Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA
DE VALLADOLID

Máster en Ingeniería Informática

Estudio de la conversión texto a voz basada en DNN: modelo base y fine-tuning

---

Alumno: Irene Peñas Pérez

Tutores: Valentín Cardeñoso Payo
David Escudero Mancebo

Fecha: Septiembre 2023

Estudio de la conversión texto a voz basada en DNN: modelo base y fine-tuning

Irene Peñas Pérez

Septiembre 2023

A mis padres, mi hermano, mis abuelos,
mis tíos y mis amigos,
que me han apoyado, animado,
y ayudado en todo momento. Que han creído en mí y
a los que siempre estaré agradecida.

No te rindas, por favor no cedas,
aunque el frío queme,
aunque el miedo muerda,
aunque el sol se esconda y se calle el viento,
aun hay fuego en tu alma,
aun hay vida en tus sueños,
porque la vida es tuya y tuyo tambien el deseo,
porque lo has querido y porque te quiero.

"No te rindas", Mario Benedetti

Usa tu sonrisa para cambiar el mundo,
no dejes que el mundo cambie tu sonrisa.

La magie consiste à rêver et à croire
que rien n´est impossible.

Tu mettici il tuo meglio,
e allora darai luce anche a tutto il resto.

Working hard is important.
But there's something that matters even more.
Believing in yourself.

"Harry Potter", J.K Rowling

"Gib niemals auf".

Agradecimientos

Querría agradecer en primer lugar a mis tutores, Valentín Cardeñoso Payo y David Escudero Mancebo por la oportunidad que me dieron de poder hacer primero las prácticas y después el TFM en el grupo de investigación ECA-SIMM. Además quiero darles las gracias por toda la ayuda que me han brindado, la disponibilidad y flexibilidad que han tenido para hacer las tutorías y por toda la paciencia y el tiempo que han dedicado al proyecto. Muchas gracias por todo. La síntesis de voz es un área de investigación en constante evolución, y que está siendo, en la actualidad un campo de investigación para las DNN generativas. Además me gustaría dar las gracias al profesor Alfonso Gutiérrez, de la UVa del campus de Segovia, por habernos prestado dos *datasets* para probar los modelos.

También me gustaría agradecer a mis padres, Mariano y Alicia, a mi hermano Mariano, a mis abuelos, a mis tíos y a mis amigos, porque han estado a mi lado en todo momento, apoyándome y animándome siempre, porque siempre han pensado que sería capaz de lo que me propusiera, incluso las veces que yo lo veía muy difícil. Pero como me dicen: "Nada es imposible".

Sin todos ellos no habría sido posible llegar hasta donde he llegado. Muchas gracias de corazón.

Resumen

La síntesis de voz es un área de investigación en constante evolución, y que está siendo, en la actualidad, un campo de investigación para las DNN generativas. En este trabajo se aborda la necesidad de desarrollar un sistema de síntesis de voz en español para superar las limitaciones lingüísticas que existen en este campo en el idioma español y tratar de mejorar la accesibilidad como puede ser en los asistentes virtuales. El objetivo del TFM se centra en explorar el uso de técnicas neuronales de última generación para crear un modelo base español, a partir de un conjunto de datos en castellano. Más tarde, se procede a optimizar, para después realizar un *fine-tuning* con otro conjunto de datos nuevo, obteniendo de esta manera una serie de modelos en español. Por último, se evalúan y se extraen una serie de conclusiones. Para la consecución de este objetivo, se hace uso de la herramienta NeMo. De esta manera, se crea un modelo base español utilizando *FastPitch* y HiFiGAN. Además se dispone de 3 conjuntos de datos diferentes para realizar los modelos y los consecuentes experimentos. Se evalúan las señales sonoras generadas por los diferentes modelos, tanto el base, como los *finetuned* y más tarde se hacen dos evaluaciones, una objetiva con un conjunto de métricas, y otra perceptual, en la que se pregunta a una serie de personas sobre la calidad e inteligibilidad de los audios. En conclusión, mediante este trabajo se aborda la necesidad imperante de desarrollar conjuntos de datos y sistemas de síntesis de voz en español para superar las limitaciones lingüísticas y mejorar la accesibilidad en aplicaciones como los asistentes virtuales en castellano.

Abstract

Speech synthesis is an area of research in constant evolution, and is currently a field of research for generative DNN. The aim of the Master's Dissertation is to explore the use of state-of-the-art neural techniques to create a Spanish base model from a Spanish dataset. Later, we proceed to optimize, and then perform a fine-tuning with another new dataset, obtaining in this way a series of models in Spanish. Finally, they are evaluated and a series of conclusions are drawn. In order to achieve this objective, the NeMo tool is used. In this way, a Spanish base model is created using FastPitch and HiFiGAN. In addition, three different datasets are available to perform the models and the consequent experiments. The sound signals generated by the different models, both the base and the finetuned, are evaluated and later two evaluations are made, an objective one with a set of metrics, and a perceptual one, in which a series of people are asked about the quality and intelligibility of the audios. In conclusion, this work addresses the imperative need to develop datasets and speech synthesis systems in Spanish to overcome linguistic limitations and improve accessibility in applications such as virtual assistants in Spanish.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. La síntesis de voz y los sistemas DNN

Para poner en contexto el tema sobre el que se cimenta el Trabajo Fin de Máster, hay que remontarse a varios siglos atrás. La historia de la síntesis de texto comienza su andadura a principios de siglo XVIII, en 1779, Christian Kratzenstein construyó modelos que podían reproducir los sonidos de las cinco vocales. Años más tarde, se inventó la *Wolfgang von Kempelen's Speaking Machine*, operada por fuelles. Esta máquina incluía modelos de labios y lengua, permitiendo producir tanto consonantes como vocales. En 1837, *Charles Wheatstone* creó una máquina parlante y en 1857 *M. Faber* construyó la máquina *Euphonia*. En la década de 1930, los laboratorios Bell desarrollan el *vocoder*, el cual automáticamente analizaba el habla por medio de su nota fundamental y resonancias. Partiendo del *vocoder*, *Homer Dudley* inventó un sintetizador operado por teclado, *The Voder* A finales de 1940 y hasta finales de 1950 se construye el *Pattern playback*. Esta máquina convierta imágenes de patrones acústicos del habla, los comúnmente conocidos como espectrogramas, en sonido. En los años 80 y 90, los sistemas dominantes eran los *DECtalk*, que hacían uso de los métodos de procesamiento de lenguajes naturales. Los primeros sintetizadores de habla sonaban robóticos y tenían poca inteligibilidad. Sin embargo, la calidad del habla sintetizada va mejorando cada vez más, aunque el audio sintético sigue diferenciándose del humano. Los sintetizadores del habla cada vez son más baratos y accesibles para todo el mundo, por lo que mediante este sistema se podrían beneficiar muchas personas, que por ejemplo, tengan problemas a la hora de hablar.

La síntesis del habla (*Text To Speech* TTS) es la tecnología que permite producir artificialmente el habla humana. Mediante dicha tecnología, se puede recibir un texto y reproducirlo con una voz artificial sintetizada [30]. En la Figura 1.1, se puede observar el proceso de conversión del texto en habla [2].

El habla sintetizada puede realizarse mediante la combinación de fragmentos de habla previamente grabados, los cuales se almacenan en una base de datos. Se pueden distinguir diferentes

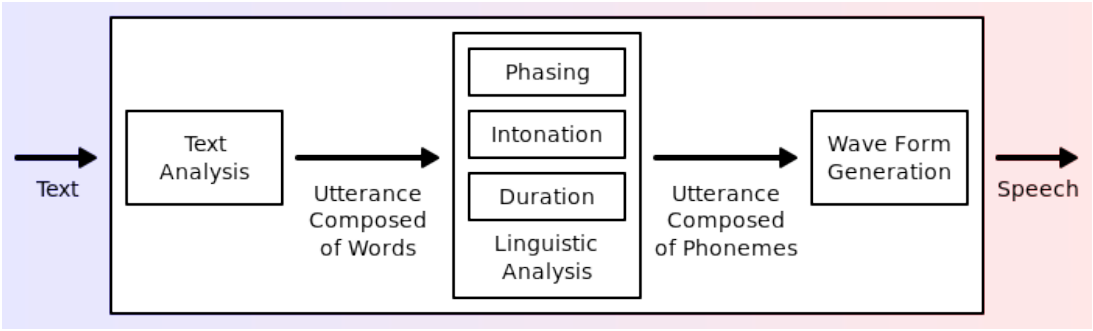


Figura 1.1: Funcionamiento de la síntesis del habla [2]

tipos de sistemas de síntesis del habla basados en el tamaño de las unidades de habla almacenadas:

- **Sistemas basados en fonos y difonos**: se almacenan unidades de habla a nivel de fonos (sonidos individuales del habla) o difonos (pares de fonos consecutivos). Esto permite un mayor rango de sonidos y una mayor flexibilidad para crear diferentes palabras y frases. Sin embargo, debido a que las unidades son muy pequeñas, la calidad de audio puede ser más baja y la síntesis carecer de claridad.
- **Sistemas basados en palabras completas u oraciones**: se almacenan unidades de habla a nivel de palabras completas u oraciones. Esto permite una mayor calidad de audio y una mayor claridad en la síntesis del habla, ya que se utilizan grabaciones de palabras o frases completas, que son más naturales y suenan más fluidas.
- **Sistemas con modelos de tracto vocal**: representaciones computacionales del tracto vocal humano, para recrear una voz sintética. Estos modelos pueden capturar características específicas de la voz humana, como la entonación, la resonancia y otros aspectos del tracto vocal, lo que permite una síntesis del habla más realista y personalizada.

1.2. Objetivos del trabajo

Primeramente se va a poner en contexto el entorno ideal, la realidad existente, las consecuencias que supone la realidad y por último la propuesta que se va a realizar. El entorno ideal es un entorno operativo en el que se aborde una revisión sistemática de las diferentes técnicas basadas en *Deep Learning* existentes, aplicadas al desarrollo de un TTS en español, además de la consecución de un *fine-tuning* para *voice-adaptation*. La realidad, es que a día de hoy existen modelos de TTS que generan audios a partir de textos, particularmente en entornos de habla en inglés. Sin embargo, en el caso del español, aunque se han realizado avances, aún existen limitaciones y desafíos en la creación de un TTS completamente funcional y de alta calidad. TTS en español enfrenta obstáculos relacionados con la variabilidad lingüística y acentos regionales, así como la

disponibilidad limitada de conjuntos de datos de calidad en español para entrenar los modelos de TTS. Si este problema no se soluciona, supondrá que se limitaría el potencial de aplicaciones y servicios que se benefician de la síntesis de voz, como asistentes virtuales, audiolibros, enseñanza asistida por voz y muchos otros campos en los cuales, TTS es muy valiosa.

La propuesta es desarrollar un modelo base de voz para TTS en español, de buena calidad y rendimiento. Para ello se emplearán DNN(*Deep Neural Network*). A partir de dicho TTS funcional se hará un *fine-tuning*, por lo que los pesos del modelo preentrenado se entrenan con datos nuevos. Actualmente se tienen una serie de audios en castellano con sus correspondientes transcripciones.

A continuación se exponen los objetivos del trabajo:

- OBJ-1: Se estudiará el entorno relacionado con TTS y DNN.
- OBJ-2: Se hará un análisis de las características y el funcionamiento de TTS, además de ver los algoritmos principales.
- OBJ-3: Mediante una serie de datos de audio con sus respectivas transcripciones, utilizando un modelo base de TTS, se creará un TTS español.
- 0BJ-4: Se realizará un *fine-tuning* del modelo base TTS español, con un *dataset* nuevo también en castellano.
- OBJ-5: Se evaluarán las voces resultantes tanto del modelo base TTS en español como del modelo con *fine-tuning*.

1.3. Estructura de la memoria

En esta sección se va a detallar cómo se ha organizado el documento, para que su posterior lectura sea más fácil.

- **Capítulo 1. Introducción.** En el primer capítulo se describe la motivación y los objetivos del TFM, además de la estructura del documento.
- **Capítulo 2. NeMo y Síntesis de Voz.** En el siguiente capítulo se detallan las metodologías utilizadas con los últimos avances.
- **Capítulo 3. Metodología.** En este capítulo se explica la creación de un TTS, desde el funcionamiento de una TTS, hasta el *fine-tuning* del modelo.
- **Capítulo 4. Experimentos y resultados.** En este capítulo se va a estudiar cómo de bien funcionan los diferentes modelos, haciendo una serie de comparaciones entre ellos, y obteniendo un conjunto de resultados.

- **Capítulo 5. Discusión y conclusiones.** En este capítulo se expondrán los resultados de todo el proyecto, analizando los aspectos más importantes y haciendo una recapitulación del TFM.

Capítulo 2

NeMo y Síntesis de Voz

2.1. Servicios LLM NeMO

NeMo *(**Ne**ural **Mo**dules)*, es un conjunto de herramientas creadas por NVIDIA para construir nuevos modelos de IA [16], con el fin de simplificar el desarrollo de modelos de inteligencia artificial para tareas de procesamiento del habla y del lenguaje. NeMo puede utilizarse para entrenar un nuevo modelo o realizar aprendizaje por transferencia sobre modelos ya entrenados. Para ello, dispone de colecciones para una serie de modelos, entre ellas, síntesis de texto a voz (TTS). Cada colección consiste en un conjunto de módulos pre-construidos y reutilizables (módulos neuronales), los cuales incluyen todo lo necesario para entrenar en el *dataset* que se desee. Cada módulo puede ser personalizado, ampliado y compuesto fácilmente para crear nuevas arquitecturas. Los módulos neuronales son bloques conceptuales de RRNN que toman entradas tipificadas y producen salidas tipificadas. Estos módulos suelen representar capas de datos, codificadores, decodificadores, modelos lingüísticos, funciones de pérdida o métodos de combinación de activaciones.

A continuación se van a explicar las herramientas de las que se dispone con la finalidad de utilizar NeMo para TTS.

La síntesis de texto a voz (TTS) se refiere a un sistema que convierte entradas de texto en habla humana natural, la cual suene inteligible y natural. La implementación de NeMo se centra en el TTS neural de última generación. Existen dos aproximaciones:

- TTS en cascada: proceso de tres etapas:

    1. Análisis del texto: translitera los grafemas de entrada en fonemas, bien buscándolos en un diccionario canónico o utilizando una conversión de grafema a fonema (G2P);
    2. Modelado acústico o Mel-espectrograma: genera características acústicas a partir de las entradas de fonemas o de una mezcla de grafemas y fonemas. NeMo opta por los mel-espectrogramas para representar características acústicas expresivas

3. *Vocoder*: sintetiza los audios en forma de onda a partir de las características acústicas correspondientes. Ejemplo: espectrograma.

- TTS de extremo a extremo: integra las tres etapas anteriores en un único modelo, es decir, sintetiza directamente los audios a partir de las entradas de grafemas/fonemas sin procesos intermedios.

Los modelos que se pueden implementar para la generación del Mel-Espectrograma son los siguientes: FastPitch [15], Mixer-TTS/Mixer-TTS-X [11, 33], RAD-TTS [29], Tacotron [31], Tacotron2 [28], WaveNet [22], Deep Voice [4], Deep Voice2 [3], etc.

Se plantea utilizar el generador de mel-espectrograma FastPitch, que es un modelo de síntesis de texto a voz, basado en *FastSpeech*. Antes de explicar FastPitch, se procederá a explicar *FastSpeech* para saber en qué modelo está basado y sus características. Después de realizar el modelado acústico con FastPitch, se detallará el *vocoder* HiFiGAN.

Tanto el generador de mel-espectrograma FastPitch, como FastSpeech y HiFiGAN quedan detallados a continuación:

2.1.1. Descripción y funcionamiento de *FastSpeech*

En los últimos años, la síntesis de voz a partir de texto (TTS) ha sido objeto de un mayor interés gracias a los avances en el aprendizaje profundo. Los sistemas basados en redes neuronales profundas se han vuelto cada vez más populares en TTS, incluyendo modelos como Tacotron y Tacotron 2 [5, 10, 12], Deep Voice 3 y ClariNet. Estos modelos generalmente siguen un proceso en el que primero generan un mel-espectrograma[1] de manera secuencial a partir de una entrada de texto y luego utilizan *vocoders* como Griffin-Lim, WaveNet, Parallel WaveGAN o WaveGlow para convertir ese mel-espectrograma en habla. Los sistemas de TTS basados en redes neuronales han demostrado ser superiores en términos de calidad de habla en comparación con los enfoques convencionales y paramétricos estadísticos.

Sin embargo, los sistemas de TTS actuales que generan mel-espectrogramas de manera secuencial enfrentan varios desafíos, como la lentitud en la generación de mel-espectrogramas, problemas de calidad en la síntesis de habla y la dificultad de controlar aspectos como la velocidad de la voz y la prosodia.

Para abordar estos inconvenientes, se propone *FastSpeech* [20], un modelo innovador que genera mel-espectrogramas de manera simultánea a partir de una secuencia de texto (fonemas). *FastSpeech* utiliza una red de alimentación directa basada en la auto-atención del *Transformer* y la convolución 1D. Para solucionar el problema de la diferencia en longitud entre las secuencias de fonemas y mel-espectrogramas, *FastSpeech* incorpora un regulador de longitud que ajusta la duración de los fonemas para que coincida con la de los mel-espectrogramas. Este regulador se basa en un predictor de duración de fonemas.

[1] Es un espectrograma que utiliza la escala MEL de frecuencias, más próxima a la escala perceptual humana.

FastSpeech afronta los desafíos mencionados anteriormente de la siguiente manera: Mediante la generación simultánea de mel-espectrogramas, *FastSpeech* acelera significativamente el proceso de síntesis. El predictor de duración de fonemas garantiza alineaciones precisas entre fonemas y sus mel-espectrogramas, evitando problemas de propagación de errores y alineaciones incorrectas, lo que reduce la cantidad de palabras omitidas o repetidas en la síntesis. El regulador de longitud permite ajustar fácilmente la velocidad de voz y controlar parte de la prosodia al modificar la duración de los fonemas. En experimentos con el conjunto de datos LJSpeech, *FastSpeech* muestra una calidad de habla cercana a la de los modelos autorregresivos de *Transformer*; pero con una aceleración de hasta 270 veces en la generación de mel-espectrogramas y 38 veces en la síntesis de habla final en comparación con los modelos auto-regresivos. Además, casi elimina el problema de omitir o repetir palabras y permite un control suave sobre la velocidad de voz. Esto hace que *FastSpeech* sea una solución prometedora para la síntesis de voz de alta calidad y velocidad. Para generar una secuencia de mel-espectrograma objetivo en paralelo, se diseña una novedosa estructura *feed-forward*, en lugar de utilizar la arquitectura basada en en codificador-atención-decodificador como la adoptada por la mayoría de generación autorregresiva basada en secuencia a secuencia y no autorregresiva. En la Figura 2.1, se muestra la arquitectura de *FastSpeech*.

