UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Closed Captions: Generador de subtítulos automáticos *offline* empleando un motor de conversión de voz a texto (STT)

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Javier Aibar Armero

Tutor: Fernando García Granada

2020/2021

Resumen

Ver películas en el idioma que se estudia es muy beneficioso para el alumno, ya que permite asentar ciertas bases lingüísticas aplicadas en un contexto realista en el que se utiliza lenguaje adecuado a la situación expuesta en el metraje.

No obstante, para facilitar la comprensión de la trama por parte del espectador, es importante que exista un refuerzo textual del lenguaje oral utilizado en la película, es decir, subtítulos.

Pese a que hoy en día resulta fácil encontrar películas subtituladas, emplear subtítulos en tu idioma materno suele derivar en prestar mayor atención a la lectura que a la escucha, por lo que, si el nivel del alumno lo permite, comenzar a leer los subtítulos en el idioma que se aprende resulta un avance muy interesante en la práctica del idioma. El problema yace en que a menudo, los subtítulos no suelen representar exactamente al audio si no que presentan un significado similar expresado de forma diferente.

Para solucionar dicho problema, en este trabajo se pretende desarrollar una aplicación que, mediante un motor Speech-To-Text, procese y transcriba archivos de audio en *closed captions* con cierto grado de confianza. Dicho texto será transcrito a un archivo SubRip (.srt) con sus correspondientes marcas de tiempo que será reconocido directamente por cualquier reproductor que permita subtitulación, como, por ejemplo, VLC media player.

Palabra clave: Reconocimiento Automático del Habla, Closed captions, Subtitulado automático.

Abstract

Watching films in the target language is very beneficial for the learner, as it allows certain linguistic foundations to be established in a realistic context in an appropriate language used to the situation depicted in the film.

However, in order to facilitate the viewer's understanding of the plot, it is important that there is textual reinforcement of the spoken language used in the film, i. e. subtitles.

Although it is easy to find subtitled films nowadays, using subtitles in your mother tongue often means paying more attention to reading than listening, therefore, if the learner's level allows it, starting to read subtitles in the language being learnt is a very interesting step forward in language practice. The problem lies in the fact that subtitles often do not exactly represent the audio, but rather present a similar meaning expressed in a different way.

To solve this problem, this work aims to develop an application that, by means of a Speech-To-Text engine, processes and transcribes audio files into closed captions with a certain degree of confidence. This text will be transcribed to a SubRip (.srt) file with its corresponding time stamps that will be directly recognised by any player that allows subtitling, such as, for example, VLC media player.

Keywords: Automatic Speech Recognition, Closed captions, Automatic subtitling.

Resum

Veure pel·lícules en l'idioma que s'estudia és molt beneficiós per a l'alumne, ja que permet assentar certes bases lingüístiques aplicades en un context realista en el qual s'utilitza llenguatge adequat a la situació exposada en el metratge.

No obstant això, per facilitar la comprensió de la trama per part de l'espectador, és important que hi haja un reforç textual de la llengua oral utilitzat en la pel·lícula, és a dir, subtítols.

Malgrat que hui en dia resulta fàcil trobar pel·lícules subtitulades, emprar subtítols en el teu idioma matern sol derivar en prestar més atenció a la lectura que a l'escolta, de manera que, si el nivell de l'alumne ho permet, començar a llegir els subtítols en l'idioma que s'aprèn resulta un avanç molt interessant en la pràctica de l'idioma. El problema rau en que sovint, els subtítols no solen representar exactament a l'àudio sinó que presenten un significat similar expressat de forma diferent.

Per solucionar aquest problema, en aquest treball es pretén desenvolupar una aplicació que, mitjançant un motor Speech-To-Text, processe i transcriga arxius d'àudio en *closed captions* amb cert grau de confiança. Dit text serà transcrit a un arxiu SubRip (.srt) amb les seues corresponents marques de temps que serà reconegut directament per qualsevol reproductor que permeta subtitulació, com, per exemple, VLC media player.

Paraules clau: Reconeixement Automàtic de la Parla, Closed captions, Subtitulat automàtic.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

Resulta muy común que los estudiantes de un idioma acudan a un soporte más entretenido para ampliar su aprendizaje de la lengua que estudian, es una práctica muy extendida debida a diversos factores, como pueden ser el contexto en el que se desarrollan los eventos dentro de una película, lo cual tiene un impacto notable en el uso del vocabulario; el entretenimiento gracias a un argumento interesante; la motivación que puede generar consumir un medio audiovisual, entre otras. Es por esto que ver películas en el idioma que se estudia es una buena práctica [1], no obstante, aunque la idea de aprender con un soporte auditivo de la lengua de destino y un soporte textual de la lengua materna es beneficioso por sí mismo, suele derivar en una lectura completa, es decir, una desconexión del apartado oral y una atención total al apartado escrito. Esto es un problema ya que se pierde el objetivo inicial de la actividad: el del aprendizaje.

En este sentido, y mientras el nivel del alumno lo permita, resulta muy interesante que ambos soportes, tanto el auditivo como el textual, estén en el idioma de aprendizaje. Se puede encontrar gran cantidad de películas ofrecidas completamente en un idioma concreto, como es el caso de plataformas de *streaming* como *Disney Plus*, *Amazon Prime Video*, entre otras, donde se pueden encontrar películas que incluyen tanto el audio como los subtítulos, sin embargo, lamentablemente, los subtítulos que acompañan al soporte auditivo en estos filmes no representan con exactitud el contenido oral de la película, en su lugar se utilizan expresiones similares, sinónimos o incluso un resumen de lo que se puede escuchar, es decir, son subtítulos, en lugar de *closed captions* [2] que es precisamente el objetivo de este trabajo y sobre cuya diferencia profundizaremos en el apartado 1.5.

Gracias al uso de las *closed captions* el alumno será capaz de leer exactamente aquello que se dice en la película y, de esta manera, maximizar su comprensión y optimizar su aprendizaje.

1.1 Motivación

Este proyecto es muy importante para mí ya que estudio alemán y me enfrento a este problema cada vez que decido ver una película con un objetivo didáctico, i. e. para aprender alemán. Por ello, creo que es necesaria la existencia de una herramienta capaz de analizar con precisión el audio de tal forma que me permita maximizar mi comprensión del lenguaje oral.

Por otra parte, durante una estancia Erasmus que disfruté en Alemania, disponía de vídeos sobre las lecciones explicadas en alemán sin transcripción y resultaba mucho más complicada la comprensión del contenido sin dicho refuerzo textual.

1.2 Objetivos

En este trabajo se pretende construir una aplicación fácil de usar con la cual extraer mediante el empleo de una red neuronal basada en el procesamiento del lenguaje natural, la transcripción de un audio y, posteriormente, estructurarla en formato SubRip [3], el cual se puede emplear por todos aquellos reproductores multimedia que permitan la adición de subtítulos para ser mostrado por pantalla de tal forma que se podrá seguir el contenido audiovisual a la vez que dicho texto es pronunciado. Para ello necesitaremos hacer uso de una aplicación capaz de realizar un reconocimiento automático del habla (RAH) [4] a partir de un fichero de audio, en nuestro caso, en alemán, por ello, se estudiarán las diferentes alternativas presentes en el mercado.

Para desarrollar el proyecto se ha decidido utilizar el motor de videojuegos Unity [5], ya que es multiplataforma y permite la creación, de forma sencilla, de una interfaz de usuario en la que poder seleccionar un idioma de los que estén disponibles para procesar el audio. Por otra parte, se ha optado por dicha herramienta para poder profundizar en su uso, ya que, al estar enfocada al desarrollo de videojuegos, permitirá demostrar una competencia que puede ayudar de cara a un posible futuro empleo en el cual se requiera el conocimiento de esta popular herramienta.

1.3 Herramientas a utilizar

Como herramienta principal se ha elegido Unity [5] ya que se trata de un motor de videojuegos propietario con modalidad gratuita muy utilizado por las empresas del sector. Aunque está mayormente centrado en los videojuegos, gracias a su versatilidad, permite al desarrollador programar también aplicaciones serias. Está programado en C++ y C#. Es una herramienta multiplataforma por lo que, aunque solo pueda ejecutarse en Windows, Mac y Linux, sus plataformas objetivo son muchas más, e. g. WebGL, iOS, Android, diferentes consolas, varios modelos de televisores y, por supuesto, PC. Otro de los motivos de peso está relacionado con el mercado laboral, ya que, gracias a su versatilidad, a sus condiciones económicas i. e. su uso es completamente gratuito hasta superar los 100 mil USD, al lenguaje de programación, i. e. C# y demás características, es muy empleado a día de hoy, lo cual abre muchas oportunidades laborales.

Por otra parte, contamos con Python porque, además de ser el lenguaje en el que está programado DeepSpeech [6], programa de RAH [4] del que hablaremos en más detalle en el estado del arte (apartado 2), permite mucha libertad y claridad al programador a la hora de desarrollar aplicaciones gracias a sus múltiples características implementadas, e. g. tipado dinámico. Se emplea Python como un nexo de unión entre Unity y DeepSpeech.

Por último, hemos decidido incluir la librería FFMPEG [7] para permitir al usuario la posibilidad, no solo de pasar archivos de audio y convertirlos al formato necesario para su procesado, i. e. con una frecuencia de 16kHz y monoaural [8], sino también de ficheros de vídeo. Por otra parte, se ha considerado necesario que, si el usuario añade un archivo de audio, pueda decidir si se genera un archivo de vídeo para que se puedan visualizar cómodamente las *closed captions* como comentaremos en más detalle en el apartado 3.5.

1.4 Reconocimiento automático del habla

El reconocimiento automático del habla o reconocimiento automático de voz, a menudo abreviado como RAH, es el nombre que recibe la técnica por la cual un sistema informático es capaz de identificar a qué caracteres corresponden ciertos sonidos gracias a un proceso de entrenamiento empleando técnicas de *machine learning* o aprendizaje automático [4].

Las palabras están formadas por una secuencia de fonemas que serán grabados como una señal de audio. Esta señal puede ser posteriormente procesada por un sistema de reconocimiento del habla. Para ello, se emplea un enfoque probabilístico que permite inferir dicha señal a un conjunto de textos probables [9].

Este proceso se puede dividir en preprocesamiento, extracción de rasgos característicos, decodificación y posprocesamiento como podemos ver en la siguiente figura.

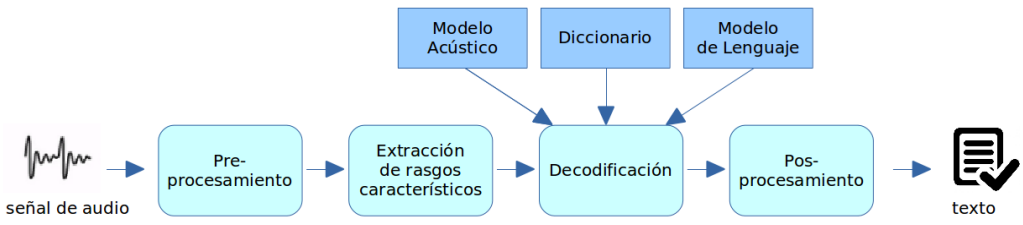

Figura 1: Esquema usual del proceso de extracción de texto a partir del audio [10].

Preprocesamiento

La señal es tratada de tal forma que se discretiza con cierta frecuencia, se minimice el ruido y se divide en partes para poder eliminar aquellas dónde no hay contenido sonoro.

Extracción de rasgos característicos

En este paso, se aplican procesos matemáticos por tramos a la señal, de tal forma que se pueda obtener un vector de coeficientes que represente dicha señal auditiva.

Decodificación

Es en este momento en el que se calcula la probabilidad de las palabras mediante un cálculo estadístico empleando para ello tres elementos que son un diccionario que contiene una lista con las palabras registradas y sus fonemas; un modelo acústico que contiene un Modelo Oculto de Markov por cada fonema o palabra y, por último, un modelo del lenguaje con las probabilidades asociadas a las palabras.

Posprocesamiento

Esta parte, aunque opcional, puede ser de gran utilidad de cara a que el resultado de la transcripción resulte de mayor calidad, pues se pueden efectuar ciertas operaciones para seleccionar no solo aquellas frases que encajen mejor con las características empleadas para decodificar, sino las que mejor encajen en el contexto.

No obstante, el reconocimiento del habla es un proceso complicado por diferentes factores como “la variación fonética interlocutor, la variación fonética entre varios locutores (multi locutor), los estilos de habla, las «disfluencias» en el habla espontánea o las características acústicas del entorno” [9] que complicarán la identificación de ciertas frases o expresiones.

1.5 Diferencia *closed captions* y subtítulos

Closed captions es el término con el que se conoce a la técnica de obtención de transcripciones a partir de audio que se ajustan todo lo posible al contenido sonoro [2]. La diferencia que existe con los subtítulos es que estos últimos permiten una reinterpretación del audio de tal forma que el texto resultante sea más breve o expresado de una forma más condensada para que el oyente tenga la oportunidad de leerlo a tiempo real.

Las ventajas que supone el uso de *closed captions* frente a los subtítulos tradicionales es que permiten al estudiante relacionar lo que oye con su transcripción exacta, mientras que, en los subtítulos, a menudo, el alumno puede llegar a perder el hilo del audio debido a que no se ajusta al texto.

No obstante, las *closed captions* también tiene la parte negativa de que habitualmente, debido a la velocidad del lenguaje oral, el espectador no sea capaz de seguir la lectura de las *closed captions* tan rápido como son pronunciadas por el interlocutor. Sin embargo, este sistema está concebido para que el estudiante pueda parar el vídeo y asimilar el contenido a su ritmo.

Además de las razones expuestas anteriormente, en este trabajo se ha decidido emplear *closed captions* porque se pueden obtener de forma automática mediante el uso de técnicas de *machine learning* con software de procesado del lenguaje natural [4] [11]. En caso de que se quisiese obtener unos subtítulos más tradicionales, las *closed captions* obtenidas, deberían pasar por una inteligencia artificial que obtuviese una simplificación del contenido que evitase la pérdida de información. A este proceso se le conoce como *automatic text summarization* o resumen automático de textos [12].

