métrica utilizada para evaluar el desempeño del modelo durante el entrenamiento.

Primero es necesario definir los términos *True Positive*, *True Negative*, *False Positive* y *False Negative*. Son los resultados de una predicción, y se definen de la siguiente manera:

- **True Positives (TP)**: Casos en los que el modelo predice correctamente la clase positiva.
- **True Negatives (TN)**: Casos en los que el modelo predice correctamente la clase negativa.
- **False Positives (FP)**: Casos en los que el modelo predice incorrectamente la clase positiva.
- **False Negatives (FN)**: Casos en los que el modelo predice incorrectamente la clase negativa.

El accuracy se calcula de la siguiente manera:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

El modelo ha obtenido un accuracy del **57 %**. Es un resultado bajo, pero hay que ponerlo en contexto.

Es importante comentar este hecho: el dataset fue sometido a un experimento en el que un grupo de personas clasificó los audios en las mismas clases que el modelo. [8] El resultado obtenido fue un accuracy del 74 %, lo cual podría indicar que es complicado clasificar correctamente los audios de este dataset, que quizás haya un problema con el etiquetado de los audios, o que quizás algunas clases no son separables.

De todos modos, es muy mejorable, pero no tenemos información sobre cómo mejorarlo. Necesitamos más métricas.

Precision

La *precision* se define como la proporción de predicciones positivas correctas entre el total de predicciones positivas realizadas por el modelo. Nos informa de cuántas de las predicciones positivas son realmente positivas. Es útil para detectar si los falsos positivos son un problema en nuestro modelo.

Se calcula de la siguiente manera:

$$Precision = \frac{TP}{TP + FP} \tag{2.2}$$

Como en este caso tenemos 5 clases, al cálculo de la predicción general se le aplica una media aritmética para obtener la precisión de cada clase.

$$Precision = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i + FP_i} \tag{2.3}$$

En este caso, se ha calculado el valor medio de la suma de las precisiones de cada clase. Hay diversos métodos para calcular la media, cada uno interpreta la precisión de una manera diferente. Este método es el más directo, pero para obtener una visión más completa del desempeño del modelo, se deberían estudiar los resultados de cada clase por separado.

Recall

El *recall* se define como la proporción de predicciones positivas correctas entre el total de predicciones positivas reales. Se utiliza para determinar cuál es el mejor modelo en caso de tener un alto número de falsos negativos.

Se calcula de la siguiente manera:

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

De nuevo, el método utilizado para calcular la media es el mismo que en el caso de la precisión.

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{TP_i}{TP_i + FN_i} \tag{2.5}$$

F1 Score

El *F1 Score* se define como la media armónica entre la precisión y el recall. Es utilizado cuando se busca un equilibrio entre ambas métricas.

Se calcula de la siguiente manera:

$$F1Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.6}$$

De nuevo, el método utilizado para calcular la media es el mismo que en el caso de la precisión.

$$F1Score = \frac{1}{n}\sum_{i=1}^{n} 2 * \frac{Precision_i * Recall_i}{Precision_i + Recall_i} \tag{2.7}$$

Tabla 2.1 Métricas de evaluación del modelo.

Metric	Precision	Recall	F1 Score
anger	0.654	0.583	0.617
disgust	1.000	0.017	0.033
fear	0.598	0.874	0.710
happiness	0.435	0.790	0.561
sadness	0.763	0.586	0.663
Average	0.690	0.570	0.517

Matriz de confusión

La matriz de confusión es una herramienta que nos permite visualizar los resultados de las predicciones realizadas por el modelo. Muestra la cantidad de predicciones correctas e incorrectas realizadas por el modelo, organizadas por clase.

Nos permite obtener información sobre las relaciones entre las clases, de modo que podemos determinar cómo de bien identifica el modelo cada clase. Es también una forma fácil de detectar si el modelo está confundiendo dos clases.

En nuestro caso, lo primero que salta a la vista al ver la Figura 2.6 es que el modelo no es capaz de identificar correctamente la clase *disgust*. Esto puede deberse a que quizás el etiquetado de audios es diferente entre el dataset de entrenamiento y el dataset de evaluación.

La matriz de confusión nos ha ayudado a descubrir este problema. Mejorando el etiquetado de los dataset nos ayudaría a mejorar el accuracy.

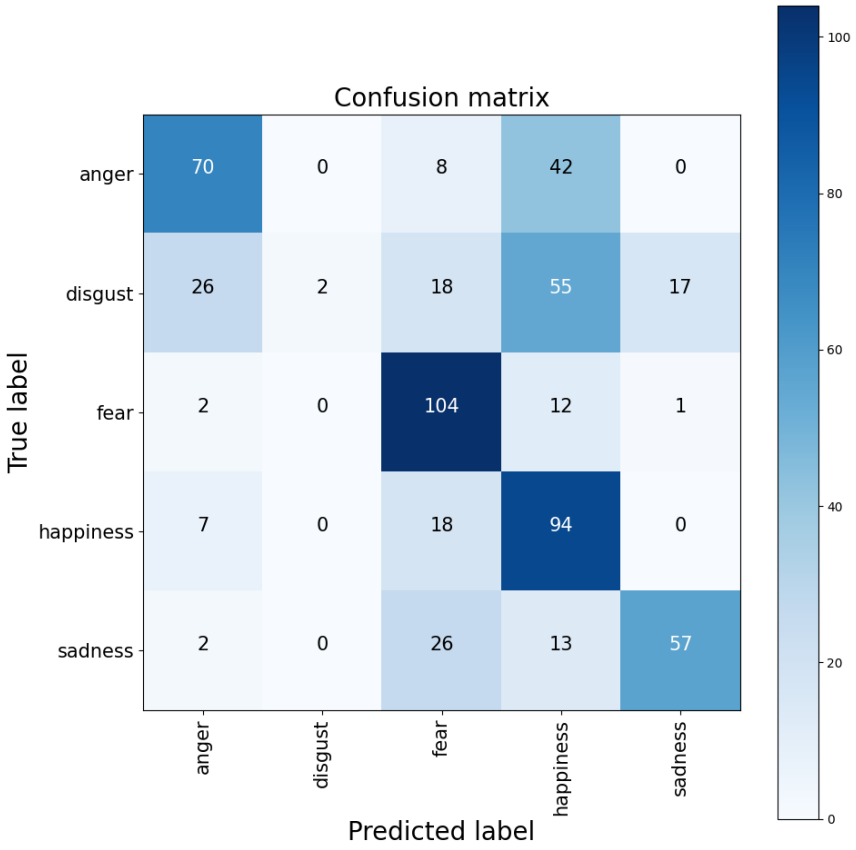

Figura 2.6 Matriz de confusión del modelo.