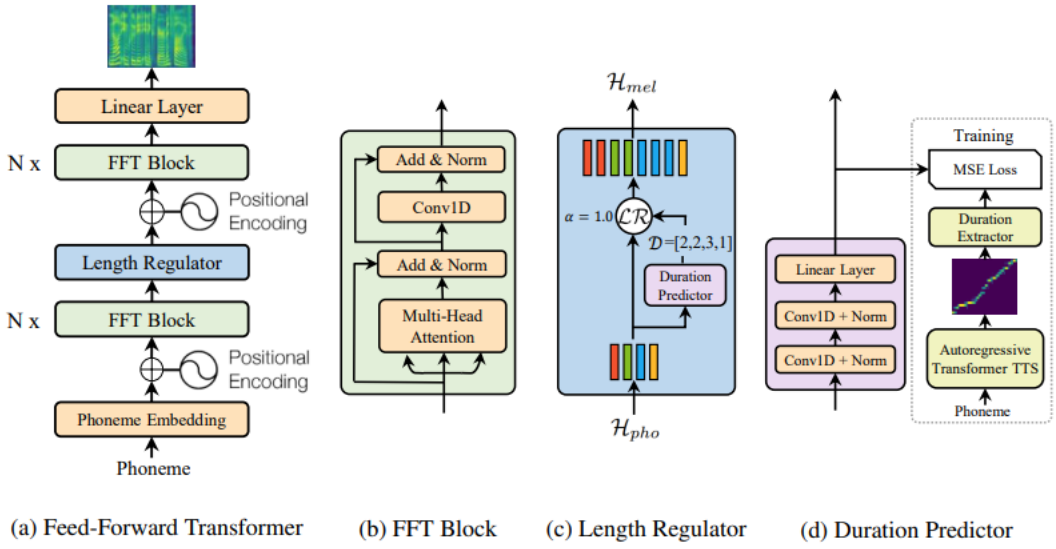


Figura 2.1: Arquitectura general de *FastSpeech*. (a) Transformador de Alimentación Directa (FFT) (b) El bloque de Transformador de Alimentación Directa. (c) El regulador de longitud. (d) El predictor de duración. [26]

Transformer de Alimentación Directa (FFT)

FastSpeech utiliza una estructura de alimentación directa basada en auto-atención en el estilo del Transformer y convolución 1D. Esta estructura se llama FFT *(Feed-Forward Transformer)* y consta de múltiples bloques FFT para transformar fonemas en mel-espectrogramas. Cada bloque

FFT incluye auto-atención y una red de convolución 1D. A diferencia del *Transformer* tradicional, *FastSpeech* utiliza una red de convolución 1D en lugar de una red densa de 2 capas. Se agregan conexiones residuales, normalización de capa y *dropout* siguiendo las redes de autoatención y convolución 1D.

Regulador de Longitud

El regulador de longitud se utiliza para abordar la discrepancia en longitud entre las secuencias de fonemas y espectrogramas en el FFT. También controla la velocidad de voz y parte de la prosodia. Ajusta la longitud de las secuencias de fonemas para que coincida con la de los espectrogramas en función de la duración de los fonemas. Se utiliza un parámetro α para controlar la velocidad de voz, y se pueden controlar las pausas entre palabras ajustando la duración de los caracteres de espacio en la oración.

Predictor de Duración

La predicción de la duración de los fonemas es esencial para el regulador de longitud. Este predictor consta de una red de convolución 1D de 2 capas con activación ReLU, normalización de capa y una capa lineal adicional para predecir la duración de los fonemas en el dominio logarítmico. El predictor se entrena junto con el modelo *FastSpeech*para predecir la longitud de los espectrogramas mel para cada fonema utilizando la pérdida de error cuadrático medio (MSE). Para entrenar el predictor de duración, se extrae la duración real de los fonemas de un modelo autoeregresivo de habla previamente entrenado. Esto se hace calculando las alineaciones de atención entre el codificador y el decodificador del modelo autoeregresivo y seleccionando la alineación que mejor se ajuste a la diagonal. Luego, se extrae la duración de los fonemas en función de estas alineaciones de atención.

2.1.2. Descripción y funcionamiento del FastPitch

FastPitch es un modelo *feed-forward*, basado en FastSpeech, sobre el cual se puede profundizar en la Subsección 2.1.1. FastPitch mejora la calidad del habla sintetizada. Al condicionar la frecuencia fundamental estimada para cada símbolo de entrada (contorno de tono), se equipara a los modelos autorregresivos. FastPitch no introduce sobrecarga, y FastPitch conserva la favorable arquitectura totalmente paralela de *transformers*, con un factor de tiempo real superior a 60x para la síntesis mel-espectrograma de un enunciado típico. El modelo aprende a predecir y utilizar el tono en una resolución baja de un valor por cada símbolo de entrada, facilita el ajuste interactivo del tono. El desplazamiento constante de F0 con FastPitch produce variaciones de tono de voz graves y agudas de sonido natural que preservan la identidad percibida del hablante. El modelo aprende a imitar la acción de las cuerdas vocales. Por lo tanto, el condicionamiento en función de

los contornos de frecuencia mejora la calidad general del habla sintetizada, haciéndola comparable a los modelos punteros.

Descripción del modelo FastPitch

- Dos pilas de *feed-forward Transformer (FFTr)* (transformadores de alimentación directa). El primer FFTr opera en la resolución de los *tokens* de entrada, y el otro en la resolución de los *frames* de salida. La primera pila FFTr produce la representación oculta, la cual se utiliza para realizar predicciones sobre la duración y la media del tono de cada letra, con una 1-D CNN. Seguidamente, se proyecta la tonalidad para que coincida con la dimensionalidad de la representación oculta. Por último, el resultado se aumenta discretamente y se envía a la red neuronal de salida FFTr, la cual produce la secuencia de mel-espectrograma de salida.
- *Pitch Predictor*: se utiliza para establecer la entonación o el tono de la voz en la síntesis de habla. Ayuda a determinar cómo deben variar las frecuencias fundamentales (F0) a lo largo del tiempo en el discurso generado.
- *Duration Predictor*: responsable de predecir las duraciones de los diferentes fonemas o unidades de habla en el discurso generado. De gran importancia para controlar la velocidad y la duración del habla sintetizada.
- Conv 1D: capa de convolución unidimensional, comúnmente el procesamiento de señales y en las redes neuronales para extraer características de los datos de entrada.
- FC: capas completamente conectadas en la red neuronal. Estas capas conectan cada neurona de una capa con todas las neuronas de la capa siguiente.
- MSE Loss: función de pérdida que se utiliza para optimizar los modelos de *Pitch Predictor* y *Duration Predictor*.
- *Embedding*:representación numérica de características importantes extraídas del audio durante el proceso de análisis. Se pueden utilizar para comparar y evaluar la autenticidad o calidad del audio en etapas posteriores del sistema.

Llegados a este punto, los *vocoders* que se pueden utilizar para sintetizar los audios son: HiFiGAN, UnivNet y WaveGlow. Finalmente se decidió escoger el *vocoder* HiFiGAN, debido a que logra una síntesis de voz eficiente y de alta fidelidad [13].

2.1.3. Descripción y funcionamiento de HiFiGAN

En los últimos tiempos, con el desarrollo de las redes neuronales, la tecnología de síntesis de voz ha experimentado un rápido progreso. La mayoría de los modelos neuronales de síntesis

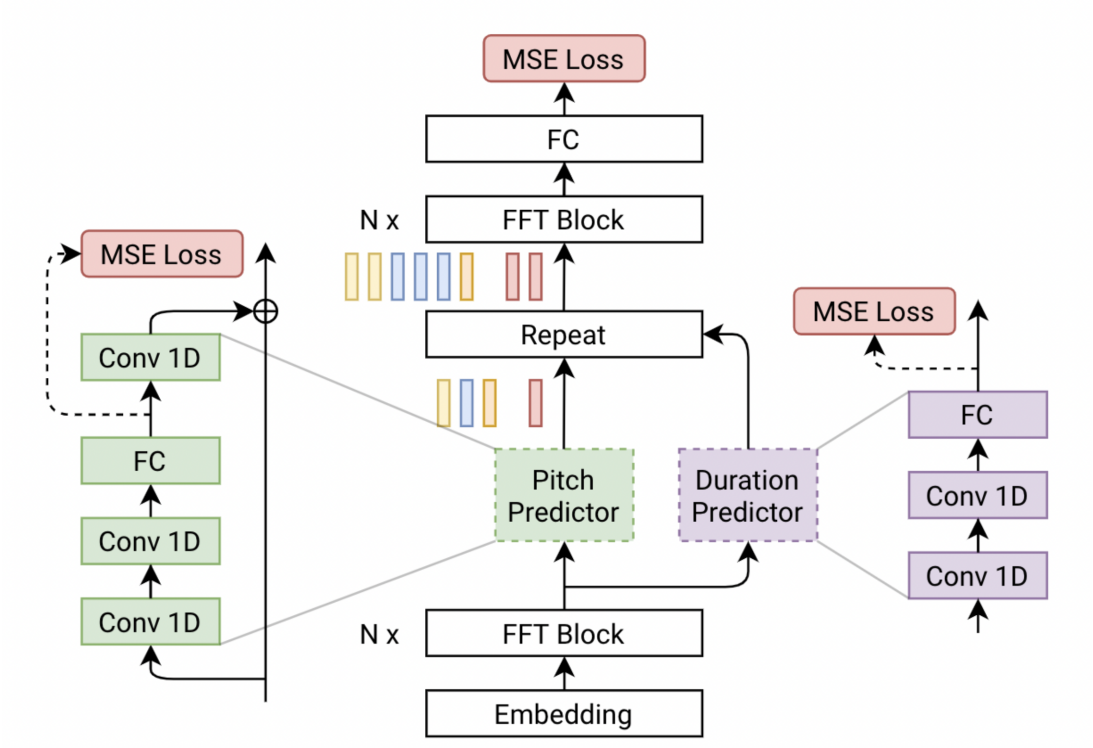


Figura 2.2: Funcionamiento de FastPitch [20][10]

de voz siguen un proceso de dos etapas: 1) la predicción de una representación intermedia de baja resolución, como mel-espectrogramas o características lingüísticas, a partir del texto. 2) La síntesis de formas de onda de audio a partir de esta representación intermedia. Se han realizado diversos esfuerzos previos para mejorar la calidad de la síntesis de audio y la eficiencia de los modelos de segunda etapa. Sin embargo, existen desafíos en términos de velocidad de generación y número de parámetros en estos modelos. En recientes investigaciones sobre síntesis de voz, se han empleado redes generativas adversarias (GANs) para crear formas de onda de audio en bruto. A pesar de mejorar la eficiencia de muestreo y el uso de memoria, la calidad de las muestras aún no se compara con la de modelos generativos autoeregresivos y basados en flujos. HiFi-GAN se enfoca en modelar patrones periódicos presentes en el audio, ya que la voz contiene señales sinusoidales con diferentes periodos. Algunas evaluaciones subjetivas (MOS) indican que HiFi-GAN produce audio de alta calidad a una velocidad 167.9 veces más rápida que el tiempo real en una sola GPU V100.

HiFi-GAN se centra en el diseño de un modelo de *vocoder* que sintetiza eficientemente audios de forma de onda en bruto, a partir de los mel-espectrogramas intermedios.

El *vocoder* consta de un generador y dos discriminadores (multiescala y multiperiodo). El generador y los discriminadores se entrenan de forma adversaria junto con dos pérdidas adicionales para mejorar la estabilidad del entrenamiento y el rendimiento del modelo [21].

- Generador: CNN que toma un espectrograma mel como entrada y lo muestrea mediante convoluciones transpuestas hasta que la longitud de la secuencia de salida coincide con la resolución temporal de las formas de onda en bruto. A cada convolución transpuesta le sigue un módulo de fusión de campos multirreceptivos (MRF). La arquitectura del generador se muestra en la Figura 2.3, destacando de esta los siguientes aspectos: El generador sobremuestrea los mel-espectrogramas hasta $|k_u|$ veces para ajustarse a la resolución temporal de las formas de onda sin procesar. Un módulo MRF añade características de $|k_r|$ bloques residuales de diferentes tamaños de núcleo y tasas de dilatación. Por último, se representa el n-ésimo bloque residual con tamaño de núcleo $k_r[n]$ y tasas de dilatación $D_r[n]$ en un módulo MRF.

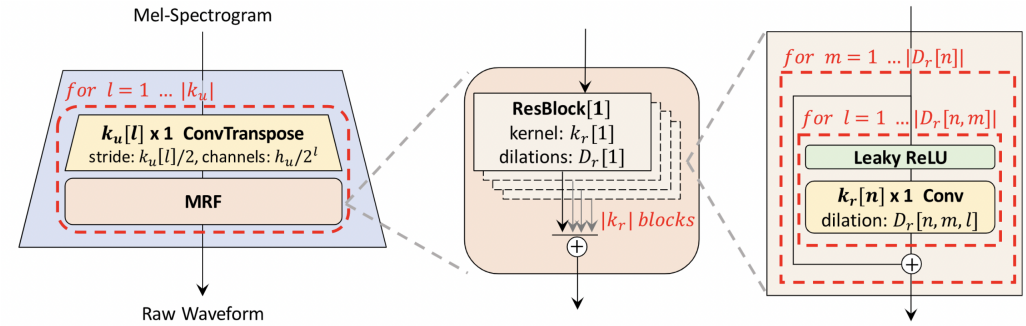


Figura 2.3: Arquitectura del Generador de HiFiGAN

A continuación se procede a explicar la estructura del discriminador, la cual se puede observar en la Figura 2.4.

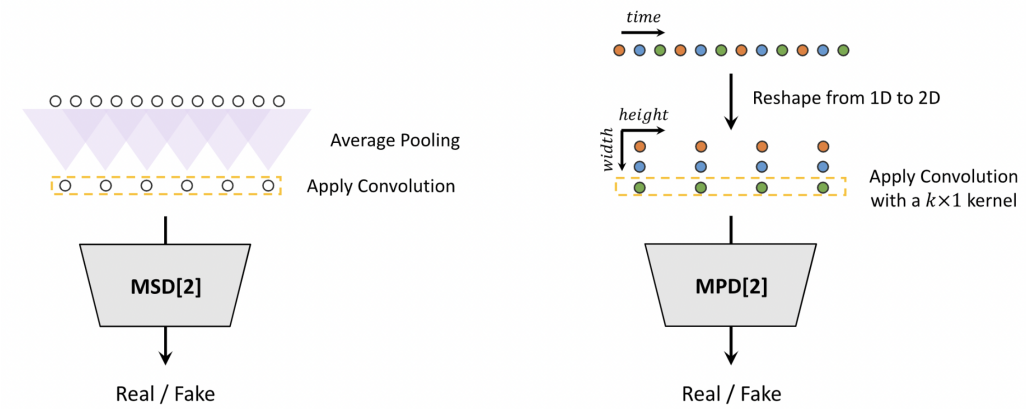


Figura 2.4: Arquitectura del Discriminador de HiFiGAN

- *Multiperiod Discriminator* (MPD): El discriminador multiperiodo se puede entender como un conjunto de discriminadores que operan en formas de onda submuestreadas, ya sean reales o generadas. La elección cuidadosa del factor de submuestreo (por ejemplo, 2, 3,

5, 7, 11) garantiza que cada discriminador observe patrones únicos en estas formas de onda submuestreadas. Para evaluar la efectividad del discriminador, se realizan pruebas en tareas de generación de formas de onda a partir de espectrogramas mel correspondientes, conocidas como tareas de inversión de espectrogramas mel [13]. Este componente consta de varios "subdiscriminadores" que trabajan juntos para evaluar un audio de entrada [1].

- Subdiscriminadores: Cada subdiscriminador es responsable de analizar una parte específica del audio de entrada. Estas partes sólo aceptan muestras igualmente espaciadas del audio de entrada. Cada subdiscriminador se especializa en buscar patrones o características particulares en su segmento asignado.
- Capturar estructuras implícitas diferentes: Cada subdiscriminador está diseñado para identificar estructuras o características diferentes en su segmento de audio. Esto significa que cada subdiscriminador puede detectar patrones únicos que pueden no ser evidentes en otras partes del audio.
- Mezclador de subdiscriminadores: Después de que cada subdiscriminador haya realizado su análisis en su segmento respectivo, sus resultados se combinan o se "mezclan" de alguna manera en el MPD. Esta mezcla puede implicar la consideración de las evaluaciones individuales de los subdiscriminadores para tomar una decisión global sobre la calidad o autenticidad del audio.

- *Multi Scale Discriminator* (MSD): Dado que cada sub-discriminador en MPD acepta muestras no superpuestas, se agrega el discriminador multiescala para evaluar consecutivamente la secuencia de audio. La arquitectura de MSD se basa en la de MelGAN y consta de tres sub-discriminadores que operan en diferentes escalas de entrada: audio en bruto, audio promediado $\times 2$ y audio promediado $\times 4$. Cada uno de los sub-discriminadores en MSD es una pila de capas de convolución estratificada y agrupada con activación ReLU con fuga. El tamaño del discriminador aumenta reduciendo el paso y agregando más capas. Se aplica normalización de pesos excepto para el primer sub-discriminador, que opera en el audio en bruto. En su lugar, se aplica la normalización espectral. Es importante destacar que MPD opera en muestras no superpuestas de formas de onda en bruto, mientras que MSD opera en formas de onda suavizadas. El discriminador podría lograr tanto una mayor eficiencia computacional como una mayor calidad de muestra que los mejores modelos autorregresivos o basados en flujo disponibles públicamente, como WaveNet y WaveGlow.

2.2. Servicios NeMo para la síntesis de voz

Después de estudiar y realizar la descripción del modelo a desarrollar, se avanza a la fase de *fine-tuning*. El *fine-tuning* es una técnica de entrenamiento que consiste en la reutilización de arquitecturas CNN predefinidas y preentrenadas. Por lo tanto, permite escoger un modelo

entrenado que realiza bien una determinada tarea y aprovechar todo sus conocimientos para resolver una nueva tarea específica [34].

El *fine-tuning* trata de un proceso en el que se realiza un ajuste fino (se ajustan ligeramente ciertas representaciones del modelo preentrenado para adaptarlo al problema) sobre algunas capas de la red para obtener las salidas deseadas. Mediante esta técnica, se evita tener que definir la estructura de la red neuronal y entrenarla desde cero, lo que podría requerir muchos esfuerzos en tiempo y recursos.

En la Figura 2.5 se puede ver una comparación de las estrategias tanto realizando *fine-tuning* como sin él.

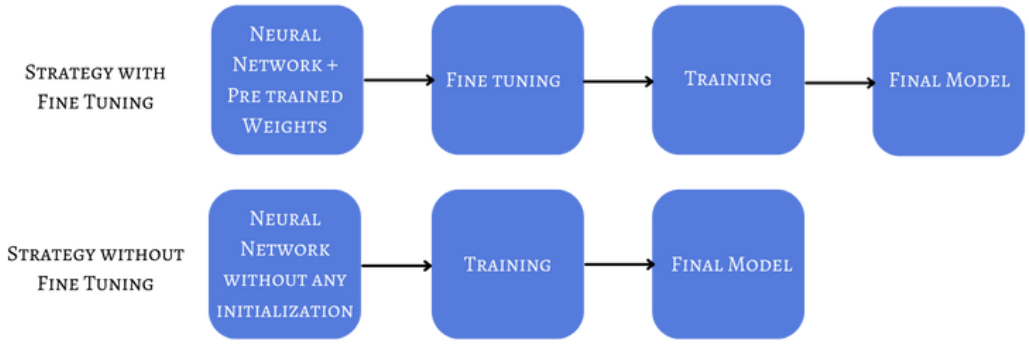


Figura 2.5: Ejemplo del funcionamiento del modelo [34]

2.2.1. Pasos a seguir para realizar el *fine-tuning*

Los pasos que se realizan para seguir el proceso del *fine-tuning* son los siguientes:

- Se añaden y quitan capas al modelo existente para adaptarlo a la nueva tarea a realizar.
- En la nueva estructura del modelo, sólo se congelan las capas de la red original cuyo conocimiento debe conservarse para el nuevo entrenamiento.
- El modelo se entrena con el nuevo *dataset* para la nueva tarea. Sólo se actualizan los pesos de las nuevas capas.