Por otra parte, como se ha comentado en la introducción, este trabajo está orientado al aprendizaje del idioma en el que está el material sonoro, y en mi opinión resulta más fácil entender el contenido y mejorar la capacidad de escucha del idioma si coincide el texto con el audio de forma precisa.

Es común, además del contenido pronunciado por los interlocutores, transcribir los sonidos no verbales como son la música o los sentimientos expresados por los actores, representados normalmente entre corchetes o paréntesis, pero en este trabajo se va a prescindir de este tipo de trascripción ya que conlleva una serie de problemas como son la necesidad de un modelo entrenado, no solo para la identificación de fonemas, sino también de sonidos como

Figura 2: Captura de una escena de Star Wars en el cual se describe el sonido que produce R2-D2 junto a su interpretación [13].

la risa, la música, explosiones, puertas, aplausos, entre otros, con la consiguiente búsqueda y etiquetado del corpus y el aumento de tamaño del modelo. Por lo que este enfoque queda fuera de este trabajo por su complejidad y consiguiente aumento de tiempo.

2. Estado del arte

Existen diferentes acercamientos que permiten la generación automática de *closed captions* [2] tanto *online* como *offline*, pero las soluciones que se han encontrado no suelen encajar con los objetivos de este TFG i. e. la generación de un archivo SubRip [3] que puedas emplear en tu propio ordenador o su uso tiene un alto coste económico o si incorporan versión gratuita, ésta está muy limitada.

Para realizar el análisis de las posibles tecnologías, se van a analizar las siguientes características:

- Si se trata de un software multiplataforma y su compatibilidad con diferentes lenguajes.
- Si existe la capacidad de aumentar las funcionalidades de la herramienta, i. e. añadir un script que obtenga el resultado procesado junto a su *timestamp* correspondiente y, de esa forma, generar las *closed captions* de acuerdo al audio proporcionado.
- Si aporta modelos ya preentrenados o no y si éstos incluyen los diferentes dialectos dentro del idioma. Concretamente si contamos con un modelo preentrenado en alemán.
- Si dispone de una comunidad en la que poder apoyarse en caso de encontrar dificultades, i. e. un foro de discusión en el que poder realizar y consultar preguntas sobre la herramienta en cuestión.
- Si es obligatoria la creación de una cuenta para la ejecución del programa.
- Si los archivos tienen un límite de tamaño a la hora de ser procesados.
- Si es preciso contar con una conexión a internet activa.
- Si existe un límite máximo diario o mensual de archivos, tiempo o tamaño de archivo procesado.
- En caso de encontrarlo, bajo qué tipo de licencia está construido, e. g. copyright, copyleft, GNU, MIT…
- Si es de código abierto.
- Si es gratuito.
- En caso de encontrarlo, ¿qué impacto tiene la herramienta en el medio ambiente?

Las herramientas que se van a tomar en cuenta son DeepSpeech de Mozilla [6], Speech-To-Text de Google Cloud [14], Microsoft Speech Service [15] y IBM Watson Speech to Text [16]:

DeepSpeech de Mozilla

Se trata de un motor de voz a texto (Speech to text STT) que permite entrenar o emplear modelos preentrenados para reconocer audio en entorno comandos.

Se trata de un programa multiplataforma y cuenta con ejemplos exportados a diferentes lenguajes como son C, .NET, Java, Javascript y Python. Es muy versátil en cuanto a añadir nuevas funcionalidades o consultar el código fuente [17].

Algunos de los ejemplos están centrados en la transcripción *online*, es decir, conectando el micrófono, va transcribiendo según la señal entra en el sistema, otros por contra, obtienen un archivo de audio y lo procesan por completo, esto se conoce como *offline* [18].

En cuanto a modelos preentrenados, Mozilla tan solo provee oficiales para inglés y chino mandarín [19], no obstante, se pueden encontrar algunos para muchos más idiomas aportados por la comunidad como por ejemplo alemán, español, francés, entre otros [20]. Sin embargo, al no ser oficiales, no siempre podemos obtener información detallada como los hiper parámetros con los que fue entrenado o el corpus empleado.

Los modelos de DeepSpeech permiten entrenar un mismo modelo con diferentes dialectos como, por ejemplo, en el alemán contamos con un porcentaje del corpus dedicado al dialecto suizo, austriaco, entre otros [21].

Uno de los puntos débiles de este software radica en su documentación, ya que está algo incompleta y es, en ocasiones, confusa. No obstante, Mozilla pone a nuestra disposición un foro en el que la comunidad pueda realizar o leer consultas de anteriores desarrolladores. Esto es muy beneficioso ya que nos permite acceder a ayuda gratuita.

El tamaño de los archivos a procesar no está limitado ni necesita internet para ejecutarlo, tampoco precisa una cuenta en Mozilla para emplearlo. Está licenciado bajo Mozilla Public License 2.0, lo cual dota al proyecto de mucha libertad.

Aunque en principio el entrenamiento y demás procesos podrían también ser multiplataforma, solo se ofrece soporte para una configuración del entorno muy concreta [22].

Speech-To-Text de Google Cloud

Se trata de una aplicación en línea de Google [14] que permite transcribir en línea u *offline* mediante el uso de su sistema de inteligencia artificial en la nube, aunque también ofrece una solución local mediante pago anticipado.

Al igual que con DeepSpeech, se trata de un software multiplataforma y soporta muchos lenguajes, como son: Go, java, Node.js, Python, C#, PHP, Ruby o directamente a través de gcloud o cualquier software que soporte peticiones POST con curl.

Las peticiones que comentamos en los párrafos anteriores deberán contener un archivo JSON con las especificaciones de nuestra petición. i. e. un enlace al audio, el idioma de transcripción con su localización regional entre otros ajustes opcionales.

También nos permite realizar una conexión empleando para ello uno de los lenguajes de programación comentados anteriormente y enviar un archivo de audio para ser procesado por la nube de Google y obtener un objeto con la transcripción, su confianza, entre otros valores.

Permite la transcripción de audio de muchas fuentes como, por ejemplo, de un teléfono (como puede ser un buzón de voz). Está limitado a archivos de hasta 10 MB. Es obligatorio tener una cuenta aportando tu tarjeta bancaria.

Guarda los resultados en la nube, donde permanecerán durante 5 días. Dispone de modelos preentrenados en 125 idiomas con sus respectivos dialectos, entrenados por la propia Google, que al ser oficiales son bastante fiables.

La versión gratuita permite transcribir archivos de audio de hasta una hora, lo cual, aunque es bastante generoso, para el objetivo del proyecto puede quedarse muy limitado, ya que a menudo se transcribirán obras audiovisuales de entre una hora y media hasta tres horas.

A partir de la hora, Google tarifa a 0,006 USD cada 15 segundos para los modelos mejorados de vídeo y 0.004 USD para los estándares de audio. La versión de pago, sin embargo, también está limitada, no se pueden superar ocho horas de audio por archivo ni las 8 horas de procesamiento al día.

Cabe destacar que todos los proyectos de Google Cloud están comprometidos con el medio ambiente, ya que, según dicen “A día de hoy, en Google registramos emisiones neutras de carbono, pero nuestro objetivo es abastecernos únicamente con energía libre de carbono de forma ininterrumpida en todos nuestros centros de datos para el 2030.” [23].

Microsoft Speech Service

Se trata del software de reconocimiento de habla de Microsoft incluido en el pack de servicios cognitivos de Azure (Azure Cognitive Services) [15] todos los servicios incluidos están licenciados bajo copyright. El paquete está compuesto por una SDK de voz, una API REST y una CLI de voz.

Cuenta con modelos en más de 85 idiomas con sus respectivas variantes entrenados por la propia Microsoft por lo que se pueden considerar como fiables, entre los que se encuentra el alemán, aunque sin la variante suiza.

Se trata de un programa multiplataforma, ya que admite clientes escritos en C#, C++, Go, Java, Node.js, Python, JavaScript (en el navegador), Objective-C/Swift, entorno comandos de Windows, Linux y macOS y cualquier lenguaje capaz de realizar peticiones REST [24].

La versión gratuita no permite peticiones concurrentes, está limitada a 5 horas de audio al mes. A partir de dicha cuota mensual, se fija un precio de 0.844€ cada hora procesada. Requiere una conexión a internet y una configuración mediante una clave y un identificador regional para poder hacer uso de este servicio de transcripciones.

Al igual que el resto de servicios revisados, Microsoft ofrece una plataforma de ayuda para obtener y ofrecer apoyo a los desarrolladores que se encuentren con algún problema, lo cual nos permitirá tener un lugar al que acudir en caso de tener dificultades durante el desarrollo.

En la versión CLI, se debe ejecutar la instrucción “spx” seguida de “recognize” y los argumentos necesarios, se puede elegir el argumento “--microphone” para realizar un reconocimiento inmediato y *online* mediante el uso de nuestro micrófono.

Adicionalmente, también permite el uso del argumento “--file” para ejecutar la instrucción en modo *offline* e, introducir la ruta a nuestro archivo de audio, por defecto trata de transcribir del inglés, pero si se sustituye “translate” por “recognize”, se puede seleccionar otro idioma añadiendo para ello, el uso de dos argumentos: el de “--source” para indicar el idioma de

origen y “--target” para indicar el de destino; ambos seguidos del código del idioma incluyendo la variable regional.

Para analizar el uso de la versión SDK, vamos a tomar como base el ejemplo que aportan escrito en Python. Deberemos crear un objeto SpeechConfig del paquete speechsdk en el que deberemos indicar nuestra clave y nuestro código regional, tal y como hacíamos con la versión CLI.

Para el reconocimiento *online* mediante el uso del micrófono, deberemos crear un objeto SpeechRecognizer con el objeto de configuración creado anteriormente y, sobre él, ejecutar los métodos recognize_once_async y get. Esto obtendrá el micrófono por defecto y devolverá un texto correspondiente a la predicción obtenida.

Además del método en línea, podemos también analizar un archivo de audio. Para ello deberemos crear un objeto AudioConfig que contenga la ruta a nuestro fichero y, al igual que hicimos con la versión *offline*, deberemos crear una instancia de SpeechRecognizer con el objeto de configuración que comentamos al principio, pero ahora, también deberemos añadir un argumento adicional correspondiente al objeto con la ruta al audio. La obtención de la predicción se realiza igual que antes.

La nube de Azure, según cuenta en su página web, dentro de la cual estaría incluido su programa de reconocimiento del habla “puede ofrecer hasta un 93 % más de eficiencia energética y hasta un 98 % más de eficiencia en materia de carbono en comparación con las soluciones locales.” [25].

IBM Watson Speech to Text

Se trata de un servicio de IBM que emplea capacidades de reconocimiento del habla para convertir un audio hablado en diferentes idiomas a texto [16].

Permite el desarrollo de clientes en múltiples lenguajes de programación, con los cuales añadir las funcionalidades deseadas, estos son: Curl, Java, Node, Python, Go, .NET, Ruby, Swift e incluso Unity.

El tamaño de los archivos está limitado a 1GB. Aunque las películas en alta calidad pueden llegar a las decenas de *Gigabytes* esto no debería ser un problema, ya que, en cualquier caso, se debería preprocesar el archivo para extraer previamente el audio en el formato correcto, i. e. una frecuencia de 16kHz y sonido monoaural [8], el cual, en principio, no debería ocupar más de 1GB [26].

Al igual que DeepSpeech, permite tanto usar los modelos preentrenados oficiales como entrenar modelos a medida. Se puede seleccionar alguno de los modelos de los que disponen, indicándolo simplemente mediante una variable con el código del idioma en la petición. Además, dispone de una gran variedad de dialectos dentro de cada idioma.

Al igual que con el resto de plataformas y programas analizados, cuenta con una comunidad donde poder realizar consultas o leerlas. Esto resulta muy interesante en caso de necesitar ayuda con algún problema relacionado con el *software* de IBM.

La versión gratuita permite hasta 500 minutos (ocho horas y 20 minutos) de procesado al mes, si se precisan más minutos, el modo *Plus* cuesta 0.02 USD por minuto hasta el millón de minutos, a partir de esta cifra, el minuto se cobra a 0.01 USD.

Se necesita una cuenta, aunque, a diferencia de la herramienta de Google, no es obligatorio introducir la tarjeta de crédito, sin embargo, desgraciadamente es necesaria una conexión a internet.

De la misma forma que hemos realizado con el análisis del servicio de voz a texto de Microsoft, vamos a basarnos en el ejemplo escrito en Python para obtener una vista previa del uso de este programa.

Lo primero que deberemos hacer una vez instalado el paquete será crear un objeto IAMAuthenticator con nuestra clave API de IBM, y de forma similar a como hacíamos con el de Microsoft, deberemos crear un objeto SpeechToTextV1 pasando como argumento el objeto identificador que acabamos de crear. Y ejecutar sobre el mismo, el método “set_service_url” para configurar nuestro servidor más cercano de entre los disponibles. Alternativamente también podremos identificarnos con nuestro usuario y contraseña empleando un objeto CloudPakForDataAuthenticator.

Deberemos leer por nosotros mismos el archivo de audio y almacenarlo en un AudioSource, que será pasado como argumento a un *websocket* indicando además el mime

correspondiente a dicho archivo. Posteriormente deberemos crear una *callback* de reconocimiento incluyendo el modelo en el idioma que necesitemos mediante su código de idioma y variante dialectal. Una vez ejecutada la *callback*, obtendremos un objeto JSON en el que podremos encontrar las posibles transcripciones, la confianza de las mismas y, en caso de haberlo indicado, también las marcas de tiempo de cada palabra.

Según un informe de responsabilidad corporativa llevado a cabo por la propia empresa, en 2020, "IBM se ha comprometido a lograr cero emisiones netas de gases de efecto invernadero (GEI) para 2030. Anunciamos este objetivo en febrero de 2021 y lo cumpliremos mejorando aún más la eficiencia energética de nuestras operaciones, obteniendo electricidad [...] a partir de fuentes renovables y luego utilizando tecnologías para eliminar el carbono en una cantidad que iguale o supere nuestras emisiones residuales." [27].