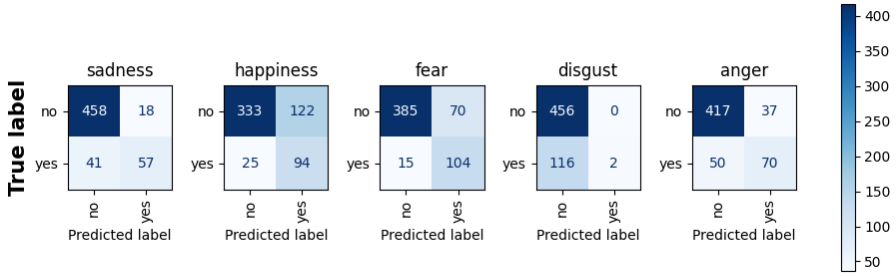

Figura 2.7 Matriz de confusión multi clase del modelo.

Vemos también que la case *fear* es la que mejor identifica el modelo, y la clase *happiness* es la que peor identifica. Para mejorar este aspecto podríamos añadir más audios de la clase *happiness* al dataset de entrenamiento.

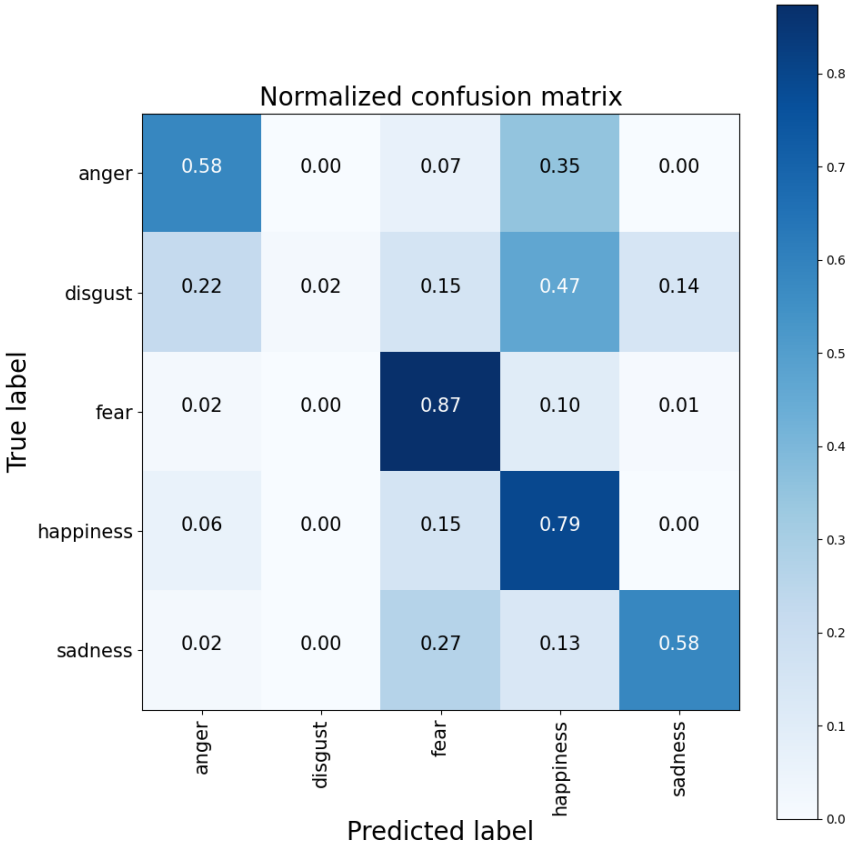

Figura 2.8 Matriz de confusión normalizada del modelo.

3 Despliegue del modelo

Less is more.

Ludwig Mies van der Rohe

3.1 Problemática y solución

Nos encontramos en esta situación: el modelo ha sido entrenado o está siendo desarrollado por otro equipo. Nuestro trabajo consiste en implementarlo en un entorno de producción para que pueda ser utilizado por los usuarios finales.

Este problema puede ser abordado de varias formas, y la solución estará altamente condicionada por los requisitos del proyecto. Al no tener ningún requisito impuesto para este trabajo, surgen muchos interrogantes cuya respuesta no es trivial:

- **Aspecto final de la solución**: ¿Cómo se va a utilizar el modelo? ¿Qué tipo de interfaz se va a utilizar? ¿Qué tipo de dispositivo se va a utilizar?
- **Requisitos de la solución**: ¿Qué requisitos de rendimiento tiene la solución? ¿Con qué riesgos podemos encontrarnos? ¿Qué requisitos de escalabilidad tiene la solución?
- **Arquitectura de la solución**: ¿En qué lenguaje se va a implementar la solución? ¿Qué tipo de arquitectura se va a utilizar? ¿Qué tipo de servidores se van a utilizar?

En este caso, se ha optado por simplificar el aspecto final de la solución para centrarnos en el despliegue del modelo. Se ha decidido crear una interfaz web que permita a los usuarios finales interactuar con el modelo, y una implementación mediante contenedores Docker para facilitar el despliegue en cualquier entorno.

3.2 Enfoque escogido

En un principio se pensó en realizar inferencia en tiempo real, es decir, que la aplicación, una vez iniciada, estuviera grabando continuamente y realizando predicciones. Esta opción sin embargo, no ha podido ser llevada a cabo de forma exitosa debido al tiempo de respuesta que ofrece el servidor en el que se ha desplegado la aplicación.

Además, para implementarla de forma correcta, habría que añadir diversos mecanismos que ayuden a filtrar los audios y nos permitieran extraer muestras que pudieran ser utilizadas para realizar predicciones. Esto implicaría detección de inicio y fin de actividad vocal, filtrado de silencios, etc. La complicación de este proceso, unido al tiempo de respuesta del servidor, ha hecho que se descarte esta opción.

