Universidad Zaragoza
1542

Trabajo Fin de Grado

Estudio de la síntesis de voz mediante técnicas de aprendizaje profundo para la creación de voces sanas y patológicas.

Study of voice synthesis using deep learning techniques for generating healthy and pathological voices

Autor

Santiago Rubio Felipo

Directores

Eduardo Lleida Solano

Dayana Ribas González

Titulación del autor

Ingeniería de Tecnologías y Servicios de Telecomunicación

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2023

AGRADECIMIENTOS

En primer lugar, quiero dar las gracias a mis tutores, Eduardo y Dayana, por su orientación y apoyo durante todo el proceso de elaboración de este Trabajo de Fin de Grado. En estos últimos meses, he tenido la oportunidad de aprender una gran variedad de conocimientos, y también he desarrollado madurez y rigor en mi enfoque hacia el trabajo, lo cual no hubiera sido posible sin vuestra ayuda.

También me gustaría agradecer a todos los compañeros del grupo Vivolab por acogerme en su ambiente de trabajo y ayudarme a resolver determinadas cuestiones que me surgieron en la realización de este. Agradecer también a Agustín por ayudarme a solucionar todos los problemas que tuve con el cluster de la universidad.

Por último quisiera agradecer tanto a mis compañeros de la carrera, quienes hicieron que todas estas tardes de estudio en la biblioteca fueran amenas y divertidas, como a mis compañeros de residencia y piso, quienes me ayudaron a que la convivencia fuera de casa fuera muy sencilla. Y en especial, agradecer a mi familia por acompañarme y apoyarme durante todos estos años de carrera, por creer en mi mucho más que yo mismo.

Estudio de la síntesis de voz mediante técnicas de aprendizaje profundo para la creación de voces sanas y patológicas

RESUMEN

La síntesis de voz ha experimentado avances significativos en los últimos años gracias a la aplicación de técnicas de aprendizaje profundo. Uno de los campos emergentes es el clonado de voces, donde los sistemas de síntesis de voz basados en redes neuronales buscan capturar y replicar con precisión las características distintivas de las voces humanas.

Este Trabajo de Fin de Grado (TFG), aborda el desafío de clonar voces sanas y patológicas con pequeñas cantidades de audio utilizando modelos. Para esto se utiliza el modelo de síntesis de voz VITS (*Variational Inference with adversarial learning for end-end Text-to-Speech*). El estudio se enfoca en explorar cómo estos sistemas pueden ser entrenados para capturar las sutilezas acústicas y fonéticas de las voces patológicas, con el objetivo de aumentar las cantidades de audio en bases de datos patológicas.

En este trabajo se plantea los siguientes interrogantes: ¿Cuánta cantidad de audio es necesaria para esta tarea? ¿Es capaz el sistema de modelar la patología de estas personas? ¿Qué características son más importantes a la hora de clonar una voz? Al explorar esta nueva técnica se aspira a abrir un nuevo enfoque en tareas de clonación de voces y en la grabación de bases de datos patológicas.

Índice

Palabras Claves

1. **AE**: Autoencoder
2. **ASR**: Automatic Speech Recognition
3. **CVAE**: Conditional Variable Autoencoder
4. **CMOS**: Comparative Mean Opinion Score
5. **CNN**: Convolutional Neural Network
6. **Divergencia KL**: Divergencia de Kullback-Leibler
7. **ELBO**: Evidence Lower BOund
8. **FFN**: Feed Forward Network
9. **GAN**: Generative Adversarial Network
10. **GRU**: Gated Recurrent Unit
11. **HiFiGAN**: High Fidelity speech Generative Adversarial Network
12. **IPA**: International Phonetic Alphabet
13. **IS**: Inteligibility Score
14. **LSTM**: Long-Short Term Memory
15. **MCD**: Mel-Cepstral Distorsion
16. **MFCC**: Mel Frequency Cepstral Coefficients
17. **MOS**: Mean Opinion Score
18. **PESQ**: Pereceptual Evaluation Speech of Quality
19. **ReLU**: Rectified Linear Unit
20. **RNN**: Recurrent Neural Network

21. **SAMPA**: Speech Assessment Methods Phonetic Alphabet
22. **SMOS**: Similiraty Score
23. **SPSS**: Statistical Parametric Speech Synthesis
24. **TTS**: Text to Speech
25. **THALENTO**: Tecnologías de HAbla y el Lenguaje para la Evualuación de Trastornos de la Comunicación
26. **VAE**: Variable Autoencoder
27. **VITS**: Variational Inference with adversarial learning for end-end Text-to-Speech

Capítulo 1

Introducción

1.1. Motivación y Objetivos

En los últimos años, la interactividad entre máquinas y seres humanos ha aumentado de manera significativa. Uno de los mayores retos a los que se enfrentan estas tecnologías es lograr la naturalidad de la comunicación humano-máquina.

Según un estudio de la empresa *serpwatch.io* [1] en 2022 alrededor del 63 % de adultos en EEUU usaban asistentes virtuales operados por voz. Mientras que un artículo publicado por la empresa *statista* [2] indica que en el mercado español en 2022 el 54,8 % de usuarios hace uso del asistente virtual de Amazon (Alexa). Además de este asistente de voz, en España son muy usados otros asistentes de voz como el de Google, con una tasa de usuarios un poco inferior a la de Alexa, y Siri con un 33 % de los usuarios.

El gran éxito de los asistentes virtuales de voz ha venido acompañado del desarrollo de nuevos métodos de procesado de lenguaje natural, transcripción de audio y síntesis de voz. Este Trabajo Fin de Grado (TFG) se centra en sistemas de síntesis de voz. El objetivo principal de este trabajo se enfoca en el uso de estos sistemas en aplicaciones personalizadas y especializadas.

Los sistemas de síntesis de voz o sistemas TTS ("*Text to Speech*") tienen como objetivo generar voces naturales e inteligibles a partir de texto. Los sistemas más recientes se basan en aprendizaje profundo [3]. Estos son entrenados con grandes cantidades de audios transcrito para que el sistema pueda aprender el modelado lingüístico del lenguaje, la representación fonética, la prosodia y el modelado acústico del habla. Recientemente *Microsoft* en el artículo sobre su modelo de síntesis de voz *VALL-E* [4] comentó que los modelos basados en mel-cepstrum se entrenan con alrededor de 600 horas de audio transcrito, siendo un gran reto conseguir ese audio para entrenar un modelo desde cero. Si conseguir 600 horas de audio transcrito ya es un gran desafío, en este artículo también se aprecia la actual tendencia de entrenar

modelos más potentes y con una mayor cantidad de datos. Para entrenar *VALL-E* se usaron 60k horas de audiolibros en inglés y en sistemas multilingües como el modelo *Voicebox*[5] de Meta se usaron 110k horas de audiolibros en diferentes idiomas.

Afortunadamente, hay algunos modelos abiertos que ya han sido pre-entrenados, dejando a libre disposición los resultados de esta etapa que supone el mayor coste computacional y de recursos. A partir del modelo pre-entrenado, se realiza un ajuste fino con los datos específicos de la aplicación para adaptarlo al escenario particular. Esta técnica es conocida en inglés por el término *Fine-tuning* [6] y requiere una menor cantidad de datos y coste computacional.

En este trabajo se utilizarán los modelos pre-entrenados de **Coqui-ai**[1]. Este repositorio pertenece a la empresa alemana de síntesis de voz *Coqui-ai* y es uno de los más activos y completos de la comunidad de investigación y desarrollo relacionada. Contiene más de 60 modelos pre-entrenados de síntesis de habla en más de 30 idiomas. Esta comunidad se encarga de recopilar todos los nuevos modelos abiertos de síntesis de voz en un solo repositorio de Github. Desde que se empezó este trabajo el repositorio ha seguido en plena actividad, publicando actualizaciones casi todas las semanas y añadiendo nuevos modelos que también se ajustan a las necesidades de este proyecto.

1.2. Contextualización del trabajo

Este TFG se enmarca en el contexto del proyecto **THALENTO** que desarrolla tecnologías de procesamiento del habla y lenguaje natural para el estudio de los trastornos de la comunicación en lengua española[2]. Una de las etapas del proyecto consiste en la creación de una aplicación de detección y seguimiento automático de patologías de la voz para asistir a los especialistas de voz del **Hospital Clínico "Lozano Bleza"** de Zaragoza. Este trabajo se plantea desarrollar modelos de síntesis de habla personalizados con el objetivo posterior de utilizarlo para aumentar datos de voces patológicas en el entrenamiento de los modelos de dicha aplicación.

1.3. Fases del Proyecto y Cronograma

El proyecto consta de cuatro fases secuenciales. Dentro de cada fase se trabaja con distintas bases de datos, partiendo de voces de personas sanas para desarrollar los primeros modelos. A continuación se ajustarán los modelos para el entrenamiento con voces de pacientes con patologías del habla.

[1] https://github.com/coqui-ai/TTS/
[2] http://dihana.cps.unizar.es/~thalento/

– **Fase 1**: Entrenamiento del modelo de un orador. *Fine-tuning* del modelo pre-entrenado elegido con la base de datos *Librivox* que contiene 50 horas de audio de un solo orador.

– **Fase 2**: Entrenamiento del modelo con varios oradores sanos. Realizar el *fine-tuning* sobre el modelo pre-entrenado en la fase 1. Los datos para el entrenamiento serán los de la base de datos **Albayzin** [7] que contiene voces de 216 personas con una cantidad de audio entre 3 y 12 minutos por persona.

– **Fase 3**: Entrenamiento del modelo de pacientes. Se empieza el *fine-tuning* con los pacientes con patologías de la base de datos **Thalento Lectura y Diálogo** que tienen entre 3 y 5 minutos de audio hablado.

– **Fase 4**: Entrenamiento del modelo de pacientes sin restricción de audio. Se hace el *fine-tuning* con todos las personas de la base de datos de **Thalento Lectura**, teniendo la mayoría de pacientes menos de un minuto de audio.