2.1 Selección de la herramienta

Tras reflexionar y sopesar los pros y contras, se ha optado por emplear DeepSpeech. Uno de sus puntos más fuertes que se han valorado es que es completamente gratuito y licenciado bajo Mozilla Public License 2.0, que nos permitirá desarrollar con tranquilidad. En segundo lugar, se ha valorado positivamente la posibilidad de ejecutar este programa bajo diferentes sistemas operativos, junto con Unity, que también es multiplataforma, nos permitirá exportar el programa final a muchos entornos diferentes.

Por otra parte, gracias a que el programa se puede descargar y ejecutar con total libertad, no es necesario contar con una conexión a internet, ni es necesario abrir ninguna cuenta, lo cual nos permite brindar al usuario la posibilidad de ejecutar nuestra aplicación sin más. Gracias a la misma razón, no tenemos ningún límite ni de archivos individuales, i. e. tamaño de archivos ilimitado, ni un límite de uso diario o mensual, i. e. podemos procesar tantos archivos como queramos, incluso de forma concurrente.

Por si lo anterior no fuera suficiente, nos permite descargar varios modelos preentrenados, la mayoría proporcionados por la comunidad[1] para poder extender de forma automática las posibilidades de este proyecto, i. e. permitiendo al usuario añadir los modelos a una carpeta para que el programa los reconozca. Y en caso de querer transcribir el audio de un idioma que no esté proporcionado ni de forma oficial, ni por la comunidad, un usuario experimentado, podría entrenar su propio modelo con DeepSpeech e incluirlo en nuestro proyecto.

[1] Mozilla solo proporciona un modelo en preentrenado en inglés y otro experimental en chino mandarín [19].

La razón principal por la que se han descartado varias de las tecnologías que se estaban valorando es el límite que tienen las versiones gratuitas. La única herramienta completamente gratuita de la lista es DeepSpeech. Google, Microsoft e IBM ofrecen versiones sin necesidad de pago, pero están relativamente limitadas y las tres requerían el uso de una conexión a internet y una cuenta, la cual habría que forzar al usuario a crearla, complicando mucho su uso e implementación.

3. Transcripción de audio a *closed captions*

En este apartado vamos hablar de la fase de diseño, ¿cómo vamos a pasar de un audio a obtener su transcripción en el formato adecuado?, es decir, ¿cómo se va a desarrollar la aplicación tanto gráfica como arquitectónicamente o a nivel de programación?, las herramientas que se van a emplear y su uso, ¿cómo vamos a desarrollar el modelo y su funcionamiento interno? y ¿cómo se ha organizado el proyecto?

3.1 Análisis de la solución

La decisión de emplear Unity [5] como herramienta principal a la hora de desarrollar la interfaz de usuario estaba bastante clara desde el primer momento ya que corresponde con un objetivo inicial del proyecto debido, por una parte, al valor que aporta tener conocimientos en dicha herramienta y, por otra parte, la gran versatilidad que brinda gracias a sus posibilidades multiplataforma.

Se ha optado por utilizar DeepSpeech [6] y Common Voice del equipo de Mozilla por ser gratuito, de código abierto entre otras características mencionadas en el apartado 2 y ofrece el corpus (Common Voice) público más grande que existe [28], en una gran variedad de idiomas (64 a febrero del 2021, aunque ya están trabajando para doblar el número de idiomas disponibles, 128), incluido el alemán, proporcionado por la propia comunidad, disponible también de forma gratuita en la página web oficial [21].

Como se comentaba en el párrafo anterior, el corpus es generado por la comunidad, esto quiere decir que cualquier persona puede grabarse leyendo en voz alta un texto concreto en su idioma nativo, otros usuarios, a su vez, validarán dicha grabación, i. e. si la grabación en cuestión es comprensible por un nativo y, efectivamente, dice lo que está escrito en el texto, la grabación debería ser entonces validada por el usuario, esto se conoce como aprendizaje activo, donde los usuarios que confirman la validez de los audios toman el rol de expertos.

Aunque se puede instalar y probar el programa original basado en entorno comandos, por defecto, tan solo transcribe el audio recibido a la consola y únicamente para el inglés. En caso de querer ampliar sus funcionalidades o emplear un idioma diferente, se deberá encontrar un modelo entrenado en el idioma seleccionado o entrenar uno propio y programar aquellas características extra que queramos o necesitemos.

DeepSpeech emplea la librería TensorFlow de Google que es gratuita y de código abierto para el procesado de redes neuronales recurrentes (RNN por sus siglas en inglés), la cual permite trabajar con los modelos, i. e. emplearlos o entrenarlos, los cuales proceden "a ingerir los espectrogramas de voz y generar transcripciones de texto" [29].

Cada muestra se compone de dos elementos, siendo el primero, un vector de entrada con una representación de las características del sonido, mediante el uso de los coeficientes cepstrales en las frecuencias de Mel (MFCC en inglés) y en segunda posición podemos encontrar la etiqueta con el contenido del sonido transcrito.

$$S = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots\}.$$

Figura 3: Representación de los datos de entrada. Siendo "x" el vector de características del audio e "y" su correspondiente transcripción [29].

La red neuronal procesa entonces los datos de entrada generando una secuencia de probabilidades, i. e. intenta deducir a partir de un sonido transformado en MFCC, a qué carácter o caracteres podría corresponder.

Esta red neuronal se compone de 5 capas ocultas, de las cuales, las 3 primeras y la quinta son no recurrentes, la cuarta es la única capa recurrente. Cada capa no recurrente trabaja con una parte diferente de los datos de entrada junto con un contexto de tantos trozos como queramos, por defecto, está configurado como 9.

"La cuarta capa contiene un conjunto de unidades ocultas con recurrencia hacia adelante" [29]. La quinta toma el resultado "hacia adelante" de la capa anterior como datos de entrada. Finalmente, obtenemos el resultado de la quinta capa, que devolverá las probabilidades de que dicha fracción de sonido corresponda con alguno de los caracteres posibles del alfabeto con el que estemos trabajando.

Del resultado obtendremos también un espacio vacío que se empleará en este momento para calcular el error conocido como pérdida Clasificación Temporal Conexionista (CTC) [30], ya que se utilizará como marcador de que hemos cambiado de carácter.

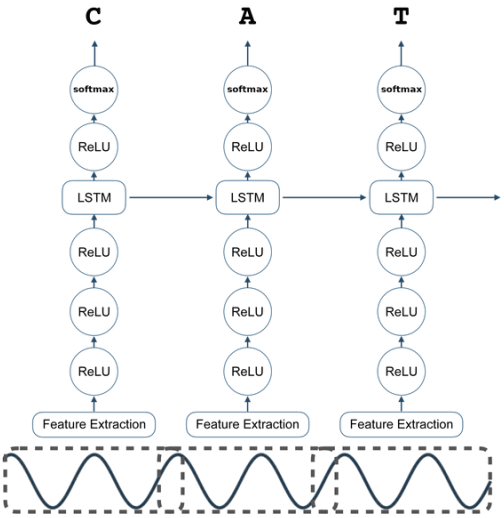

Figura 4: Representación del modelo de la RNN de DeepSpeech [29].

3.2 Diseño de la solución

En el siguiente apartado hablaremos sobre el diseño de la solución, es decir, la obtención de recursos y el diseño gráfico inicial de la aplicación.

3.2.1 Obtención de recursos

El primer paso para poder comenzar a desarrollar es obtener los diferentes recursos que nos permitirán comenzar a juntar las piezas. Se trata del programa base, DeepSpeech, que se puede encontrar en la página oficial de GitHub [6]; Unity [5], el cual podemos descargar desde su programa principal centralizado Unity Hub; Python 3.6, desde la web oficial para el sistema operativo correspondiente, en nuestro caso, Windows 10.

Por otra parte, deberemos descargar el Corpus oficial de Common Voice [21], perteneciente a Mozilla, para poder entrenar nuestro propio modelo en alemán, está disponible en su página web oficial y habrá que preprocesar dicho corpus, i. e. generar una serie de archivos .csv correspondientes a entrenamiento, test y desarrollo que recogerán la ruta a los archivos de audio, su transcripción, idioma, dialecto y demás información interesante para el entrenamiento.

3.2.2 Diseño preliminar de la interfaz gráfica de usuario

Se preparó un diseño inicial de la interfaz gráfica para configurar las diferentes opciones que debería recoger la aplicación: un selector de idioma, la ruta del archivo de audio o vídeo y un área de texto para la salida de mensajes.

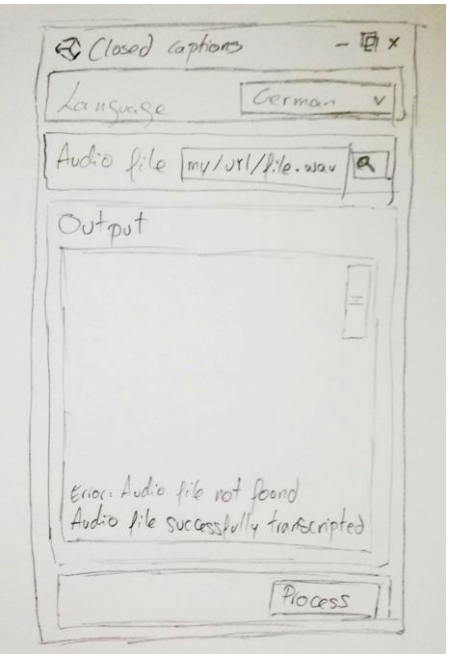

Figura 5: Diseño preliminar de la interfaz, dibujado a mano en papel.

La interfaz consta, por este orden, de un selector de idioma que consiste en un desplegable que lista los modelos existentes en la carpeta designada para ello[2], i. e. un directorio designado para albergar los modelos de los idiomas que el usuario quiera disponer.

Lo siguiente que podemos observar es uno de los campos más importantes de la aplicación, que es el selector de la ruta de audio o vídeo. En este apartado contamos con dos posibilidades: escribir manualmente la ruta o clicar en el botón para explorar el sistema de archivos en busca del fichero de multimedia que queramos transcribir.

Por otra parte, podemos apreciar un cuadro de texto, de solo lectura, con *scroll*, que nos permitirá realizar un seguimiento de la ejecución de la aplicación ya que nos devolverá los mensajes de error, advertencia y éxito que puedan generar Python, el entorno comando o el propio Unity. De esta forma podremos obtener información sobre posibles fallos del usuario o problemas a solucionar.

Por último, tenemos el botón de procesar, que nos permitirá ejecutar el comando con el idioma y audio seleccionados por el usuario. Y, en caso de éxito, se crearía un archivo SubRip [3] con el contenido del audio transcrito.

Este diseño se adaptó posteriormente a nuevos requisitos que se identificaron más adelante como, por ejemplo, nombre y ubicación de salida por si el usuario decidiese que el resultado

[2] Se encuentra dentro de la carpeta “StreamingAssets”. Se habla de esta carpeta en más detalle en el apartado 3.5.

estuviera en una ubicación diferente o ajustes avanzados del modelo i. e. ajustes de hiper parámetros.

Más tarde se vio necesaria la creación de un menú extra ocultable para añadir opciones avanzadas. Por lo pronto, se van a añadir aquellas que corresponden a los hiper parámetros alfa y beta, necesarios para la correcta ejecución del software ya que se emplean para calibrar el modelo como comentaremos en el apartado 5.4.

3.3 Arquitectura

Como se comentó en los apartados 1.2, 1.3, 2 y 3.1, tenemos gran interés en que la aplicación sea multiplataforma para poder ser ejecutada por la mayor cantidad de ordenadores posible y, por tanto, más usuarios puedan ampliar su repertorio de estudio de forma gratuita sin la limitación del sistema operativo. No obstante, la opción de ejecutar la aplicación en diferentes sistemas operativos se plantea para el futuro debido a la limitación de tiempo.

En cuanto a la arquitectura en la que se va a desarrollar este trabajo, se divide entre Windows 10 para el desarrollo de la interfaz de usuario (ejecución de Unity) y redacción de esta misma memoria y Ubuntu 18.04 para el entrenamiento del modelo. Una vez acabada la aplicación se procederá a desplegarla y probarla en Windows 10.

3.4 Organización del proyecto

Para la organización de las tareas derivadas de la escritura de la memoria y el desarrollo de la aplicación, se ha optado por emplear la herramienta de gestión de tareas Trello [31], la cual permite crear tarjetas muy completas, i. e. no solo permite la asignación de un título sino también una descripción detallada, añadir imágenes e incluso adjuntar archivos o enlaces.

Dichas tarjetas pueden ser clasificadas dentro de las categorías (columnas) que decidamos, en el caso de este proyecto se ha optado por las siguientes: “Tareas pendientes” para las tareas que están en cola, “En proceso” para aquellas que están en desarrollo en este momento, “En pausa” son aquellas tareas que por su naturaleza deben esperar a un estado del proyecto más avanzado, “Pendiente de terceros” para aquellas actividades que no puedan continuar hasta obtener una respuesta de alguna persona, “Acabado” para aquellas tarjetas cuya misión haya sido cumplida y “Fijado” para almacenar enlaces o información importante durante el completo desarrollo del trabajo.

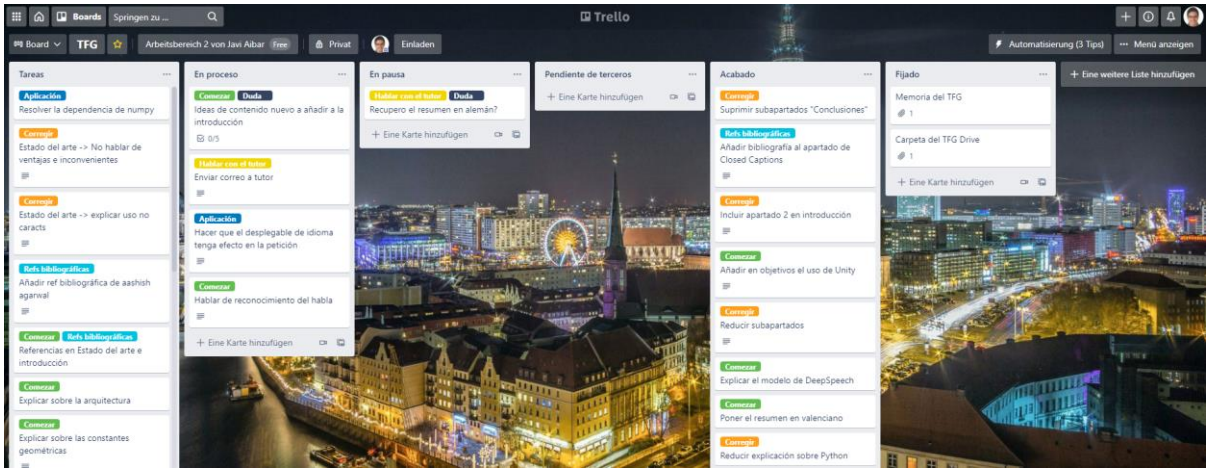

Figura 6: Captura de pantalla del tablero de Trello del proyecto.