En el caso de NeMo, el proceso de *fine-tuning* es como sigue: Todos los *scripts* TTS permiten un ajuste fácil mediante la carga parcial o total de los pesos preentrenados de un punto de control en el modelo instanciado en ese momento. Los pesos preentrenados pueden ser proporcionados de diferentes maneras:

- Proporcionando una ruta a un modelo NeMo.
- Proporcionando un nombre de un modelo NeMo preentrenado.

- Proporcionar una ruta a un archivo de punto de control de *Pytorch Lightning*.

Existen varios scripts para realizar ajustes del modelo TTS.

2.2.2. Medición de la calidad del resultado

Medir la calidad del resultado de un modelo de TTS tiene una cierta complejidad. Existen métricas basadas en la distancia. Sin embargo, son pobres porque solo miden la similitud con el conjunto de pruebas, y no el realismo del discurso generado. Es por esto que, la mayoría de los artículos de TTS se basan en las puntuaciones medias de opinión para informar la calidad del modelo (MOS *(Mean opinion score)*). Realizar MOS involucra a personas, lo que significa que es costoso y consume mucho tiempo. Más importante aún, no se puede utilizar durante el entrenamiento para evaluar el rendimiento en tiempo real de un modelo.

Para este proyecto, primero se investigó sobre las técnicas CLVP. El campo de generación de imágenes se ha establecido en el uso de las métricas *Fréchet Inception Distance* y *Inception Score* para medir el rendimiento en vivo [19]. Por otro lado, en este proyecto, se va a utilizar la librería *tts-scores*, la cual ofrece una modernización de estas técnicas [6]. Para realizar dicha mejora, crea un Modelo contrastivo de lenguaje y voz preentrenado (CLVP). Para ello, se entrena una arquitectura similar a CLIP *(Contrastive Language-Image Pre-Training)* con algunos cambios: en lugar de medir la similitud entre texto e imágenes, mide la similitud entre texto y clips de voz.

A partir de esta arquitectura, la herramienta ofrece dos métricas:

- *CLVP Score*: mide la distancia predicha por CVLP entre el texto y un clip de audio en el que se habla ese texto. Una puntuación más baja es mejor.

- *CLVP Fréchet Distance*: compara la distribución del texto hablado real con lo que genera el modelo TTS. Resulta especialmente útil si se tiene mucho texto hablado con el que se quiere comparar pero no hay transcripciones. Funciona calculando la distancia de *Fréchet* de las salidas de la última capa del modelo CLVP cuando se alimentan datos de ambas distribuciones. Una puntuación más baja es mejor.

Sin embargo, finalmente se optó por utilizar las técnicas descritas en la Sección 3.9.1.

2.2.3. Descripción del proceso de adaptación de NeMo a un nuevo *speaker*

La adaptación de NeMo de NVIDIA a un nuevo hablante para la generación de voz a partir de texto implica los siguientes pasos:

- Tener a disposición un *dataset* en la lengua que se requiera: el primer paso es recopilar datos de voz del nuevo hablante. En este caso, se tiene un *dataset* español, con una voz diferente a aquella del corpus original.

- Entrenamiento del modelo base: se necesita contar con un modelo base previamente entrenado, que se entrena utilizando una gran cantidad de datos de voz. Durante este proceso, el modelo aprende a generar voz a partir de texto de manera general.
- Adaptación del modelo: se utiliza el nuevo corpus, modificando los ficheros de configuración del modelo base y entrenando para que se ajuste a su voz específica. Durante la adaptación, el modelo aprenderá las características distintivas del hablante y podrá generar voz que se asemeje más a su estilo y entonación.
- Evaluación y refinamiento: se verifica la calidad de la voz generada por el modelo adaptado, comparando la salida del modelo con las grabaciones originales. Se realizan ajustes si es necesario. Este proceso ayuda a refinar el modelo y mejorar su desempeño.
- Uso del modelo adaptado: el modelo está listo para generar voz a partir de texto para el nuevo hablante.

Capítulo 3

Metodología

En este capítulo se explicará la metodología utilizada, en este caso CRISP DM y la propuesta de trabajo de manera detallada.

3.1. Metodología utilizada: CRISP DM

CRISP-DM *(Cross Industry Standard Process for Data Mining)* es un marco estándar para proyectos de análisis de datos que se asemeja al ciclo de vida de desarrollo de software [32]. Esta metodología proporciona una estructura para comprender, preparar y utilizar datos con el objetivo de abordar problemas comerciales.

El proceso CRISP-DM consta de seis fases que pueden ser flexibles y cíclicas, lo que significa que se puede retroceder y avanzar según sea necesario. Estas fases son:

- Selección de datos y *features*: Aquí, se recopilan y exploran los datos para comprender su estructura y calidad. Más tarde, esos datos deben ser limpiados y transformados, para así adecuarlos al futuro modelo.
- Preparación de un modelo inicial: Se seleccionan y aplican técnicas de modelado para crear un modelo que resuelva el problema.
- Optimización de modelo: una vez obtenido el modelo, se procede a aplicar una serie de hiperpatámetros para optimizar el modelo y de esta manera encontrar el mejor resultado. Más tarde, con el modelo optimizado, se realiza el *fine-tuning* sobre FastPitch y HiFiGAN.
- Evaluación de resultados: el modelo se evalúa para determinar si cumple con el objetivo de negocio.

Además, CRISP-DM reconoce que los proyectos de análisis de datos no son lineales; pueden iterar y mejorar con el tiempo. Es esencial documentar todo el proceso y mantener una comunicación clara con los interesados.

3.2. Datasets

3.2.1. Descripción

Para la realización de un TTS se dispone de varios *dataset* o conjuntos de datos, que más tarde se usarán para implementar el modelo y realizar consecuentemente una serie de pruebas. El corpus utilizado para entrenar el modelo base se llama *Spanish Single Speaker Speech Dataset*, descargado de Kaggle [14]. Dicho *dataset* contiene 3 carpetas, pero por razones de tiempos de entrenamiento y memoria se han utilizado los audios de una sola de las carpetas, llamada *19de-marzo*. Dicho *dataset* consiste en una serie de grabaciones en español con sus correspondientes transcripciones. Más tarde, para la fase de *fine-tuning* se utilizarán dos *datasets*, llamados "datasetAlfonsoGutierrezFinetuning" y "datasetAlfonsoGutierrezFinetuning". Estos han sido cedidos por el profesor de la UVa Alfonso Gutiérrez, del campus de Segovia. Ambos conjuntos de datos son relativos a una clase de la universidad. Todos los *datasets* tienen que pasar por el proceso ETL .En el Apéndice C.2 se describe en profundidad.

3.2.2. Operativa

La operativa previa a la generación del modelo base son:

- Completar la instalación del entorno adecuado. En primera instancia, se utilizó Google Colab, pero surgieron una serie de problemas. Más tarde, se ha utilizado VSCode para acceder a una máquina, la cual contiene un contenedor Docker, en el cual se pueden realizar todas las pruebas y crear los diferentes modelos. En el Apéndice A se explican más detalles.
- Realizar el proceso de extracción, transformación y carga del *dataset*.
- Tener en el entorno todas las librerías que se van a utilizar en versiones que sean compatibles entre sí, para que no haya ningún impedimento a la hora de implementar la solución.
- Encontrar un modelo que se ajuste a las necesidades del proyecto. En este caso, se ha creado un modelo en español partiendo de cero.
- Realizar el *fine-tuning* del modelo.
- Crear una serie de modelos combinando diferentes parámetros.
- Evaluar los resultados de los modelos y compararlos entre ellos para ver cuál ofrece unas mejores métricas de rendimiento.
- Guardar el mejor modelo.

3.3. Adaptación de un modelo inglés al castellano

En una primera aproximación, se probó el modelo FastPitch con audios en inglés. Una vez comprobado que funcionaba, se decidió realizar un *fine-tuning* de dicho modelo, pero esta vez con un *dataset* en castellano. Dicho conjunto de datos, es descargado de Kaggle y el modelo FastPitch, se intenta en primera instancia adaptar el modelo inglés a un nuevo idioma, el castellano. Durante la realización de dicha adaptación, surgieron una serie de inconvenientes. El más serio fue un error que hacía que abortara la ejecución del entrenamiento. Se investigó su posible causa, y se concluyó que debía tratarse de un problema por intentar realizar el *fine-tuning* del inglés al español. En el Apéndice C.4 se describe en profundidad. Finalmente, esa idea fue descartada, y se optó por otro camino.

3.4. Creación de un modelo base español desde cero

Investigando sobre qué soluciones se podían encontrar, se optó por realizar un modelo base desde cero a partir de un ejemplo en alemán [18], en vez de realizar el *fine-tuning* en castellano. Los pasos que se han realizado para conseguir crear un modelo castellano a partir de un modelo base son:

1. En primer lugar, el proceso de extracción, transformación y carga de los datos (ETL) se automatizó en un cuaderno de Jupyter, en el cual se procede a realizar desde la descarga de los datos hasta la obtención de los tres ficheros .csv resultantes. En el Apéndice C.2 se describe en detalle los pasos dados en el cuaderno de Jupyter.

2. Una vez con el *dataset preparado*, se necesita un modelo base. El cuaderno en el que se cimenta el modelo español es el creado por Thorsten Müller, el cual utiliza FastPitch con HiFiGAN [18]. En este cuaderno se realiza el proceso desde la normalización del *dataset* hasta el entrenamiento y el *fine-tuning*. Para adaptar el modelo alemán al español, se han hecho una serie de cambios tanto en el cuaderno como en los ficheros de configuración. En el Apéndice C.5 se describe de manera detallada. En líneas generales, estos cambios sirven para realizar las siguientes tareas:

   - Obtener los ficheros normalizados de entrenamiento, validación y test.
   - Obtener una serie de estadísticas de tono, relevantes para el entrenamiento.

3. A continuación, se procede a ejecutar el código correspondiente al entrenamiento. Una vez completado este, se evalúa la calidad del modelo generado FastPitch.

4. Después de oír los audios resultantes, probando con 3 modelos diferentes (con diferente

tamaño de *batch*[1], siendo este 4, 8 y 12), como se puede observar en las Figuras 3.1, 3.2 y 3.3, se ve que la calidad del audio no es tan buena como debería. La prosodia en cambio sí lo es. Para mejorar la calidad del audio, se realiza un *fine-tuning* tanto de FastPitch como de HiFiGAN.

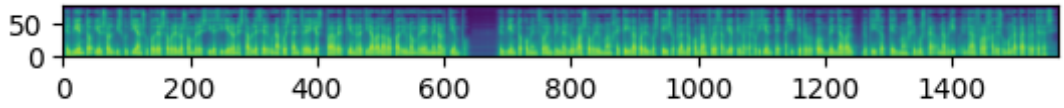


Figura 3.1: Espectrograma después del entrenamiento con tamaño de *batch* 4

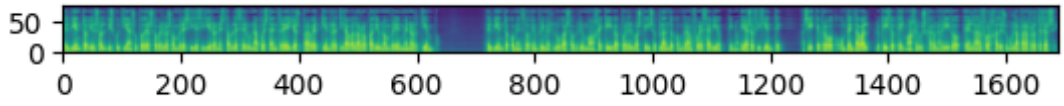


Figura 3.2: Espectrograma después del entrenamiento con tamaño de *batch* 8

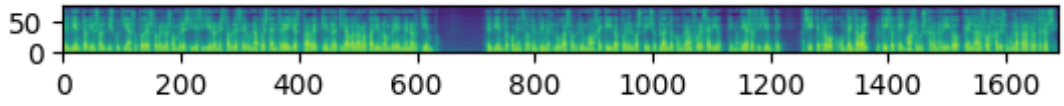


Figura 3.3: Espectrograma después del entrenamiento con tamaño de *batch* 12

3.5. *Fine-tuning* de FastPitch del modelo base español con un *dataset* nuevo

Después de obtener el modelo base castellano desde cero, se hizo un *fine-tuning* de dicho modelo, con el *dataset* "datasetAlfonsoGutierrezFinetuning". Una vez realizado todo el proceso, se pasa a realizar los pasos que siguen, los cuales se encuentran detallados en el Apéndice C.6:

1. Transformación y limpieza del *dataset*.
2. Extracción de los *manifest*.
3. Extracción de los datos suplementarios.
4. *Fine-tuning* del modelo base: en esta etapa se hicieron una serie de ejecuciones cambiando y combinando diferentes hiperparámetros. Los resultados obtenidos, medidos mediante diferentes métricas, mejoraron el modelo base.
5. Creación de los audios a partir del modelo al que se le ha realizado el *fine-tuning*.

[1]Número de ejemplos de entrada que se procesan simultáneamente en una sola iteración durante el entrenamiento o la inferencia de un modelo de TTS.

3.5.1. Transformación y limpieza del *dataset*

Se extraen los datos, si están comprimidos, y, más tarde, se realizan algunas transformaciones para que dicho fichero tenga el formato adecuado para poder entrenar. También se eliminan las filas nulas. Por último, se debe dividir el fichero en 3 csv: metadata_train, metadata_test y metadata_val.

3.5.2. Obtención de los ficheros normalizados

A continuación se llama a es_get_data.py, para obtener los ficheros train_manifest.json, test_manifest.json, val_manifest.json, train_manifest_text_normed.json,

test_manifest_text_normed.json y val_manifest_text_normed.json.

De esta manera, se obtienen los ficheros en formato .json, además de los ficheros normalizados, que más adelante se utilizarán para entrenar el modelo.

3.5.3. Conversión de audios mp4 a wav

En este apartado se explica cómo se realiza la conversión de los ficheros de audio del formato mp4 a wav. En un primer momento no se convirtieron, pero después se vio que había un problema al entrenar. Una vez ejecutado el código, el problema se solucionó.

3.5.4. Calculo de rangos de variación del pitch

Una vez se tienen los ficheros anteriores, es el momento de llamar al fichero *extract_ sup_ data.py*. De este fichero se va a extraer la siguiente información importante:PITCH_MEAN, PITCH_STD PITCH_MIN y PITCH_MAX. De estos valores, se escogerán tanto PITCH_MEAN como PITCH_STD, para el posterior entrenamiento. Aunque este paso es opcional, estos valores son muy útiles para acelerar y estabilizar el entrenamiento.

3.5.5. Entrenamiento del modelo

Después de ejecutar los diferentes *scripts*, tenemos los ficheros necesarios para poder realizar el entrenamiento, que son:

- train_manifest_text_normed.json: es el *dataset* de entrenamiento, que posee la siguiente estructura: ”audio_filepath“: audio ”duration“: duración, ”text“: texto, ”normalized_text“: texto normalizado
- val_manifest_text_normed.json: es el *dataset* de validación.
- sup_data: conjunto de archivos que servirán para acelerar y estabilizar el entrenamiento.

Ahora se va a llamar al *script* fastpitch_finetune.py, con el fichero de configuración fastpitch_align_44100.yaml.

Además, en el comando que se utiliza de Python, se establecen algunos parámetros, que son:

- *batch_size*: tamaño de *batch*
- train_dataset: conjunto de datos de entrenamiento.
- validation_dataset: conjunto de datos de validación.
- init_from_nemo_model: modelo preentrenado, en este caso sería el modelo base español creado de cero.
- sup_data_path: información suplementaria.
- exp_manager: gestor de los entrenamientos.
- check_val_every_n_epoch:
- pitch_mean: media extraída de ejecutar el *script* sup_data.
- pitch_std: desviación estándar extraída de ejecutar el *script* sup_data.

Por último, se realiza el entrenamiento, y se obtienen una serie de modelos, cada uno con una pérdida diferente. De esta manera, se ha realizado un *fine-tuning* del modelo, utilizando otro *dataset* español. Una vez con el modelo creado, se procederá a ejecutarlo para ver qué voz crea.

3.6. *Fine-tuning* de HiFiGAN del modelo base español con el dataset original

El proceso de *fine-tuning* de HiFiGAN es como sigue, aunque se encuentra en mayor detalle en el Apéndice C.7:

1. Se tienen dos tipos de mel espectrogramas, que se pueden usar para realizar el *fine-tuning* de HiFiGAN:
   - Utilizando el mel espectrograma original generado del archivo de audio de origen. Este se puede observar en la Figura 3.4
   - Mel-espectrograma predicho por FastPitch. Este se puede ver en la Figura 3.5
2. No obstante, es importante destacar que el espectrograma pronosticado tiene una duración de 241 marcos, lo que lo diferencia de manera significativa del espectrograma original que abarca 345 marcos. Para llevar a cabo el proceso de *fine-tuning* en HiFiGAN, resulta esencial contar con un mel-espectrograma pronosticado a partir de FastPitch que conserve

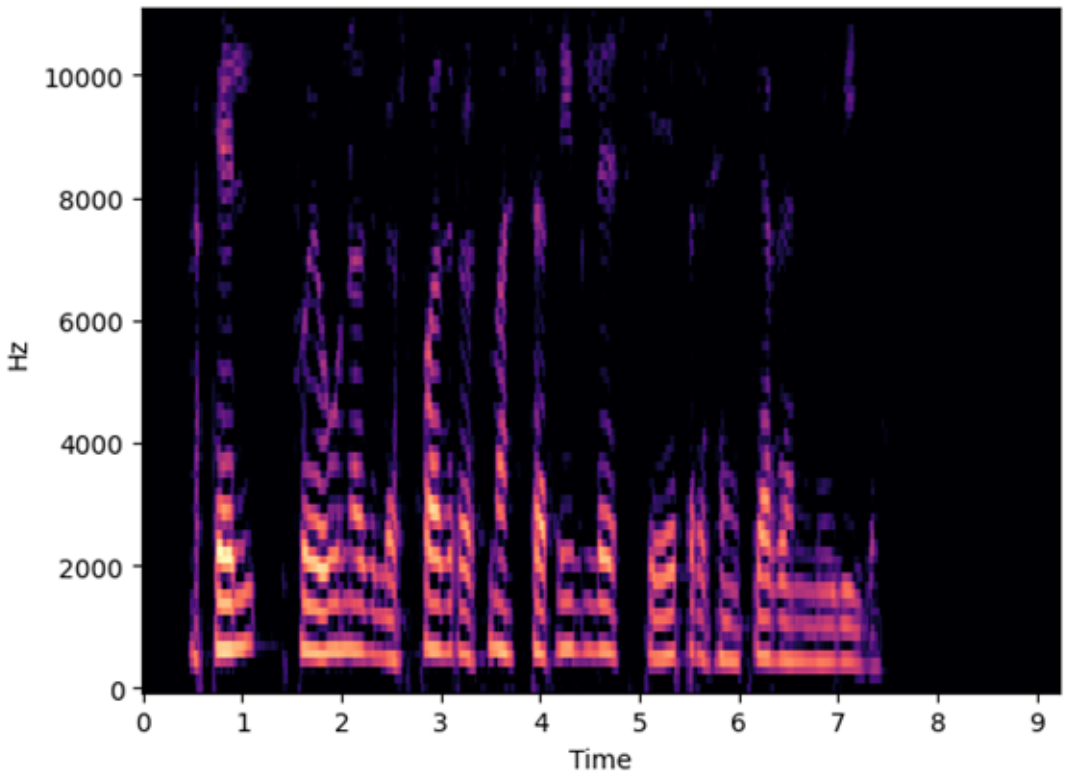


Figura 3.4: Espectrograma original

tanto la alineación como la duración originales. Esto cobra una importancia crucial, ya que se busca generar audio de síntesis de habla que no solo sea de alta calidad, sino también que suene de manera natural y coherente.