Finalmente, contamos con una aplicación web que permite a los usuarios finales grabar audios y obtener una predicción de la clase a la que pertenece el audio, se presuponen audios válidos para clasificar por emociones. La aplicación está pensada para ser utilizada en una sola dirección: el usuario debe iniciar la grabación, grabar un mensaje, parar la grabación y esperar el resultado. Una vez obtenido el resultado, puede volver a grabar otro mensaje.

La aplicación es accesible desde cualquier dispositivo que tenga un navegador web a través de la dirección *https://www.classifier-web.com/*.

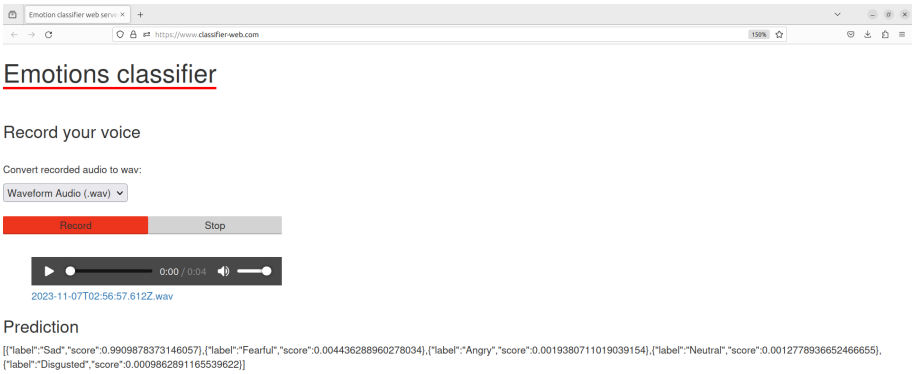

Figura 3.1 Resultado mostrado en la interfaz tras realizar una predicción.

3.3 Aplicación web

Una forma sencilla de crear una interfaz que sea accesible desde cualquier dispositivo es crear una aplicación web.

Aunque el modelo creado puede ser integrado utilizando cualquier lenguaje de programación, se ha optado por utilizar *Python* para la implementación de la aplicación web. *Python* cuenta con una gran cantidad de librerías que facilitan la implementación de aplicaciones web, como *Flask* o *Django*, quizás las más populares.

Se ha optado por utilizar *Flask*, debido a que es una librería más ligera que Django y a que es más sencilla de utilizar. Además, contamos con cierta experiencia previa en el uso de *Flask*, lo que nos permite acelerar el desarrollo de la aplicación.

3.3.1 *Flask*

Flask es un micro framework para *Python* que permite crear aplicaciones web de forma sencilla.

Está diseñado para ser extensible, por lo que es posible añadirle funcionalidades mediante extensiones, aunque en este caso no vamos a utilizar ninguna. Sin embargo, estas extensiones de alto nivel nos abren las puertas a posibles líneas futuras, como lecturas de bases de datos, autenticación de usuarios, etc.

Utilizar *Python* para la creación web no siempre es la solución idónea, ya que existen otros lenguajes de programación que están más orientados a la creación de aplicaciones web. Sin embargo, si no se tiene experiencia previa en estos lenguajes, o el objetivo es lanzar una aplicación web de forma rápida, *Flask* es idóneo. No estamos exentos de tener que crear plantillas en otros lenguajes propios de la web, como HTML, CSS o JavaScript, pero *Flask* nos permite crear una aplicación web funcional en muy poco tiempo.

3.3.2 Estructura de la aplicación

Aunque *Flask* ofrece muchas posibilidades, en este caso se ha optado por crear una aplicación web muy sencilla, que consta de dos partes principales.

Primero el modelo es cargado en memoria, y se crea una instancia de *Flask*. El modelo es cargado en memoria para evitar tener que cargarlo cada vez que se realiza una predicción, lo que se traduciría en un aumento del tiempo de respuesta de la aplicación. Esto puede ralentizar sin embargo el arranque de la aplicación, pero es una operación que se realiza una única vez.

Posteriormente se crean las rutas de la aplicación, que son las direcciones a las que se puede acceder desde un navegador web. Contamos con la ruta principal, que es la que se utiliza para cargar la página principal de la aplicación, y la ruta de predicción, que es la que se utiliza para realizar las predicciones.

La ruta principal simplemente carga un fichero HTML que contiene el código de la página principal.

La ruta de predicción es llamada internamente mediante una petición POST lanzada por el código del lado del cliente escrita en *JavaScript* cuando un usuario termina una grabación. La grabación se guarda localmente en el servidor momentáneamente para que el modelo pueda realizar la predicción sobre ella, y posteriormente se borra, por cuestiones de espacio y privacidad.

Código 3.1 Aplicación Flask..

```
from flask import Flask, request, jsonify, render_template, redirect
import os
import random

from transformers import pipeline

app = Flask(__name__)
pipe = pipeline("audio-classification", model="antonjaragon/
    emotions_6_classes_small")

@app.route("/", methods=["GET", "POST"])
def index():
predict = ""
if request.method == "POST":
    print("FORM DATA RECEIVED")

    if "file" not in request.files:
        return redirect(request.url)

    file = request.files["file"]
    if file.filename == "":
        return redirect(request.url)

    if file:

        predict = pipe(file.filename)



return render_template('index.html')

@app.route('/save_recording', methods=['POST'])
def upload():
if request.method == 'POST':
    f = request.files['audio_data']
    # audio folder is static/audios/random int/filename
    audio_folder = 'static/audios/' + str(random.randint(1, 100000000000000000)
        )
    # create folder if not exists audio_folder
    if not os.path.exists(audio_folder):
        os.makedirs(audio_folder)
    f.save(os.path.join(audio_folder, f.filename))
    output = pipe(audio_folder + '/' + f.filename)
    # print(output)

    # delete the audio folder after prediction
    os.remove(audio_folder + '/' + f.filename)
    os.rmdir(audio_folder)

    return output


@app.route('/cache-me')
def cache():
return "server will cache this response"
```

```
@app.route('/info')
def info():

resp = {
    'connecting_ip': request.headers['X-Real-IP'],
    'proxy_ip': request.headers['X-Forwarded-For'],
    'host': request.headers['Host'],
    'user-agent': request.headers['User-Agent']
}

return jsonify(resp)

@app.route('/flask-health-check')
def flask_health_check():
return "success"
```

3.3.3 Interfaz web

El desarrollo de interfaces web es un mundo complejo, y no es el objetivo de este trabajo crear una interfaz especialmente atractiva, sino más bien que nos proporcione la funcionalidad básica. Existen desarrolladores especializados únicamente en el desarrollo de interfaces web, y es un campo que requiere de un conocimiento muy amplio, además de experiencia.