Para obtener un diagrama de Gantt de forma sencilla, semiautomática, se decidió añadir al tablero trello del proyecto, una extensión llamada TeamGantt [32] que permite asignar inicio y fin a nuestras tareas, además podemos indicar el porcentaje de realización que llevamos hasta el momento en cada una de ellas o marcarlas como hitos, para poder realizar una buena planificación del tiempo y posterior seguimiento.

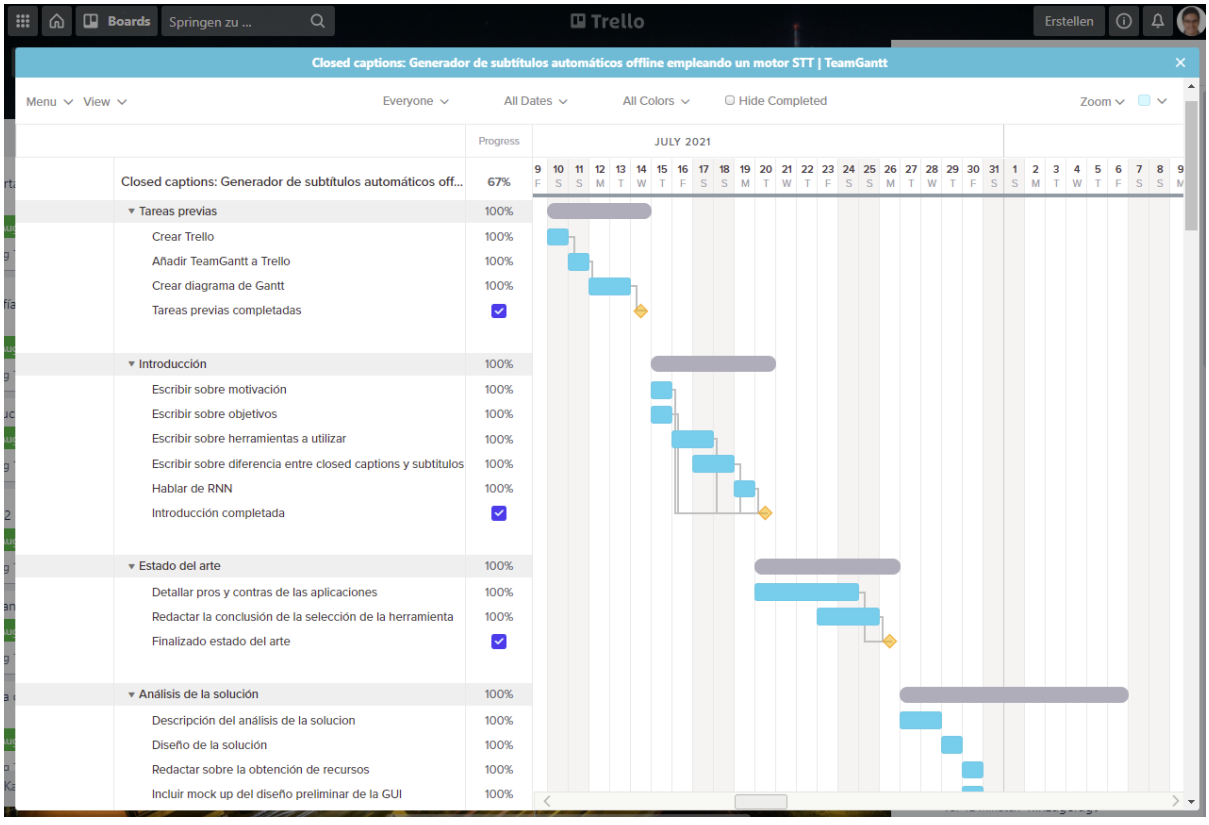

Figura 7: La parte del inicio del diagrama de Gantt del TFG.

Para el control de versiones se emplea Git con la herramienta de apoyo GitHub Desktop, la cual proporciona un entorno gráfico amigable con el usuario, fácil de usar y aporta una

cómoda conexión con el repositorio remoto alojado en GitHub, lo que lo convierte en una buena opción para trabajar desde cualquier dispositivo que cuente con acceso a Internet.

Por último, se ha elegido la herramienta Google Docs para la redacción de la memoria ya que, al igual que con Git, nos permite trabajar de forma remota, accediendo desde diferentes dispositivos y permite su control en tiempo real, así como anotaciones, correcciones, control de versiones, entre otras ventajas.

3.5 Diseño detallado

La aplicación va a contar con una interfaz de usuario desarrollada en Unity que permitirá configurar todas las opciones esenciales para el correcto funcionamiento del programa base DeepSpeech [6] y deberá mostrar por pantalla información de la ejecución, e. g. mensajes de error o de completación de la tarea.

Además, deberá contener todos los elementos externos (requisitos) necesarios para prescindir de cualquier conexión a Internet[3] como son el ejecutable de Python, DeepSpeech y la librería FFMPEG [7].

Cabe destacar que para que Unity pueda emplear todas las herramientas i. e. el ejecutable Python, la librería FFMPEG y el propio DeepSpeech, se debe crear un subproceso llamando al entorno comandos del sistema operativo en el que se ejecute la aplicación. Esto se consigue gracias a una librería que emplea Unity procedente de C# llamada Process. Dicha librería obtendrá los parámetros necesarios y creará, de forma transparente al usuario i. e. en segundo plano, un proceso en el cual ejecutar dichos parámetros.

Más tarde, los resultados de la ejecución, tanto satisfactorios como fallidos, serán redirigidos por el subproceso de nuevo a nuestro programa, permitiendo así mostrarlos por pantalla dentro de nuestra aplicación.

[3] Siempre que tan solo sea necesaria la transcripción al alemán, si fuese necesario otro idioma, el usuario debería descargar el modelo correspondiente y ubicarlo en la carpeta de modelos para ser reconocido automáticamente por la aplicación.

En cuanto a la estructura de carpetas tenemos la siguiente:

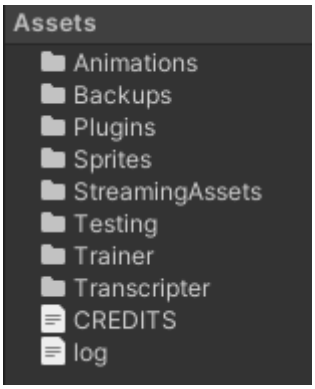

Figura 8: Estructura de archivos y carpetas.

Está ordenado automáticamente por orden alfabético y mostrando primero directorios y después archivos. Podemos observar y comentar el contenido de las diferentes carpetas y archivos por orden de aparición:

En primer lugar, tenemos “Animations” donde van a estar todas las animaciones necesarias para un correcto funcionamiento de la aplicación, en nuestro caso y por ahora, la única animación existente corresponde a una espiral que rota en señal de carga.

Figura 9: Símbolo de carga.

En la carpeta “Backups” se encuentran archivos de respaldo correspondientes a versiones de la interfaz funcionales. Esto es así por dos razones, por una parte, desafortunadamente, las escenas de Unity no son del todo compatibles con Git y por otra, resulta útil para tener una forma sencilla y rápida de crear grandes cambios en la interfaz sin que suponga un riesgo para la versión funcional actual. Las versiones del resto de elementos y de la misma interfaz se mantienen mediante Git.

El siguiente directorio de la lista es “Plugins” que contiene un par de extensiones de Unity de terceros. Estos son “SimpleFileBrowser” [33] que permite explorar de forma sencilla el almacenamiento del sistema operativo actual. Por otra parte, tenemos “TextMeshPro” [34] que consiste en un conjunto de herramientas para crear formularios.

Después podemos encontrar la carpeta “Sprites” que contiene todos los iconos e imágenes que serán utilizados dentro de la aplicación. En principio tendremos el símbolo de carga visto en la figura 9 y dos iconos para la gestión del cuadro de texto, i. e. el de borrar y el de guardar el contenido en un archivo.

Después tenemos el directorio “StreamingAssets” que se trata de una carpeta especial de Unity, el contenido de la cual será transferida sin cambios a la versión compilada. Es una carpeta muy útil ya que podemos agregar las librerías y modelos para ser ejecutadas posteriormente, una vez compilada la aplicación.

En la carpeta “Testing” se almacenarán todas las pruebas unitarias necesarias para comprobar el correcto funcionamiento de la aplicación según se vayan realizando cambios en los métodos.

Por último, podemos encontrar los directorios “Trainer” y “Transcripter” que serán las dos funcionalidades que será capaz de realizar la aplicación.

Por una parte, “Trainer” nos permitirá seleccionar la ubicación del corpus en el idioma que queramos y entrenar un nuevo modelo. Por otra parte, “Transcripter” permite transcribir el contenido de un audio o vídeo, indicando la ruta del mismo y eligiendo el idioma de entre los modelos disponibles.

Se ha detectado que la adición de un archivo de subtítulos a un fichero de audio no siempre es posible[4], lo cual nos obliga a crear una solución para mantener la simplicidad de uso de la aplicación de cara al usuario. Para ello, si el usuario selecciona un archivo de audio, la aplicación lo detecta automáticamente y muestra una opción para “crear vídeo falso”.

La opción para crear el vídeo falso creará un proceso desde Python que llamará a la biblioteca FFMPEG y, aprovechando su potencia y versatilidad, crearemos a partir del audio, un vídeo con el mismo nombre y duración, de extensión .mkv y con la posibilidad de añadir subtítulos.

Para poder generar el vídeo, se ha creado una imagen de relleno con el logo de la UPV y un texto con créditos de DeepSpeech y Aashish Agarwal.

[4] Aparentemente, los reproductores no están preparados para mostrar subtítulos en archivos de audio, solo pueden en los de vídeo.

SUBTITULOS GENERADOS AUTOMÁTICAMENTE
POR JAVIER AIBAR EMPLEANDO DEEPSPEECH
DE MOZILLA Y EL MODELO ENTRENADO POR
AASHISH AGARWAL

Figura 10: Fondo utilizado para generar el vídeo falso.

4. Desarrollo de la solución

A continuación, comentaremos cómo ha ido evolucionando el desarrollo de la solución. Comenzaremos por hablar sobre los primeros pasos, continuaremos con el apartado del entrenamiento del modelo en alemán y los problemas encontrados durante dicho proceso, y hablaremos del desarrollo de las capas de presentación y negocio en las que se ha dividido la aplicación.

4.1 Primeros pasos

El primer problema con el que nos topamos es que Unity [5] emplea C# como lenguaje de programación para crear videojuegos, sin embargo, el equipo de DeepSpeech [6] decidió crear su herramienta utilizando Python, por tanto, deberemos ajustar uno de los dos para que puedan trabajar juntos.

El equipo de DeepSpeech pone a nuestra disposición una configuración alternativa mediante la generación por parte del usuario de ciertas librerías orientadas al uso con Java, JavaScript, .NET, entre otras, ya que C# forma parte del universo de .NET, podría ser una buena idea avanzar en esa dirección, i. e. generar una librería de DeepSpeech que pudiera ser ejecutada directamente desde Unity. Lamentablemente, solo ofrecen soporte para la versión de Python y, tras intentar generar dicha librería, en el último paso, daba un error recurrente que no se logró solucionar.

En ese momento, se decidió intentar la solución opuesta e intentar ajustar Unity para que fuera capaz de ejecutar Python. De esa forma, ser capaces de emplear DeepSpeech de forma nativa, sin necesidad de generar las librerías. Esto es posible gracias al poder que proporciona C#, ya que nos permite ejecutar comandos externos de forma sencilla, de manera que referenciando al ejecutable, el archivo binario de Python correspondiente a la plataforma en la que nos encontremos y a los archivos necesarios, i. e. el fichero con la lógica de negocio escrito en Python, seremos capaces de ejecutarlo directamente desde Unity.

4.2 Entrenamiento del modelo

En este apartado podremos analizar el proceso de entrenamiento con detalle y posteriormente comentaremos cuáles fueron los problemas encontrados.

4.2.1 Proceso de entrenamiento

Debido a que no existe un modelo oficial preentrenado en alemán por Mozilla, se ha determinado que para comenzar el desarrollo era interesante intentar entrenarlo por nosotros mismos con el corpus en alemán que proporcionan en su apartado Common Voice [21].

Además de lo comentado en el párrafo anterior, también se pensó en primera instancia en incorporar la posibilidad de que la aplicación pudiese entrenar un modelo automáticamente dentro del mismo proyecto de Unity debido a que el corpus de voces en todos los idiomas aumenta día a día, lo que cual se acaba de forjar en una nueva versión del mismo cada 6 meses, por lo que se estimó importante la posibilidad de que la propia aplicación pudiese actualizarse fácilmente.

Es decir, el usuario podría descargar el paquete de idioma actualizado desde la página oficial de Common Voice, ubicarlo dentro de la carpeta concreta, especificar el nombre del idioma y demás parámetros y procesar, de tal forma que el proyecto se adapte sin cambios adicionales a las mejoras en el corpus, y por consiguiente pudiese reconocer en el futuro con mejor precisión de la que dispone actualmente.

No obstante, una vez se analizaron los requisitos con mayor detenimiento, se pudo determinar que esta posibilidad era inequívocamente imposible, ya que como se comentará en el apartado de problemas encontrados durante el entrenamiento (4.2.2), requiere de un entorno muy específico y una capacidad de procesamiento elevada.