El espectrograma-mel predicho a partir de FastPitch con alineación y duración reales. Este se puede observar en la Figura C.5.

3. Después de ver todos los mel-espectrogramas, se procede a generar en la carpeta mels, unos archivos con formato .json, que serán iguales a los archivos train_manifest_text_normed.json, test_manifest_text_normed.json y val_manifest_text_normed.json, pero añadiendo el atributo mel_filepath.

4. Más tarde, una vez con los nuevos archivos .json, se realiza el *fine-tuning*.

   En la Figura 3.6 se puede observar el espectrograma que resulta después de realizar la fase de *fine-tuning* del vocoder HiFiGAN. Después de obtener las muestras de audio definitivas y evaluar su métrica de pérdida, se notó que, a pesar de observar una mejora en el valor de pérdida, los resultados no cumplían con las expectativas deseadas. La calidad del audio apenas mostraba una mejoría significativa. Se realizaron diversos experimentos de ajuste de hiperparámetros, pero ninguno de ellos sobresalió en términos de calidad de audio.

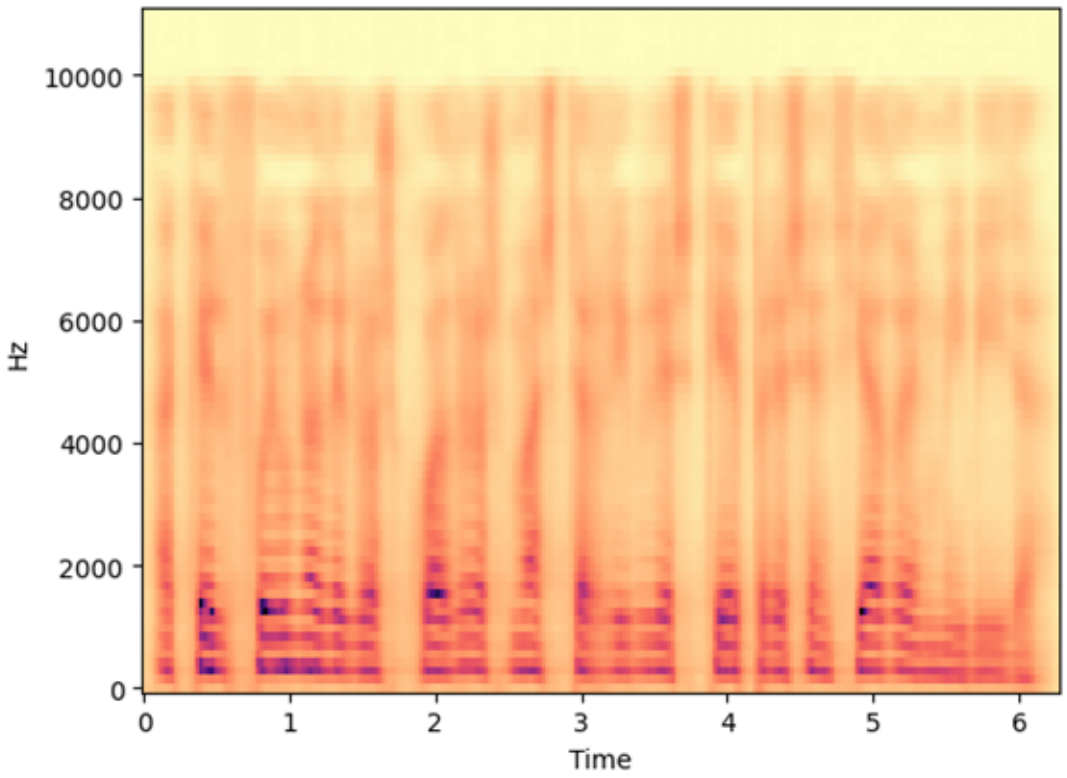


Figura 3.5: Espectrograma predicho

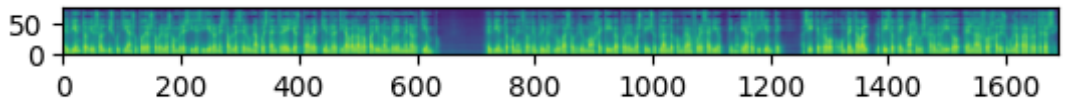


Figura 3.6: Espectrograma después de realizar el *fine-tuning* de HiFiGAN

3.7. Ejecución de las métricas

Por último, hay una serie de métricas que se han escogido para evaluar una serie de características de los audios creados. Estas se pueden observar en la Sección 3.9.1. Los pasos que se realizan son los siguientes. Estos pueden ser ampliados en el Apéndice C.8:

1. Se cargarán los audios guardados del cuaderno de evaluación.
2. Más tarde se calcularán todas las métricas, de tal manera, que se compare el audio original con el creado por el modelo, para cada una de las métricas.
3. Para cada audio, se obtendrán las métricas calculadas, que más tarde, se estudiarán, como se puede observar en el capítulo 4.

3.8. Ejecución del cuaderno de evaluación

A continuación, pasamos a ejecutar el cuaderno de evaluación, que contiene los siguientes pasos, aunque estos se encuentran más detallados en el Apéndice C.9:

1. Primero de todo, se cargan una serie de frases, tanto las de evaluación del *dataset*, como otras, previamente buscadas, del *paper* de *International Communication Union (ITU)* [35]. Algunas de estas últimas servirán para realizar la evaluación perceptual.
2. Más tarde se cargan los modelos, tanto el mel-espectrograma como el *vocoder*.
3. Por último se crean los audios a partir del modelo entrenado. Se cargan varios audios de varios modelos diferentes, para así realizar comparaciones entre ellos.
4. Se guardarán tanto los audios originales como los creados por el modelo.

3.9. Evaluación

En la siguiente sección se explican tanto las métricas de evaluación como la evaluación perceptual.

3.9.1. Métricas de calidad del modelo resultante

A continuación se van a explicar una serie de métricas que han sido utilizadas para la medición de la calidad del modelo resultante [9]. Estas métricas ayudan a comparar cuantitativamente las señales sonoras de habla original con muestras de test generadas por el modelo, identificando diferencias y evaluando el impacto.

- Métricas de medición de la calidad del audio
  - fwSNRseg (Frequency-weighted Segmental Signal-to-Noise Ratio) y SNRseg (Segmental Signal-to-Noise Ratio): Estas métricas evalúan el nivel de ruido o distorsión en la señal hablada (la creada por el modelo) en comparación con la señal original. Proporcionan información sobre la calidad de la señal en términos de relación señal-ruido.
  - LLR (Log-likelihood Ratio): se puede utilizar para comparar cómo de bien el habla del modelo coincide con las características del habla original.
  - WSS (Weighted Spectral Slope): mide la pendiente espectral de la señal. Puede servir para evaluar si las características espectrales de la señal sonora creada por el modelo difieren significativamente del habla original.
  - Cepstrum Distance (CD): Esta métrica se puede utilizar para cuantificar la similitud o diferencia entre la señal original y la señal hablada en el dominio cepstral. Puede ayudar a calcular cuánto han cambiado las señales.

- Métricas de medición de la inteligibilidad del audio
  - STOI (Short-time Objective Intelligibility): evalúa la capacidad de comprensión de la señal hablada en relación con la señal original, lo cual es crucial.
  - CSII (Coherence and Speech Intelligibility Index): es similar a STOI. CSII evalúa la inteligibilidad del habla basada en la coherencia. Puede proporcionar información adicional sobre cómo de bien la señal hablada retiene la inteligibilidad.
- Métricas de medición de la desverberación
  - BSD (Distorsión Espectral en la Escala Bark): BSD es una métrica que mide la distorsión en el dominio de frecuencia Bark. Evalúa cuánto difieren las características espectrales de la señal hablada de la señal original cuando ambas se transforman en la escala de frecuencia Bark. BSD se basa en la idea de que la calidad del habla está relacionada con la intensidad del habla, que es una medida psicoacústica que refleja la magnitud de la percepción auditiva. Para calcular esta intensidad, se procesa la señal de habla utilizando datos de mediciones psicoacústicas. El BSD evalúa la distorsión global comparando las intensidades de la señal de referencia y la señal de habla distorsionada a través de la distancia euclidiana promedio. Funciona bien cuando la distorsión en las regiones de habla representa la distorsión general, ya que se enfoca solo en estas regiones y requiere la detección de las mismas.

En la Tabla 3.1 se muestran los umbrales de las diferentes métricas:

Métrica	Descripción	Umbral
fwSNRseg	Frequency-weighted Segmental Signal-to-Noise Ratio	> 1 sugiere mejora en relación señal-ruido
SNRseg	Segmental Signal-to-Noise Ratio	> 0 indica mejora en relación señal-ruido
LLR	Log-likelihood Ratio	Comparación de similitud, sin umbral específico
WSS	Weighted Spectral Slope	Sin umbral específico, contexto-dependiente
Cepstrum Distance (CD)	Similitud en dominio cepstral	Sin umbral específico, contexto-dependiente
STOI	Short-time Objective Intelligibility	$> 0{,}5$ sugiere buena inteligibilidad
CSII	Coherence and Speech Intelligibility Index	Sin umbral específico, contexto-dependiente
BSD	Distorsión Espectral en Escala Bark	Sin umbral específico, contexto-dependiente

Tabla 3.1: Métricas de audio y umbrales

3.9.2. Evaluación perceptual

En esta parte se va a proceder a explicar unas pruebas de inteligibilidad básicas. Para realizar dichas pruebas, se ha creado un cuestionario en Google Forms, el cual se pasará a una serie de personas para que evalúen las diferentes voces [24]. El citado cuestionario ha sido creado a partir del *paper* de *International Communication Union (ITU)* [35]. El cuestionario consiste en una batería de preguntas sobre diferentes audios, cuyas transcripciones se pueden observar en la Figura 3.7 creados por diferentes modelos, con dos temáticas diferenciadas. En la primera parte del cuestionario, 2 tipos de preguntas. El primer tipo es sobre compras por correo. El segundo es sobre información de tráfico ferroviario. A cada una de las personas se les pregunta las cuestiones contenidas en las Figuras 3.8 y 3.9.

```
M1: Miss Robert, las zapatillas color: blanco, talla: 11, referencia: 501-97-52,
precio: 319 francos, se le entregarán en 1 semana.
R3: El tren número 4320 procedente de Birmingham llegará a las 5:44, andén 2, vía C.
```

Figura 3.7: Transripciones de los audios usados en la evaluación perceptual, sobre los cuales se realizan una serie de preguntas, Estas se pueden observar en las Figuras 3.8 y 3.9 La primera transcripción, M1, trata sobre compras por Internet; la segunda transcripción, R3, trata sobre tráfico ferroviario

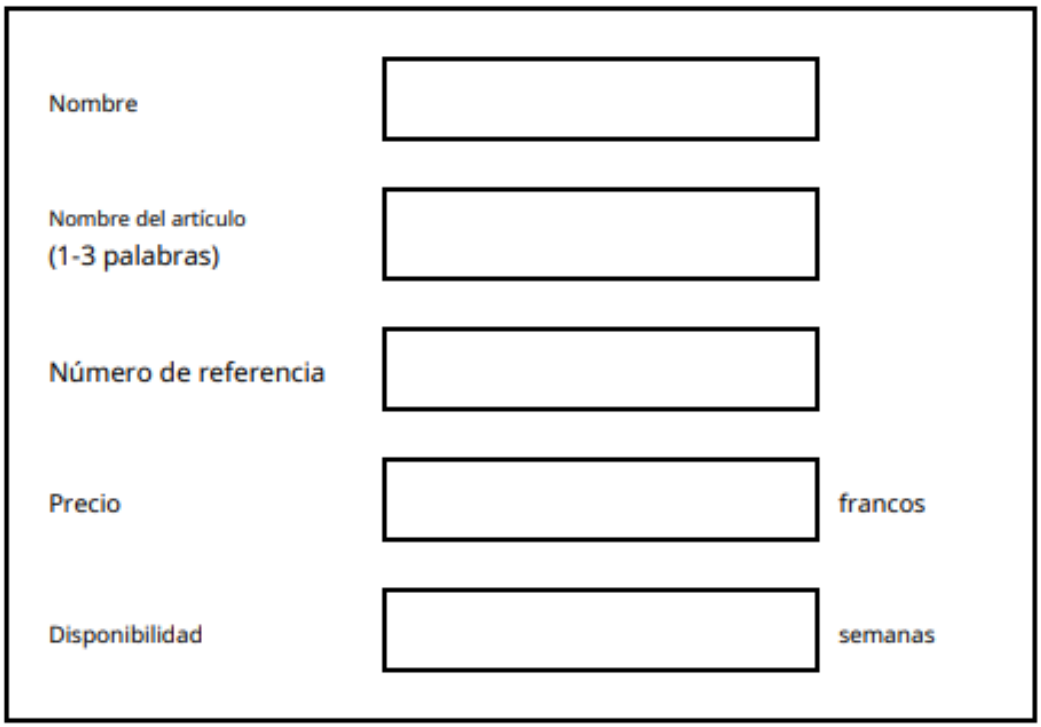


Figura 3.8: Cuestiones relacionadas con las compras por Internet (Transcripción M1)

En la segunda parte del cuestionario se realizan 5 preguntas diferentes, que, esta vez, tienen relación con la calidad y la inteligibilidad de cada uno de los audios. En la Figura 3.10 se pueden

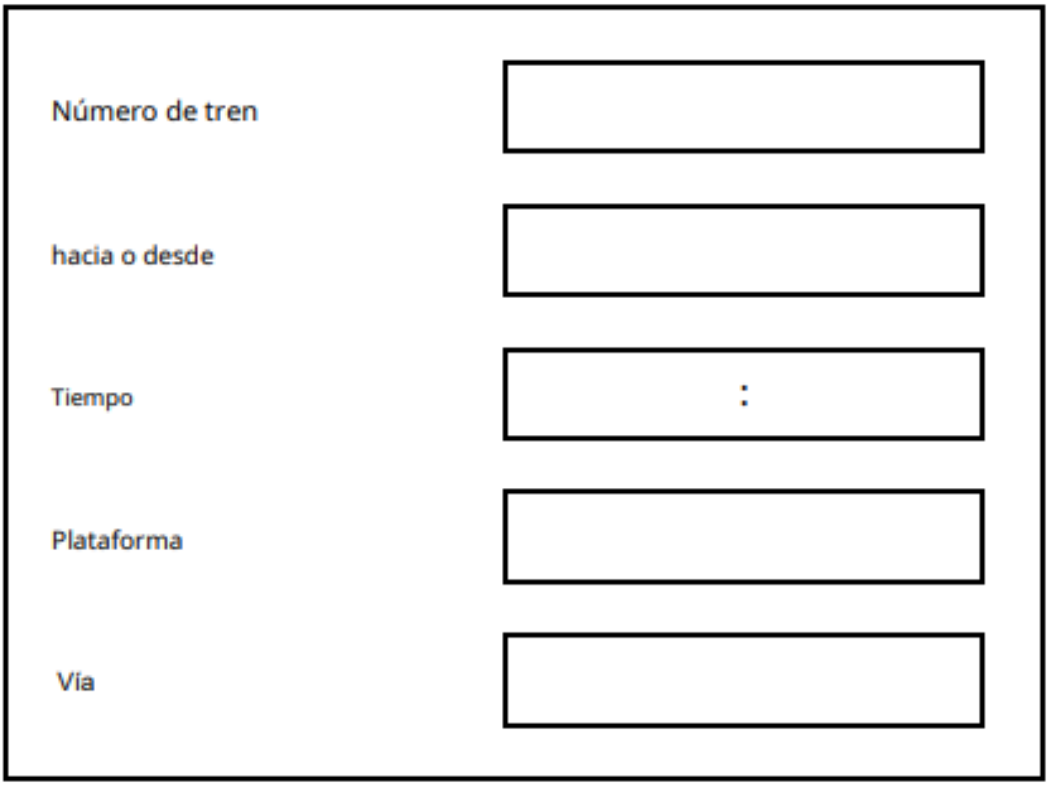


Figura 3.9: Cuestiones relacionadas con el tráfico ferroviario (Transcripción R3)

ver las preguntas.

Overall impression

How do you rate the quality of the sound of what you have just heard?

- ☐ Excellent
- ☐ Good
- ☐ Fair
- ☐ Poor
- ☐ Bad

Pronunciation

Did you notice any anomalies in pronunciation?

- ☐ No
- ☐ Yes, but not annoying
- ☐ Yes, slightly annoying
- ☐ Yes, annoying
- ☐ Yes, very annoying

Speaking rate

The average speed of delivery was:

- ☐ Much faster than preferred
- ☐ Faster than preferred
- ☐ Preferred
- ☐ Slower than preferred
- ☐ Much slower than preferred

Voice pleasantness

How would you describe the voice?

- ☐ Very pleasant
- ☐ Pleasant
- ☐ Fair
- ☐ Unpleasant
- ☐ Very unpleasant

Acceptance

Do you think that this voice could be used for such an information service by telephone?

- ☐ Yes
- ☐ No

Observations:

Figura 3.10: Cuestiones relacionadas con la calidad e inteligibilidad de los audios

Capítulo 4

Experimentos y resultados

4.1. Experimentos

En esta sección se van a detallar los diferentes modelos obtenidos y la valoración de los mismos.

En la Tabla 4.1 se pueden observar los modelos base creados con sus diferentes hiperparámetros.

4.1.1. Modelos base en castellano

Modelo	**Batch size train**	**Batch size val**	**Check val every n ephoc**	**Pitch mean**	**Pitch std**
1	4	4	1	126.73	38.10
2	8	8	1	126.73	38.10
3	12	12	1	126.73	38.10

Tabla 4.1: Comparación de los modelos base

En la Tabla 4.2 se puede observar el valor del atributo *val_loss* y el número de épocas durante las cuales ha sido entrenado el modelo. El modelo 3 tiene el valor de pérdida de validación más

Modelo	**Val loss**	**Épocas**
1	0.77	50
2	0.73	100
3	0.71	265

Tabla 4.2: Rendimiento de los modelos base

bajo (0.7140), lo que sugiere que es el mejor modelo en términos de minimización de la pérdida durante el entrenamiento.

En la Tabla 4.3 se ven los resultados de las diferentes métricas, en las cuales se comparará el audio original del *dataset* con el audio creado por cada uno de los modelos, para una frase del conjunto de datos, de la parte de validación.

Métrica	Modelo 1	Modelo 2	Modelo 3
fwSNRseg	2.79	3.33	2.77
SNRseg	-3.40	-3.40	-3.41
LLR	1.97	1.96	1.95
WSS	98.32	95.38	95.54
Cepstrum Distance (CD)	9.80	9.80	9.82
STOI	0.06	0.11	0.05
CSII	(0.0, 0.0, 0.0)	(0.0, 4.82e-05, 0.01)	(0.0, 0.00, 0.0)
BSD	2299267594.20	460804545.33	2772008744.81

Tabla 4.3: Métricas de audio para los diferentes modelos base

Después de observar los resultados de las métricas para cada uno de los modelos, el Modelo 2 parece ser la mejor opción. Este modelo tiene un buen rendimiento en varias métricas importantes, como fwSNRseg, SNRseg, LLR, WSS, Cepstrum Distance, STOI, y CSII, superando a los otros dos modelos en la mayoría de estas. Además, tiene un valor de pérdida de validación (val_loss) razonablemente bajo. El Modelo 2 a su vez, es la elección más eficiente en términos de tiempo de entrenamiento y recursos computacionales, ya que logra un buen rendimiento con menos épocas que el modelo 3, que tiene más del doble.