Debido a tratar esta parte como algo secundario, sumado a la falta de conocimiento acerca del manejo de audios en la web, se ha optado por basar la interfaz en trabajo previo realizado por otros desarrolladores. En concreto, se ha utilizado como base el proyecto con licencia para uso público alojado en *GitHub* titulado *Voice-to-text-with-Python-in-Flask* [14].

La interfaz web final es muy sencilla, y se puede ver en la Figura 3.1. Contiene lo básico para que un usuario pueda realizar la grabación de un audio y obtener una predicción de la clase a la que pertenece el audio.

3.4 Aplicación *Flask* en producción

Flask integra un servidor web apto para entornos de desarrollo, que es el que se utiliza por defecto cuando se lanza la aplicación, llamado *Werkzeug*. Este servidor es muy sencillo de utilizar, pero no está pensado para ser utilizado en producción, ya que no está optimizado para soportar un gran número de peticiones simultáneas.

3.4.1 *Gunicorn*

Para lanzar la aplicación en producción, se ha optado por utilizar *Gunicorn*, un servidor web *HTTP WSGI* (Web Server Gateway Interface) para *Python*. Es uno de los servidores más utilizados para lanzar aplicaciones *Flask* en producción, y es además recomendado en la documentación oficial de *Flask*.

Gunicorn es un servidor web que se encarga de gestionar las peticiones HTTP que llegan a la aplicación, y de lanzar procesos de la aplicación para atender estas peticiones. Esto permite que la aplicación pueda atender varias peticiones simultáneamente, lo que se traduce en un aumento del rendimiento de la aplicación.

Código 3.2 Código guardado en 'wsgi.py'..

```
from app import app
import os

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8000, debug=True)
```

Código 3.3 Comando para lanzar la aplicación con *Gunicorn*..

```
gunicorn -b 0.0.0.0:8000 wsgi:app
```

El Código 3.3, lanzará un servidor web en el puerto 8000, el puerto que se suele usar por defecto, que permite conexiones desde cualquier dirección IP.

El siguiente paso es configurar un servidor web que actúe como proxy inverso, para que las peticiones HTTP que lleguen al servidor web sean redirigidas al servidor *Gunicorn*.

3.4.2 *Traefik*

Para configurar el servidor web que actúe como proxy inverso, se ha optado por utilizar *Traefik*. Aunque *Nginx* es quizás el servidor web más utilizado para realizar esta tarea, la elección de *Traefik* se debe principalmente a su facilidad de configuración.

En la actualidad, *Nginx* ofrece más funcionalidades, pero para este caso con una configuración básica es suficiente. La mayor ventaja que nos ha brindado *Traefik* es la facilidad de generar certificados SSL para la aplicación, lo que nos permite utilizar HTTPS. Esto es importante, ya que si no se utiliza HTTPS, los navegadores web no permiten acceder al micrófono del dispositivo, lo que hace imposible la grabación de audios.

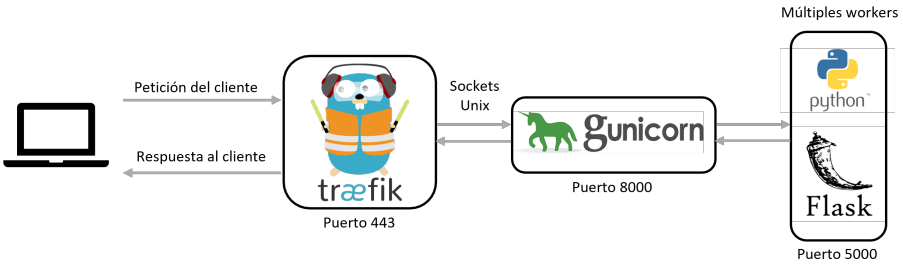

Figura 3.2 Esquema completo del sistema.

No es una tarea difícil de realizar correctamente para un desarrollador experimentado mediante un servidor web como *Nginx*, pero es mucho más sencillo de realizar con *Traefik*, y además, al contar con poca experiencia en este campo, nos ha permitido solventar este problema de forma rápida y sencilla. Además, *Traefik* aún está dando sus primeros pasos, y está ganando popularidad entre desarrolladores, por lo que quizás en un futuro sea una alternativa a *Nginx* también en entornos reales de producción.

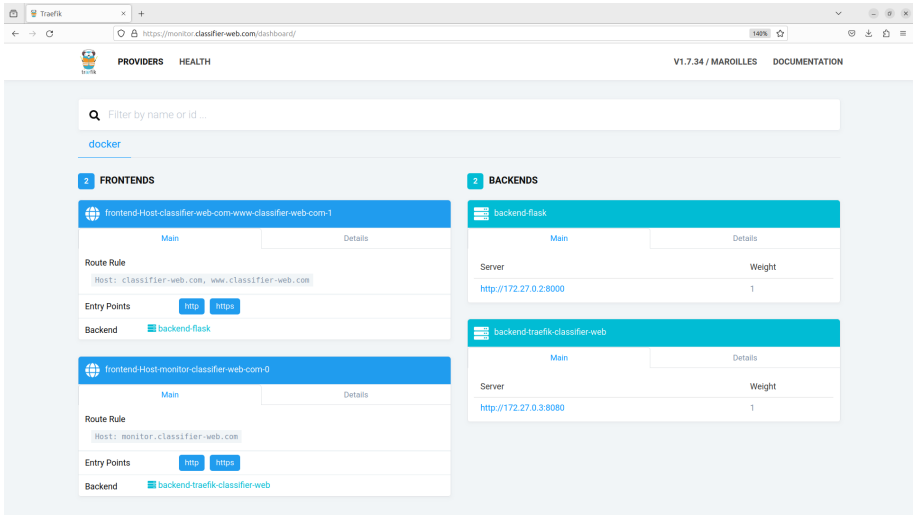

Figura 3.3 Panel de control de *Traefik*, accesible a través de la dirección *https://monitor.classifier-web.com* con contraseña admin:admin.