Sin embargo, para el desarrollo de este proyecto, se procedió a configurar el entorno necesario siguiendo cuidadosamente las instrucciones disponibles en la página web oficial de DeepSpeech para tal efecto.

4.2.2 Problemas encontrados durante el entrenamiento

Durante la fase de entrenamiento, fueron surgiendo diversos errores, los cuales están detallados en el apéndice A. Los problemas encontrados durante esta fase están relacionados con ciertas dependencias que no podían ser satisfechas, bien por el entorno, por ambigüedades del manual o por falta de permisos en una máquina remota de la UPV a

la que nos dieron acceso para este objetivo. Otro de los problemas recurrentes fue el reconocimiento de la tarjeta gráfica o su compatibilidad con el software CUDA.

Desgraciadamente, debido al límite de tiempo y sin una certeza de éxito, se decidió renunciar. No obstante, afortunadamente, logramos encontrar un modelo preentrenado en alemán por internet [35] [36] con el que poder trabajar.

4.3 Desarrollo de la capa de presentación

Una vez obtenido un modelo válido en alemán, es momento de desarrollar la interfaz. Para ello es necesario crear el proyecto en Unity [5]. Se ha decidido generarlo con la configuración en 2D ya que el objetivo es desarrollar un formulario.

Se procede, por una parte, a crear un objeto *Canvas* o lienzo que nos permitirá mostrar el formulario al usuario, y por otro, un objeto de juego[5] que contendrá el script con la lógica de negocio y eliminar el objeto con la iluminación ya que el *Canvas* no se ve afectado por las luces de la escena.

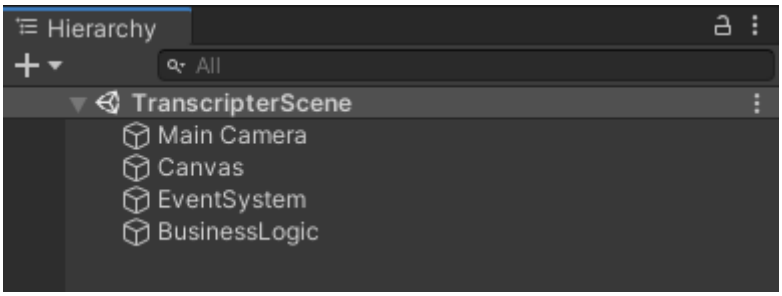

Figura 11: Estructura inicial de la interfaz de usuario.

Ahora podemos construir sobre el *Canvas* todos los elementos que queramos incluir. Como son el selector de idioma, la ruta del archivo de audio, campos opcionales como la ruta de salida o el nombre de salida del archivo, el campo para generar un vídeo falso a partir de un audio, el campo de texto para mostrar el resultado de la ejecución al usuario y, por último, la sección de botones.

Para organizar las secciones mencionadas en el párrafo anterior y con idea de que la aplicación sea responsiva, se ha optado por emplear el componente *Vertical Layout Group* o grupo de diseño vertical [37] en el contenedor de dichas secciones. Este componente permite

[5] Unity denomina a todo elemento de un proyecto GameObject, es decir objeto de juego.

ajustar diferentes opciones y mantiene unos elementos encima de otros de forma automática en lugar de que se superpongan.

Para que el *Vertical Layout Group* realice correctamente su función, es necesario que cada uno de los elementos contenga un *Layout Element* o elemento de diseño [38], el cual nos permite configurar ciertas características tales como alto mínimo, preferido o flexible y la prioridad de este elemento con respecto al resto. Un buen ajuste de todas estas opciones nos permite una correcta visualización incluso en espacios muy ajustados.

Sin embargo, es necesario también controlar los elementos que formarán parte de cada uno de los apartados que acabamos de crear, por ejemplo, necesitamos que cada sección tenga, al menos, una etiqueta, algunos de ellos requerirán un campo de texto, otros además requerirán un botón, o incluso en el caso del área de texto, necesitaremos un *scroll*.

Es por esta razón que necesitaremos añadir en muchas de las secciones descritas, un componente muy similar al ya empleado, se trata de *Horizontal Layout Group* o grupo de diseño horizontal [39], el cual tiene el mismo efecto que el ya visto, permite ajustar diferentes opciones y mantiene unos elementos junto a otros de forma automática en lugar de que se superpongan. Al igual que con su homólogo vertical, nos permitirá de una forma relativamente sencilla configurar la responsividad de la aplicación.

En este punto, el problema que presenta la interfaz es la capacidad de ajustar el tamaño del área de texto, ya que lo interesante sería dar un tamaño mínimo y preferido a las secciones que no necesiten gran cantidad de espacio y reservar el resto del espacio a dicha área de texto para que el usuario pueda leer el resultado de la ejecución cómodamente.

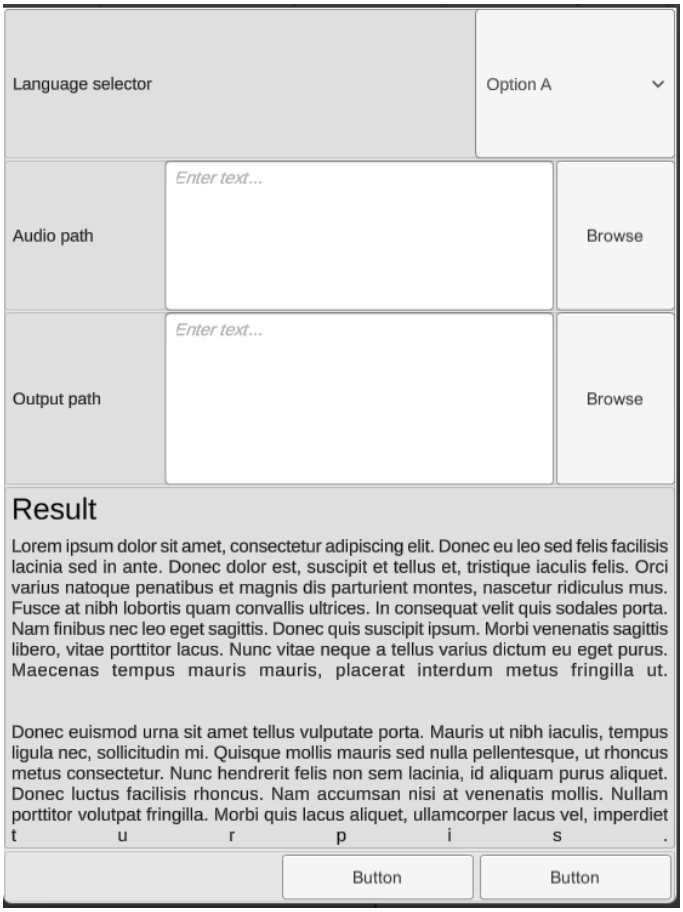

Figura 12: Tamaño de secciones y ventana de resultados inapropiado.

Gracias al aumento en la prioridad del componente, al ajuste del tamaño preferido y flexible y al ajuste de altura fija del resto de opciones, por fin se logró que la sección de resultados obtuviese el espacio sobrante.

Ahora es necesario incluir las opciones avanzadas, para lo cual, se ha decidido añadir un menú ocultable y, de esa manera, para mantener una interfaz lo más limpia posible. De forma que es necesario incluir un botón que nos permita ocultar o mostrar dicho menú.

Para ello se ha creado una nueva sección que actuará como contenedor, a la misma altura en la jerarquía que el resto de apartados vistos en los párrafos anteriores, es decir, compartiendo contenedor con el selector de idioma, el selector de ruta de audio o el cuadro de resultado. Esto es así ya que queremos que se adapten los espacios correctamente y compartiendo dicho contenedor, el sobrante se podrá distribuir entre el cuadro de resultado y las opciones avanzadas si necesitáramos añadir alguna nueva.

Dentro de la nueva sección, la de opciones avanzadas, se han creado dos objetos de juego, estos son, el botón, y un panel contenedor donde se ubicarán todos los ajustes necesarios. Este panel será el que se active y desactive para visualizar u ocultar las opciones, respectivamente. Estos dos nuevos elementos serán gobernados por un grupo de diseño vertical, tal y como hicimos con el contenedor padre.

Los elementos hijos del panel ocultable funcionan igual que los ya vistos, con su etiqueta y su cuadro de texto para ser más tarde usados en la petición de transcripción junto a las opciones avanzadas. Se ha decidido crear un sistema de guardado de variables ya que podría ser muy incómodo tener que configurar las opciones cada vez que se abre la aplicación.

Por otra parte, se ha decidido incluir un par de funciones a la gestión del cuadro de resultado, i. e. incluir dos botones, el de borrado y el de guardado del texto. El botón de borrado eliminará el texto de resultado, para facilitar su lectura en la próxima ejecución, el botón de guardado nos permitirá crear un archivo con el contenido del cuadro de texto para su posterior revisión o para reporte de errores.

Figura 13: Resultado final de la interfaz de usuario.

4.4 Lógica de la aplicación

En este proyecto se está llevando a cabo un patrón de diseño por capa [40] como aprendimos en la asignatura de Ingeniería del software. En el apartado anterior, el 4.3, hemos visto el desarrollo de la capa de presentación, en este apartado veremos el de la capa de negocio tanto en C# como en Python.

4.4.1 Capa de negocio en C#

El ciclo de vida de una aplicación en Unity [5] comienza por la ejecución de las funciones Awake y Start[6] en este orden. En la función Awake se inicializan todas las variables necesarias antes de que la aplicación se ejecute, en nuestro caso, vamos a inicializar el entorno comandos y las variables previamente guardadas por el usuario.

Como se comentaba en el párrafo anterior, se ejecuta a continuación la función Start, la cual contempla aquellas acciones que se tienen que ejecutar solo una vez, al principio de la aplicación, aunque después de la función Awake, como son la detección de los idiomas disponibles y su consiguiente representación en el desplegable o la configuración del *plugin* del explorador de archivos, que en nuestro caso consistente en añadir un enlace de acceso rápido a nuestra carpeta especial *StreamingAssets*, así como redirigir los mensajes de Unity a nuestra ventana de salida.

Una vez ejecutados dichos comandos, la aplicación queda a la espera de las órdenes del usuario, es decir, el usuario podrá ahora ejecutar los diferentes métodos disponibles gracias a los botones de la interfaz.

Para el guardado de las opciones, ha sido necesaria la creación de una clase adicional con el marcado *"Serializable"*, el cual nos permite convertir el contenido de un objeto instancia de dicha clase en texto, en nuestro caso hemos optado por una serialización mediante JSON por su facilidad de uso.

```
[Serializable]
6 referencias
public class ConfigFile
{
    1 referencia
    public ConfigFile(float alpha, float beta)
    {
        this.alpha = alpha;
        this.beta = beta;
    }
    1 referencia
    public ConfigFile() { }
    public float alpha;
    public float beta;
}
```

Figura 14: Clase serializable para el guardado y recuperado de las opciones avanzadas.

[6] Se han seleccionado solo aquellas funciones interesantes para el proyecto. El ciclo de vida completo puede consultarse en el apartado Orden de ejecución del manual de unity [41].

El botón de guardar opciones ("*Save values*") llamará al método de mismo nombre que creará un objeto de la ya mencionada clase *ConfigFile* reuniendo en ella las opciones seleccionadas por el usuario y, posteriormente, procederá a serializar dicho objeto en el archivo *"config.json"* para poder ser cargado posteriormente al iniciar la aplicación.

Por otra parte, tenemos el botón *"Reset to default"*, es decir, el botón de restaurar, que regresará los valores a los originales, guardados en el propio código fuente para que no puedan ser accidentalmente modificados.

El botón de guardado del cuadro de texto con el resultado obtenido, también llamará al explorador de archivos a través del método *"PrintLogToFile"* el cual tomará el texto actual en dicho cuadro y lo guardará en la ubicación seleccionada por el usuario con el nombre "log.txt".

Por último, el botón más importante de la interfaz, el que nos da la posibilidad de ejecutar DeepSpeech, de nombre "Process audio", saneará los valores del usuario para comprobar que son válidos y ejecutará el método *"ProcessAudioToText"* que activará la animación de carga, formará la orden a ejecutar y la pasará como argumento al entorno comandos para ser procesada.

En caso de que la opción "vídeo falso" haya sido seleccionada, se añadirá a la orden comentada en el párrafo anterior, el argumento *"--fake_video"* que le indicará a nuestro cliente (del que hablaremos en el siguiente apartado, el 4.4.2) que deberá abrir un proceso para ejecutar un comando de la librería FFMPEG [7].

4.4.2 Capa de negocio en Python

Se trata de un archivo Python llamado *"client.py"* ubicado en la carpeta especial *StreamingAssets* del proyecto Unity, la cual será ejecutada mediante un entorno comandos y accederá a las dependencias que necesite dentro de la misma carpeta en el que está ubicado.

Parte del código que se puede encontrar en dicho archivo pertenece al equipo de DeepSpeech y está licenciado bajo Mozilla Public License 2.0, lo cual nos permite hacer uso del mismo y realizar las modificaciones necesarias.

Se ha editado la parte, a partir de la cual, se mostraba por pantalla, sin cambios, el resultado de la inferencia, i. e. la transcripción del audio, y se ha añadido en ese punto la lógica que

convierte el texto a formato SubRip [3] y se guarda en un archivo para que pueda ser empleado por los reproductores compatibles.

Como se comentó en el apartado de objetivos (1.2) y en el párrafo anterior, el formato elegido para formatear las closed captions [2] es SubRip, en el cual, el texto se divide en bloques que puedan mostrarse por pantalla, cuya primera línea indica el índice del mismo con un número sucesivo, en la siguiente línea del bloque, se recogen los tiempos en los que debe ser mostrado por pantalla, por último, en las sucesivas líneas, se recoge el texto con un máximo práctico de 2 líneas para no entorpecer la visualización.

Para ello se ha creado un contador que indicará el número de bloque en el que estamos. El resultado que nos devuelve DeepSpeech nos indica el tiempo preciso de cada carácter, incluídos espacios, por tanto, mientras se concatenan dichos caracteres, es necesario identificar si un espacio indica un final de palabra, o de bloque, además, hay que tener en cuenta el número máximo de palabras que caben en una sola línea y mantener el límite de dos líneas por bloque para una mejor legibilidad y no molestar al visionado de la imagen.