4.1.2. Modelos *finetuning* FastPitch castellano con un nuevo *dataset*

En la Tabla 4.4 se pueden observar los modelos base creados con sus diferentes hiperparámetros.

Modelo	**Batch size train**	**Batch size val**	**Check Val every n ephoc**	**Pitch mean**	**Pitch std**
1	7	7	1	109.04	25.55
2	9	9	1	109.04	25.55
3	4	4	1	109.04	25.55

Tabla 4.4: Comparación de los modelos *fine-tuned* con FastPitch

En la Tabla 4.5 se puede observar el valor del atributo *val_ loss* y el número de épocas durante las cuales ha sido entrenado el modelo.

Modelo	**Val loss**	**Épocas**
1	1.53	105
2	1.59	58
3	1.40	65

Tabla 4.5: Rendimiento de los modelos *fine-tuned* con FastPitch

En la Tabla 4.6 se ven los resultados de las diferentes métricas, en las cuales se comparará el audio original del *dataset* con el audio creado por cada uno de los modelos, para una frase del conjunto de datos, de la parte de validación.

Métrica	Modelo 1	Modelo 2	Modelo 3
fwSNRseg	3.98	4.09	4.13
SNRseg	-2.74	-2.46	-2.75
LLR	1.85	1.82	1.83
WSS	84.91	86.04	86.77
Cepstrum Distance (CD)	9.28	9.18	9.20
STOI	0.14	0.14	0.12
CSII	(0.0, 0.0030, 0.0010)	(0.0, 0.0004, 0.0010)	(0.0, 0.0, 0.0110)
BSD	33552977.71	5856353.36	19790573.20

Tabla 4.6: Métricas de audio los modelos *fine-tuned* con FastPitch

Después de analizar los resultados, el Modelo 2 parece ser la mejor elección en esta comparación.Aunque es el peor en términos de pérdida de validación, en relación a las métricas de rendimiento, obtiene resultados sólidos en fwSNRseg, SNRseg, LLR, WSS, Cepstrum Distance, y STOI. Además, se entrenó en menos épocas, lo que puede ser un factor importante a considerar en términos de eficiencia.

4.1.3. Modelos *finetuning* HiFiGAN castellano con el *dataset* original

En la Tabla 4.7 se pueden observar los modelos base creados con sus diferentes hiperparámetros.

Modelo	**Max steps**	**Learning rate**	**L1 loss factor**	**Val check interval**
1	2500	0.00001	25	25
2	2500	0.0002	25	25
3	2500	0.0001	25	25

Tabla 4.7: Comparación de los modelos *fine-tuned* con HiFiGAN

En la Tabla 4.8 se puede observar el valor del atributo *val_ loss* y el número de épocas durante las cuales ha sido entrenado el modelo.

Modelo	**val loss**	**Épocas**
1	0.56	0
2	0.97	0
3	0.98	1

Tabla 4.8: Rendimiento de los modelos *fine-tuned* con HiFiGAN

Después de ver los resultados de *val loss*, se podría pensar que el mejor modelo es el primero. Sin embargo, cuando se reprodujeron los audios, el mejor era el del modelo 3, puesto que sonaba más natural que los demás. En la Tabla 4.9 se ven los resultados de las diferentes métricas, en las cuales se comparará el audio original del *dataset* con el audio creado por cada uno de los modelos, para una frase del conjunto de datos, de la parte de validación.

Métrica	Modelo 1	Modelo 2	Modelo 3
fwSNRseg	3.12	2.83	2.96
SNRseg	-3.27	-3.02	-3.11
LLR	1.93	1.81	1.79
WSS	95.23	88.43	94.73
Cepstrum Distance (CD)	9.79	9.27	9.24
STOI	0.07	0.05	0.13
CSII	(0.0, 0.00014, 0.0)	(0.0, 0.0, 0.0)	(0.0, 0.0, 0.0)
BSD	22463302875.56	28757804441.12	80947561617.72

Tabla 4.9: Métricas de audio de los modelos *fine-tuned* con HiFiGAN

En este caso, se puede observar que la evaluación basada únicamente en el valor de pérdida de validación (val_loss) no refleja adecuadamente la calidad del modelo. A pesar de que el Modelo 1 tiene el val_loss más bajo, el Modelo 3 produce resultados de audio más naturales y de mejor calidad. En las métricas, el Modelo 3 parece ser el mejor, ya que supera a los otros modelos en la mayoría de las métricas, incluido STOI, que está relacionada con la inteligibilidad del audio. El Modelo 1, a pesar de tener el val_loss más bajo, no se desempeña tan bien en las métricas de calidad de audio, lo que destaca la importancia de evaluar el rendimiento de un modelo utilizando múltiples métricas relevantes.

4.2. Resultados

4.2.1. Resultados de la evaluación de los modelos con métricas

En esta sección se van a exponer los resultados de los experimentos. Para realizar una evaluación correcta de los diferentes modelos obtenidos, se han escogido ficheros de audio creados por diferentes modelos de los tres *datasets* que se han empleado. En la Tabla 4.10 se puede observar una comparación entre el mejor modelo que se ha creado haciendo un *fine-tuning* de FastPitch, frente al mejor modelo, realizando el *fine-tuning* de HiFiGAN.

Métrica	Mejor Modelo base	Mejor Modelo Finetuned FastPitch	Mejor Modelo Finetuned HiFiGAN
fwSNRseg	3.33	4.09	2.96
SNRseg	-3.40	-2.46	-3.11
LLR	1.96	1.82	1.79
WSS	95.38	86.04	94.73
Cepstrum Distance (CD)	9.80	9.18	9.24
STOI	0.11	0.14	0.13
CSII	(0.0, 4.82E-05, 0.01)	(0.0, 0.00036, 0.0014)	(0.0, 0.0, 0.0)
BSD	4.61E8	5.86E06	8.09E09
Épocas	100	58	1
Val loss	0.7326	1.5916	0.9774

Tabla 4.10: Métricas de audio

El "Mejor Modelo Finetuned HiFiGAN" se erige como la elección preferida, no solo por su sobresaliente desempeño en métricas críticas de calidad de audio, sino también por su perceptible mejora en la experiencia auditiva. Esta es la mayor diferencia con respecto a los otros dos

modelos, ya que la señal sonora es mucho más inteligible y natural. En definitica, es mejor con respecto a la calidad auditiva. El modelo "Mejor Modelo Finetuned HiFiGAN" exhibe notables mejoras, especialmente en la métrica WSS, lo que se traduce en una calidad de sonido superior y una reducción significativa de la distorsión en comparación con los otros modelos analizados. Además, el HiFiGAN demuestra eficacia en la minimización de la pérdida de validación, lo que podría significar una adaptación sólida a los datos y una capacidad de generalización efectiva. Es destacable que alcanza estos resultados en tan solo una época de entrenamiento, lo que subraya su capacidad de ajuste rápido.

4.2.2. Resultados de la evaluación perceptual

Además, para los resultados referentes a la evaluación perceptual, se toman como referencia dos textos, en vez de señales de audio. Estos pueden ser consultados en la Subsección 3.9.2. Con respecto a los modelos, se van a utilizar los dos mejores modelos que se han obtenido, los cuales se pueden observar en la Tabla 4.10. Los textos de la evaluación perceptual, no se pueden utilizar para la evaluación objetiva con las diferentes métricas, debido a que no se poseen los ficheros de audio, sino que sólo se tienen las transcripciones. Para la realización del cuestionario se recurre a la herramienta *Google Forms* [24].

Después de pasar el formulario a un total de dos personas, los resultados se pueden ver en la Tabla 4.11. De estos, se extraen una serie de conclusiones. La primera de ellas es que ambos modelos necesitan mejorar, pues no son aceptables en la mayoría de los casos. El *fine-tuning* de FastPitch degrada la calidad. Por otro lado, el *fine-tuning* de HiFiGAN mejora el audio con respecto a FastPitch. No obstante, la calidad de audio tampoco es tan buena como debería. Es por esto que existe un gran margen de mejora. Una posible razón relativa a la pobre calidad de los resultados podría ser debido a las diferencias entre el conjunto de datos de entrenamiento utilizado en los artículos originales del modelo base, FastPitch y HiFiGAN y el conjunto de datos en el que realizamos el *fine-tuning*. Los modelos de TTS son altamente sensibles a la calidad y cantidad de datos de entrenamiento disponibles. En este sentido, los *datasets* que se utilizaron para entrenar los modelos originales contienen 24 horas de audio. Sin embargo, el conjunto de datos utilizado para realizar tanto el modelo base, como los modelos *fine-tuned*, es pequeño y no tiene demasiadas horas de audio. Esto revela otra conclusión: no existen a día de hoy *datasets* muy grandes en castellano peninsular. Por lo tanto, al ser el conjunto de datos de *fine-tuning* más pequeño y menos diverso que el conjunto de datos original, esto podría resultar en una calidad de audio inferior.

Por otro lado, es importante tener en cuenta los hiperparámetros de entrenamiento utilizados durante el *fine-tuning*. La elección de la tasa de aprendizaje, el *batch size*, el número de épocas y otros hiperparámetros, puede influir significativamente en el rendimiento del modelo. Una configuración inadecuada de estos hiperparámetros podría de igual manera conducir a una

degradación en la calidad del audio generado.

		Cuestiones		Mejor Modelo Finetuned FastPitch		Mejor Modelo Finetuned HiFiGAN	
		1ª parte	2ª parte	Respuestas 1ª parte	Respuestas 2ª parte	Respuestas 1ª parte	Respuestas 2ª parte
Usuario 1	Audio 1	Nombre: Ms Robert	Impresión general	-	Pobre	-	Justo
		Nombre del artículo: Zapatillas	Esfuerzo de escucha	Zapatillas	Esfuerzo requerido	Zapatillas color blanco	Esfuerzo moderado requerido
		Nº referencia: 501-97-52	Problemas de comprensión	587-52	A menudo	597-52	Ocasionalmente
		Precio (francos) 319	Articulación	319	No, no muy claro	319	Bastante claro
		Disponibilidad (semanas) 1	Aceptación	1	No	1	Sí
	Audio 2	Nº tren 4320	Impresión general	-	Malo	4320	Justo
		Hacia o desde Birmingham	Esfuerzo de escucha	-	Esfuerzo requerido	Birmingham	Atención necesaria: no se requiere un esfuerzo apreciable
		Hora 5:44	Problemas de comprensión	5 y 44	A menudo	5 y 44	Ocasionalmente
		Plataforma 2	Articulación	2	No, no muy claro	2	Bastante claro
		Vía C	Aceptación	C	No	C	Sí
Usuario 2	Audio 1	Nombre Ms Robert	Impresión general	-	Pobre	-	Pobre
		Nombre del artículo Zapatillas	Esfuerzo de escucha	Zapatillas	Esfuerzo requerido	Zapatillas	Esfuerzo moderado requerido
		Nº referencia 501-97-52	Problemas de comprensión	5019752	A menudo	59752	Ocasionalmente
		Precio (francos) 319	Articulación	319	Bastante claro	319	Bastante claro
		Disponibilidad (semanas) 1	Aceptación	1	No	1	No
	Audio 2	Nº tren 4320	Impresión general	4320	Pobre	4320	Pobre
		Hacia o desde Birmingham	Esfuerzo de escucha	Birmingham	Esfuerzo moderado requerido	Birmingham	Esfuerzo requerido
		Hora 5:44	Problemas de comprensión	5:44	Ocasionalmente	17:44	Ocasionalmente
		Plataforma 2	Articulación	2	No, no muy claro	2	No, no muy claro
		Vía C	Aceptación	C	No	C	No

Tabla 4.11: Resultados de la evaluación perceptual

Capítulo 5

Discusión y conclusiones

En este capítulo se ven los resultados obtenidos, la discusión, las conclusiones y la valoración global de la actividad.

5.1. Resultados obtenidos

Primero de todo, se logró ejecutar el cuaderno del modelo FastPitch con HiFiGAN inglés, generando audio en inglés y el cuaderno de *customizing pronunciation*. Al tener el cuaderno inglés funcionando, se está adaptando este y el alemán, al español. Más tarde, la adaptación del modelo inglés al español causó una serie de problemas, por lo que se decidió continuar con el cuaderno alemán. A partir de este, se creó un nuevo modelo base español. Más tarde, se optimizó dicho modelo mediante la aplicación de una serie de hiperparámetros. Después se realizó el *fine-tuning* de FastPitch, el cual mejoró de manera notable la calidad de audio del modelo base. A su vez, se hizo el *fine-tuning* de HiFiGAN. Sin embargo, esta vez el resultado fue aparentemente infructuoso, debido a que no se mejoraban apenas los resultados anteriores, ya que en una época alcanzaba su mejor resultado. Sin embargo, en las pruebas auditivas se vio que la calidad era mejor. Por último, se han calculado una serie de métricas para medir tanto la calidad como la inteligibilidad de los modelos, además de realizar una evaluación subjetiva.

5.2. Discusión

Se han decidido utilizar FastPitch y HiFiGAN porque FastPitch tiene gran capacidad para generar habla natural y de alta calidad. Emplea un enfoque basado en atención para capturar las características prosódicas del habla. Produce audio con detalles finos y alta fidelidad. Asimismo, FastPitch y HiFiGAN tienen cierta velocidad de generación, por lo que son más eficientes. Se decidió utilizar tres corpus en castellano, y realizar el *fine-tuning* a partir del modelo español creado desde cero, por un motivo concreto. En un principio, se probó a utilizar un modelo previamente

entrenado en inglés. A la hora de realizar el *fine-tuning* con un corpus español, aparecía un error a la hora de entrenar; *RuntimeError: Error(s) in loading state_ dict for FastPitchModel: size mismatch for fastpitch.encoder.word_ emb.weight: copying a param with shape torch.Size([115, 384]) from checkpoint, the shape in current model is torch.Size([59, 384])..* Después de investigar de dónde podía provenir el error, se vio que tenía que ver con los fonemas y caracteres propios de cada idioma. Al no ser estos el mismo número en español y en inglés, el entrenamiento abortaba. Se intentó realizar un *tokenizer* personalizado, para adecuar el número de letras y fonemas, pero no dio resultados, por lo que se decidió realizar el *fine-tuning* con el modelo castellano.

5.3. Conclusiones y valoración global del TFM

La mayoría de los objetivos propuestos se han alcanzado. Se ha analizado el entorno TTS y las disfluencias (OBJ-1). Se han estudiado las características y funcionamiento de TTS, y sus tecnologías (OBJ-2). Se ha aprendido sobre NeMo NVIDIA y sobre *Text-To-Speech*. Se ha creado un modelo base desde cero en español, mediante un *dataset* en castellano. (OBJ-3). Se ha realizado un *fine-tuning*, tanto de FastPitch como de HiFiGAN, del modelo base TTS español con un conjunto de datos nuevo (OBJ-4). Además, se han evaluado las voces resultantes tanto del modelo base como del modelo con *fine-tuning*, con respecto a los audios originales (OBJ-5).

Durante la realización del TFM, he adquirido muchos conocimientos. Al empezar tenía conocimientos básicos de TTS, por lo que he profundizado en estos. Además, he probado varios modelos, explorando las estructuras de *vocoders* y modelos existentes. Por último, he visto modelos en varios idiomas con sus fonemas, por lo que también estoy aprendiendo sobre ello.

5.3.1. Perspectiva del proyecto

A continuación se va a valorar la perspectiva del proyecto.

- Objetivos específicos: todos los objetivos han sido completados satisfactoriamente. Se ha investigado sobre DL, TTS, se han visto los algoritmos principales tanto de creación de mel-espectrogramas como de *vocoders*. Por último, se ha realizado un modelo partiendo de cero, que permite crear voz en castellano, a partir de un *dataset* en el mismo idioma, además de realizar los *fine-tuning* de FastPitch y HiFiGAN.

- Utilidad para el futuro: este proyecto puede resultar de gran utilidad puesto que establece una manera de crear un modelo TTS español y de realizar los correspondientes *fine-tuning* y pruebas.

- Desarrollo del proyecto y metodología de trabajo: El proyecto se ha desarrollado siguiendo la metodología de trabajo UVagile, descrita en el Capítulo D. Gracias al empleo de la misma, se ha conseguido llegar al resultado final, de manera satisfactoria. A su vez, se

ha conseguido realizar un uso efectivo del tiempo, que, aun habiendo surgido problemas y cuellos de botella, gracias a la metodología se han conseguido subsanar mediante las tutorías online y los canales destinados a la comunicación.

5.3.2. Perspectiva personal

La realización del presente TFM ha supuesto un aprendizaje muy grande, por varias razones:

- Gracias a la investigación, se ha conseguido llegar entender de una manera amplia TTS y el proceso de creación de modelos.
- Gracias a UVagile, se ha gestionado el tiempo de una manera más eficaz.
- Mediante Teams, se ha conseguido lograr una comunicación muy efectiva.
- Se han mejorado mucho las habilidades con Python y con LaTeX.
- He crecido como estudiante y persona.
- Ser resiliente y ante un problema, buscar la solución hasta encontrarla, sin importar la cantidad de horas o el esfuerzo que se necesite, puesto que al final, el trabajo siempre acaba dando sus frutos si se le dedica tiempo y esfuerzo.

5.4. Trabajo futuro

Debido a las restricciones de tiempo, no se han podido realizar otras mejoras del proyecto, pero podrían ser realizadas en un futuro, y que aportarían mayor valor al proyecto:

- Mejorar los modelos *fine-tuned* tanto de FastPitch como de HiFiGAN, para que los modelos resultantes produzcan unas voces de más calidad.
- Conseguir realizar el *fine-tuning* de un modelo en inglés para que cree audios en español, y que estos sean de buena calidad y prosodia.
- Probar más configuraciones en los diferentes modelos para buscar soluciones más efectivas.
- Incorporar más datos de audios a los diferentes *datasets* para que el entrenamiento se realice con más muestras de audio y sea más efectiva.

Parte I

Apéndices

Apéndice A

Entregables del TFM

Las instrucciones para descargar y ejecutar los diferentes pasos especificados en los capítulos TFM, así como los *scripts* y *notebooks* necesarios, podrán encontrarse en el repositorio GitHub `https://github.com/irenepenasperez/tfm-tts`. El fichero README.md contiene las indicaciones necesarias para poder localizar y descargar las diferentes componentes.

Los experimentos y pruebas asociados al proyecto comenzaron realizándose en Google Colab [1]. Después se pasó a utilizar una computadora del grupo de investigación ECA-SIMM. El cambio fue motivado por dos razones: se pueden dejar almacenados los ficheros de manera permanente y no existe límite de tiempo para realizar las ejecuciones. En vista de que se debía experimentar con diferentes entornos de ejecución Python y librerías tanto de ML como de TTS, se optó por un entorno de trabajo basado en contenedores usando Docker [2] y accediendo a ellos a través de Visual Studio Code y los *plugins* adecuados para manejar y conectar contenedores Docker. En suma, el entorno de trabajo estaba compuesto por:

- Docker: solución de PaaS (Plataforma como Servicio) que permite empaquetar distribuciones Linux y aplicaciones junto a todas sus depedencias en una suerte de máquina virtual que se denomina **contenedor**. Un contenedor es una instancia aislada y ligera de un sistema operativo que contiene la aplicación y todas las bibliotecas y dependencias necesarias para ejecutarla. Docker proporciona una forma eficiente y portátil de distribuir y ejecutar aplicaciones en diferentes entornos.