3.4.3 Docker

Para facilitar el despliegue de la aplicación en cualquier entorno, se ha optado por utilizar contenedores Docker. Esta tecnología permite encapsular una aplicación y sus dependencias en un contenedor, que puede ser ejecutado en cualquier entorno que tenga instalado Docker. De este modo nos aseguramos que únicamente tenemos que preocuparnos de que el entorno tenga instalado Docker.

Esta tecnología ayuda a eliminar muchos problemas a la hora de desplegar servicios, pero incorpora otros de los que hay que ser conscientes. En particular, Docker presenta un problema de seguridad, ya que los contenedores son ejecutados por defecto con privilegios de root. Esto implicaría que si un atacante consigue acceder al contenedor, puede tener acceso a todo el sistema.

Este problema se ha solventado creando un usuario no privilegiado dentro del contenedor al construir la imagen de la aplicación, y ejecutando la aplicación con este usuario. Sin embargo, las implicaciones de seguridad de Docker son un tema muy amplio y precisamente pueden llegar a ser determinantes para no utilizar esta tecnología en entornos de producción con requisitos de seguridad muy estrictos. No es el caso de este trabajo, pero es un tema que hay que tener en cuenta y debería ser estudiado en profundidad antes de utilizar Docker en entornos de producción.

A pesar de ello, las ventajas que ofrece Docker son muy interesantes, y es una tecnología que ha ganando mucha popularidad en los últimos años. Las principales ventajas que ofrece son las siguientes:

- **Portabilidad**: Docker permite encapsular una aplicación y sus dependencias en un contenedor, que puede ser ejecutado en cualquier entorno que tenga instalado Docker.
- **Escalabilidad**: Docker permite crear múltiples contenedores de una misma aplicación, lo que permite escalar la aplicación de forma horizontal.
- **Aislamiento**: Docker permite aislar una aplicación y sus dependencias en un contenedor, lo que permite que la aplicación no se vea afectada por otras aplicaciones que se estén ejecutando en el mismo entorno.

Para este proyecto, hemos utilizado la imagen oficial de *Traefik* y una imagen personalizada para la aplicación *Flask*. Para la construcción de la imagen de Flask, se ha utilizado una imagen base de *Python* y se han instalado las dependencias necesarias para la aplicación.

Código 3.4 Dockerfile para la aplicación *Flask*.

```
FROM python:3.10.13

RUN apt update && apt upgrade -y
RUN apt install -y ffmpeg

# get curl for healthchecks
RUN apt install curl

# permissions and nonroot user for tightened security
RUN adduser nonroot
RUN mkdir /home/app/ && chown -R nonroot:nonroot /home/app
RUN mkdir -p /var/log/flask-app && touch /var/log/flask-app/flask-app.err.log
    && touch /var/log/flask-app/flask-app.out.log
RUN chown -R nonroot:nonroot /var/log/flask-app
WORKDIR /home/app
USER nonroot

# copy all the files to the container
COPY --chown=nonroot:nonroot . .

# venv
ENV VIRTUAL_ENV=/home/app/venv

# python setup
```

```
RUN python -m venv $VIRTUAL_ENV
ENV PATH="$VIRTUAL_ENV/bin:$PATH"
RUN export FLASK_APP=app.py


# upgrade pip
RUN pip install --upgrade pip

RUN pip install torch torchvision torchaudio --index-url https://download.
    pytorch.org/whl/cpu

# RUN pip install 'transformers[torch]'

RUN pip install -r requirements.txt


# define the port number the container should expose
EXPOSE 5000

CMD ["python", "app.py"]
```

3.4.4 Docker Compose

Docker Compose es una herramienta que permite definir y ejectutar aplicaciones Docker de forma sencilla. Permite definir las imágenes de los contenedores, las redes, los volúmenes, etc., en un fichero YAML, y ejecutarlos con un único comando.

Es especialmente útil cuando se tienen varias aplicaciones que dependen unas de otras, ya que permite definir todas las aplicaciones en un único fichero. En nuestro caso contamos solo con dos contenedores, pero crear un fichero Docker Compose nos permite definirlos de forma sencilla, construir las imágenes con las dependencias que nosotros definamos y levantar el despliegue con un único comando.

La sintaxis general de un fichero Docker Compose consiste en definir los servicios que se van a utilizar, las imágenes que se van a utilizar para cada servicio, los volúmenes que deben ser creados, las redes, variables de entorno, etc.

En este caso, algunos aspectos a destacar de la definición del fichero son los siguientes:

- **Servicios**: Han sido definidos dos servicios, uno para la aplicación *Flask* y otro para el servidor *Traefik*.
- **Imagen**: Se ha utilizado la imagen oficial para el servidor *Traefik*, y una imagen de *Python* personalizada con las dependencias necesarias para la aplicación *Flask*.
- **Volúmenes**: Para el servicio de *Traefik* se han definido varios volúmenes para almacenar los certificados SSL y la configuración de *Traefik*.
- **Redes**: No ha sido necesario definir ninguna red, ya que por defecto Docker Compose crea una red interna que es suficiente para que los contenedores se comuniquen entre sí.
- **Variables de entorno**: Han sido definidas varias variables principalmente para el servicio de *Traefik*, de modo que pueda servir la aplicación *Flask* con HTTPS.

El proyecto completo se puede encontrar en *https://github.com/antaramol/classifier-web*.

Código 3.5 Fichero Docker Compose..

```
version: '3'

services:
    flask:
    build: ./flask
    image: flask_app
```

```
    command: gunicorn -w 3 -t 60 -b 0.0.0.0:8000 app:app
    labels:
        - "traefik.enable=true"
        - "traefik.backend=flask"
        - "traefik.frontend.rule=Host: $DOMAIN_NAME, www.$DOMAIN_NAME"
        - "traefik.port=8000"

    traefik:
    image: traefik:1.7-alpine
    environment:
        - DOMAIN_NAME=$DOMAIN_NAME
    volumes:
        - /var/run/docker.sock:/var/run/docker.sock:ro
        - ./traefik/traefik.toml:/etc/traefik/traefik.toml:ro
        - ./traefik/acme:/etc/traefik/acme
        - ~/.htpasswd:/etc/htpasswd/.htpasswd
    labels:
        - "traefik.enable=true"
        - "traefik.frontend.rule=Host: monitor.$DOMAIN_NAME"
        - "traefik.port=8080"
    ports:
        - "80:80"
        - "443:443"
```