Por otra parte, se ha visto necesaria la creación de dos variables que ajustaran el umbral de tiempo máximo de silencio antes de cambiar de frase o bloque. Esto es debido a que en ocasiones los interlocutores hacen pausas, pero su frase no ha terminado, por lo que el texto se debería mantener en escena aún un tiempo extra. Sin embargo, si la escena ha acabado, el texto debería desaparecer.

```
1
00:00:00,000 --> 00:00:03,810
Das hier is sozusagen das Beweisstück
Ich bin inzwischen geimpft und zwar

2
00:00:03,820 --> 00:00:07,220
zweimal gegen Corona beziehungsweise
COVID-19 hab also den vollen Schutz

3
00:00:07,300 --> 00:00:12,050
und ich bin froh darüber aber auch
wenn grade breit darüber diskutiert
```

Figura 15: Ejemplo de *closed captions* extraídas del vídeo de MrWissen2go [42].

Posteriormente, el texto generado que representan las *closed captions*, con el formato adecuado, es guardado en un archivo con extensión .srt para que pueda ser empleado por cualquier reproductor con capacidad para mostrar subtítulos SubRip.

4.5. Implantación

Como se ha comentado en el apartado 3.5, queremos que la aplicación sea lo más cómoda posible y eso pasa por ser autosuficiente, i. e. el usuario no debería necesitar descargar ningún software ni dependencia adicional, por lo que se ha puesto especial atención en la recolección de todos y cada uno de los requisitos para ser incluido en la carpeta fuente de la aplicación y sean accesibles directamente por la misma.

Otra de las ventajas de Unity, como se comentó en el apartado 1.3 es que permite el despliegue en diversas plataformas, como son Windows, Linux, Android, iOS, entre otras, lo cual nos permite implantar este proyecto de forma sencilla para muchos sistemas operativos.

No obstante, por el momento, vamos a restringir el uso de la aplicación a su implementación para Windows debido a la limitación de tiempo, no obstante, queremos que la aplicación pueda ser utilizada en una mayor cantidad de plataformas en el futuro.

Para probar en Windows, es necesario utilizar otro dispositivo o una máquina virtual, ya que el software empleado en el ordenador de desarrollo podría incluir (y se ha comprobado que, de hecho, lo incluye) dependencias globales que podrían no estar instaladas en otro dispositivo como, por ejemplo, la instalación de Python.

En este caso, hemos podido acceder a otro ordenador con Windows 10 instalado, sin ninguna dependencia, por lo que podemos comprobar si todos los requisitos están correctamente satisfechos en la versión final.

Al probar en otro dispositivo Windows, nos damos cuenta que falta el ejecutable de Python y posteriormente también las dependencias de numpy y el propio DeepSpeech, ya que estaban instalados de forma global mediante pip.

La solución que se ha alcanzado para resolver dichas dependencias ha sido crear un entorno virtual temporal de Python e instalar ahí las diferentes librerías necesarias. Una vez descargadas, han sido copiadas en el proyecto de forma que sean accesibles por el ejecutable del proyecto y tras repetir la prueba, ha funcionado.

5. Pruebas

En este apartado hablaremos de las pruebas llevadas a cabo por Aashish Agarwal en el modelo que estamos utilizando y qué corpus empleó para su entrenamiento, posteriormente comentaremos acerca de las pruebas unitarias creadas tanto para el código implementado en C# como para el de Python y acabaremos con las pruebas llevadas a cabo de forma manual sobre la aplicación completada y su resultado final.

5.1 Pruebas realizadas sobre el modelo

Debido a los problemas comentados en el apartado 4.2.2, no hemos podido entrenar nuestro propio modelo, no obstante, es importante resaltar ¿qué pruebas se han llevado a cabo? y ¿qué resultados se han obtenido en el entrenamiento del modelo en uso en este trabajo, cortesía de Aashish Agarwal? [35].

Como nos comenta Aashish Agarwal en su artículo, en la primera versión del modelo, se emplearon los corpus de VoxForge [43], Tuda-De [44] y Mozilla Common Voice [21], los cuales se fueron alternando para obtener los mejores resultados posibles [36].

Dataset	Size	Median Length	# Speakers	Condition	Type
Voxforge	35h	4.5s	180	noisy	read
Tuda-De	127h	7.4s	147	clean	read
Mozilla Common Voice	140h	3.7s	>1,000	noisy	read

Tabla 1: Descripción de los corpus utilizados por Aashish Agarwal para el entrenamiento del modelo [36]

En la versión que nos ocupa, el número de corpus disponibles gratuitos en la web ha aumentado, por lo que el modelo actual está cimentado en los siguientes: Spoken Wikipedia Corpora [45], Common Voice de Mozilla, M-AILABS Speech Dataset [46], Tuda-De y Voxforge, los cuales suman un total de 3 172 horas de audio validadas.

El vocabulario que podemos encontrar en los diferentes corpus es bastante variado, por un lado, podemos encontrar un lenguaje culto y con vocabulario relacionado con la política en el corpus como por ejemplo en el de VoxForge que también contiene vocabulario de gran variedad de especialidades ya que contiene frases leídas de la Wikipedia en alemán. Por otra parte, podemos encontrar un lenguaje más neutro en el corpus de M-AILABS ya que contiene la lectura de cerca de 1 000 horas de audio procedente de libros escritos entre 1884 y 1964.

Las transcripciones de los audios de todos los corpus fueron limpiadas, combinadas y luego divididas en tres conjuntos: entrenamiento, validación y *test*, en un 70, 15 y 15 por ciento respectivamente [36]. El entrenamiento permite a la red neuronal crear una relación estadística entre los datos de entrada y las etiquetas, mientras que los conjuntos de validación y *test* nos permiten determinar el grado de precisión que ha alcanzado nuestro modelo.

Para obtener resultados comparables y, por tanto, poder cualificar los diferentes escenarios propuestos, se ha empleado la métrica WER (*Word Error Rate)*, la cual mide el ratio de predicciones de errores en cada palabra frente a los aciertos. Consiste en un sencillo cálculo que nos muestra el porcentaje de fallos frente a aciertos cometidos por el modelo a la hora de procesar el conjunto de test.

$$WER = \frac{S + D + I}{N}$$

where...
S = number of substitutions
D = number of deletions
I = number of insertions
N = number of words in the reference

Figura 16: Fórmula de la tasa de error por palabra «Word error rate and who is winning» [47].

Adicionalmente, se ha hecho un estudio de los mejores hiper parámetros mediante el fijado de dos de ellos y el ajuste del tercero, i. e. se han fijan el abandono (*dropout*), el tamaño de lote (*batch size*) y se ha buscado cuál es el mejor valor para el ratio de aprendizaje (*learning rate)*.

Una vez obtenido un buen valor para el ratio de aprendizaje, este es fijado y comienza la búsqueda del mejor valor de abandono. Y de la misma forma, una vez obtenidos ambos, se busca el mejor valor para el tamaño del lote que se ajuste a los ya obtenidos.

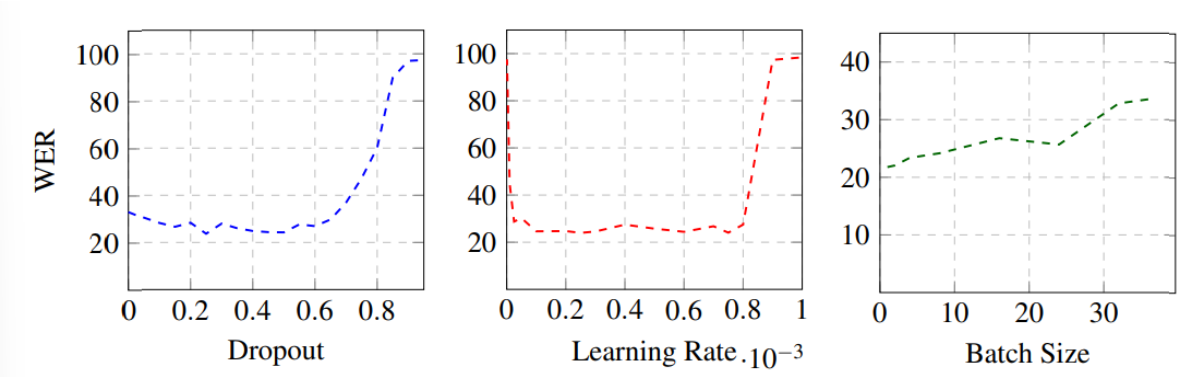

Figura 17: Búsqueda de los mejores valores para los hiper parámetros abandono (dropout), ratio de aprendizaje (learning rate) y tamaño del lote (batch size) llevada a cabo por Aashish Agarwal [36].

Tras realizar las pruebas con cada combinación de los corpus disponibles y empleando los mejores valores obtenidos para los hiper parámetros indicados, se puede concluir que la mejor combinación para la primera versión del modelo (la que devuelve el WER más bajo) es el que emplea Tuda-De con VoxForge con un error del 15.1%, seguido por el mismo conjunto, pero añadiendo Common Voice de Mozilla que obtiene un total de 21.5% de error.

Dataset	WER
Mozilla	79.7
Voxforge	72.1
Tuda-De	26.8
Tuda-De + Mozilla	57.3
Tuda-De + Voxforge	15.1
Tuda-De + Voxforge + Mozilla	21.5

Tabla 2: Resultados obtenidos en los diferentes corpus sometidos a prueba por Aashish Agarwal [36].

5.2 Pruebas unitarias sobre Python

En cuanto al código realizado en Python, se ha desarrollado un conjunto de pruebas unitarias que nos permitirán probar las funcionalidades que hemos desarrollado nosotros, i. e. las que corresponden a la conversión del resultado de DeepSpeech al formato SubRip [3].

Para lo cual se ha creado un archivo llamado *"testing.py"* que recoge los métodos creados junto con sus datos de entrada y los datos que esperamos como salida para realizar una batería de pruebas que los valide.

Esto es muy sencillo en Python ya que disponemos de la palabra reservada *assert* que nos permite comprobar si un valor es igual a otro, en cuyo caso, el método termina, por contra, si el resultado de dicha operación booleana es falso, elevará una excepción que parará el resto de pruebas, permitiéndonos, además, añadir un comentario para que sea mostrado junto con la excepción y nos facilite la tarea de solución de errores.

Al final de todas las pruebas unitarias y de integración, se ha añadido un mensaje indicando que todas las pruebas han sido satisfactorias, ya que, como se ha comentado en el párrafo

anterior, si alguna prueba fallase, la ejecución terminaría y, por tanto, no podríamos leer dicho mensaje.

Para poder ejecutar las pruebas desde este nuevo archivo ha sido necesaria la importación de todos los métodos de nuestro fichero, eso ha sido posible gracias a la instrucción *"from client import *"* al principio del archivo de pruebas. Esto las convierte en accesibles y, por tanto, ejecutables. De esta forma podremos confirmar si dichas funciones tienen la funcionalidad deseada.

Los resultados de las pruebas unitarias llevadas a cabo han sido satisfactorios, todos los métodos probados han devuelto los valores esperados para los datos proporcionados.

5.3 Pruebas unitarias sobre Unity

Unity [5] dispone de un *framework* integrado de pruebas unitarias conocido como Unity Testing Framework o UTF [48]. Dicho sistema permite pruebas de dos tipos, las de modo editor y las del modo *play* o en tiempo de ejecución [49].

Las pruebas del modo editor son aquellas que validan todas las funcionalidades o variables qué no precisan que la aplicación esté en ejecución, por ejemplo, valores de constantes o funciones matemáticas.

Por otra parte, tenemos las pruebas del modo *play* que son las que requieren que la aplicación se esté ejecutando. Dichas "pruebas deben cumplir las siguientes condiciones: tener una definición de ensamblaje con referencia a *nunit.framework.dll*, tener el código de *test* cargado en una carpeta junto el archivo .asmdef y que el test de ensamblaje debe referenciar a un ensamblaje dentro del código que quieres probar" [49].

Se trata de un sistema muy visual y usable ya que podemos pasar todos los test de un solo clic y podremos ver el resultado de nuestras pruebas marcado mediante un símbolo de verificación o, por el contrario, con círculo rojo tachado, indicando que dicha prueba ha resultado fallida.

Para crear las pruebas, es necesario emplear el marcado "Test" para las del modo editor y "UnityTest" para las del modo ejecución, los cuales explicamos en el segundo y tercer párrafo de este apartado 5.3.

Los métodos marcados como editor, deberán devolver *void* mientras que los de ejecución devolverán una interfaz enumerador ya que son corrutinas. Esto es así para poder realizar esperas en el código y, por tanto, que tenga tiempo suficiente para ejecutar las acciones correspondientes sin paralizar la aplicación.

Una vez programados todos los métodos de pruebas, podemos ejecutarlos mediante una ventana integrada de Unity llamada “TestRunner”. En nuestro caso, las pruebas realizadas han resultado satisfactorias.

5.4 Pruebas de transcripción con la aplicación

Adicionalmente a las pruebas ya mencionadas en los puntos 6.1, 6.2 y 6.3, se ha considerado ineludible validar de forma manual la aplicación ya que el valor devuelto de la inferencia puede diferir tanto en tiempo como en resultado.

Para facilitar esta tarea, se ha creado un conjunto archivos que contienen segmentos cortos de audio de tal forma que se puedan probar rápidamente diferentes fonemas o palabras. Y para facilitar estas pruebas manuales, se ha decido poner por defecto la ruta al audio más empleado debido a su sencillez de vocabulario y claridad en el habla.

Este audio se corresponde con el primer segundo de habla de un vídeo del canal de YouTube de Easy German en el que la profesora Cari Schmidt saluda al público con un *“Hallo Leute”* ("hola gente").

Las pruebas manuales no fueron, en principio, satisfactorias ya que las predicciones del modelo empleado estaban resultando muy diferentes a lo esperado. A juzgar por los valores WER obtenidos por Aashish Agarwal comentados en apartado 5.1, pensamos que podría estar relacionado con una configuración errónea del modelo a la hora de lanzar la ejecución.