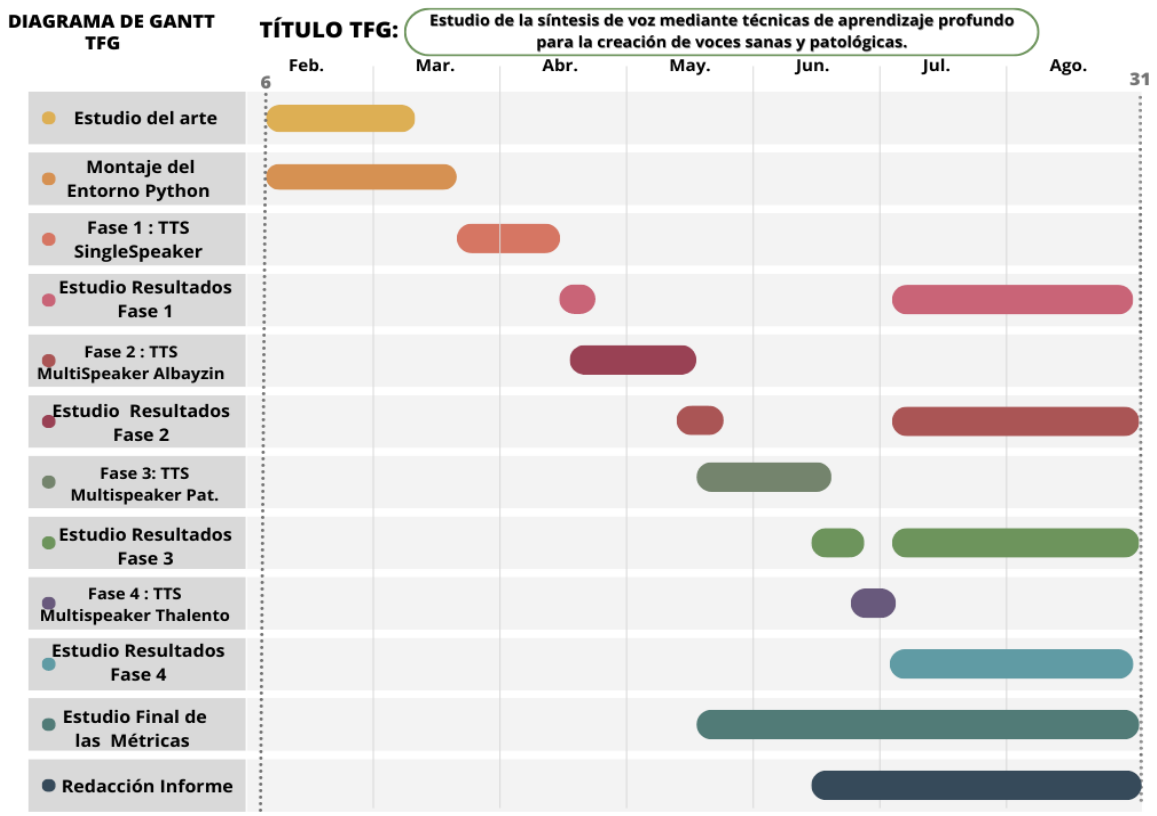


Figura 1.1: Diagrama de Gantt del proyecto

1.4. Estructura de la Memoria

El contenido de la memoria está estructurado en siete capítulos y anexos organizados de la siguiente manera:

- **Capítulo 1**: Introducción. Se han expuesto las motivaciones para empezar el trabajo, los objetivos propuestos y las fases que se siguieron en el proyecto.
- **Capítulo 2**: Estado del Arte. Se expondrá la evolución histórica de los sintetizadores de habla, los nuevos modelos de generación de voz basados en aprendizaje profundo y las posibles apliaciones en el mundo de las patologías de la voz.
- **Capítulo 3**: Modelo de Síntesis de Voz: VITS. Se explicará el modelo que se ha decidido usar en el proyecto, su funcionamiento y sus bloques en tareas que solo implican uno o varios oradores.
- **Capítulo 4**: Bases de Datos. Se describirán las bases de datos usadas, el preprocesado que se ha tenido que hacer en cada una, sus características principales y la partición de los datos entre entrenamiento, validación y prueba.
- **Capítulo 5**: Fases y Métodos. Se empezará explicando las pérdidas que utiliza el modelo, el tipo de entrenamiento, los hiperparámetros, y las métricas usadas para analizar los resultados.
- **Capítulo 6**: Simulación y Análisis de los Resultados. Se expondrán los resultados que hemos obtenido en cada fase de nuestro proyecto y se compararán las métricas usadas para estudiar las prestaciones del sistemas en cada etapa.
- **Capítulo 7**: Conclusiones y Líneas Futuras. Se finalizará con las conclusiones y posibles líneas futuras.

Capítulo 2

Estado del arte

2.1. Evolución Histórica de los sistemas TTS hasta la actualidad

La síntesis de voz artificial ha sido una tarea que la humanidad ha estado intentando desarrollar durante los últimos siglos. A lo largo de la historia se puede clasificar la síntesis de voz en dos etapas diferenciadas por la naturaleza del sintetizador.

2.1.1. Sintetizadores Mecánicos

En 1773, Euler planteó una pregunta acerca de las cualidades tonales que caracterizaban a las diferentes letras, especulando sobre la posibilidad de construir un instrumento musical que produjera sonidos similares a la voz si estas diferencias fueran definidas con precisión. Con esa idea, en 1791 Wolfgang von Kempelen desarrolló su "**Máquina de Voz Acústica-Mecánica**"que podía sintetizar ya todos los sonidos aislados similares al habla humana incluyendo alguna combinación sonora. Está máquina se fue perfeccionando surgiendo nuevos modelos en el siglo XIX que mejorarían la síntesis de voz mediante sintetizadores mecánicos. Aun así estos sintetizadores eran muy complejos de utilizar y la calidad de la voz generada no era buena.

2.1.2. Sintetizadores Electrónicos

Homer Dudley, físico de los *Laboratorios Bell*, desarrollo el primer vocoder electrónico[8] en la década de los 30. Este vocoder consistía en un analizador y un sintetizador de voz artificial usados para la compresión de señales de voz. Esto hizo que se investigará sobre la síntesis de voz de manera electrónica, así en la decada de los 50 surgieron los primeros sistemas de síntesis de habla basados en ordenadores. Estos sistemas evolucionaron hasta que en 1968 se consiguió el primer sistema completo de texto a voz.

Estos sistemas utilizaban diversas técnicas para generar voz, como la síntesis articulatoria, la síntesis por formantes y la síntesis concatenativa. Con los avances del aprendizaje automático estadístico, se desarrollaron otras técnicas basados en la predicción de parámetros estadísticos (SPSS).

Síntesis articulatoria

La síntesis articulatoria[9] produce voz simulando las articulaciones humanas como labios, lengua, glotis y tracto vocal. Idealmente, esta síntesis es la más efectiva porque es como los humanos generan voz, sin embargo, en la práctica es difícil modelar estas articulaciones.

Síntesis por formantes

La síntesis por formante [10] produce voz basándose en un modelo de fuente-filtro sencillo controlado por una serie de reglas. Estas normas son generadas por lingüistas para intentar imitar la voz. La voz generada puede variar parámetros como la frecuencia fundamental, el tono o el ruido, logrando una voz inteligible con un coste computacional moderado y sin necesidad de grandes cantidades de audio de referencia. Por contra, la voz generada no suena muy natural y, en muchos casos, es difícil definir claramente las normas.

Síntesis concatenativa

La síntesis concatenativa[11] se basa en la concatenación de segmentos de audios almacenados en una base de datos. En la inferencia, estos sistemas buscan segmentos de voz e intentan emparejar las entradas de texto con el audio concatenando en estos segmentos. Esta técnica genera audio de alta inteligibilidad y un timbre cercano a la voz original, sin embargo, requiere grandes bases de datos para abarcar todas las posibles concatenaciones y a veces esta unión entre unidades puede resultar en voces poco naturales.

Exiten dos tipos de sítnesis concatenativa: la síntesis difónica y la síntesis basada en selección de unidades. La síntesis difónica[12] se basa en el uso de *difonos*, unidades que empiezan en la mitad de un fonema y continua hasta la mitad del siguiente fonema. Mientrás que la síntesis de unidades pueden abarcar desde un simple fonema hasta una oración completa.

Síntesis por predicción de parámetros estadísticos

Estos sistemas se basan en analizar y modelar las propiedades estadísticas de la voz humana para generar nueva voz. Para eso primero generaban los parámetros acústicos

necesarios para producir el habla y luego recuperar la voz usando parámetros acústicos.

Estos modelos solían consistir en tres módulos: módulo de análisis de texto, módulo de predicción de parámetros acústicos y módulo de vocoder. Algunos de los parámetros acústicos que se modelan son el pitch, frecuencias de resonancias, coecifientes MFCC, etc. Con estos se podrá crear un sistema que simule de forma eficiente la voz.

Algunos de los sistemas más utilizados para estimar estos parámetros eran modelos ocultos de Markov (**HMM**)[13] o modelos mixtos Gaussianos (**GMM**).

2.2. Estado del arte TTS y uso en procesado de voces patológicas

En la actualidad, con el desarrollo del aprendizaje profundo, las redes neuronales se empezaron a considerar como base de la generación de voz. A todos estos modelos en inglés se les considera como *Neural Text To Speech Systems* [3]. Estos sistemas introdujeron una mejora en calidad de audio tanto en inteligibilidad como naturalidad, con menor coste de preprocesado humano y desarrollo de características.

Desde el desarrollo de estos sistemas se intentó la implementación de un modelo *end-to-end*, que integre todas las etapas en un único sistema sin tener intervenciones intermedias. Debido a dificultades en el entrenamiento esto no se pudo conseguir de manera simple y siguió una progresión en las etapas como se ve en la Figura 2.1.

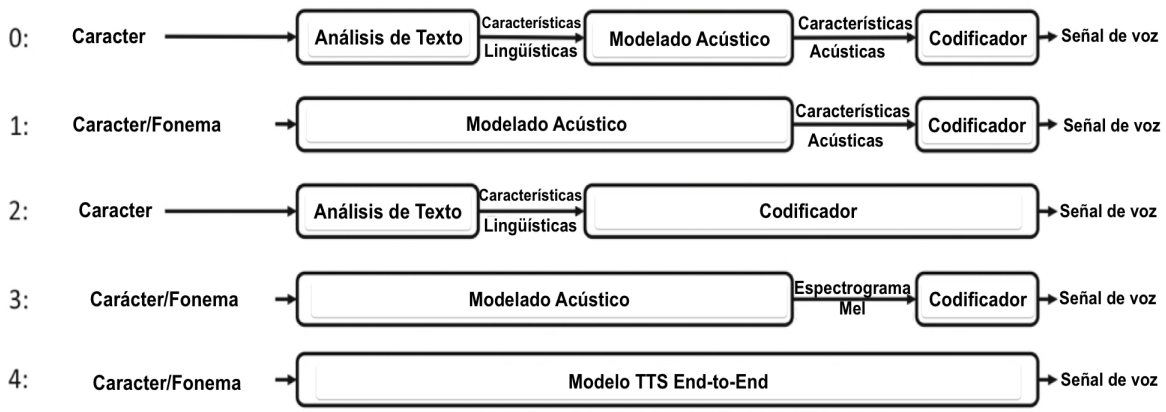


Figura 2.1: Evolución de las etapas en modelos TTS

Se empezaron a usar modelos neuronales incorporados a los sistemas SPSS para reemplazar el modelado HMM. Como los modelos SPSS estos sistemas tenían tres etapas: 1) Conversión texto a características lingüísticas, 2) Generación las características acústicas a través de estas de las lingüísticas, 3) Síntesis la señal de voz a través de las características fonéticas.