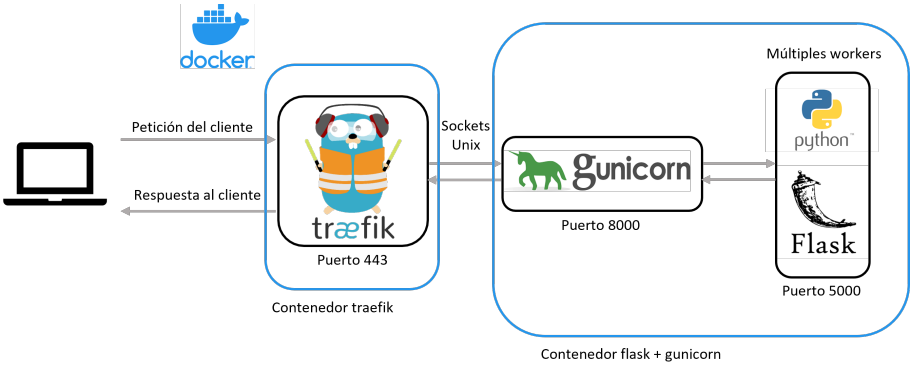

Figura 3.4 Esquema completo del sistema con Docker Compose.

3.5 Despliegue

Una vez que la aplicación está lista para ser desplegada, es necesario elegir un entorno de producción.

Varias opciones han sido probadas para este trabajo, ya que no se tenía ninguna restricción en cuanto al entorno de producción. Al no tener un servidor propio, se ha optado por utilizar servicios de terceros, que ofrecen servidores virtuales a un precio muy asequible.

De nuevo, igual que comentábamos en el apartado anterior, no es posible utilizar un ordenador personal para este servicio, ya que debería estar conectado todo el tiempo. Además, existen cada vez más opciones de hosting y es muy interesante utilizarlas, ya que nos permiten centrarnos en el desarrollo de la aplicación y no en la gestión del servidor.

Esta es la principal ventaja, nos olvidamos de gestionar la infraestructura. Además muchos servicios permiten gran facilidad para escalar horizontalmente, lo que nos permite aumentar la capacidad de la aplicación de forma sencilla.

Sin embargo, el principal problema es el coste, ya que estos servicios no son gratuitos. Es un inconveniente que, sobre todo para iniciados en este mundo, puede ser determinante para no utilizar estos servicios.

Varias opciones han sido probadas para este trabajo, ya que cada una de ellas ofrece diferentes opciones y precios.

3.5.1 Amazon Web Services

Amazon Web Services (AWS) es una plataforma de servicios en la nube que ofrece servicios de computación, almacenamiento, bases de datos, etc. Esta ha sido la primera opción probada debido a su gran popularidad y que se contaba con cierto conocimiento previo.

""" Insertar gráfico de popularidad"""

AWS ofrece una gran cantidad de servicios, y es una de las plataformas más completas que existen. Este ha sido el principal problema, ya que está pensado para ser utilizado por empresas que necesitan una gran cantidad de servicios por su fácil integración entre ellos. Como este trabajo solo necesita un servidor virtual, quizás AWS no sea la mejor opción.

En concreto, ha sido probado el servicio EC2, que ofrece servidores virtuales en la nube. Cuenta con una capa gratuita, que permite utilizar un servidor virtual de forma gratuita durante un año, pero con ciertas limitaciones.

La capacidad de cómputo en la capa gratuita es muy limitada, pero es suficiente para probar la aplicación. Sin embargo, el número de horas de uso es limitado, y una vez que se agotan, hay que pagar por cada hora de uso. Esto hace que no sea una opción viable para este trabajo, ya que el coste sería muy elevado. Este segmento está quizás pensado para pruebas puntuales de aplicaciones, pero no para desplegar aplicaciones en producción.

Sin embargo, sirvió para realizar diversas pruebas de funcionamiento, pruebas de rendimiento, etc., y para familiarizarse con este tipo de servicios.

3.5.2 Google Cloud Platform

El resultado es similar al de AWS, ya que Google Cloud Platform (GCP) está enfocado igualmente a grandes proyectos.

Además, la documentación de estos servicios es tan extensa que puede llegar a ser abrumadora para un desarrollador que no esté familiarizado con este tipo de servicios. Una mejor opción para este trabajo es utilizar servicios más sencillos, que estén pensados para pequeños proyectos y que sean más fáciles de utilizar, suavizando así la curva de aprendizaje.

La atracción hacia Google Cloud Platform es que ofrece una gran cantidad de crédito gratuito para utilizar sus servicios, lo que permite utilizarlos de forma gratuita durante un tiempo. El resultado es muy similar a AWS, esta vez se ha utilizado el servicio Compute Engine, que ofrece servidores virtuales en la nube.

3.5.3 *Contabo*

Una opción más sencilla y económica es utilizar un VPS (Virtual Private Server), que es un servidor virtual que se encuentra alojado en un servidor físico. Este tipo de servidores cuentan con una ventaja con respecto a los anteriores, y es que el precio es fijo, y no depende del uso que se haga del servidor.

En concreto, se ha utilizado el servicio de VPS de *Contabo*, recomendado por un compañero de la universidad. Este servicio ofrece servidores virtuales a un precio muy asequible, y con una gran cantidad de recursos.

Es muy sencillo desplegar una instancia de un servidor virtual, a la que podemos conectarnos mediante SSH. Si a esto le sumamos la facilidad que nos ofrece Docker Compose para desplegar la aplicación, junto con herramientas de edición como VSCode que nos permiten conectarnos y editar los ficheros de la aplicación de forma remota, el resultado es muy satisfactorio.

Contabo además nos ofrece muchas opciones que nos ayudan a perfilar el resultado final. En nuestro caso, se ha utilizado el área de manejo de DNS para configurar el dominio de la aplicación, de modo que podemos otorgarle un nombre más amigable a la aplicación. *Contabo* permite crear registros DNS de forma sencilla, y además ofrece un servicio de DNS dinámico, que permite asignar un nombre de dominio a una IP dinámica, que es la que nos ofrece el servidor virtual.