Fueron realizadas muchas pruebas al respecto, cambiando tanto el valor alfa, como el beta o el *beam width* e incluso eliminando el diccionario del lenguaje obteniendo resultados similares hasta quedar sin ideas.

No obstante, comenzamos a hacer pruebas empleando los archivos originales del corpus, concretamente del set de test y cuyos resultados eran los esperados coincidiendo con el trabajo de Aashish Agarwal, finalmente acabamos dando con la clave: los audios que

generábamos nosotros estaban por defecto en estéreo. Una vez generado un fichero de audio en versión monoaural [8], los resultados fueron completamente satisfactorios.

Figura 18: Captura de las closed captions en un vídeo de Easy German. [50]

En la fig. 18 podemos apreciar dos pequeños errores. En la primera línea, la profesora Schmidt dice “Easy German” mientras que el modelo ha entendido “isi german”; en la segunda tenemos un error fonético, y es que en el vídeo se hace referencia a la enfermedad Coronavirus empleando su abreviación, i. e. “Corona”, la cual, se escribe con “C”. Probablemente dicha palabra no se encuentre dentro del vocabulario del corpus.

6. Conclusiones

Estamos muy satisfechos con el resultado final de la aplicación. Hemos creado una interfaz muy usable y cómoda que permite, no solo emplear modelos en alemán, como era el objetivo inicial de este proyecto, sino también los disponibles en internet o nuevos modelos entrenados por los usuarios. Lo cual permite mejorar los resultados futuros de la aplicación simplemente sustituyendo el modelo por uno más completo.

Por otra parte, hemos obtenido una interfaz que, aunque no tenga un diseño gráfico sofisticado, sí es agradable a la vista y cumple los requisitos estéticos mínimos. Además, gracias a los diversos componentes que nos brinda Unity [5], hemos conseguido una interfaz completamente responsiva incluso en dispositivos, muy pequeños o muy grandes.

La aplicación está preparada para obtener unas closed captions [2] precisas que permitan al alumno mejorar su aprendizaje mediante películas u otros soportes digitales en alemán y demás idiomas en caso de que el usuario decidiese añadir modelos adicionales.

Gracias a este proyecto hemos podido mejorar en el uso de Unity, especialmente en el diseño y construcción de interfaces gráficas de usuario. Además, hemos tenido la oportunidad de emplear, por primera vez, pruebas unitarias en Unity.

Adicionalmente, aunque externo a la aplicación, debido al desarrollo de este trabajo, hemos conocido el complemento para Trello [31] TeamGantt, el cual será utilizado en el futuro para mejorar la organización de nuevos proyectos. Por último, la escritura de la memoria ha supuesto una mejora en la capacidad de expresión.

Los conocimientos adquiridos en diferentes asignaturas han resultado muy útiles a la hora de desarrollar este proyecto: las clases de Ingeniería del software fueron muy útiles a la hora de estructurar el código por capas o para componer las pruebas unitarias. Por otra parte, la organización del proyecto se ha basado en lo aprendido en la asignatura Gestión de proyectos. Para comprender el modelo han resultado útiles las clases tomadas durante la estancia Erasmus *Autonomous Learning*. Por último, durante las clases de Entornos de desarrollo de videojuegos y *Animation and design of videogames* sirvieron para tener una base en el uso de Unity.

6.1 Futuras mejoras

Debido a la limitación de tiempo, y aunque los objetivos iniciales de este trabajo están completados, sí sería buena idea mejorar algunos aspectos de la aplicación.

En cuanto al diseño gráfico de la aplicación, como comentamos en el apartado 6, es correcto pero mejorable. Se propone un rediseño del apartado estético, creando para ello una interfaz que se vea moderna. Además, sería interesante dar al usuario la posibilidad de personalizarla realizando ciertos cambios, como pueden ser la selección de colores, seleccionar una fuente tipográfica diferente, reubicación de las secciones o incluso ocultar las que no le sean interesantes.

Además de lo ya comentado, se podría crear un menú adicional que permitiera descargar de internet nuevos modelos preentrenados y ser ubicados automáticamente en la carpeta correspondiente para ser localizados por la aplicación, es decir, se podrían incluir tantos enlaces a modelos de idiomas como seamos capaces y, permitir al usuario mediante un clic, que dichos idiomas sean descargados y ubicados correctamente para su posterior uso. Lo cual nos lleva a que se debería dar al usuario la opción de eliminar un modelo ya existente en la aplicación.

En nuestro proyecto, al abrir el explorador de archivos, dicha ventana queda dentro de los límites de nuestra interfaz, es decir, en lugar de ser dos ventanas independientes, se trata de una que contiene ambas interfaces, por tanto, la propuesta sería encontrar el modo de ejecutar dicho explorador en una ventana propia.

Adicionalmente, sería muy interesante poder aprovechar el potencial multiplataforma de Unity y DeepSpeech. De esa forma, la aplicación no estaría limitada a su uso en un dispositivo con Windows, sino que podría ser ejecutada, por ejemplo, en Linux o Android.

Otra posible mejora podría ser permitir al usuario seleccionar más de un archivo de audio, de tal forma que podamos explotar la ventaja que nos brinda DeepSpeech que permite la ejecución concurrente.

Por último, sería interesante que la interfaz de la aplicación estuviera traducida a más idiomas. Para lo cual, se debería crear un archivo por cada idioma que contenga el texto de la aplicación junto a la clave que sustituirá para su posterior traducción y se permitiera al usuario emplear la aplicación en otros idiomas. A este proceso se le conoce como localización.

7. Referencias

- [1] N. Escobar, «7 consejos para aprender idiomas más fácil y rápido», *Hipertextual*, may 21, 2015. http://hipertextual.com/2015/05/consejos-para-aprender-idiomas (accedido sep. 01, 2021).

- [2] «Subtítulos cerrados», *Wikipedia, la enciclopedia libre*. feb. 11, 2021. Accedido: sep. 03, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Subt%C3%ADtulos_cerrados&oldid=133153512

- [3] «SubRip - Wikipedia, la enciclopedia libre». https://es.wikipedia.org/wiki/SubRip (accedido sep. 04, 2021).

- [4] «Reconocimiento del habla», *Wikipedia, la enciclopedia libre*. ago. 29, 2021. Accedido: sep. 03, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Reconocimiento_del_habla&oldid=137989723

- [5] «Unity (motor de videojuego)», *Wikipedia, la enciclopedia libre*. jul. 01, 2021. Accedido: sep. 03, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Unity_(motor_de_videojuego)&oldid=136709983

- [6] *Project DeepSpeech*. Mozilla, 2021. Accedido: sep. 05, 2021. [En línea]. Disponible en: https://github.com/mozilla/DeepSpeech

- [7] «FFmpeg», *Wikipedia, la enciclopedia libre*. abr. 10, 2021. Accedido: sep. 05, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=FFmpeg&oldid=134660936

- [8] «Sonido monoaural», *Wikipedia, la enciclopedia libre*. oct. 22, 2019. Accedido: sep. 05, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Sonido_monoaural&oldid=120654321

- [9] «El reconocimiento automático del habla». http://liceu.uab.es/~joaquim/speech_technology/tecnol_parla/recognition/speech_recognition/reconocimiento.html (accedido ago. 19, 2021).

- [10] «¿Cómo funciona el reconocimiento automático del habla? | by Diego Campos Sobrino | SoldAI | Medium». https://medium.com/soldai/c%C3%B3mo-funciona-el-reconocimiento-autom%C3%A1tico-del-habla-eb038ecfe72e (accedido ago. 20, 2021).

- [11] A. Hannun *et al.*, «Deep Speech: Scaling up end-to-end speech recognition», *arXiv:1412.5567 [cs]*, dic. 2014, Accedido: ago. 14, 2021. [En línea]. Disponible en: http://arxiv.org/abs/1412.5567

- [12] M. G. Ozsoy, I. Cicekli, y F. N. Alpaslan, «Text Summarization of Turkish Texts using Latent Semantic Analysis», en *COLING 2010, 23rd International Conference on Computational Linguistics, Proceedings of the Conference, 23-27 August 2010, Beijing, China*, 2010, pp. 869-876. Accedido: sep. 05, 2021. [En línea]. Disponible en: https://aclanthology.org/C10-1098/

- [13] «Subtitles for the Deaf or Hard-of-Hearing (SDH) - Subtitles, Closed Captions, and SDH.» https://blog.ai-media.tv/blog/what-is-sdh (accedido sep. 05, 2021).

- [14] «Speech-to-Text: reconocimiento de voz automático». https://cloud.google.com/speech-to-text?hl=es (accedido sep. 01, 2021).

- [15] «Speech to Text: conversión de audio en texto | Microsoft Azure». https://azure.microsoft.com/es-es/services/cognitive-services/speech-to-text/ (accedido sep. 01, 2021).

- [16] «IBM Watson - Visión general», jun. 30, 2021. https://www.ibm.com/es-es/cloud/watson-speech-to-text (accedido sep. 01, 2021).

- [17] «Supported platforms for inference — DeepSpeech 0.9.3 documentation». https://deepspeech.readthedocs.io/en/v0.9.3/SUPPORTED_PLATFORMS.html (accedido sep. 01, 2021).

- [18] *DeepSpeech 0.9.x Examples*. Mozilla, 2021. Accedido: sep. 01, 2021. [En línea]. Disponible en: https://github.com/mozilla/DeepSpeech-examples

- [19] «Release DeepSpeech 0.9.3 · mozilla/DeepSpeech», *GitHub*. https://github.com/mozilla/DeepSpeech/releases/tag/v0.9.3 (accedido sep. 01, 2021).

- [20] «Links to pretrained models», *Mozilla Discourse*, jun. 26, 2020. https://discourse.mozilla.org/t/links-to-pretrained-models/62688 (accedido sep. 01, 2021).

- [21] «Common Voice by Mozilla». https://commonvoice.mozilla.org/ (accedido sep. 01, 2021).

- [22] «Training Your Own Model — DeepSpeech 0.9.3 documentation». https://deepspeech.readthedocs.io/en/v0.9.3/TRAINING.html#transfer-learning-new-alphabet (accedido ago. 24, 2021).

- [23] «Sostenibilidad», *Google Cloud*. https://cloud.google.com/sustainability?hl=es (accedido sep. 01, 2021).

- [24] «Introducción a la conversión de voz en texto: el servicio Voz - Azure Cognitive Services». https://docs.microsoft.com/es-es/azure/cognitive-services/speech-service/speech-to-text (accedido sep. 01, 2021).

- [25] «Sostenibilidad de Azure: tecnologías sostenibles | Microsoft Azure». https://azure.microsoft.com/es-es/global-infrastructure/sustainability/ (accedido sep. 01, 2021).

- [26] «AudioMountain.com». http://www.audiomountain.com/tech/audio-file-size.html (accedido sep. 01, 2021).

- [27] «IBM's ambitious energy and climate goals», jun. 30, 2021. https://www.ibm.org/responsibility/2020/priorities/protecting-the-environment (accedido sep. 01, 2021).

- [28] R. Ardila *et al.*, «Common Voice: A Massively-Multilingual Speech Corpus», *arXiv:1912.06670 [cs]*, mar. 2020, Accedido: sep. 03, 2021. [En línea]. Disponible en: http://arxiv.org/abs/1912.06670

- [29] «DeepSpeech Model — Mozilla DeepSpeech 0.9.3 documentation». https://deepspeech.readthedocs.io/en/r0.9/DeepSpeech.html (accedido sep. 03, 2021).

- [30] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, vol. 385. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-24797-2.

- [31] «Trello», *Wikipedia, la enciclopedia libre*. ago. 04, 2021. Accedido: sep. 03, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Trello&oldid=137447987

- [32] «Home». https://www.teamgantt.com (accedido sep. 05, 2021).

- [33] S. Y. KULA, *Unity Simple File Browser*. 2021. Accedido: sep. 05, 2021. [En línea]. Disponible en: https://github.com/yasirkula/UnitySimpleFileBrowser

- [34] U. Technologies, «Unity - Manual: TextMeshPro». https://docs.unity3d.com/Manual/com.unity.textmeshpro.html (accedido sep. 05, 2021).

- [35] A. Agarwal y Torsten, Zesch, «Automatic Speech Recognition (ASR) - DeepSpeech German». [En línea]. Disponible en: https://github.com/AASHISHAG/deepspeech-german.

- [36] A. Agarwal y T. Zesch, «German End-to-end Speech Recognition based on DeepSpeech», oct. 2019. Accedido: ago. 30, 2021. [En línea]. Disponible en: https://www.researchgate.net/publication/336532830_German_End-to-end_Speech_Recognition_based_on_DeepSpeech

- [37] U. Technologies, «Vertical Layout Group (Grupo de diseño vertical) - Unity Manual». https://docs.unity3d.com/es/2018.4/Manual/script-VerticalLayoutGroup.html (accedido sep. 04, 2021).

- [38] U. Technologies, «Layout Element (Elemento de Diseño) - Unity Manual». https://docs.unity3d.com/es/2018.4/Manual/script-LayoutElement.html (accedido sep. 04, 2021).

- [39] U. Technologies, «Horizontal Layout Group (Grupo de diseño horizontal) - Unity Manual». https://docs.unity3d.com/es/2019.4/Manual/script-HorizontalLayoutGroup.html (accedido sep. 04, 2021).

- [40] «Programación por capas», *Wikipedia, la enciclopedia libre*. jun. 10, 2021. Accedido: sep. 04, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Programaci%C3%B3n_por_capas&oldid=136238285

- [41] «Unity - Manual: Execution Order of Event Functions (Orden de Ejecución de Funciones de Evento)». https://docs.unity3d.com/es/530/Manual/ExecutionOrder.html (accedido ago. 29, 2021).

- [42] Corona-Impfpflicht? Nein, danke! | #mirkosmeinung, (jul. 26, 2021). Accedido: ago. 29, 2021. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=jET5873K-KA

- [43] «Open Speech Data Corpus for German - voxforge.org». http://www.voxforge.org/home/forums/other-languages/german/open-speech-data-corpus-for-german (accedido ago. 24, 2021).