Esto se intentó mejorar simplificando el módulo de análisis de texto utilizando los fonemas como entrada. Una mejora significativa fue la invección de ***WaveNet***[14], el primer modelo basado en redes neuronales capaz de generar audio a través de características lingüísticas. Tras esto surgieron varios modelos como *DeepVoice 1/2*[15], que eliminaban el bloque de modelado acústico y usando un codifcador como WaveNet obtuvienen la señal a través de características lingüísticas.

Otros modelos intentaron simplificar el módulo de análisis de texto y tener directamente como entrada el Caracter/Fonema y obtener de salida las características acústicas simplificadas por espectrogramas Mel. Para generar finalmente la voz se usaba un vocoder y así surgieron los primeros modelos casi *end-to-end*. Algunos de estos modelos son *Tacotron 1/2* [16], *DeepVoice3* [17] o *FastSpeech 1/2* [18].

Aún así el problema de representar las características acústicas como un espectrograma Mel es un cuello de botella en la calidad de estos modelos. El espectrograma Mel es una representación física, lo que hace que se tenga dependencias de características intermedias e impide la aplicación de representaciones ocultas de vectores en estas fases, empeorando las prestaciones del sistema. Además, estos sistemas al tener dos etapas es necesario hacer un entrenamiento o *fine-tuning* secuencial lo que es una pérdida de rendimiento tanto computacional como temporal.

Por eso, en los últimos años, ya empezaron a surgir los primer modelos *end-to-end* que permiten un entrenamiento sencillo en una sola etapa. Algunos de los modelos *end-to-end* que se han implementado son *VITS*[19], *NaturalSpeech*[20] o *FastSpeech 2s*[21].

Para observar la proliferación de estos nuevos avances con la invención de nuevos sistemas, se adjunta la Figura 2.2 donde se evidencia claramente el crecimiento exponencial de estos modelos.

Se aprecia un notable avance en los sistemas de síntesis de voz en los últimos años. Sin embargo, dicho progreso no ha venido acompañado por un desarrollo equivalente en su aplicación para voces afectadas por algún tipo de patología. Recientes tesis doctorales [22, 23] estudian la aplicación de estas nuevas técnicas en voces patológicas. En la primera parte de ambos trabajos, se centran en técnicas ASR en voces patológicas de media y baja inteligibilidad, mientras que en los últimos capítulos, se enfocan en el estudio de generación de nuevas voces patológicas, específicamente voces disártricas [1]. En ese entonces no estaban a disposición los modelos TTS *end-to-end* pre-entrenados que hay actualmente. Así que en estos trabajos se centraron en otras técnicas para

[1]Voz Disártrica: Es una condición caracterizada por dificultades en la articulación, prosodia y resonancia del habla debido a problemas neuromusculares que afectan los músculos responsables del control vocal.

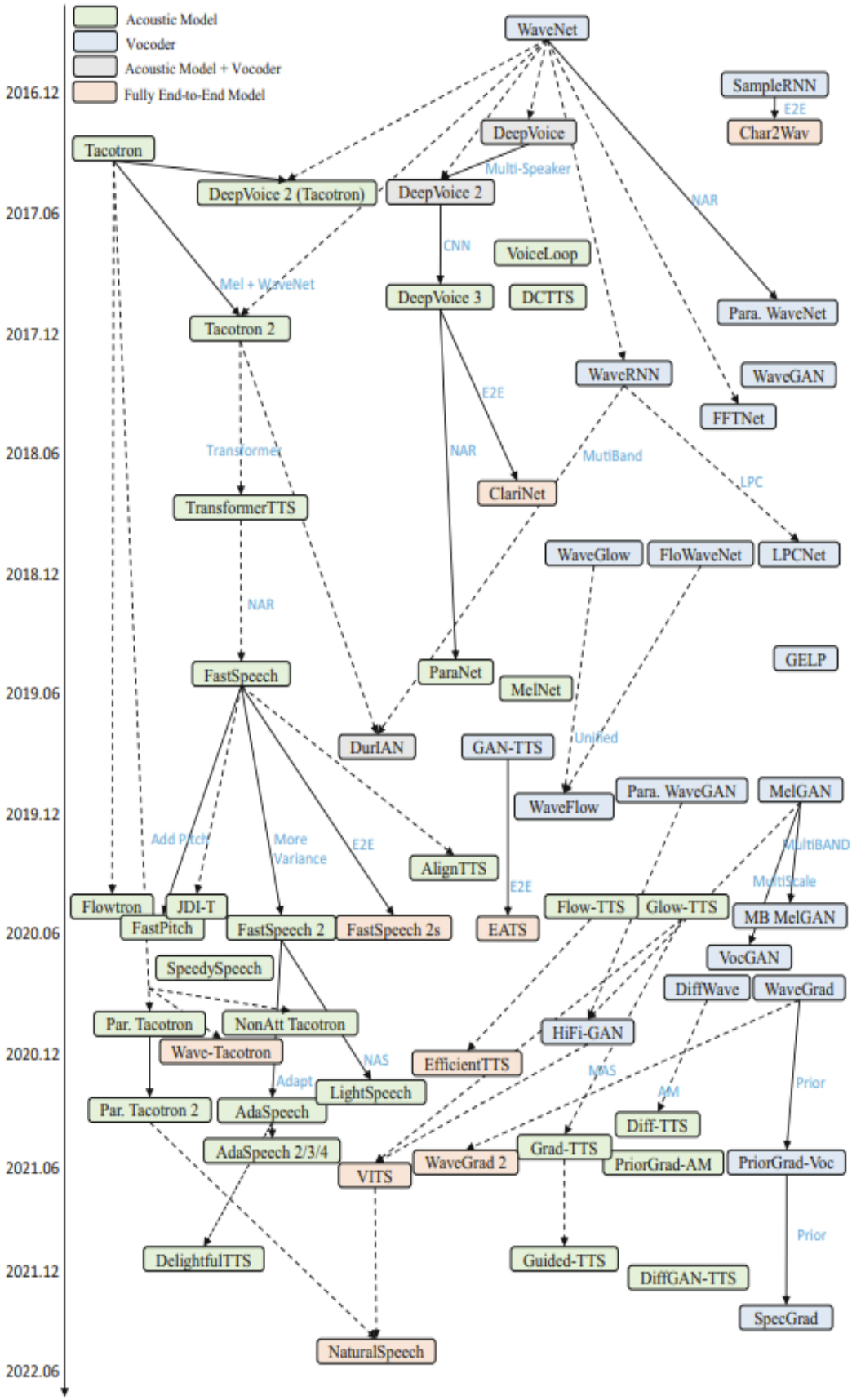


Figura 2.2: Estado del Arte TTS. *Referencia: "Neural Text-to-Speech" [3].*

la generación de nuevas voces. En [24] se hace uso de ténicas de conversión de voz basadas en DCGAN (vea Anexo E.4) para intentar generar nuevas voces con diferentes características a las iniciales. Los resultados de este artículo indican que con pequeñas cantidades de datos se pueden generar nuevas voces disártricas, aunque los resultados no llegan a ser muy concluyentes en algunos pacientes. En [25] se utiliza el modelo FastSpeech2[21] empleando un bloque de *Pause Insertion*, que introduce silencios entre palabras y fonemas, y un bloque de *Dysarthia Severity Predictor*, que busca predecir y modelar la patología de la voz. Los resultados obtenidos no coinciden con el objetivo del trabajo porque lo que consigue con estas modificaciones es aumentar la severidad de la patología introduciendo más pausas al respirar y/o ronquera en la voz. Aún así se elige este artículo de referencia debido a que consigue simular las patologías con un modelo de síntesis de voz antecesor al utilizado en este trabajo.

Aunque estos trabajos se centran en la generación de nuevas voces patólogicas[2], su idea principal es generar una versión más inteligible de las voces de los pacientes. Además, la falta previa de una base de datos de voces patológicas en castellano ha llevado a la realización de todos estos trabajos con voces en inglés pertenecientes a unas pocas bases de datos, como **TORGO**[26], **Nemours**[27] y **UASpeech**[28]. Gracias al proyecto THALENTO, ahora se dispone de una de las primeras bases de datos de voces patológicas en español, lo que permite acompañar el progreso de las tecnologías del habla en el ámbito de las voces patológicas.

[2]Note que estos trabajos se basan en voces patológicas-disártricas, mientras que este TFG se centra en voces patologicas-disfónicas como por ejemplo el **Edema de Reinke**

Capítulo 3

Modelo de Síntesis de Voz: VITS

En el capítulo anterior se estudiaron todos los posibles modelos de generación de texto a voz, lo que reveló un amplio abanico de sistemas potenciales. La búsqueda de modelos se limitó a aquellos que ya tuvieran una base pre-entrenada en español y que ofrecieran un enfoque *end-to-end*. Finalmente, debido a los buenos resultados obtenidos en **YourTTS**[29] se decidió usar el modelo **VITS**[19], que demostró una gran simplicidad y versatilidad. VITS es un modelo *end-to-end* no autoregresivo por lo que su ejecución en tarjetas gráficas será computacionalmente más eficiente. La sencillez en la implementación facilita su uso tanto para expertos en la materia como para aquellos menos familiarizados con esta tecnología, mientras que su versatilidad permite la adaptabilidad del modelo a los diferentes escenarios planteados (Figura 3.1).