Para esta aplicación, se ha elegido un servidor de 4 núcleos, 8 GB de RAM y 50 GB de almacenamiento. Si en un futuro se necesitara más capacidad, se podría escalar horizontalmente, creando más instancias de la aplicación y balanceando la carga entre ellas. Para esta implementación, quizás sería más conveniente estudiar en profundidad los servicios de AWS o GCP, ya que ofrecen más facilidades para escalar horizontalmente, pero para el propósito de este trabajo, es más que suficiente.

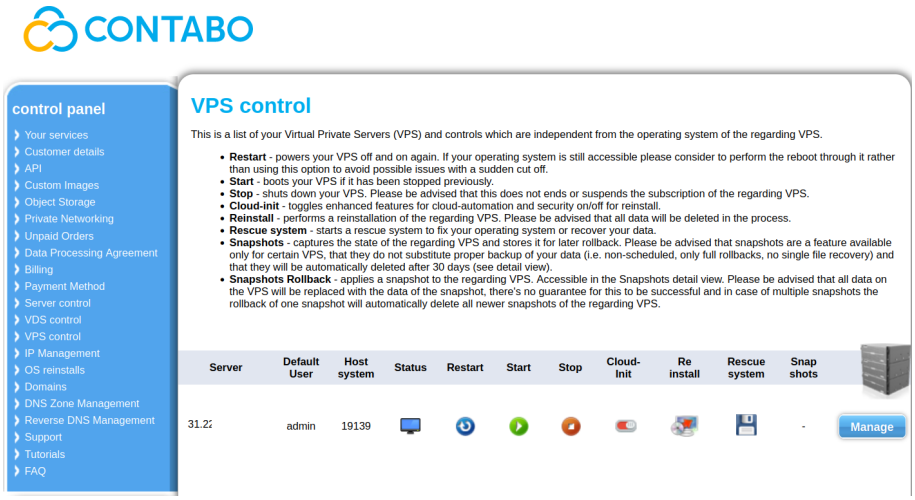

Figura 3.5 Instancias activas en Contabo.

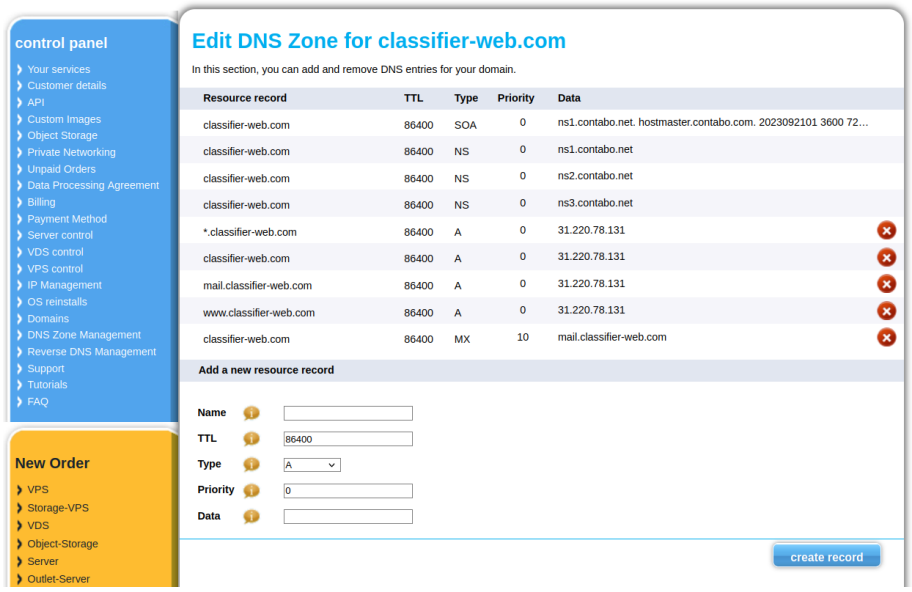

Figura 3.6 Configuración de DNS en Contabo.

4 Conclusiones

If we knew what it was we were doing, it would not be called research, would it?

Albert Einstein

Una vez finalizado el proyecto, es momento de hacer una valoración del mismo. En este capítulo se exponen las conclusiones obtenidas tras la realización del proyecto, así como las posibles mejoras que se podrían realizar en un futuro.

4.1 Hitos conseguidos

Después de haber estado enfocado en un proyecto exigente durante un periodo de tiempo prolongado, tener sentimientos encontrados es normal. Pensar que se podrían haber realizado más pruebas, o que se podría haber mejorado algún aspecto, es inevitable.

Sin embargo, hay que echar la vista atrás y ver de dónde venimos para entender el camino recorrido. El objetivo principal del proyecto, que era crear y desplegar un modelo clasificador de audios ha sido cumplido.

Al no haber utilizado los requisitos de forma rigurosa durante el desarrollo del proyecto, es difícil determinar el grado de éxito del mismo. Algunos de los aspectos más importantes que se han conseguido son los siguientes:

- El modelo es capaz de clasificar los audios del dataset original con una precisión superior al 70 %.
- Conocemos las debilidades del modelo y tenemos ideas sobre cómo mejorarlas.
- El sistema es accesible desde cualquier navegador web.
- El modelo puede ser reentrenado con nuevos datos, o entrenar un nuevo modelo.
- Todo el código ha sido desarrollado utilizando librerías de código abierto.
- La utilización de contenedores permite desplegar el sistema en cualquier máquina que tenga instalado Docker.
- El servidor web utiliza el protocolo HTTPS para asegurar la comunicación con el cliente.

Algunos requisitos como los tiempos de respuesta del sistema dependen en gran medida de la capacidad de computación. Además, pueden ser variables dependiendo de la carga de trabajo del servidor, de la calidad de la red, etc. Por lo tanto, se necesitaría un estudio más profundo para medirlos correctamente.

4.2 Lecciones aprendidas

El aprendizaje adquirido durante el desarrollo del proyecto ha sido muy valioso. Más allá de las cuestiones puramente técnicas, se han adquirido múltiples habilidades que serán de gran utilidad en el futuro.

El hecho de enfocar el proyecto como un problema real, obliga a imaginarse qué tipo de problemas pueden surgir y qué elección es la más adecuada para resolverlos. Aunque muchos requisitos han sido obviados,

sobre todo los relacionados con la seguridad, entender las implicaciones que tienen las decisiones tomadas es muy enriquecedor.