- *Visual Studio Code*: IDE que proporciona herramientas y extensiones. Es altamente personalizable y admite una variedad de lenguajes de programación y marcos.

- Docker for VSCode: VS Code tiene extensiones y herramientas que permiten la integración con Docker. Esto significa que se puede configurar un entorno de desarrollo para que trabajar con contenedores Docker dentro de VSCode.

[1] `https://colab.research.google.com/`
[2] `https://www.docker.com/`

A continuación, se explica el proceso de montaje de manera más detallada:

1. Se crea un archivo de configuración de Docker llamado Dockerfile que define cómo se debe construir la imagen de la máquina y qué dependencias debe contener (ver repositorio GitHub).

2. Se crea a partir del Dockerfile una imagen del contenedor, con la orden `docker build`.

3. Se ejecuta un contenedor Docker basado en esta imagen, que contiene la máquina y todas sus dependencias.

4. Mediante VSCode se puede establecer una conexión con este contenedor Docker para acceder a la máquina.

Apéndice B

Siglas y acrónimos

A continuación se van a exponer todas las Siglas y acrónimos que han sido utilizadas durante la redacción de la memoria del TFG.

- TTS: *Text-To-Speech* (Texto a Voz).
- G2P: *Grapheme to Phoneme* (Grafema a fonema).
- CNN: *Convolutional Neural Networks* (Redes neuronales convolucionales).
- DNN: *Deep Neural Networks* (Redes neuronales profundas), también llamadas RNN.
- MOS: *Mean Opinion Score* (Puntuación media de la opinión).
- HiFiGAN: *High Fidelity Generative Adversarial Network* (Red Adversarial Generativa de Alta Fidelidad).
- LLM: *Large Language Model* (Modelos de Lenguaje Grande).

Apéndice C

Detalles del cuaderno del proyecto

C.1. Generación de los *datasets*

A continuación, se va a explicar cómo se han generado los *datasets* a partir de los archivos con los audios y sus correspondientes transcripciones. El cuaderno que se detalla a continuación se llama ExperimentoCompleto.ipynb, y contiene la totalidad del proyecto. Durante el cuaderno, se llaman a otros cuadernos o *scripts*, para simplificar.

Para la realización del experimento se han utilizado un total de 3 conjuntos de datos:

- datasetCastellanoReducido: se tiene que disponer de un *dataset* conveniente. En un primer momento, se buscó cuidadosamente un *dataset* en castellano, llamado *Spanish Single Speaker Speech Dataset* [14]. El conjunto de datos contiene un total de 3273 grabaciones. Al descargar este cuaderno, venían 3 carpetas con diferentes grabaciones. Debido a que ocupaba mucho espacio, se decidió escoger solamente una de las carpetas, llamada 19demarzo. Con la descarga, también venía un fichero con las transcripciones, con el siguiente formato: rutaCarpetaAudio|texto|textoNormalizado|duraciónAudio. Algunos ejemplos del fichero se pueden observar en la Figura C.1.

```
1  19demarzo/19demarzo_0000.wav|Durante nuestra conversación advertí que la multitud aumentaba, apretándose más.|Durante nuestra conversación
   advertí que la multitud aumentaba, apretándose más.|5.88
2  19demarzo/19demarzo_0001.wav|Componíanla personas de ambos sexos y de todas las clases de la sociedad,|Componíanla personas de ambos sexos
   y de todas las clases de la sociedad,|4.52
3  19demarzo/19demarzo_0002.wav|espontáneamente venidas por uno de esos llamamientos morales, íntimos, misteriosos, informulados, que no
   parten de ninguna voz oficial,|espontáneamente venidas por uno de esos llamamientos morales, íntimos, misteriosos, informulados, que no
   parten de ninguna voz oficial,|8.31
4  19demarzo/19demarzo_0003.wav|y resuenan de improviso en los oídos de un pueblo entero, hablándole el balbuciente lenguaje de la
   inspiración.|y resuenan de improviso en los oídos de un pueblo entero, hablándole el balbuciente lenguaje de la inspiración.|7.2
5  19demarzo/19demarzo_0004.wav|La campana de ese arrebato glorioso no suena sino cuando son muchos los corazones dispuestos a palpitar en
   concordancia con su anhelante ritmo,|La campana de ese arrebato glorioso no suena sino cuando son muchos los corazones dispuestos a
   palpitar en concordancia con su anhelante ritmo,|7.5
```

Figura C.1: Extracto del dataset castellano, carpeta 19demarzo [14]

  En este fichero venían las transcripciones de las 3 carpetas de audio, por lo que más adelante se eliminarán las filas pertenecientes a las otras dos carpetas de audio, dejándose únicamente la carpeta 19demarzo. Además, se eliminará la columna de la duración del audio, pues se calculará más adelante.También se eliminarán las filas vacías.

- "datasetAlfonsoGutierrezFinetuning": el *dataset* ha sido cedido por el profesor de la UVa Alfonso Gutiérrez, del campus de Segovia. El conjunto de datos es relativo a una clase de la universidad. Los temas tratan diferentes aspectos. El conjunto de datos tiene un total de 92 grabaciones.

- "datasetEsMapa152Finetuning": el *dataset* ha sido cedido también por el profesor de la UVa Alfonso Gutiérrez, del campus de Segovia El conjunto de datos es relativo a otra clase de la universidad. Los temas tratan diferentes aspectos. El conjunto de datos tiene un total de 152 grabaciones.

Todos los *datasets* tienen que pasar por el proceso ETL. En el caso de los *datasets* "datasetAlfonsoGutierrezFinetuning" y "datasetEsMapa152Finetuning", se debe hacer previamente lo siguiente:

1. los ficheros venían separados, por lo que se creó un *script* en Colab para juntar todos los ficheros en uno solo [25]. En este *script* se descomprime el *dataset*. Después se crea un fichero, y mediante un bucle se van abriendo las diferentes transcripciones y se van pegando en el fichero creado anteriormente. Por último, el archivo se almacena y ya se tienen todas las transcripciones unidas.

2. A continuación, se transformó el fichero resultante para que tuviera la misma estructura que el fichero con el que se entrenó el modelo base.

   En las Figuras C.2 y C.3 se pueden observar los ficheros transcript.txt de los *datasets* "datasetAlfonsoGutierrezFinetuning" y "datasetEsMapa152Finetuning", los cuales servirán como entrada al *script* de la fase ETL.

   Estos ficheros tienen la siguiente estructura:

   rutaCarpetaAudio|texto|textoNormalizado|duraciónAudio.

```
1  asignaturaBueno/resumen_asignatura_bueno0.wav|Saludos a todos. Voy a tratar de presentar en unos veinte minutos una visión global para la formación del
   profesorado en TIC y medios para la era digital.|Saludos a todos. Voy a tratar de presentar en unos veinte minutos una visión global para la formación del
   profesorado en TIC y medios para la era digital.|5.88
2  asignaturaBueno/resumen_asignatura_bueno1.wav|A lo largo de la historia se han venido utilizando distintas tecnologías e instrumentos para transmitir
   información, para enseñar y para aprender:|A lo largo de la historia se han venido utilizando distintas tecnologías e instrumentos para transmitir
   información, para enseñar y para aprender:|4.52
3  asignaturaBueno/resumen_asignatura_bueno2.wav|desde las tablillas sumerias 3000 años antes de Cristo hasta las representaciones tridimensionales en la
   actualidad|desde las tablillas sumerias 3000 años antes de Cristo hasta las representaciones tridimensionales en la actualidad|5.88
4  asignaturaBueno/resumen_asignatura_bueno3.wav|Dentro de las tecnologías audiovisuales existen dispositivos específicamente diseñados para facilitar los
   procesos de enseñanza-aprendizaje, como el retroproyector|Dentro de las tecnologías audiovisuales existen dispositivos específicamente diseñados para
   facilitar los procesos de enseñanza-aprendizaje, como el retroproyector|4.52
5  asignaturaBueno/resumen_asignatura_bueno4.wav|que se diseña para la instrucción en el ejército norteamericano,|que se diseña para la instrucción en el
   ejército norteamericano,|5.88
```

Figura C.2: Extracto del *dataset* Asignatura Bueno

```
1  esmapa152/esmapa0000|¿Acaso estaban reñidas con las más pronunciadas curvas?|¿Acaso estaban reñidas con las más pronunciadas curvas?|5.88
2  esmapa152/esmapa0001|¿Y qué me importa la audiencia?|¿Y qué me importa la audiencia?|5.88
3  esmapa152/esmapa0002|¿Es que vuestra majestad no concederá la audiencia acostumbrada?|¿Es que vuestra majestad no concederá la audiencia acostumbrada?|5.88
4  esmapa152/esmapa0003|¿Por qué había interferido en el desenlace?|¿Por qué había interferido en el desenlace?|5.88
5  esmapa152/esmapa0004|¿En la que estáis gaztelu, gabriel y tú?|¿En la que estáis gaztelu, gabriel y tú?|5.88
```

Figura C.3: Extracto del *dataset* esmapa152

3. Más tarde, los ficheros de audio también se pasaron del formato mp4 a wav. Para ello se creó otro *script* [23], el cual convierte los audios en formato .mp4 de una carpeta en audios con formato .wav. Por último, se guardan para utilizarlos posteriormente.

Una vez realizado todo el proceso, y tanto con la transcripción como con los audios preparados, se pasa a realizar la transformación y limpieza del *dataset*

C.2. Transformación y limpieza de los *datasets*

Para la realización de la etapa de ETL, se utiliza el cuaderno FP_ES_TTS_ExtraccionTransformacionLimpiezaDataset.ipynb, en el que se ejecutan los siguientes pasos:

1. Se crea una carpeta en donde se alojarán los ficheros que se vayan creando durante la ejecución del cuaderno.

2. Una vez descargados los datos (para el *dataset* 19demarzo, al descargarse de Kaggle, se necesita el nombre de usuario y la clave) y realizadas las transformaciones descritas previamente, se deben extraer estos, en caso de que estén comprimidos.

3. Después, se cambia a la carpeta en la cual están alojados los archivos.

4. En el caso de los ficheros del 19demarzo (con los que se entrena el modelo base), se tenían que quitar del fichero transcript.txt las filas que no fueran de la carpeta 19demarzo, puesto que en el fichero transcript.txt se alojan todas las transcripciones de las tres carpetas, de las cuales se descargó únicamente una.

5. Más tarde, se realizan algunas transformaciones para que dicho fichero tenga el formato adecuado para poder entrenar. Estas se detallan a continuación:

   - Se quitan las filas que no pertenezcan al *dataset* 19demarzo.
   - Se divide cada columna por el separador |, para adecuarse al formato que el modelo acepta.
   - Por último, a partir del fichero transcript.txt, se crean tres ficheros csv: metadata_dev, metadata_train y metadata_test.

   Para los otros dos *datasets*, se realizan las mismas transformaciones, a excepción de la de quitar las filas que no pertenezcan al *dataset* 19demarzo. Una vez ejecutado este *cuaderno* nos encontramos con los ficheros csv creados. Estos nos servirán como entrada para los siguientes pasos.

C.3. Prueba del modelo FastPitch

En una primera aproximación, se probó el modelo FastPitch con audios en inglés. Una vez comprobado que funcionaba, se decidió realizar un *fine-tuning* de dicho modelo, pero esta vez con un *dataset* en castellano.

C.4. Adaptación de un modelo inglés al castellano

Con el conjunto de datos en castellano descargado de Kaggle y el modelo FastPitch, se intenta en primera instancia adaptar el modelo inglés a un nuevo idioma, el castellano. Durante la realización de dicha adaptación, surgieron una serie de inconvenientes. El más serio fue un problema que hacía que abortara la ejecución del entrenamiento. El error era el siguiente: *RuntimeError: Error(s) in loading state_dict for FastPitchModel: size mismatch for fastpitch.encoder.word_emb.weight: copying a param with shape torch.Size([115, 384]) from checkpoint, the shape in current model is torch.Size([59, 384])..*

Se investigó su posible causa, y se concluyó que debía tratarse de un problema por intentar realizar el *fine-tuning* del inglés al español. Este consistía en que el número de fonemas de ambos idiomas no coincide, ya que el inglés tiene más. Este error se producía al cambiar el *Tokenizer* del modelo, para que fuera español en vez de inglés, ya que se quería hacer el *fine-tuning* del modelo con datos en español. Primeramente se intentó crear un nuevo *tokenizer* añadiendo más fonemas al *tokenizer* español. Esto suponía varios problemas, debido a que los fonemas que se añadían no se correspondían con el idioma. El segundo problema, era que, aunque funcionaba, no se obtenía un audio de buena calidad. Por lo tanto, esa idea fue descartada, y se optó por otro camino.

C.5. Creación de un modelo base español desde cero

Investigando sobre qué soluciones se podían encontrar, se optó por realizar un modelo base desde cero a partir de un ejemplo en alemán [18], en vez de realizar el *fine-tuning* en castellano. El cuaderno alemán sobre el que se cimenta el modelo español, es el creado por Thorsten Müller, el cual utiliza FastPitch con HiFiGAN [18]. En este cuaderno se realiza el proceso desde la normalización del *dataset* hasta el entrenamiento y el *fine-tuning*. Para adaptar el modelo alemán al español, se han hecho una serie de cambios tanto en el cuaderno como en los ficheros de configuración. Es por esto que se van a detallar concienzudamente todos los pasos que se han realizado hasta lograr el modelo base español creado desde cero:

1. Ante todo, se han instalado todas las librerías necesarias.

2. Ejecución del fichero *get_ data.py*: en este fichero se realiza la normalización de los ficheros metadata_train, metadata_test y metadata_dev. Durante la etapa de normalización se obtiene una forma canónica del texto. Normalmente incluye la corrección de errores ortográficos, y la uniformización en el uso de mayúsculas, acentos, signos de puntuación, acrónimos y abreviaciones [7]. Los parámetros de entrada del *script* son los siguientes:

   - data-root: ruta al directorio raíz donde se encuentran los conjuntos de datos que serán utilizados por el script.
   - manifests-root: indica la ruta al directorio donde se encuentran los *manifests*.
   - *–val-size*: Tamaño del conjunto de validación.
   - *–test-size*: Tamaño del conjunto de prueba.
   - *–seed-for-ds-split*: semilla para la división de datos.
   - *–num-workers*: número de trabajadores para procesamiento de datos.

   Hay unos ficheros resultantes intermedios, que son: train_manifest.json, test_manifest.json y val_manifest.json. La forma que tienen estos ficheros es la siguiente: {“audio_filepath”: “ruta”, “duration”: duraciónSegundos, “text”: “Texto”}. Por último, el *script* crea unos ficheros llamados train_manifest_text_normed.json, test_manifest_text_normed.json y val_manifest_text_normed.json. La forma que tienen estos ficheros es la siguiente: {“audio_filepath”: “ruta”, “duration”: duraciónSegundos, “text”: “Texto”, “normalized_text”: “TextoNormalizado”}. Algunos ejemplos de uno de los ficheros se pueden observar en la Figura C.4

{"audio_filepath": "../../datasets/datasetCastellanoReducido/19demarzo/19demarzo_3176.wav", "duration": 6.096145, "text": "Yo tambi\u00e9n tengo mucho miedo. Pero Vd. tiembla, Vd. est\u00e1 malo... En efecto,", "normalized_text": "Yo tambi\u00e9n tengo mucho miedo. Pero vosotros tiembla, vosotros est\u00e1 malo... En efecto,"}
{"audio_filepath": "../../datasets/datasetCastellanoReducido/19demarzo/19demarzo_1213.wav", "duration": 15.484807, "text": "Muy al contrario de esto, nuestro personaje ten\u00eda, sin duda, en su organismo un resorte para la risa, de la cual pasaba a la seriedad tan bruscamente como si un dedo misterioso se quitara de la tecla de lo alegre para oprimir la de lo grave.", "normalized_text": "Muy al contrario de esto, nuestro personaje ten\u00eda, sin duda, en su organismo un resorte para la risa, de la cual pasaba a la seriedad tan bruscamente como si un dedo misterioso se quitara de la tecla de lo alegre para oprimir la de lo grave."}
{"audio_filepath": "../../datasets/datasetCastellanoReducido/19demarzo/19demarzo_1058.wav", "duration": 6.435374, "text": "A \u00e9ste no le hemos podido coger, y seg\u00fan las noticias que hoy recib\u00ed, ha desaparecido del Real Sitio.", "normalized_text": "A \u00e9ste no le hemos podido coger, y seg\u00fan las noticias que hoy recib\u00ed, ha desaparecido del Real Sitio."}

Figura C.4: Fichero normalizado

   Estos ficheros servirán como entrada para sacar unas estadísticas necesarias, además de para realizar el entrenamiento.

3. Antes de proseguir, se debe hacer un paso extra necesario para los *datasets* “datasetAlfonsoGutierrezFinetuning” y “datasetEsMapa152Finetuning”, debido a que los audios se encuentran en estéreo. Se deben cambiar a mono, y de esta manera, funcionan los siguientes pasos.

4. Ejecución del *script* extract_sup_data.py: este fichero se utiliza con el fin de extraer el tono de cada audio y estimar las estadísticas de tono (media, estándar, mínimo y máximo),

iterando sobre los datos una vez. Para conseguir extraer estas características, se han retocado dos parámetros del fichero ds_for_fastpitch_align.yaml, para que en vez del lenguaje y la normalización alemana use la española:

- text_tokenizer: cambiado de GermanCharsTokenizer a SpanishCharsTokenizer.
- text_normalizer: lang: de DE a ES.

También se ha utilizado el fichero train_manifest_text_normed.json. Las estadísticas de tono resultantes son las que siguen:

- PITCH_MEAN = 126.73465728759766
- PITCH_STD = 38.099849700927734
- PITCH_MIN = 65.4063949584961
- PITCH_MAX = 2021.7091064453125

Cabe destacar que, aunque este paso no es obligatorio, es útil para acelerar y estabilizar el entrenamiento.

5. Entrenamiento del modelo: a continuación, se procede a ejecutar el código correspondiente al entrenamiento. Antes de ejecutar, se van a realizar una serie de explicaciones:

   a) En el comando que se utiliza de Python, se establecen algunos parámetros, que son:
   - *batch_size*: tamaño de *batch*
   - train_dataset: conjunto de datos de entrenamiento.
   - validation_dataset: conjunto de datos de validación.
   - init_from_nemo_model: modelo preentrenado, en este caso sería el modelo base español creado de cero.
   - sup_data_path: información suplementaria.
   - exp_manager: gestor de los entrenamientos.
   - check_val_every_n_epoch:
   - pitch_mean: media extraída de ejecutar el *script* sup_data.
   - pitch_std: desviación estándar extraída de ejecutar el *script* sup_data.

   b) Los ficheros que se utilizan son: *fastpitch.py*: este script de Python utiliza PyTorch Lightning y NeMo para entrenar un modelo de síntesis de voz FastPitch. Carga la configuración del modelo y entrenamiento desde archivos de configuración Hydra, que son archivos utilizados para personalizar y gestionar parámetros. Además, configura *callbacks*, como el "LogEpochTimeCallback", que se utilizan para registrar información relevante durante el entrenamiento, como el tiempo que tarda cada época. Este script es fundamental para ajustar eficientemente el modelos de síntesis de voz. Además, se usan

los ficheros train_manifest_text_normed.json, val_manifest_text_normed.json y el fichero de configuración modificado *fastpitch_ align_ 44100*. A continuación se explican las modificaciones sufridas por el último fichero:

- text_tokenizer: cambiado de GermanCharsTokenizer a SpanishCharsTokenizer.
- text_normalizer: lang: de DE a ES.
- num_workers: pasan de 12 a 8.

c) Además, se ajustan las estadísticas de tono resultantes, el tamaño de *batch*, en este caso 12 y el número de épocas, 10.