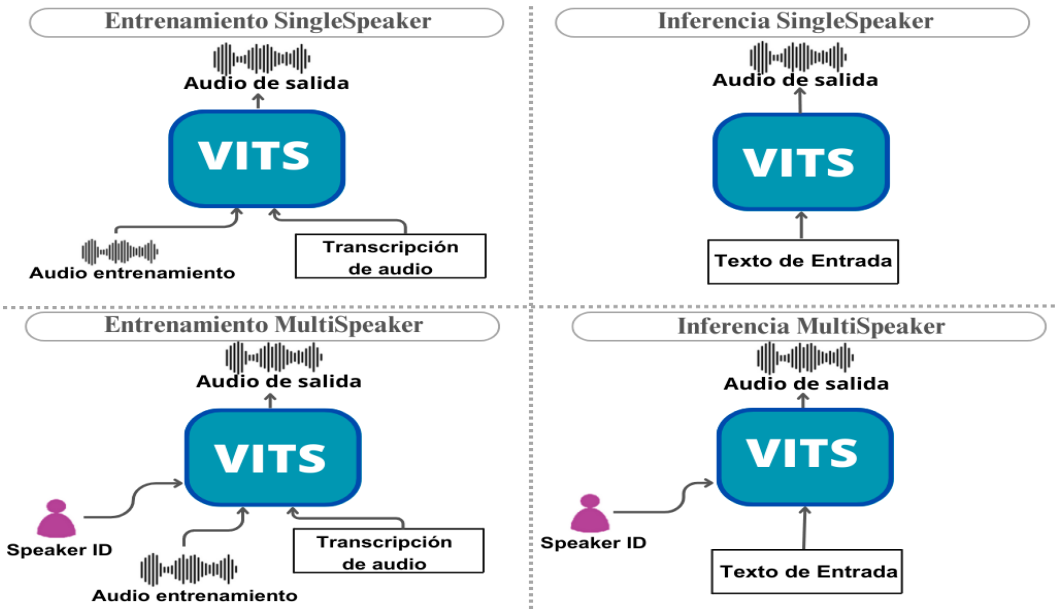
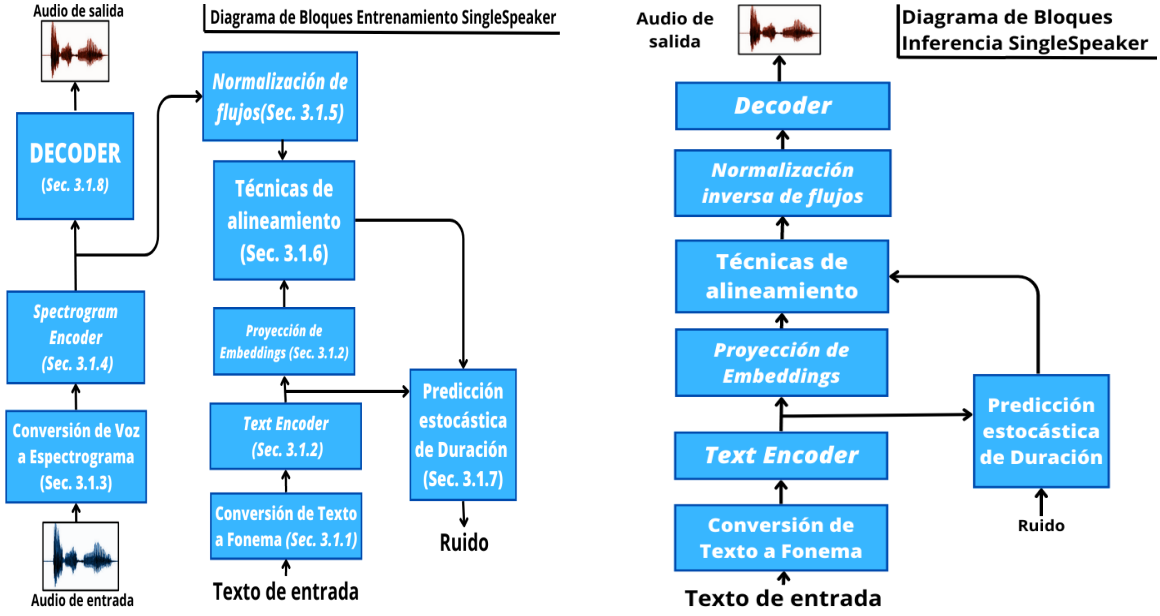


Figura 3.1: Esquema VITS End-to-End

3.1. VITS *SingleSpeaker*

3.1.1. Bloques VITS *SingleSpeaker*

En la Figura 3.1 se explica el comportamiento del modelo *end-to-end*, esto facilita una comprensión sencilla del sistema pero en realidad este modelo está formado por varios bloques de gran complejidad cada uno, como se ve en la Figura 3.2. A continuación, se explicará de manera breve cada uno de los bloques y para una mayor explicación del comportamiento y función de cada uno vea Anexo A.



(a) Entrenamiento. (b) Inferencia.

Figura 3.2: Diagrama de bloques modelo VITS con un solo orador

3.1.1 **Conversión de texto a Fonema** : Recibe como entrada el texto y lo convierte a fonemas de distintos diccionarios fonéticos como IPA[30] o SAMPA[31] (Vea Anexo D).

3.1.2 **Text Encoder** : Es un modelo *transformer* que procesa los fonemas de entrada (c_{text}) y los convierte en una representación compacta dentro de un espacio vectorial que representa al fonema (h_{text}).

3.1.3 **Conversión de voz a Espectrograma y MFCC** : Transforma el audio de dominio temporal al dominio frecuencial su forma de espectrograma y extrae sus características mediante los MFCC.

3.1.4 **Spectrogram Encoder** : Calcula el espacio latente a partir del audio de entrada que está en el dominio frecuencial.

3.1.5 **Normalización de flujo** : Realiza una proyección del espacio latente para mejorar la flexibilidad de la representación procesada por el *autoencoder* (Vea Anexo E.3.1).

3.1.6 **Técnicas de Alineamiento** : Estima el alineamiento entre los fonemas y los grafemas de entrada, relacionando la voz y el texto. El algoritmo usado se denomina *"Monotonic Alignment Search"* o (**MAS**).

3.1.7 **Predicción Estocástica de la duración** : Proporciona una estimación probabilística del tiempo que debe durar cada unidad de habla en la salida.

3.1.8 **Decoder** : Transforma la representación latente de las características acústicas (**z**) en una señal de audio. El bloque se trata de una GAN, cuyo modelo es HiFiGAN [32],

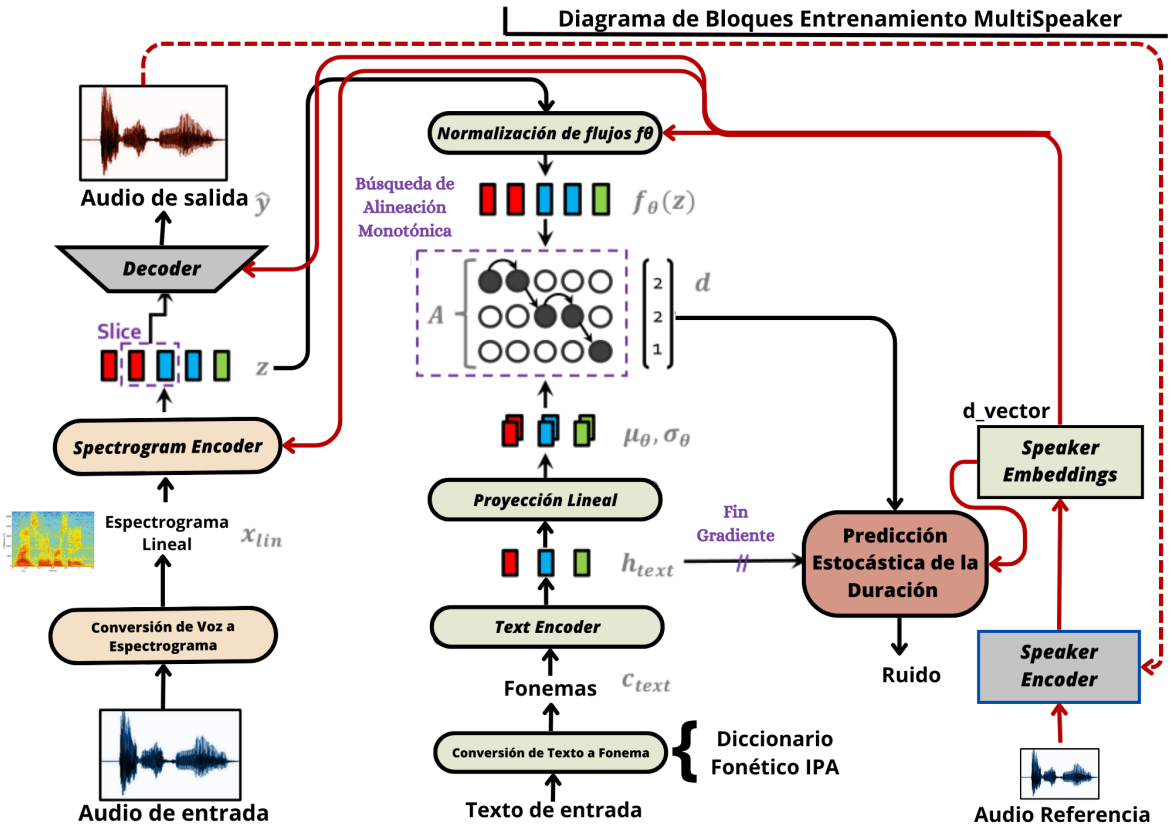
3.2. VITS *MultiSpeaker*

En la parte inferior de la Figura 3.1 se explica el comportamiento del modelo *end-to-end* cuando se incorporan varias voces. En esta sección se añade un nuevo bloque para dotar al sistema de la capcidad de clasificar diferentes personas y como se varía los bloques explicados en la anterior sección. (Figura 3.3)

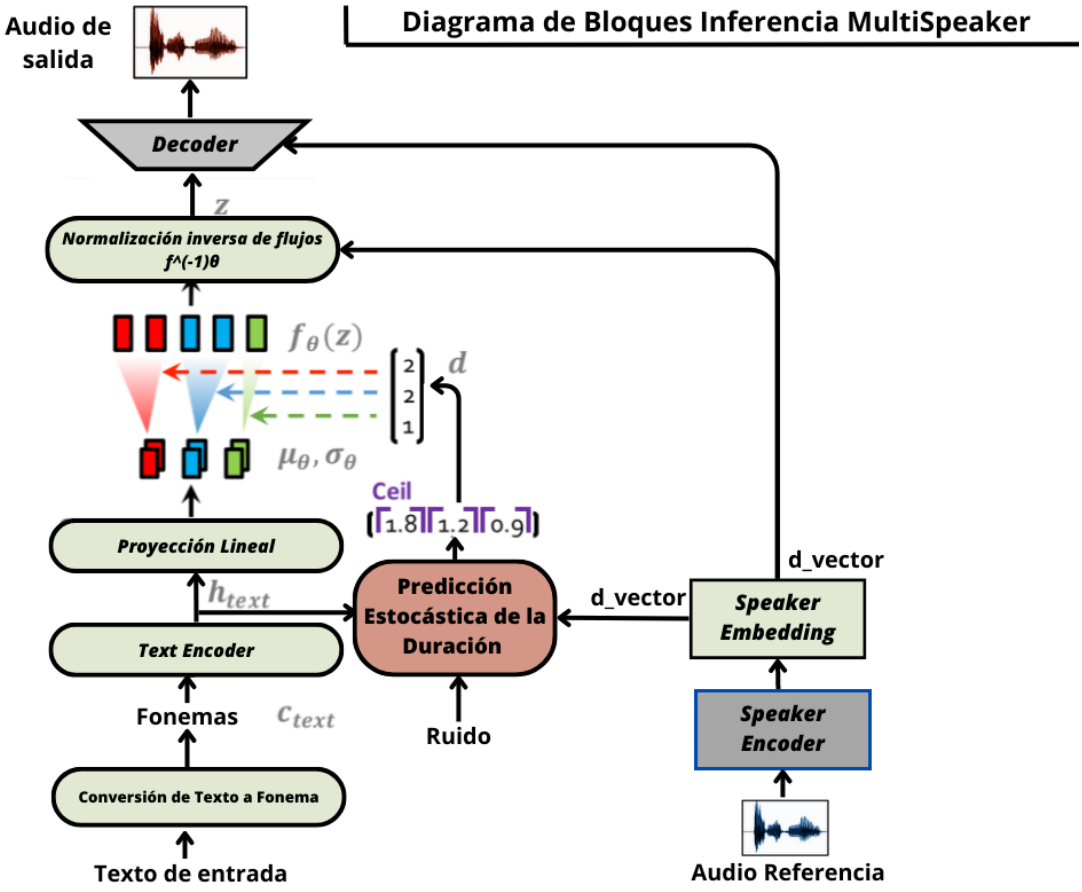
3.2.1. *Speaker Encoder* y *Speakers Embeddings*

El *Speaker Encoder* es un bloque que reconoce las características acústicas de cada persona y las representa en un *embedding*. Se utiliza el modelo ***H/ASP*** [33], que se basa en un *encoder* entrenado con la base de datos *VoxCeleb2*[34], con las funciones de pérdidas *Softmax* y *Prototipical Angular*[35]. Durante el entrenamiento, el *Speaker Encoder* recibe el audio generado y va actualizando los pesos para conseguir una mejor representación optimizando su función de pérdidas.