- [44] R. Zierke, «Open Source Acoustic Models for German Distant Speech Recognition». https://www.inf.uni-hamburg.de/en/inst/ab/lt/resources/data/acoustic-models.html (accedido sep. 05, 2021).

- [45] T. Baumann, A. Köhn, y F. Hennig, «The Spoken Wikipedia Corpus collection: Harvesting, alignment and an application to hyperlistening», *Lang Resources & Evaluation*, vol. 53, n.º 2, pp. 303-329, jun. 2019, doi: 10.1007/s10579-017-9410-y.

- [46] «The M-AILABS Speech Dataset – caito». https://www.caito.de/2019/01/the-m-ailabs-speech-dataset/ (accedido ago. 24, 2021).

- [47] Sonix. https://sonix.ai/articles/word-error-rate (accedido ago. 30, 2021).

- [48] «About Unity Test Framework | Test Framework | 1.1.29». https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/index.html (accedido sep. 05, 2021).

- [49] «Edit Mode vs. Play Mode tests | Test Framework | 1.1.29». https://docs.unity3d.com/Packages/com.unity.test-framework@1.1/manual/edit-mode-vs-play-mode-tests.html (accedido ago. 31, 2021).

- [50] Easy German: Learn German From the Streets!, *Coronavirus Vaccines in Germany | Easy German 407*, (jul. 04, 2021). Accedido: sep. 04, 2021. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=Nix9rXsqKXU

- [51] «CUDA on WSL User Guide». http://docs.nvidia.com/cuda/wsl-user-guide/index.html (accedido ago. 29, 2021).

- [52] «Windows Subsystem for Linux», *Wikipedia, la enciclopedia libre*. ene. 06, 2021. Accedido: ago. 23, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Windows_Subsystem_for_Linux&oldid=132188860

- [53] «Training Your Own Model — DeepSpeech 0.9.3 documentation». https://deepspeech.readthedocs.io/en/v0.9.3/TRAINING.html#transfer-learning-new-alphabet (accedido ago. 24, 2021).

Apéndice A

Detalle de los problemas a la hora de entrenar el modelo

La primera opción que se intentó fue configurar el entorno de entrenamiento en Windows ya que es el sistema operativo instalado en nuestro equipo y un requisito, hasta hace poco, de Unity, por lo que, aunque no esté indicado en las instrucciones de DeepSpeech, en este momento solo se dispone de un equipo con la capacidad de *hardware* suficiente como para entrenar un modelo, y su sistema operativo es Windows.

El primer paso consiste en crear el entorno virtual de Python en Windows (virtualenv), al intentar instalar los requisitos, el instalador de Python no reconoce correctamente el archivo que contiene las dependencias del proyecto con sus versiones correspondientes. No obstante sí reconoce alguna de ellas, lo cual fue confuso ya que, en efecto, la lectura del fichero parecía correcta.

Tras descubrir el error y constatar que este problema no ocurre en Linux (plataforma dónde recomienda el equipo de Mozilla que se debe entrenar), se decide crear una máquina virtual con Linux Mint, ya que es el sistema operativo GNU/Linux que más hemos empleado y, por tanto, mejor conocemos.

Con la máquina virtual, se procede de nuevo a crear el entorno virtual Python tal y como se indica en las instrucciones, sin embargo, hay otra dependencia que no puede ser satisfecha y se instala de forma manual, se trata de SoX que se debe instalar tanto como dependencia de Python como instalable de Linux.

Se procede entonces a instalar CUDA [51], pero entonces nos encontramos con más problemas. Los manuales de CUDA son numerosos y confusos y al proceder a instalar la versión que indican desde el equipo de DeepSpeech, el instalador refiere no disponer de una versión válida de Visual Studio, por lo que se procede a instalar la versión de 2015.

La instalación de Visual Studio falla y tras investigar por internet posibles causas, otros usuarios de la red reportan haberlo conseguido en Ubuntu 18.04, simultáneamente se encuentra que es una de las apuestas más sólidas a la hora de entrenar con DeepSpeech.

Por lo que se decide volver a comenzar instalando Ubuntu 18.04 y eliminando Linux Mint. En esta ocasión, pese a lo que indicaban los comentarios encontrados en internet, la instalación

de las dependencias sigue fallando, esta vez nos indica que una de las dependencias no es compatible con otra dependencia. Tras investigar de nuevo, conseguimos finalmente y por primera vez, instalar todas las dependencias. La solución fue incluir el siguiente argumento al comando *--use-feature=2020-resolver*.

Desgraciadamente, los problemas vuelven a surgir ya que, para poder hacer uso de la tarjeta gráfica en el entrenamiento, es necesario instalar el complemento de VirtualBox[7] Guest Additions[8], para lo cual será necesario resolver una gran cantidad de errores. Una vez solucionados todos los problemas e instalado dicho complemento, no parecía tener efecto[9].

Finalmente descubrimos que el problema era tan simple como, además de reiniciar el ordenador virtual como indican las instrucciones, también se debe reiniciar el ordenador anfitrión.

Una vez instalados de forma satisfactoria todos los requisitos para el entrenamiento, se procede a ejecutar el preprocesado del corpus, en el que los archivos se convierten de mp3 a .wav, se remuestrea a 16kHz y se generan archivos .csv que serán posteriormente usados para el entrenamiento, desarrollo y test del modelo como se comentó en el apartado 3.2.1.

Tras una serie de errores solucionados y ciertos problemas de espacio también solventados, el preprocesado de datos con el corpus de Common Voice queda completado y se procede finalmente al entrenamiento del modelo.

Tras un día completo esperando, se aprecia que dicho entrenamiento no ha avanzado de la Epoch 0. Más tarde, en un nuevo intento, se obtiene un mensaje de error, que, de forma resumida contiene el siguiente mensaje *"No OpKernel was registered to support Op 'CudnnRNNCanonicalToParams'"*. Tras una investigación de las causas se determina que podría ser una incompatibilidad con CUDA o la propia gráfica con la máquina virtual, es decir, la virtualización no sería compatible con el uso de una tarjeta gráfica.

[7] Software empleado para virtualizar.
[8] Conjunto de ampliaciones de VirtualBox que permiten, entre otras, aceleración hardware o integración de pantalla y ratón.
[9] La pantalla no se ajustaba al tamaño total disponible, prueba de que GuestAdditions no está teniendo efecto.

Por lo que se decide cambiar una vez más de enfoque. En este punto o incluso antes, la instalación de un *dual boot*[10] parece la opción más obvia, ya que, en principio, es la opción más práctica y sencilla pero lamentablemente el factor limitante aquí es el equipo, pues no se dispone de espacio suficiente en el disco duro del ordenador para mantener dos sistemas operativos simultáneamente. Y para poder utilizar la gráfica sin renunciar al sistema operativo se decide instalar Windows Subsystem for Linux (WSL)[11], opción que han explorado exitosamente algunos usuarios de la red.

Siguiendo el ejemplo de algunos internautas que se han decantado por esta opción, se procede a explorarla instalando Ubuntu para WSL y configurando el entorno de forma normal, tal y como se indica en el manual de DeepSpeech.

En esta ocasión, el proceso de instalación de dependencias no da problemas y el manual de instalación de CUDA en WSL es claro y conciso[12] por lo que la configuración e instalación del conjunto de requisitos se completa con relativa rapidez y se procede al entrenamiento, es entonces cuando obtenemos de nuevo el mensaje de error de *"No OpKernel was registered to support Op 'CudnnRNNCanonicalToParams'"* lo cual parece indicar que el sistema no está reconociendo la gráfica, hipótesis que parece confirmarse tras ejecutar el comando nvidia-smi para obtener la información de la gráfica, y su resultado es *"segmentation fault"*. De nuevo es necesario investigar a qué se debe y concluimos que el problema podría ser la versión de WSL.

De acuerdo con el manual, es necesaria la versión 2 de WSL, pero para poder instalar dicha versión, es necesario entrar en el programa Insider de Microsoft Windows[13], lo cual nos requiere activar dicho complemento y formatear el ordenador. Se procede a ello.

Una vez más, es momento de configurar desde el principio el entorno, incluyendo la instalación y posterior actualización de WSL a WSL 2 y Ubuntu 18.04 para dicho subsistema. Tal y como ocurrió con la versión 1 de WSL y en parte gracias a la experiencia adquirida de tantas instalaciones, conseguimos instalar todos los requisitos sin problemas.

[10] Consiste en la partición del disco duro en dos sistemas operativos, usualmente Windows y Linux para obtener lo mejor de ambos entornos.
[11] Se trata de una capa de compatibilidad que permite emplear Ubuntu desde Windows de forma nativa [35].
[12] Contiene una entrada dedicada a la instalación de CUDA en WSL [52].
[13] Consiste en un programa en el cual se permite explorar las últimas características de Windows antes de su salida estable, es decir, en beta.

No obstante, al intentar entrenar una vez más, volvemos a obtener el mismo mensaje de error. Repetimos la investigación, ahora incluyendo que se trata de la versión 2. Esta vez podría estar producido por una mala configuración de CUDA/CuDNN.

Paralelamente a este problema, el equipo empieza a tener pequeños errores y está afectando a otras asignaturas ya que al intentar compilar algo que antes funcionaba y que no está relacionado con este TFG, comienza a dar pantallas azules BSoD, pero de color verde (las correspondientes a Windows Insider). En mi opinión, parece estar relacionado con la actualización de Windows a características en beta.

Después de cierta investigación y una vez arreglado el problema de la otra asignatura, logramos que funcione gracias a la reinstalación de la librería cuda-toolkit-10-0 que, por alguna razón, habría quedado corrupta. Esta vez, se consigue que comience el entrenamiento pero no parece que alcance nunca las 10 Epoch, se barajan dos hipótesis: es posible que requiera mucho más tiempo según avanzan las Epoch y, por diferentes razones, no se ha podido dejar hasta el final, por tanto la solución sería conseguir que quedara varios días seguidos entrenando, por otra parte podría ser que quedase bloqueado en alguna Epoch y, por tanto, habría un problema subyacente no identificado y no avanzaría independientemente del tiempo empleado.

La opción más sencilla es tratar de dejarlo por más tiempo, pero desgraciadamente, en un momento determinado el disco duro extraíble donde estaba ubicado el corpus y demás archivos necesarios para el entrenamiento, por alguna razón, se desconectó.

Al intentar iniciar el entrenamiento, sin explicación, volvió a dar el error de *“No OpKernel was registered to support Op 'CudnnRNNCanonicalToParams'”*, sospechamos que podría tratarse de CUDA y siguiendo la sugerencia de un usuario de la red, se prueba a ejecutar el siguiente comando en los ficheros binarios de CUDA *strace ./BlackScholes* lo cual nos indica que, efectivamente, no está siendo capaz de detectar la gráfica o hay alguna incompatibilidad con la instalación de CUDA.

Después de varios intentos fallidos de hacerlo funcionar de nuevo, nos rendimos y gracias al tutor, la UPV nos cede un equipo remoto con tarjeta gráfica y Ubuntu 18.04 en la UPV, de acceso por VPN mediante SSH. Además, tiene la ventaja de que de la instalación de CUDA se encargan los administradores del equipo.

Tras instalar una vez más el entorno en la máquina de la UPV, es necesario instalar SoX pero no tenemos permisos de administrador. Por cortesía, intentamos instalar dicho requisito de forma local, de tal forma que solo sería accesible por nosotros y no requeriría una instalación que afectase al resto del equipo. Esto es posible descargando, descomprimiendo y ejecutando la compilación de los componentes del *software*.

No obstante, pese a que la recién instalada herramienta estaba siendo reconocida por el programa de entrenamiento, éste reportaba que no era capaz de procesar archivos mp3 y que un *software* adicional debía ser instalado para tal efecto. Tal y como se hizo con SoX, se procede a instalar dicho componente sin éxito, ya que tenía varios paquetes como dependencia, los cuales, a su vez, tenían más dependencias. Tras una minuciosa búsqueda e instalación local de todas las dependencias, el programa seguía solicitando la ampliación de funcionalidad antes mencionada.

Por este motivo, se decidió que la mejor solución era, sin duda, solicitarlo al administrador. Se procede sin más problemas al preprocesado del corpus que termina sin fallos.

Es el momento de intentar entrenar. Otra vez. Y de nuevo, nos volvemos a topar con el mismo fallo de *"No OpKernel was registered to support Op 'CudnnRNNCanonicalToParams'"*, pero esta vez sí podemos determinar que la gráfica es reconocida por el sistema con la orden *nvidia-smi*, es decir, la gráfica está correctamente configurada y es accesible. De nuevo sospechamos que la instalación de CUDA podría no ser buena, ya que nos percatamos que falta la carpeta de ejecutables *bin*. Adicionalmente, observamos que en el manual especifica que es necesario instalar CuDNN a través de Docker.

No obstante, y tras haber dedicado alrededor de un mes en el intento de entrenamiento, nos vemos obligados a renunciar a este apartado y avanzar en el resto. Los motivos de abandonar en este momento se deben, en parte, a la cantidad de incógnitas, i. e. cuál es la verdadera razón por la que no se ha conseguido entrenar, si es por la versión o configuración de CUDA, o es por CuDNN, entre otras posibles causas. La otra razón por la que se ha decidido parar es que la administración de la máquina remota estaría de vacaciones en agosto y no podría avanzar en el TFG.

Afortunadamente, tras una búsqueda por la web, se ha encontrado un modelo entrenado, no solo con el corpus en alemán de Common Voice de Mozilla, sino también con su homólogo en inglés mediante la técnica de *transfer learning* que permite DeepSpeech [53], y los corpus de Spoken Wikipedia Corpora [45], Common Voice de Mozilla [21], M-AILABS Speech

Dataset[14], TUDA[15] y Voxforge[16], los cuales junto a los de Mozilla suman un total de 3 172 horas de audio validadas.

[14] El primer gran conjunto gratuito y libre para reconocimiento y síntesis de habla [46].
[15] Modelos acústicos de código abierto para el reconocimiento distante del habla, provisto por la universidad de Hamburgo [44].
[16] Un corpus libre adicional en alemán cedido por el usuario kmaclean [43].