Quizás el mayor aprendizaje ha sido el relacionado con cómo los obstáculos han sido sorteados. Es inevitable que surjan problemas durante el desarrollo de un proyecto, y muchos de ellos parecen bloquear el avance. Sin embargo, es en estos puntos cuando hay que entender bien el problema para poder encontrar una solución, o un camino alternativo.

En relación con el previo acercamiento a esta problemática, muchos de estos obstáculos que antes bloqueaban el proyecto por completo, han podido ahora ser resueltos o evitados. En esta línea, es importante entender que muchas veces la solución no es un camino recto, sino que a veces hay que tomar desvíos para poder llegar al destino.

4.3 Líneas futuras

Esta sección pretende servir de autocrítica y de guía para futuros proyectos que quieran continuar con el trabajo realizado. Se van a enumerar los puntos que se consideran más importantes para mejorar el proyecto.

4.3.1 Mejorar el modelo

A pesar de contar con un modelo capaz de clasificar audios, hemos comprobado que los resultados pueden ser mejorables. En particular, sería conveniente prestar atención a las clases más problemáticas para subir el porcentaje de acierto.

Para ello, se podrían realizar las siguientes acciones:

- Aumentar el número de muestras de las clases más problemáticas.
- Ajustar parámetros de entrenamiento.
- Probar distintos modelos preentrenados.
- Probar técnicas de Data Augmentation.

4.3.2 Añadir funcionalidad al sistema

Otro punto particularmente débil de la implementación final del sistema es que tiene una funcionalidad muy limitada. El sistema es capaz de clasificar audios de entrada que se suponen válidos, pero nada más.

Sería interesante añadir funcionalidades que permitan al usuario interactuar con el sistema de forma más natural. Por ejemplo, grabar de forma continua y detectar emociones dependientes del tiempo. Esta nueva funcionalidad no se consigue simplemente grabando de forma continua y haciendo predicciones, sería conveniente añadir un bloque detector de actividad vocal.

Una posible futura mejora podría ser un sistema capaz de detetar palabras clave. El desarrollo de este tipo de sistemas es muy similar al de los clasificadores de emociones, pero se necesitaría un dataset de grabaciones de la palabra clave en cuestión. Acompañado de un estudio de las palabras clave que pudieran resultar de interés, este nuevo sitema podría ser de gran utilidad al clasificador de emociones.

También se podría investigar cómo identificar el hablante. Hasta el momento hemos supuesto que hay un solo hablante, pero en caso de que hubiera más de uno, sería interesante identificarlos para poder obtener información temporal de cada uno de ellos.

4.3.3 Verificación de funcionamiento

La verificación del funcionamiento de un sistema puede llegar a consumir la mayor parte del tiempo de desarrollo. En proyectos cuya disponibilidad es crítica, o se necesita un alto grado de confianza en el sistema, es necesario realizar una verificación exhaustiva del mismo.

Asegurar que el sistema no va a fallar es muy complejo, y requiere de un estudio profundo de los posibles casos de uso.

Para nuestro caso de uso, la evaluación del modelo debe ser mucho más exhaustiva, contando con un dataset de test mucho más variado (audios en varios idiomas, con ruido de fondo, audios de baja calidad, audios sin palabras, grabaciones genuinas, etc.).

En cuanto al servidor web, se debería realizar una verificación de la disponibilidad del sistema, pruebas de carga, pruebas de seguridad, etc.

En este proyecto, se ha realizado una verificación básica del funcionamiento del sistema, pero no se ha profundizado en ella.

4.3.4 Optimización de la solución

Reducir los recursos necesarios que necesita el sistema es siempre algo positivo. No sirve de nada un modelo con un 99 % de acierto si necesita más del tiempo crítico para realizar la predicción.

Por ello, paralelamente a buscar la mejor solución, en un proyecto real sería necesario balancear la solución con los recursos disponibles.

En caso de faltar recursos para realizar la predicción, se podría barajar la implementación de una solución 'b' que sea menos precisa pero más rápida.

La aplicación también debe ser optimizada según el número de usuarios que se espera que la utilicen. En caso de que el número de usuarios sea muy elevado, se debería estudiar la posibilidad de utilizar un servidor más potente, o incluso utilizar varios servidores en paralelo. El uso de contenedores facilita esta tarea, ya que se podrían desplegar instancias del servidor web en varios servidores y balancear la carga entre ellos.

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli, *wav2vec 2.0: A framework for self-supervised learning of speech representations*, 2020.

[2] Gaurab Banerjee, Emily Huang, and Allison Lettiere, *Understanding emotion classification in audio data*, 2021.

[3] Alexis Conneau, Alexei Baevski, Ronan Collobert, Abdelrahman Mohamed, and Michael Auli, *Unsupervised cross-lingual representation learning for speech recognition*, 2020.

[4] Universidad Internacional de Valencia, *¿cuáles son las aplicaciones de la ia actuales y futuras?*, April 2023.

[5] Hugging face docs, *Transformers*, 2021.

[6] Mehrdad Farahani, *Emotion recognition in greek speech using wav2vec 2.0*, GitHub, 2021.

[7] Sage Kelly, Sherrie-Anne Kaye, and Oscar Oviedo-Trespalacios, *What factors contribute to the acceptance of artificial intelligence? a systematic review*, Telematics and Informatics **77** (2023), 101925.

[8] M3C: Multidisciplinary Media and Mediated Communication Research Group, *Acted emotional speech dynamic database (aesdd)*, 2023.

[9] Peter Sullivan, Toshiko Shibano, and Muhammad Abdul-Mageed, *Improving automatic speech recognition for non-native english with transfer learning and language model decoding*, 2022.

[10] Uldis Valainis, *Audio emotions*, Kaggle, 2020.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, 2023.

[12] Nikolaos Vryzas, Rigas Kotsakis, Aikaterini Liatsou, Charalampos Dimoulas, and George Kalliris, *Speech emotion recognition for performance interaction*, Journal of the Audio Engineering Society. Audio Engineering Society **66** (2018), 457–467.

[13] Nikolaos Vryzas, Maria Matsiola, Rigas Kotsakis, Charalampos Dimoulas, and George Kalliris, *Subjective evaluation of a speech emotion recognition interaction framework*, Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion (New York, NY, USA), AM '18, Association for Computing Machinery, 2018.

[14] Ruslan Magana Vsevolodovna, *Voice to text with python in flask*, 2021.