Con este *encoder* se consiguen obtener unos vectores de dimensión 512, denominados *d-vectors*[36]. Estos vectores representan las características de cada orador y actuarán como una nueva condición en la representación del espacio latente. Esta condición afectará a los bloques que evaluen las características del espacio relacionada con la voz sintetizada incluyendo: *Spectrogram Encoder*, Normalización de flujos, Predictor estocástico de la duración y *Decoder*. En este nuevo escenario las distribuciones de probabilidad tendrán una nueva condición de entrada que será el *d-vector*. Así que si se consigue tener una correcta representación de las características de cada orador se podrá llegar a condicionar un nuevo espacio latente que permita trabajar y distinguir a varias personas con el mismo modelo.



(a) Entrenamiento



(b) Inferencia

Figura 3.3: Diagrama en bloques del modelo VITS *MultiSpeaker*

Capítulo 4

Bases de Datos

Una de las partes más importantes a la hora de entrenar un modelo basado en redes neuronales, especialmente en tareas de aprendizaje supervisado, es la elección de las bases de datos etiquetadas. Para la tarea TTS, se requiere que las bases de datos contengan audio transcrito.

4.1. Elección de las Bases de Datos

La razón por la cual necesitamos bases de datos etiquetadas es debido a que el objetivo del entrenamiento de la red neuronal es ajustar los pesos y las conexiones internas de la red para que pueda generalizar patrones y realizar predicciones precisas sobre nuevos datos de entrada. Las etiquetas proporcionan la "respuesta correcta" asociada a cada ejemplo de entrenamiento, permitiendo que la red aprenda a mapear correctamente los datos de entrada a la salida deseada.

Al iniciar esta búsqueda, se enfrentaron limitaciones significativas, incluyendo la necesidad de contar con un modelo pre-entrenado en español y confiable, así como el requerimiento de bases de datos con múltiples locutores en español y pacientes que presentaran alguna patología vocal.

Se encontraron varios modelos pre-entrenados en español, sin embargo estos no tenían una voz de alta calidad desde el incio del entrenamiento. Por lo tanto, se decide escoger uno de estos modelos inciales y seleccionar la base de datos **Librivox**, que tiene numerosas horas de datos de voz para utilizarla de pre-entrenamiento. También se requiere una segunda base de datos que tenga una significativa cantidad de audio transcrito de varios locutores y se escoge la base de datos **Albayzin**.

En la búsqueda de base de datos de pacientes con voces patológicas en español se estaba más limitado. Se escogió las base de datos **Thalento**, que aunque no está desarrollada específicamente para tarea de clonado de voces, se trata de una de la base de datos con voces patológicas más completa en español. A la vez, esta base de

datos se utiliza en dos etapas, primero se escogen unos pocos pacientes que tienen una mayor cantidad de audio, y luego se utiliza toda la base de datos. La tabla 5.1 presenta información detallada sobre las bases de datos.

Base de datos	Horas de audio	Número de personas
Librivox	53,86	1
Albayzin	8,48	216
Thalento	1,51	29
Thalento sin restricciones	2,2	198

Tabla 4.1: Resumen de bases de datos

4.2. Preprocesamiento de las Bases de Datos

Las bases de datos escogidas son de diversidad significativa y formatos dispares, por lo que antes de entrenar los modelos se realiza un preprocesado de los audios para que tengan un formato acorde a la tarea de TTS. La figura 4.1 describe el esquema de pasos para obtener un formato adecuado y común a todos los audios.

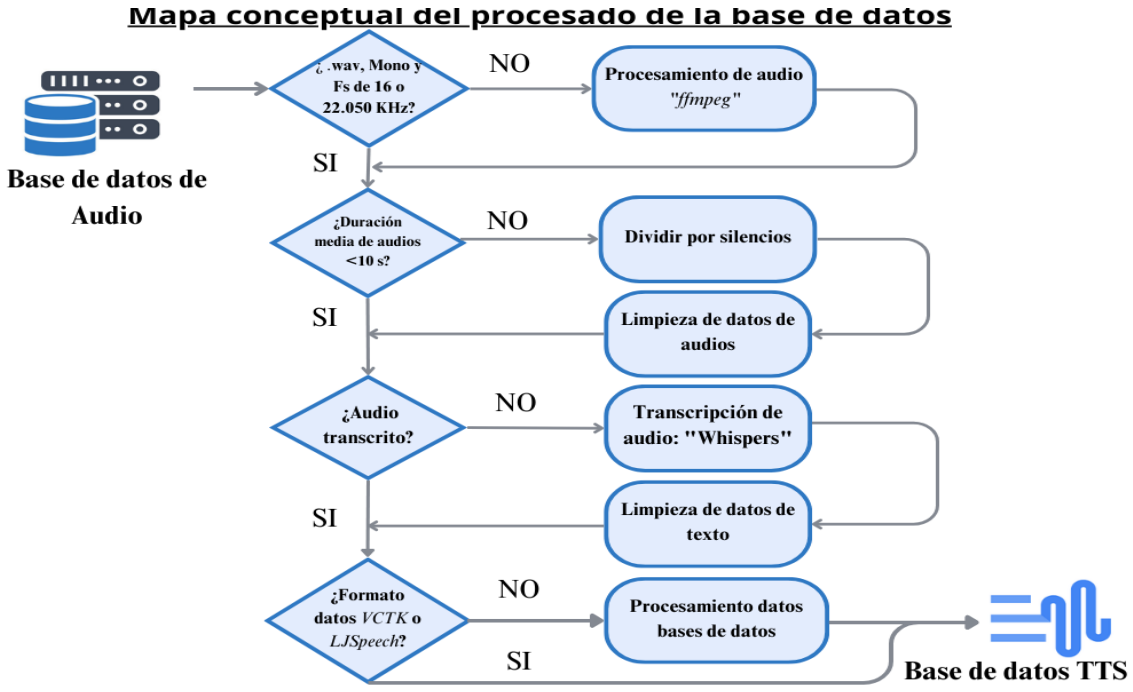


Figura 4.1: Mapa conceptual preprocesamiento de datos

1. **Formato de audio**: Se utilizan audios monocanal, muestreados entre 16000 y 22050 Hz. Una mayor frecuencia de muestreo o emplear audio de dos canales aumentaría la calidad, pero también incrementaría la memoria requerida por el

modelo, lo cual podría ser un factor limitante en estas tareas. Para asegurar un formato adecuado se utilizó el software libre ***ffmpeg***[1].

2. **Fracción de los datos**: Los audios utilizados para entrenar modelos neuronales son de corta duración, entre 2 y 10 segundos aproximadamente. Estos audios ocupan menos espacio de almacenamiento y ayudan a una transmisión eficiente de la información. Así que se detectan y dividen los audios de larga duración en fragmentos de corta duración, utilizando la librería de python *pydub*[2]. Solucionar esto de manera automática y sin cortar palabras es un gran desafío por lo tanto se decidió fraccionar por silencios de duración mayor a 0.4 segundos. Al estar todos los audios grabados en cámaras semianecoicas, se considerará como silencio los fragmentos con potencia inferior a -40 dBs. Sin embargo, esta fragmentación genera problemas cuando existan silencios de larga duración, donde se generan audios de silencio o de muletillas del habla. Tener varios audios de este tipo es un problema a la hora de su entrenar la red, por lo que tras fragmentar el audio se ejecuta otro programa que busque estos audios no deseados con el fin de eliminarlos.

3. **Transcripción de audio**: En el contexto de TTS, contar con audios con transcripciones precisas y revisadas es lo ideal. Sin embargo, en la práctica, muchas bases de datos no cuentan con transcripciones de calidad, lo que dificulta la clonación de voces. Para superar este desafío y obtener las transcripciones de los audios se utiliza el modelo ASR ,*Whisper*[37]. *Whisper* tiene un rendimiento satisfactorio en la transcripción de la mayoría de los audios, generando transcripciones correctas o con errores que fonéticamente no se alejan del audio real. No obstante, en algunos casos, *Whisper* genera errores garrafales usualmente provocados por mala inteligibilidad del audio. Esto hará que se tenga que realizar una limpieza de datos, donde se revisan la transcripciones erróneas para garantizar la calidad y coherencia de los datos.

4. **Formato de la base de datos**: Por último, para que la red neuronal pueda leer los datos adecuadamente, es necesario que estos tengan un formato estandarizado. Existen varios formatos de lectura de datos para tareas TTS, en este caso, se decide usar para tareas de un solo orador el formato **LJSpeech** y para tareas de varios oradores el formato **VCTK**.

[1] https://www.ffmpeg.org/
[2] https://pypi.org/project/pydub/

4.2.1. Preprocesamiento Librivox

Librivox[3] es un sitio web de dominio público que se dedica a audiolibros. Los usuarios tienen tanto la opción de leer libros y subirlos, como de escuchar los audiolibros disponibles. Entre los usuarios, se encuentra uno denominado "*tux753*"[4], que ha subido alrededor de 100 horas de audiolibros grabadas con su voz en español.

Otros usuarios, verificaron que 50 horas de audio dentro de todas las grabaciones coincidían con las transcripciones de los libros y los dividieron en audios de corta duración con su transcripción correspondiente. Desde entonces esta colección de audiolibros se ha establecido como una base de datos de audio en español de uso frecuente y gran importancia[38].