6. Una vez completado el entrenamiento, se evalúa la calidad del modelo generado FastPitch. Esta se mide utilizando el cuaderno FP_ES_TTS_Evaluate.ipynb, el cual contiene los siguientes pasos:

   a) Primero de todo se carga un texto, correspondiente o no a un audio de validación del *dataset*. Además se carga su ruta.

   b) Después, una función se encarga de generar audio a partir de una representación espectrograma utilizando un modelo FastPitch y HiFiGAN Primero, se genera un número de semilla aleatorio para garantizar la reproducibilidad. Luego, se procesa el texto de entrada utilizando el modelo "spec_gen_model" para obtener un espectrograma normalizado. Más tarde, se convierte este espectrograma en audio.. Finalmente, se realiza el procesamiento necesario para normalizar el audio y se devuelve tanto el audio como el espectrograma generados.

   c) Se cargan los modelos FastPitch (el creado ahora) y HiFiGAN (preentrenado)

   d) Además, para comparar con el modelo FastPitch creado, se pueden cargar otros modelos preentrenados.

   e) Finalmente se crean los espectrogramas y los audios, y se guardan aquellos que sean interesantes para estudiarlos posteriormente.

   Después de oír los audios resultantes, probando con 3 modelos diferentes (con diferente tamaño de *batch*, siendo este 4, 8 y 12), se cree que la calidad del audio no es tan buena como debería. La prosodia en cambio sí que lo es. Para mejorar la calidad del audio, se procederá a realizar un *fine-tuning* tanto de FastPitch como de HiFiGAN.

C.6. *Fine-tuning* de FastPitch del modelo base español con un *dataset* nuevo

Después de obtener el modelo base castellano desde cero, se pensó en realizar un *fine-tuning* de dicho modelo, pero con el *dataset* "datasetAlfonsoGutierrezFinetuning", también en castellano.

Se realizan los siguientes pasos, los cuales son similares a los de la creación del modelo base, hasta la etapa de *fine-tuning*, donde se escoge un modelo preentrenado (el modelo base español).

1. Transformación y limpieza del *dataset*: se realiza del mismo modo que en el apartado C.2. De manera sucinta, en la ETL se realiza la conversión de los ficheros de audio del formato mp4 a wav, además se tienen que convertir a mono (cambiar el número de canales a 1), debido al problema que se comentó de la diferencia de dimensiones al entrenar.

2. Extracción de los ficheros *manifest*

3. Extracción de los datos suplementarios: Una vez se tienen los ficheros anteriores, es el momento de llamar al fichero *extract_sup_data.py* de nuevo.

4. *Fine-tuning* del modelo base: Después de ejecutar los diferentes *scripts*, tenemos los ficheros necesarios para poder realizar el entrenamiento, que son:

   - train_manifest_text_normed.json: es el *dataset* de entrenamiento, que posee la siguiente estructura: "audio_filepath": audio "duration": duración, "text": texto, "normalized_text": texto normalizado
   - val_manifest_text_normed.json: es el *dataset* de validación.
   - sup_data: conjunto de archivos que servirán para acelerar y estabilizar el entrenamiento.

   Ahora se va a llamar al *script* fastpitch_finetune.py, con el fichero de configuración fastpitch_align_44100.yaml A continuación se van a explicar estos ficheros:

   - fastpitch_finetune.py: este *script* utiliza PyTorch Lightning y la biblioteca NeMo para entrenar FastPitch. Carga una configuración desde archivos Hydra, inicializa el modelo y el entrenamiento, y luego registra el progreso del entrenamiento, incluyendo información sobre la tasa de aprendizaje y el tiempo que tarda cada época.
   - fastpitch_align_44100.yaml: es el mismo fichero que en la etapa de creación del modelo base.

   Además, en el comando que se utiliza de Python, se establecen algunos parámetros, que son:

   - *batch_size*: tamaño de *batch*
   - train_dataset: conjunto de datos de entrenamiento.
   - validation_dataset: conjunto de datos de validación.
   - init_from_nemo_model: modelo preentrenado, en este caso sería el modelo base español creado de cero.

- sup_data_path: información suplementaria.
- exp_manager: gestor de los entrenamientos.
- check_val_every_n_epoch:
- pitch_mean: media extraída de ejecutar el *script* sup_data.
- pitch_std: desviación estándar extraída de ejecutar el *script* sup_data.

Por último, se realiza el entrenamiento, y se obtienen una serie de modelos, cada uno con una pérdida diferente. De esta manera, se ha realizado un *fine-tuning* del modelo, utilizando otro *dataset* español.

5. Ejecución del cuaderno de evaluación: una vez con el modelo creado, se procederá a ejecutarlo para ver qué voz crea. Esto se realiza con el mismo cuaderno de evaluación (FP_ES_TTS_Evaluate.ipynb) que en la etapa de creación del modelo base. Una vez ejecutado el cuaderno, se comprueba mediante una serie de métricas y la pérdida de validación, que los resultados obtenidos mejoraron el modelo base.

C.7. *Fine-tuning* de HiFiGAN del modelo base español con el *dataset* original

Después de realizar el *fine-tuning* de FastPitch, se procedió a hacer el de HiFiGAN, cuya ejecución se puede realizar en el cuaderno FP_ES_TTS_Finetuning_HiFiGAN.ipynb. El proceso es como sigue:

1. Se escoge un audio y se indica su ruta.
2. Se cargan tanto el modelo FastPitch como el HiFiGAN.
3. Ahora, se tienen dos tipos de mel espectrogramas, que se pueden usar para realizar el *fine-tuning* de HifiGAN:
   - Utilizando el mel espectrograma original generado del archivo de audio de origen. Este se puede observar en la Figura 3.4
   - Mel espectrograma predicho por FastPitch. Este se puede contemplar en la Figura 3.5

   Sin embargo, el espectrograma predicho tiene la duración de 241 fotogramas, y por tanto no es igual al original, con una duración de 345 fotogramas. Para poder realizar el *fine-tuning* de HiFiGAN, se necesita un espectrograma mel predicho a partir de FastPitch con la alineación y la duración reales. El espectrograma debe tener la alineación y duración reales puesto que es necesario que se genere audio de síntesis de habla de alta calidad y que además suene natural y coherente.

Espectrograma Mel predicho a partir de FastPitch con alineación y duración reales. Este se puede observar en la Figura C.5.

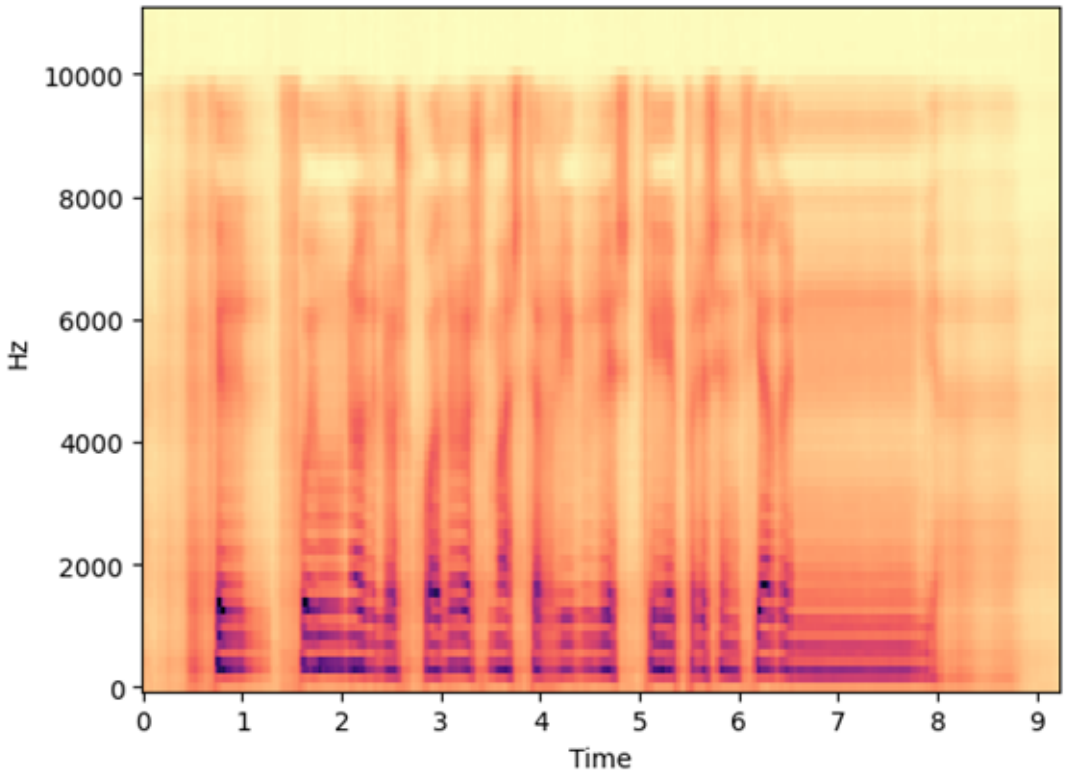


Figura C.5: Mel espectrograma alineado

4. Ejecución del *script* fptts-finetuningHifigan.sh Después de ver todos los mel-espectrogramas, se procede a ejecutar este *script*, el cual llama a dos *scripts*:

   - Llamada a generate_mels: se genera en la carpeta mels, unos archivos con formato .json, que serán iguales a los archivos train_manifest_text_normed.json, test_manifest_text_normed.json y val_manifest_text_normed.json, pero añadiendo el atributo mel_filepath. Ahora, una vez con los nuevos archivos .json, se realiza el *fine-tuning*.
   - Llamada a hifigan_finetune.py. Este tiene como entrada los siguientes parámetros:
     - *–config-path*: ruta al archivo de configuración.
     - *–config-name*: nombre del archivo de configuración.
     - *model.max_steps*: número máximo de pasos de entrenamiento.
     - *model.optim.lr*: tasa de aprendizaje del optimizador.
     - *model.l1_loss_factor*: factor de pérdida L1.

- *model.optim.sched*: desactiva el programador de tasa de aprendizaje.
- *train_dataset*: *manifest* de datos de entrenamiento.
- *validation_datasets*: *manifest* de datos de validación.
- *exp_manager.exp_dir*: directorio de salida de la experimentación.
- *+init_from_pretrained_model*: inicialización desde un modelo preentrenado (tts_en_lj_hifigan_ft_mixerttsx).
- *+trainer.val_check_interval*: intervalo de validación del entrenador.
- *trainer.check_val_every_n_epoch*: frecuencia de validación por época.
- *model/train_ds*: conjunto de datos de entrenamiento del modelo.
- *model/validation_ds*: conjunto de datos de validación del modelo.

5. Ejecución del cuaderno de evaluación de HiFiGAN: el cuaderno, FP_ES_TTS_Evaluate-FTHifiGAN.ipynb es similar al de evaluación de FastPitch, con la única diferencia, de que en el paso de cargar el *vocoder*, en vez de cargar uno preentrenado, se carga el que acaba de pasar por el proceso de *fine-tuning*.

C.8. Ejecución de las métricas

Para evaluar cada uno de los modelos creados, hay un conjunto de métricas que se han escogido para evaluar una serie de características de los audios creados, las cuales se pueden observar en la Sección 3.9.1. Se escogerán los audios guardados del cuaderno de evaluación y se cargarán. Más tarde se calcularán todas las métricas, obteniendo un valor por cada una de ellas. Estos cálculos se realizan en el cuaderno ExperimentoCompleto.ipynb, en la sección llamada Métricas calculadas con Pysepm.

1. Se proporciona una lista de rutas de archivos de audio, incluyendo tres archivos del modelo base, tres archivos del mejor modelo *fine-tuned* de FastPitch y tres archivos del mejor modelo *fine-tuned* de HiFiGAN.

2. Se carga el archivo de referencia (base_speech) y su frecuencia de muestreo (fs_orig) a partir de la última entrada en la lista de archivos de audio.

3. Se inicializa un diccionario metrics_dict para almacenar las métricas de calidad de señal para cada archivo.

4. Luego, se recorren los archivos de audio en la lista, y para cada uno:

   - Se carga el archivo de audio (orig_speech) y su frecuencia de muestreo (fs).
   - Se verifica y ajusta la frecuencia de muestreo y la longitud para que coincidan con el archivo de referencia.

- Se calculan varias métricas de calidad de señal, como fwSNRseg, SNRseg, LLR, WSS, Cepstrum Distance, STOI, CSII y BSD, utilizando funciones de la biblioteca pysepm.
- Se imprimen las métricas para cada archivo de audio comparado con el archivo de referencia.

C.9. Ejecución del cuaderno de evaluación perceptual

Por último, se ejecuta el cuaderno evaluateEvaluacionPerceptual.ipynb. Se han escogido los dos mejores modelos de FastPitch y de HiFiGAN, los cuales se pueden observar en la Tabla 4.10. A su vez, se han escogido 6 frases específicas, de las cuales, las dos mejores serán elegidas para realizar un cuestionario a una serie de personas, que evaluarán la calidad y otros parámetros de las señales sonoras. Por cada frase hay dos audios, uno por cada modelo. Este cuaderno es igual que aquellos de evaluaciónde FastPitch y de HiFiGAN.

Más tarde, se han subido a *Google Drive* todos los audios, y en el cuestionario se redirecciona a los mismos, para que se puedan escuchar [24].

Apéndice D

Planificación

En este capítulo se presenta la metodología utilizada para llevar a cabo el TFM, además de las herramientas utilizadas y la planificación temporal. El proyecto se ha realizado siguiendo la metodología ágil UVagile, descrita en la Sección D.1. El proyecto se ha dividido en 5 sprints de aproximadamente la misma duración cada uno. La organización de cada sprint ha sido descrita en el *backlog* de la aplicación Trello.

D.1. Metodología de trabajo

El presente Trabajo Final de Máster se va a estructurar y desarrollar empleando la metodología de trabajo ágil UVagile [17], adaptada para su uso en Trabajos de Fin de Grado. Esta metodología ágil se basa en los valores y principios del manifiesto ágil, mejorando los proyectos de aprendizaje. El proyecto se desarrollará atendiendo a todas las normas establecidas por la metodología, para lograr un trabajo eficiente y con unos resultados mejores. A continuación se pasa a explicar qué es UVagile.

D.1.1. El marco de trabajo UVagile

UVagile es una metodología de enseñanza y aprendizaje desarrollada como un proyecto de innovación docente de la Universidad de Valladolid. Se quiere implantar una metodología Agile, en concreto Scrum [27] en el ámbito de la universidad. Mediante la implementación de UVagile se pretenden abordar proyectos de aprendizaje y establecer una dinámica de trabajo iterativa e incremental, al tiempo que se motivan el aprendizaje activo, la comunicación entre el profesor y los alumnos, y generar de manera más temprana la retroalimentación.

En este trabajo se usarán los siguientes artefactos:

- Espacio de trabajo compartido en Teams: aquí se guardarán todos los subproductos que se vayan generando durante los sprints. Este espacio será visible tanto para el alumno como

para los profesores. Contará con un canal privado para que la comunicación sea más sencilla entre los profesores y el alumno; Dicho espacio cuenta con control de versiones y una copia de respaldo en la nube. El espacio de trabajo compartido está formado por un conjunto de directorios.

- Tablero del proyecto en Trello: el tablero de Trello cuenta con 6 columnas cada una dedicada a un propósito en concreto: *Backlog* del proyecto, Objetivo del sprint, Tareas pendientes, En curso, Bloqueado y Finalizado. En el tablero se ponen las historias de usuario con los objetivos del proyecto. Cada historia corresponde a una serie de tareas que se van a ir realizando durante la consecución del sprint. Las tareas en un principio están en Pendientes. Una vez empezadas se mueven a En curso. Si surgen bloqueos, se moverán a dicha columna, En caso de que la tarea se complete, se llevará la ficha a Finalizada. En el tablero se especifican los objetivos mediante dichas historias de proyecto, con una descripción de lo que se debe realizar. Las tareas cuentan con una descripción también, con unos puntos de esfuerzo para realizarla, la fecha de finalización prevista y una lista con todos los resultados que tienen que darse para que dicha tarea se marque como completada. Este tablero es accesible tanto al alumno como a los profesores, para que se pueda ir siguiendo el desarrollo del proyecto de manera ordenada y eficaz.

- Eventos:

  - Sprint de trabajo: periodo de tiempo definido, en el que se establecen una serie de objetivos, que al final del sprint tienen que haber sido realizados. Todos los sprints tienen que tener la misma duración (de 4 a 5 semanas), aunque durante la estancia en el grupo de investigación la duración tuvo que ser menor.
  - Reunión de inicio: es el punto de partida de cada sprint. Se tiene como propósito consolidar el objetivo del sprint y establecer una cierta planificación. En la reunión de inicio el equipo analiza el alcance y se derivan las tareas. A su vez se fijan las fechas de las reuniones de sincronización. En esta reunión, participan los profesores y el alumno. La reunión se realiza la primera semana del sprint, con una duración de 30 minutos.

D.2. Estimación del esfuerzo

A la hora de estimar el esfuerzo necesario para llevar a cabo las tareas planificadas, existen diversas técnicas disponibles, como las tallas de camisetas y el *"planning poker"*. En este contexto, se ha optado por utilizar la metodología del *"planning poker"* debido a su simplicidad y efectividad para asignar puntos de historia a cada tarea. Su funcionamiento se detalla a continuación:

- Se utiliza una baraja de cartas con valores que siguen la secuencia de Fibonacci: 1, 2, 3, 5, 8, 13 y 21. Además, la baraja incluye tres cartas especiales con los símbolos de infinito

(∞), interrogación (?) y una taza de café. Las cartas se replican para que cada miembro del equipo tenga su propia baraja. Cada miembro selecciona una carta de cada valor, preparándose para la planificación.

Figura D.1: Baraja de cartas de "planning poker"

- Los valores de las cartas representan los puntos de historia, es decir, la dificultad asociada a cada tarea. Estos valores van desde 0 (sin dificultad) hasta 21 (máxima dificultad). La carta con el símbolo de infinito (∞) indica que la tarea debe dividirse en subtareas para ser manejable. La carta de interrogación (?) indica que la dificultad es desconocida, y el símbolo de la taza de café se utiliza para realizar una pausa durante la planificación.

- A medida que se leen las tareas en voz alta, los miembros del equipo seleccionan una carta que representa la dificultad de la tarea. Cuando se llega a la tarea con el valor 3, los miembros del equipo revelan sus cartas seleccionadas. Si coinciden, se registran los puntos de historia en el tablero del proyecto, junto con las descripciones de las tareas. Si no hay consenso, se abre un debate para entender las razones detrás de las asignaciones de puntos de cada miembro. Luego, se repite la estimación y estos dos pasos hasta que se alcance un acuerdo.

En este caso, al tratarse de una única persona, las estimaciones se realizan en función de las consideraciones individuales. Cada tarea en Trello se asigna con un número de puntos de historia correspondiente.