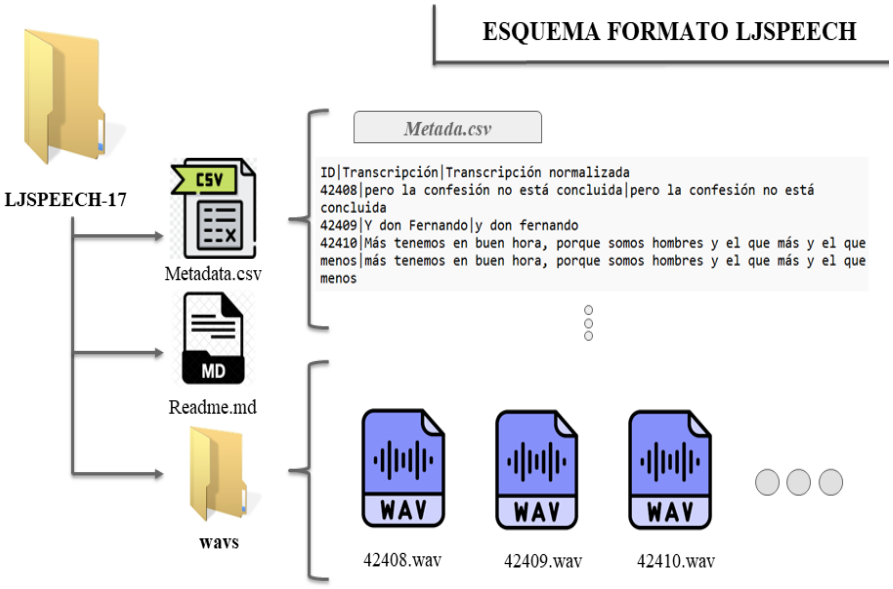


Figura 4.2: Formato *LJSpeech*

Esta base de datos se definió con un formato de distribución similar al usado anteriormente por **LJSpeech**[39]. Tal y como se ve en la Figura 4.2, el formato LJSpeech se basa en una carpeta *wavs* que contiene los audios y un archivo .csv denominado *metadata.csv*, que contiene tres columnas delimitadas por '|'. En el archivo csv, se observa que en la primera columna se indica el ID o nombre del audio, la segunda columna se trata del texto transcrito en formato UTF-8 y la tercera columna de la transcripción normalizada también en formato UTF-8. La transcripción normalizada es el texto transcrito en minúsculas, con números, tanto cardinales como ordinales, y unidades monetarias transcritas. Mientras que en la carpeta *wavs*, tenemos todos los audios formato .wav, monocanal, codificación en 16 bits PCM lineal y frecuencia de

[3] https://librivox.org/
[4] https://librivox.org/reader/3946

muestreo 22050 Hz.

Al ser una base de datos específica para TTS, su formato coincide con el requerido y no habrá que realizar ninguna tarea de preprocesado previo al entrenamiento. En la Figura 4.3, se puede ver como la duración de la mayoría de los audios está entre 2 y 10 segundos, correspondiendo con la duración ideal para entrenar estos modelos. Otro parámetro importante a la hora de estudiar la calidad de las bases de datos, es la cobertura fonética de estas. Una base de datos con variabilidad fonética es aquella en la que aparecen todos los fonemas usados en el lenguaje con una distribución similar a la probabilidad de ocurrencia en el habla. En la Figura 4.4, se aprecia como aparecen todos los fonemas hablados en español de manera equilibrada. Finalmente se hizo otro estudio donde se analiza la velocidad de habla de las personas, con los resultados recogidos en una página de Github[5]. En este estudio se aprecia como su ritmo de habla es de alrededor de 15 letras por segundo y con poca variación de ritmo en sus audios.

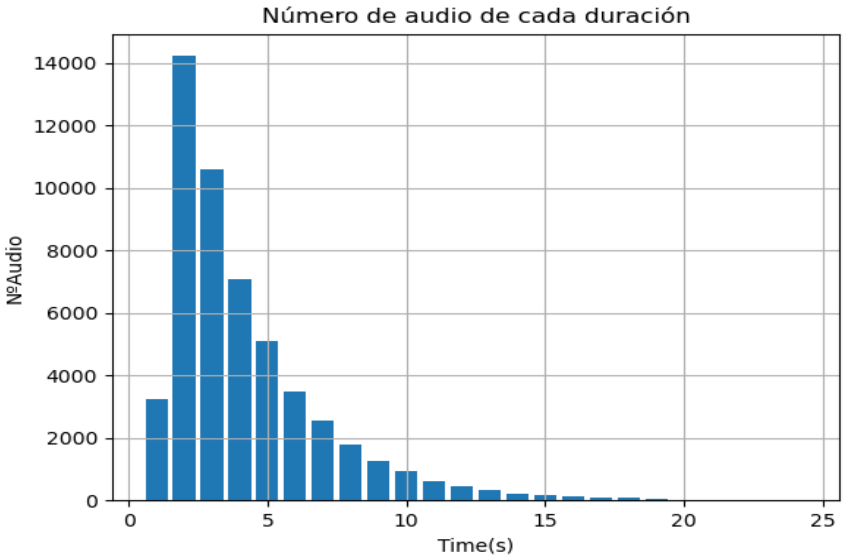


Figura 4.3: Duración de los audios Librivox

4.2.2. Preprocesamiento Albayzin

Albayzin[7] se trata de una base de datos que consiste de 5 sesiones de grabaciones de oraciones cortas en español. Dentro de esta base de datos hay grabaciones de 216 personas que dicen 25, 50, 75 o 250 oraciones, de duración entre 2 y 5 segundos.

En la tabla 4.2, se observa que el número de personas con 25 oraciones es bastante superior a los que tienen una cantidad mayor de audio (75 o 250 oraciones). En la gráfica 4.5, se ve que casi el tiempo medio de las personas con 25 oraciones es de 80

[5] https://santirf01.github.io/TFG_Image_page.io/

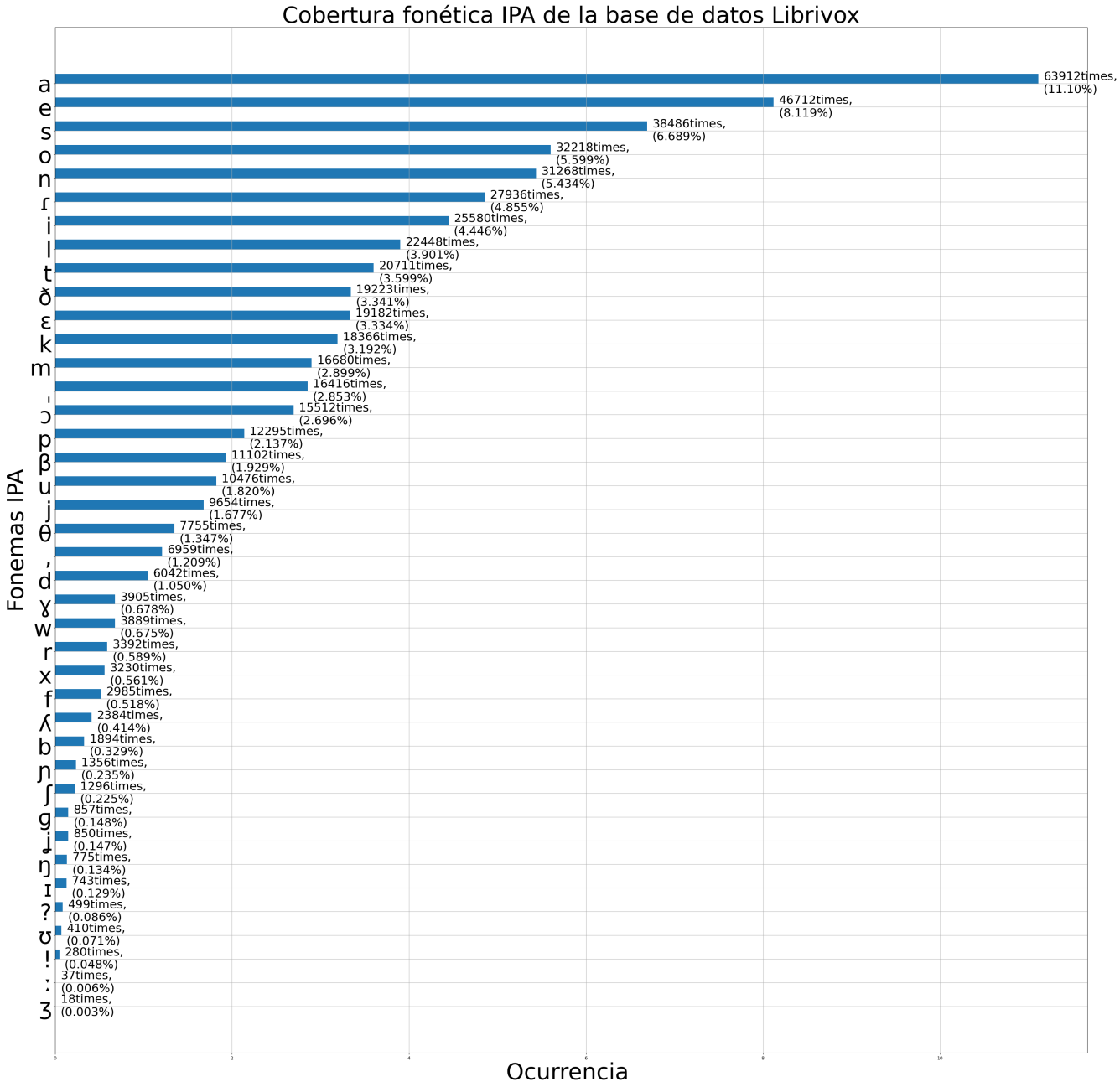


Figura 4.4: Cobertura Fonética IPA Librivox

segundos y de los de 50 oraciones alrededor de 3 minutos, siendo bastante limitado. En [3] se comenta que con estos sistemas la cantidad mínima de audio para poder conseguir resultados inteligibles es de 20 oraciones y que la calidad va aumentando

Número de audios	Persona
25	“nf, tc, oj, ta, tp, jn, op, tg, ch, ba, cj, tb, xp, bf, tj, oo, nk, go, ja, cn, ga, gk, ck, bk, cb, be, og, ob, gb, tm, xd, te, cl, oh, bc, nm, jk, cp, om, xh, xk, xo, oa, jm, nb, gn, gj, tk, cg, xe, bn, xl, gi, jd, cc, gc, bg, gm, jf, oi, oe, ci, xc, ng, ti, jo, nd, nl, on, np, bp, jj, jc, xg, nj, bb, gl, bi, bl, cm, xb, bd, jh, jb, nn, xm, ca, tn, od, xa, oc, gf, cf, ce, tl, gh, ge, nc, td, xf, na, to, xi, bo, gd, cd, ok, th, bh, ne, jg, co, xj, jp, bj, je, gp, ni, bm, ji, tf, gg, of, xn, nh, no, ol, jl”
50	“ha, df, he, dj, ke, hc, vn, hb, kd, pa, ye, db, yc, kb, hd, yb, di, ua, uc, de, ph, pj, im, ez, in, pb, dh, dd, rz, ub, rx, pg, pe, zk, ud, ka, ue, ya, pc, pd, vm, ey, lk, dc, ry, pf, pi, ex, yd, kc, da, dg”
75	“gt, tr, xt, nv, tu, xr, jt, gr, gs, tt, nt, nr, ts, bw, nw, js, bv, ns, br, nu, ju, by, bu, nx, jr, bt, gu, xu, ny, bs, xs, bx”
250	“ma, mb, ab, aa”

Tabla 4.2: Número de audios por persona Albayzin

de forma progresiva conforme mayor cantidad de audio se tiene. Por eso para estas personas se necesitará todo el audio en entrenamiento.

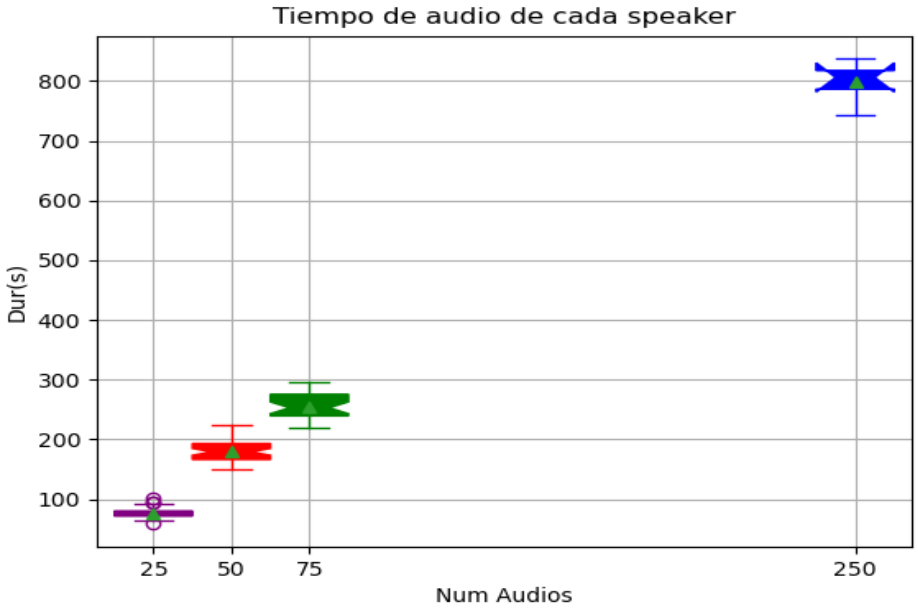


Figura 4.5: Tiempo medio por cada persona

Aunque está base de datos no tenga excesivo audio, sus grabaciones están fonéticamente balanceadas. Esto quiere decir que los contextos fonéticos de mayor relevancia en el español se encuentran repetidos al menos en cuatro instancias, es decir, aquellos que están presente en un mínimo del 10 % de las veces que nos comunicamos a través del habla. Esto es de gran importancia y tendrá un impacto beneficioso en los resultados. En la Figura 4.8, se puede apreciar como la distribución fonética que

siguen las cuatro personas con 250 oraciones como cuando se utiliza toda la base de datos está muy equilibrada, haciendo la conversión vista en Anexo D se representan todos los sonidos revelantes en el español.

Respecto al formato de la base de datos, Albayzin está formada por archivos *.seo* y *.ses*. Los archivos *.seo* son archivos de texto que contienen información como la edad de la persona, la ciudad, la tasa de muestro, la transcripción del audio, etc. Así que se hizo una lectura de estos archivos para copiar la transcripción del audio y guardarlo en un archivo *.txt*. Mientrás que el archivo *.ses* es el audio sin cabecera. Este audio tenía una codificación 16 bits PCM lineal, *little endian*, monocanal y una frecuencia de muestreo de 16 kHz. Aunque el formato del audio coincide con el formato requerido para que sea aceptado por el modelo había que introducirle cabecera usando el comando:

```
ffmpeg -f s16le -ar 16000 -ac 1 -i {dir_entrada} {dir_salida}
```

El formato de salida seleccionado para la lectura es **VCTK** [40]. Este se estructura en dos directorios principales denominados *txt* y *wav48*. Estos directorios contienen un subdirectorio por persona dentro de la base de datos. Dentro de estos subdirectorios, se encuentran todos los archivos de audio y ficheros de texto con el mismo nombre de fichero, lo que permite establecer una correspondencia uno a uno entre los archivos de audio y su transcripción textual asociada.

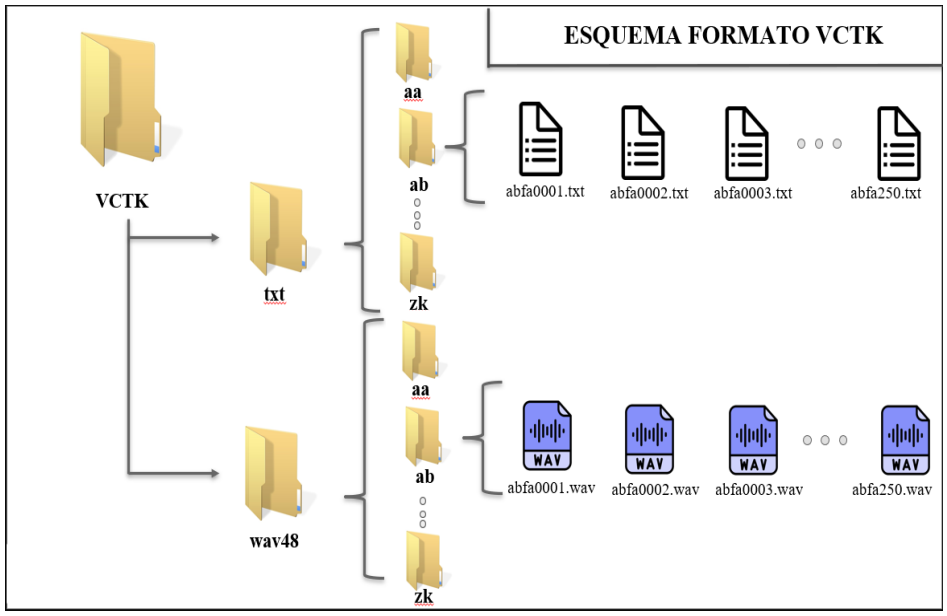


Figura 4.6: Esquema VCTK

En las gráficas obtenidas en formato html[6], se observa como el ritmo de habla de esta base de datos es menor que Librivox, con una media de 12 letras por segundo, y con una mayor variación en la duración de audios con el mismo número de letras. Esto

[6] https://santirf01.github.io/TFG_Image_page.io/

último es debido a que al tener varios oradores, el ritmo de cada uno será diferente. Además, se representa los valores de los *embeddings* d-vectors de los *Speakers*. Como estos vectores tiene una dimensión-512, es necesario realizar una proyección de estos a través de UMAP[41] (Figura 4.7). En estas representaciones se aprecia como los audios son clasificados correctamente dentro del espacio dimensional donde se encuentran todos los audios de cada persona. Así se llega a la conclusión de que el *Speaker Encoder* consigue obtener las características acústicas de cada persona y es capaz de caracterizarlo vectorialmente, aunque una proyección no deje de ser una aproximación del espacio vectorial y no sea exacta.

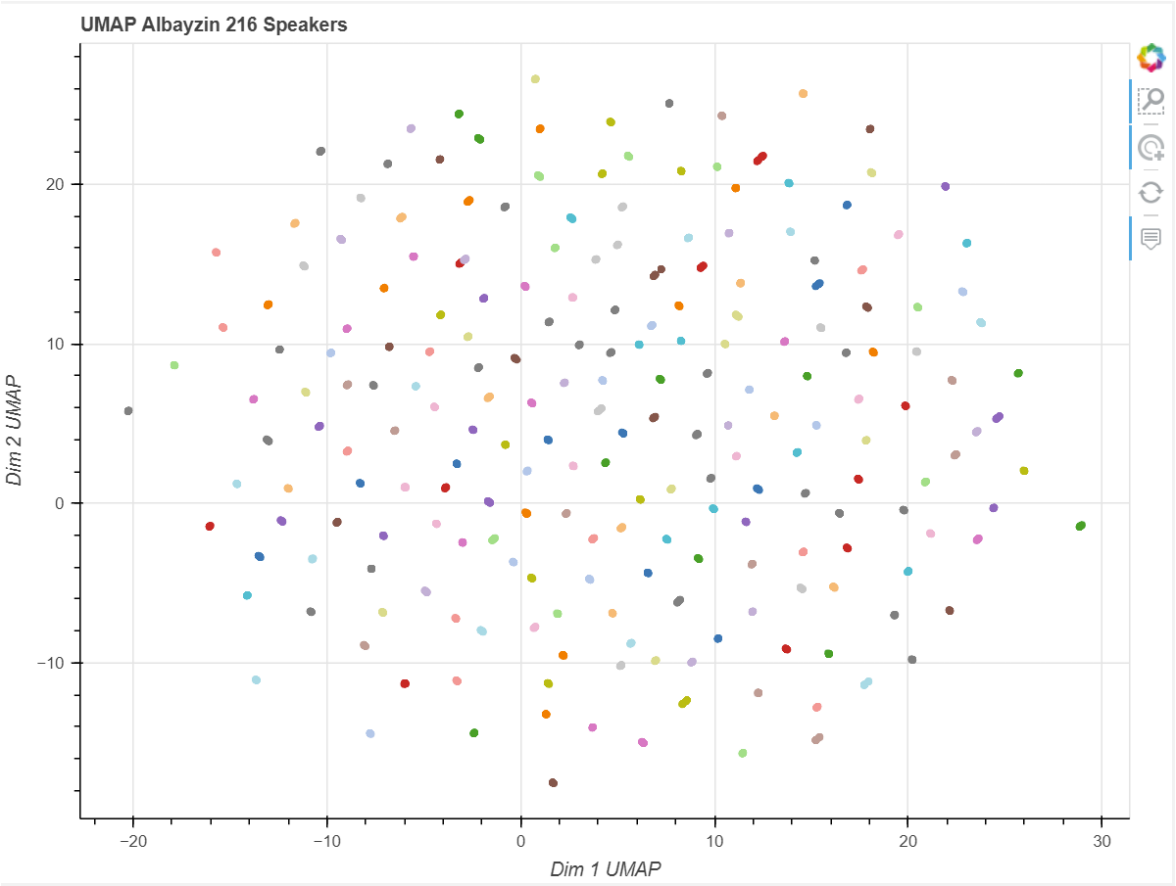


Figura 4.7: Proyección UMAP Albayzin 216 personas. Figura html: https://santirf01.github.io/TFG_Image_page.io/

4.2.3. Preprocesamiento Thalento

Thalento es una base de datos grabada por el grupo Vivolab de la Universidad de Zaragoza que actualmente dispone de la voz de 196 personas divididas entre voces sanas y patológicas. Está diseñada para detectar y evaluar automáticamente las patologías de la voz. El protocolo de grabación está asociado a los exámenes que realizan los otorrinolaringólogos y logopedas del hospital Clínico de Zaragoza. Los