D.2.1. Herramientas empleadas

Para el desarrollo de este proyecto se han empleado las siguientes herramientas:

- Overleaf: herramienta en línea para la generación de código LaTeX.
- Teams: herramienta de comunicación entre el tutor y el alumno, además de ser el sitio de entrega de los diferentes incrementos.
- Visual Studio Code: se ha utilizado esta herramienta para establecer la comunicación mediante ssh con la máquina virtual del grupo, cronos. Los profesores han creado mediante *docker* un contenedor, sobre el cual se ha montado todo lo necesario para la consecución del proyecto.
- Colab: se ha utilizado Colab para la ejecución del cuaderno de Python. Colab se ha enlazado con la máquina cronos, por lo que se ha podido utilizar su GPU en Colab, haciendo mucho más fácil todo el proceso, puesto que se han podido guardar de manera permanente todos los ficheros y los datos.
- Trello: herramienta para crear una comunicación efectiva entre los profesores y la alumna. En la Figura D.2 se pueden ver las diferentes columnas del tablero de Trello, además de las tareas.

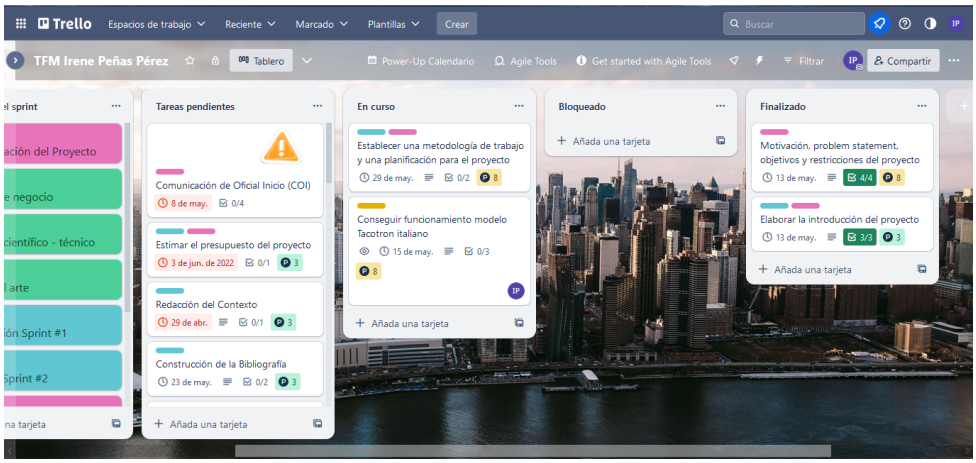
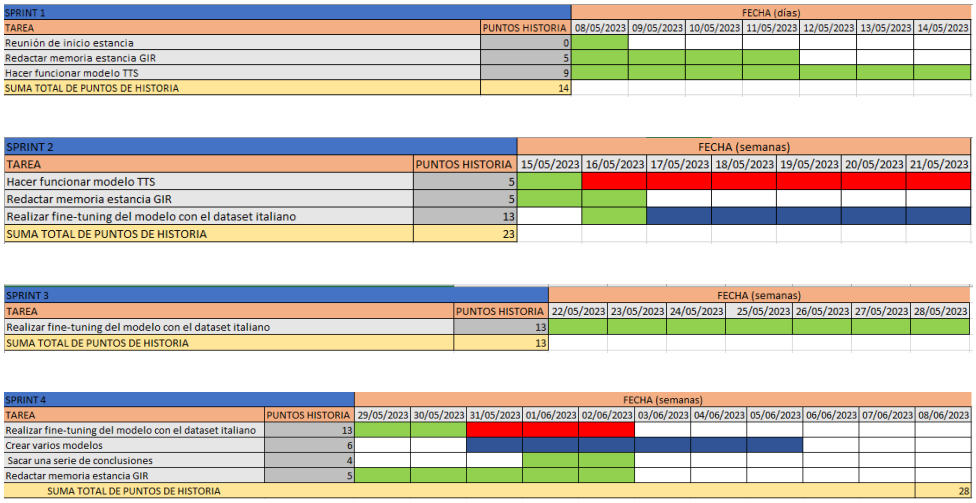


Figura D.2: Tablero TFG Trello

D.3. Planificación temporal

Al comienzo del proyecto, se hizo la estancia en el grupo de investigación ECA-SIMM, por lo que se hizo una planificación específica para la misma siguiendo la metodología UVagile. En

la Figura D.3 se muestra la planificación temporal de los tres *sprints* de esta primera fase de estancia en el grupo. Se han asignado una serie de puntos de historia a las diferentes tareas.



SPRINT 1		FECHA (días)						
TAREA	PUNTOS HISTORIA	08/05/2023	09/05/2023	10/05/2023	11/05/2023	12/05/2023	13/05/2023	14/05/2023
Reunión de inicio estancia	0							
Redactar memoria estancia GIR	5							
Hacer funcionar modelo TTS	9							
SUMA TOTAL DE PUNTOS DE HISTORIA	14							

SPRINT 2		FECHA (semanas)						
TAREA	PUNTOS HISTORIA	15/05/2023	16/05/2023	17/05/2023	18/05/2023	19/05/2023	20/05/2023	21/05/2023
Hacer funcionar modelo TTS	5							
Redactar memoria estancia GIR	5							
Realizar fine-tuning del modelo con el dataset italiano	13							
SUMA TOTAL DE PUNTOS DE HISTORIA	23							

SPRINT 3		FECHA (semanas)						
TAREA	PUNTOS HISTORIA	22/05/2023	23/05/2023	24/05/2023	25/05/2023	26/05/2023	27/05/2023	28/05/2023
Realizar fine-tuning del modelo con el dataset italiano	13							
SUMA TOTAL DE PUNTOS DE HISTORIA	13							

SPRINT 4		FECHA (semanas)										
TAREA	PUNTOS HISTORIA	29/05/2023	30/05/2023	31/05/2023	01/06/2023	02/06/2023	03/06/2023	04/06/2023	05/06/2023	06/06/2023	07/06/2023	08/06/2023
Realizar fine-tuning del modelo con el dataset italiano	13											
Crear varios modelos	6											
Sacar una serie de conclusiones	4											
Redactar memoria estancia GIR	5											
SUMA TOTAL DE PUNTOS DE HISTORIA												28

Figura D.3: Planificación Sprints asociados a la fase de estancia en Prácticas (4 semanas). El código de colores empleado es: en verde la planificación original; en rojo la planificación real; en azul la tarea atrasada.

De la misma manera, se realizó una planificación para el resto del TFM. En la Figura D.4 se puede observar la planificación que se ha seguido en estos 4 sprints finales. Por último, en la Figura D.5 se puede contemplar una recapitulación de todos los sprints con sus tiempos.

Durante la etapa de la estancia en el grupo de investigación, se siguió la planificación tanto en el primer sprint como en el tercero. Sin embargo, en el segundo sprint hay una tarea retrasada (Hacer funcionar el modelo TTS). Hubo una serie de problemas que hicieron que la tarea se atrasara, puesto que no se conseguía hacer funcionar el modelo. Durante la consecución del cuarto y último sprint de la estancia, hubo otra tarea que se demoró (Realizar *fine-tuning* del modelo con el *dataset* italiano), la cual se retrasó por el mismo motivo que la anterior. En un punto determinado saltó un error en el código que no se conseguía subsanar. De hecho, en la Figura D.4, sprint 5, sigue estando la tarea, la cual se volvió a atrasar, dejándola para el sexto sprint. Sin embargo, y como se puede contemplar en la Figura D.4, sprint 6, se tomó otro rumbo, y en vez de realizar un *fine-tuning* del modelo inglés para crear uno italiano, se decidió crear un modelo base castellano.

Siguiendo con el Sprint 6, no se detectaron retrasos. Se creó el modelo base en castellano y se procedió a experimentar con diferentes modelos. Además se empezó a realizar el *fine-tuning* tanto de FastPitch como de HiFiGAN. En el Sprint 7 se siguieron realizando esas tareas, además de construir un formulario para completar una evaluación subjetiva. En el último Sprint, se creó un cuaderno que englobara todo lo realizado hasta la fecha. Además se realizó la evaluación perceptual con una serie de usuarios.

Durante los 8 sprints se ha ido completando la memoria. A su vez, en todo este tiempo, se

SPRINT 5		FECHA (semanas)		
TAREA	PUNTOS HISTORIA	12-18 jun	19-25 jun	26-30 jun
Conexión a la máquina Cronos	5			
Crear modelo finetuned italiano a partir de uno inglés	20			
Creación del modelo base castellano a partir del modelo alemán	21			
Redacción de la memoria	3			
Reunión de sincronización	0			
SUMA TOTAL DE PUNTOS DE HISTORIA	49			

SPRINT 6		FECHA (semanas)		
TAREA	PUNTOS HISTORIA	3-9 jul	10-16 jul	17-23 jul
Creación del modelo base castellano	13			
Experimentar con distintos modelos	3			
Realización fine-tuning FastPitch segundo dataset	10			
Realización fine-tuning HiFiGAN	3			
Redacción de la memoria	3			
Reunión de sincronización	3			
SUMA TOTAL DE PUNTOS DE HISTORIA	35			

SPRINT 7		FECHA (semanas)				
TAREA	PUNTOS HISTORIA	1-6 ago	7-13 ago	14 -20 ago	21 -27 ago	28 -31 ago
Realización fine-tuning FastPitch datos tercer dataset	8					
Realización fine-tuning HiFiGAN	8					
Experimentar con distintos modelos	3					
Construcción formulario Google Forms	6					
Redacción de la memoria	5					
Coordinación a través de Teams	1					
SUMA TOTAL DE PUNTOS DE HISTORIA	31					

SPRINT 8		FECHA (semanas)	
TAREA	PUNTOS HISTORIA	4-10 sep	11-15 sep
Redacción de la memoria	12		
Construcción cuaderno global	8		
Corrección de la memoria	8		
Realización y evaluación de resultados evaluación perceptal	5		
Reunión de sincronización	0		
Entregar TFM	1		
SUMA TOTAL DE PUNTOS DE HISTORIA	34		

Figura D.4: Planificación Sprints asociados a la fase posterior, de realización del TFM como tal. El código de colores es el mismo que en la figura D.3.

han ido realizando una serie de reuniones de sincronización, además de la contínua coordinación con los tutores a través de Teams.

RESUMEN SPRINTS						
TAREA	PUNTOS HISTORIA	mayo	junio	julio	agosto	septiembre
Sprint 1	14					
Sprint 2	23					
Sprint 3	13					
Sprint 4	28					
Sprint 5	49					
Sprint 6	35					
Sprint 7	31					
Sprint 8	34					
SUMA TOTAL DE PUNTOS DE HISTORIA	227					

Figura D.5: Resumen global de la planificación por sprints

Al finalizar los 8 Sprints, los puntos de historia resultantes son 227, como se puede observar en la Figura D.5. Con esto, se puede concluir que cada Sprint estaba equilibrado. Si bien los primeros cuatro sprints tienen menos carga de trabajo, puesto que su duración es menor, los siguientes sprints hasta la finalización del proyecto, tienen una cantidad similar de puntos de

historia, lo que es un aspecto a destacar, debido a que la carga de trabajo ha sido equitativa durante todo el TFM.

Bibliografía

[1] Kong et al. *HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis*. [Consultado el 9 de septiembre, 2023]. 2023. URL: https://paperswithcode.com/method/hifi-gan.

[2] Andy0101. *TTS System*. [Consultado el 17 de abril, 2023]. 2011. URL: https://commons.wikimedia.org/w/index.php?curid=11876828.

[3] Sercan Ömer Arik et al. «Deep Voice 2: Multi-Speaker Neural Text-to-Speech». En: *CoRR* abs/1705.08947 (2017). arXiv: 1705.08947. URL: http://arxiv.org/abs/1705.08947.

[4] Sercan Ö. Arık et al. «Deep Voice: Real-time Neural Text-to-Speech». En: *Proceedings of the 34th International Conference on Machine Learning*. Ed. por Doina Precup y Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, ago. de 2017, págs. 195-204. URL: https://proceedings.mlr.press/v70/arik17a.html.

[5] *Behind Tacotron 2: Google's Incredibly Real Text To Speech System*. [Consultado el 22 de mayo, 2023]. 2023. URL: https://analyticsindiamag.com/tacotron-2-google-ai-text-to-speech-system/.

[6] James Betker. *TTS-scores*. Ver. 1.0.0. [Consultado el 24 de mayo, 2023]. Abr. de 2022. URL: https://github.com/neonbjb/tts-scores.

[7] Alan Bracco. *Normalización de Texto en Español de Argentina*. Ver. 1.0.0. [Consultado el 27 de junio, 2023]. Jun. de 2018. URL: https://rdu.unc.edu.ar/bitstream/handle/11086/11707/Bracco.pdf?sequence=1&isAllowed=y.

[8] Aníbal Bregón Bregón. *Apuntes Sistemas Inteligentes*. [Consultado el 24 de julio, 2023]. 2022. URL: https://campusvirtual.uva.es/.

[9] *examplesForCalculatingMeasures*. [Consultado el 9 de septiembre, 2023]. 2023. URL: https://github.com/schmiph2/pysepm/blob/master/examples/examplesForCalculatingMeasures.ipynb.

[10] Sergios Karagiannakos. *Speech synthesis: A review of the best text to speech architectures with Deep Learning*. [Consultado el 22 de mayo, 2023]. 2021. URL: https://theaisummer.com/text-to-speech/#tacotron-2.

[11] Kyle Kastner et al. «Representation Mixing for TTS Synthesis». En: *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2019, págs. 5906-5910. DOI: 10.1109/ICASSP.2019.8682880.

[12] Ustym Khaburskyi. *How Text-to-Speech Models Work: Theory and Practice*. [Consultado el 22 de mayo, 2023]. 2021. URL: https://www.it-jim.com/blog/how-text-to-speech-models-work-theory-and-practice/.

[13] Jungil Kong, Jaehyeon Kim y Jaekyoung Bae. «Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis». En: *Advances in Neural Information Processing Systems* 33 (2020), págs. 17022-17033.

[14] Shweta Kyubyong Park Tommy Mulc. *Spanish Single Speaker Speech Dataset*. Ver. 1.0.0. [Consultado el 27 de junio, 2023]. Jun. de 2018. URL: https://www.kaggle.com/datasets/bryanpark/spanish-single-speaker-speech-dataset?resource=download.

[15] Adrian Łańcucki. «Fastpitch: Parallel Text-to-Speech with Pitch Prediction». En: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2021, págs. 6588-6592. DOI: 10.1109/ICASSP39728.2021.9413889.

[16] Raghav Mani et al. *Speeding Up Development of Speech and Language Models with NVIDIA NeMo*. [Consultado el 24 de mayo, 2023]. 2023. URL: https://developer.nvidia.com/blog/announcing-nemo-fast-development-of-speech-and-language-models/.

[17] Miguel A. Martínez-Prieto et al. «Una metodología basada en prácticas ágiles para la realización de Trabajos Fin de Grado». En: *JENUI 2023*. 2023. URL: https://uvadoc.uva.es/bitstream/handle/10324/60235/JENUI_2023_paper_44.pdf?sequence=2&isAllowed=y.

[18] Thorsten Müller. *Modifying FastPitch to Train on a Non-English (German) Dataset*. Ver. 1.0.0. [Consultado el 27 de junio, 2023]. Jun. de 2018. URL: https://github.com/NVIDIA/NeMo/blob/main/tutorials/tts/FastPitch_GermanTTS_Training.ipynb.

[19] neonbjb. *TTS Scores - Better evaluation metrics for text to speech models*. [Consultado el 4 de septiembre, 2023]. 2023. URL: https://github.com/neonbjb/tts-scores/blob/main/README.md.

[20] NVIDIA. *Fastpitch 2*. [Consultado el 3 de septiembre, 2023]. 2023. URL: https://pytorch.org/hub/nvidia_deeplearningexamples_fastpitch/.

[21] NVIDIA. *Speech Synthesis HiFi-GAN*. [Consultado el 3 de septiembre, 2023]. 2023. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/teams/tao/models/speechsynthesis_hifigan.

[22] Aäron van den Oord et al. «WaveNet: A Generative Model for Raw Audio». En: *Arxiv*. 2016. URL: https://arxiv.org/abs/1609.03499.

[23] Irene Peñas Pérez. *Convertir de mp4 a wav*. [Consultado el 12 de agosto, 2023]. 2023. URL: https://colab.research.google.com/drive/14izC7G5e-R2LlxvdRG8oCUH25QwZmlmH#scrollTo=OXVrAcJ0ebrQ.

[24] Irene Peñas Pérez. *Formulario comprensión audios síntesis de voz*. [Consultado el 10 de septiembre, 2023]. 2023. URL: https://docs.google.com/forms/d/e/1FAIpQLSdqz-bUytl33xz9NvNwfYZ9UkGb5ihzu3W-qglYEmnMhtOwpA/viewform?usp=sf_link.

[25] Irene Peñas Pérez. *Unificar archivos y formatear txt*. [Consultado el 12 de agosto, 2023]. 2023. URL: https://colab.research.google.com/drive/1btYdX4B6VPYdxwtES5BlfuZB4R2v7p32.

[26] Yi Ren et al. «FastSpeech: Fast, Robust and Controllable Text to Speech». En: *Advances in Neural Information Processing Systems*. Ed. por H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/f63f65b503e22cb970527f23c9ad7db1-Paper.pdf.

[27] Paweł Rola y Dorota Kuchta. «Implementing Scrum Method in International Teams—A Case Study». En: *Open Journal of Social Sciences* 03 (ene. de 2015), págs. 300-305. DOI: 10.4236/jss.2015.37043.

[28] Jonathan Shen et al. «Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions». En: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2018, págs. 4779-4783. DOI: 10.1109/ICASSP.2018.8461368.

[29] Kevin J Shih et al. «RAD-TTS: Parallel flow-based TTS with robust alignment learning and diverse synthesis». En: *ICML Workshop on Invertible Neural Networks, Normalizing Flows, and Explicit Likelihood Models*. 2021.

[30] *Síntesis de habla*. [Consultado el 17 de abril, 2023]. 2023. URL: https://es.wikipedia.org/wiki/S%5C%C3%5C%ADntesis_de_habla.

[31] RJ Skerry-Ryan et al. «Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron». En: *Proceedings of the 35th International Conference on Machine Learning*. Ed. por Jennifer Dy y Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, jul. de 2018, págs. 4693-4702. URL: https://proceedings.mlr.press/v80/skerry-ryan18a.html.

[32] SNGULAR. *CRISP-DM: La metodología para poner orden en los proyectos*. [Consultado el 6 de septiembre, 2023]. 2023. URL: https://www.sngular.com/es/data-science-crisp-dm-metodologia/.

[33] Oktai Tatanov, Stanislav Beliaev y Boris Ginsburg. «Mixer-TTS: Non-Autoregressive, Fast and Compact Text-to-Speech Model Conditioned on Language Model Embeddings». En: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2021), págs. 7482-7486. URL: https://api.semanticscholar.org/CorpusID:238419141.

[34] Moisés Barrios Torres. *What Is Fine-Tuning and How Does It Work in Neural Networks?* [Consultado el 22 de mayo, 2023]. 2023. URL: https://blog.pangeanic.com/what-is-fine-tuning.

[35] International Telecommunication Union. «Telephone Transmition Quality Subjective Opinion Tests. A method for subjective performance assesment of the quality of speech voice output devices». En: (2023). [Consultado el 7 de agosto, 2023]. URL: https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-P.85-199406-I!!PDF-E&type=items.