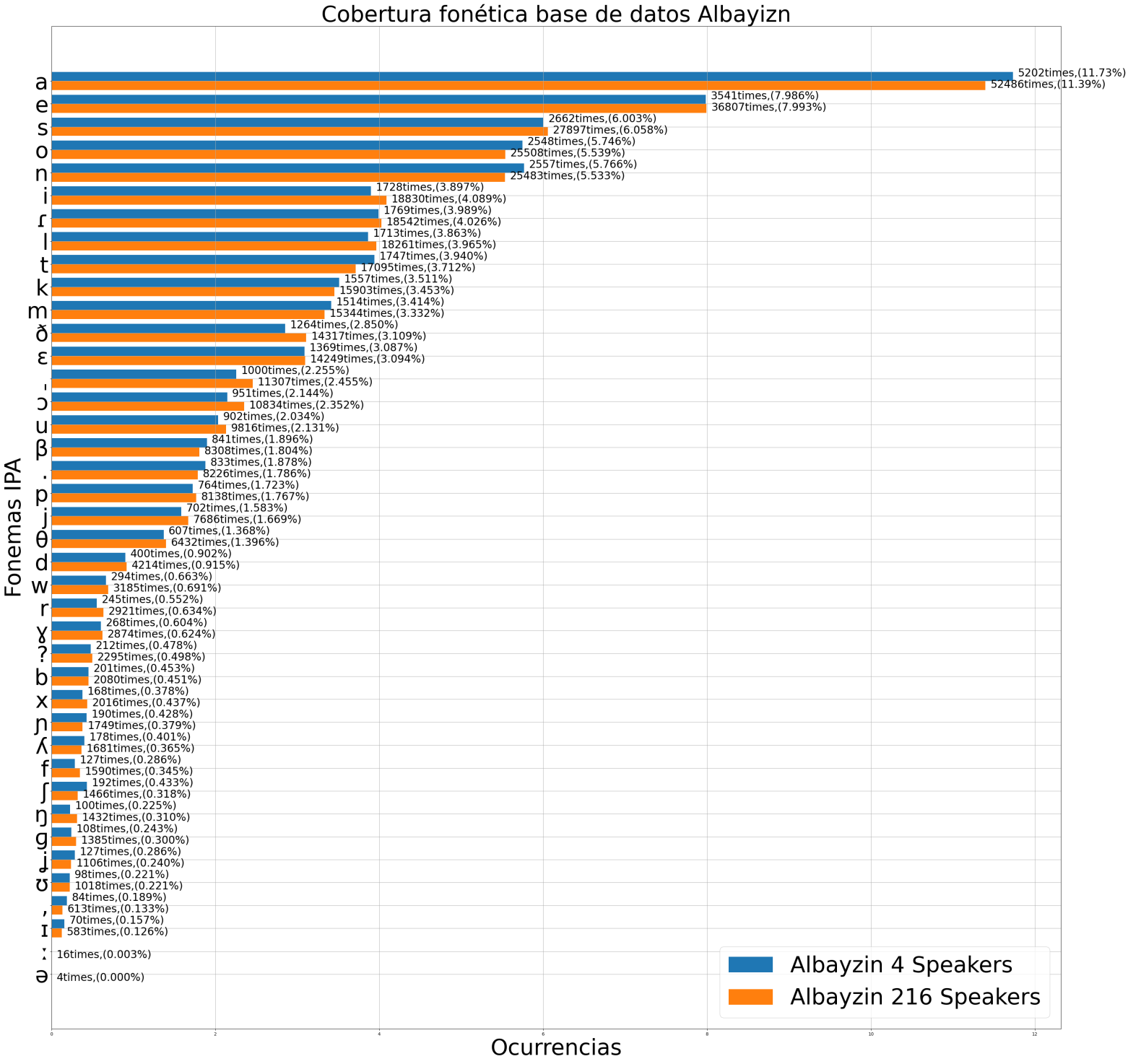


Figura 4.8: Cobertura fonética Albayzin

audios se grabaron en la cabina insonorizada de la consulta de otorrinolaringología del hospital e incluyen vocales sostenidas, cambios de escala, lectura, diálogo y canto. En este trabajo se utilizaron los audios donde los participantes leen la fábula “El

pastor y el lobo" y describen una imagen sumando alrededor de cuatro minutos de audio. Aquí surgen dos problemas: 1) Estos cuatro minutos de audio incluyen silencios e imprecisiones propias del habla espontánea, por lo que el audio neto disponible es mucho menos de cuatro minutos. 2) El audio de habla espontánea se dejó de grabar tras la época de pandemia, por lo que la mayoría de personas solo disponen del audio de lectura correspondiendo a una duración inferior a un minuto. Entonces se decide dividir esta base de datos en dos tareas: 1) pacientes con diálogo y lectura, 2) pacientes con lectura.

Todos los audios están grabados a una frecuencia de muestreo de 48 kHz, por lo que se debe hacer una conversión de formato a una frecuencia 16 kHz, manteniendo las características de codificación 16 bits PCM lineal, *little endian* y monocanal. Para eso se aplica el comando:

```
ffmpeg -ac 1 -i {dir_entrada} -ar 16000 {dir_salida}.
```

Con el formato correcto, se procede a separar el audio por silencios y eliminan las muletillas (ejemplo: *"Ehmmm"*) en los dialogos. Este paso es importante porque un uso excesivo de muletillas afectaría a los resultados. Además, los pacientes con una patología severa tienden a hacer pausas para respirar y se generan muchos audios de solo respiración. Al fragmentar el audio, también se procede a la transcripción del audio utilizando *Whisper* y se revisan estas transcripciones buscando errores graves. Por último se vuelve a utilizar el formato VCTK como en la Figura 4.6.

En el estudio hecho en la página[7], se ve como el tiempo medio al procesar los audios ha disminuido considerablemente, teniendo en la primera tarea una cantidad cercana a los cuatro minutos en los mejores casos y dos minutos y medio en los peores. Respecto a la segunda tarea, la cantidad de audio media ronda los 40 segundos, siendo una cantidad pequeña para esta tarea. Además de tener poca cantidad de audio, se ve como la duración es bastante disforme, con un ritmo de habla superior a 20 letras por segundo. Esto es debido a que al eliminar los silencios y muletillas solo hay audios de corta duración donde se comentan varias palabras. También se ve que la proyección de *Embeddings* con *UMAP* hay algunos audios que se clasifican fuera del espacio vectorial conforme a todos los audios de esa persona, aunque se haya realizado una revisión de los audios del paciente. Cuando se realiza esta proyección de todos los pacientes de la base de datos se ve un espacio más confuso y con mayor mezcla entre distintos pacientes. Aunque, en realidad en el espacio vectorial si se encuentran separados pero menos que las personas de Albayzin (Figura 4.9). Esto hará que sea más complicada la tarea de clonado de voces.

[7] https://santirf01.github.io/TFG_Image_page.io/

Por último, en la Figura 4.10 se aprecia como la distribución fonética es bastante pobre y no sigue una distribución balanceada. Además en esta figura, se aprecia que aparecen fonemas no usado en el español y que no se encuentran en los audios. Estos fonemas son generados por errores de transcripción de Whisper. Se llega a la conclusión que esta base de datos será la más difícil de abordar debido a que su función principal es la detección de patologías. Encontrarse con bases de datos no ideales, va a ser una situación que muchas veces se enfrentan estos tipos de sistemas.

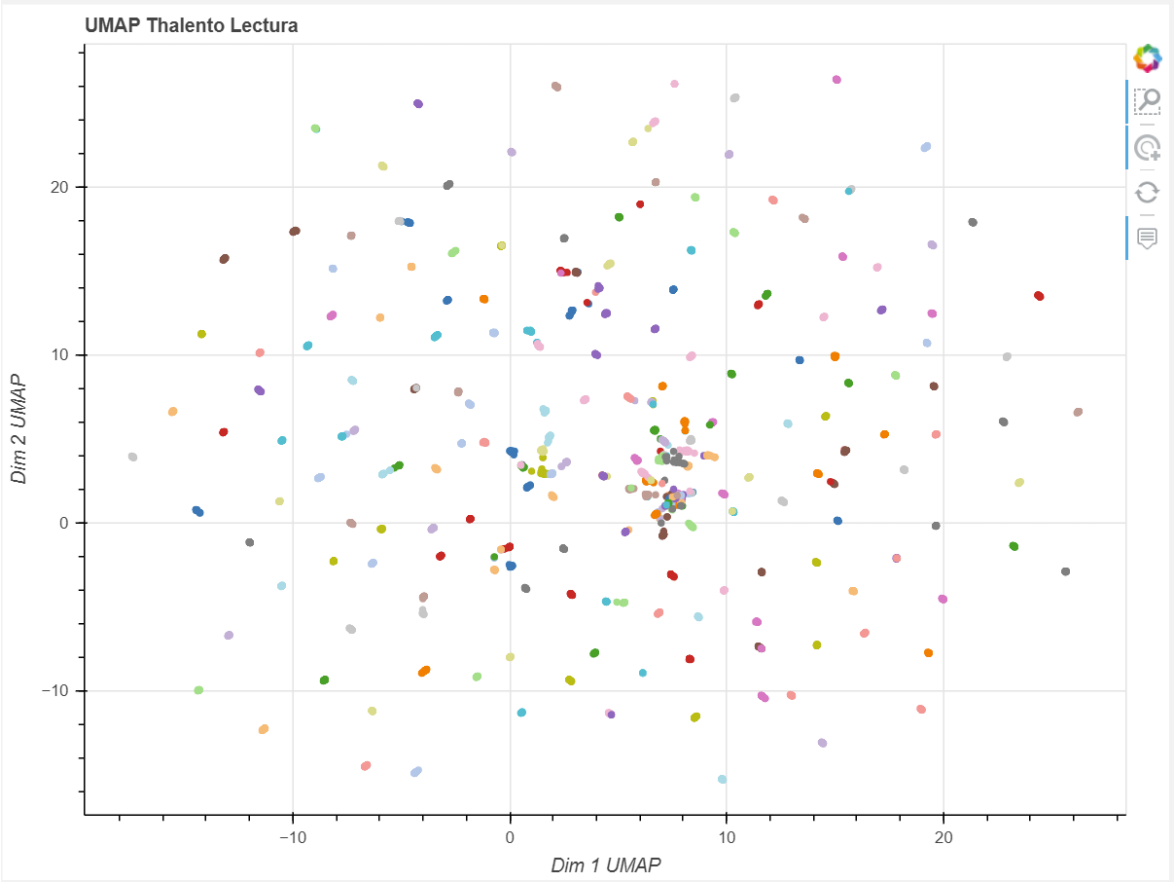


Figura 4.9: Proyección UMAP Thalento Lectura. Figura html: https://santirf01.github.io/TFG_Image_page.io/

4.3. Partición de los datos

Una vez ya procesados los datos, se procede a la partición de estos en tres bloques: entrenamiento, validación y prueba. La cantidad de datos usados en validación está limitada por el consumo de memoria en la tarjeta gráfica. Durante el bloque de validación de datos se comprobó que el consumo de memoria aumentaba de manera significativa, siendo esto un factor limitante a priori. Para mitigarlo, se disminuyó el tamaño del batch a 16 en la evaluación, incluso en algún caso llegando a tener que

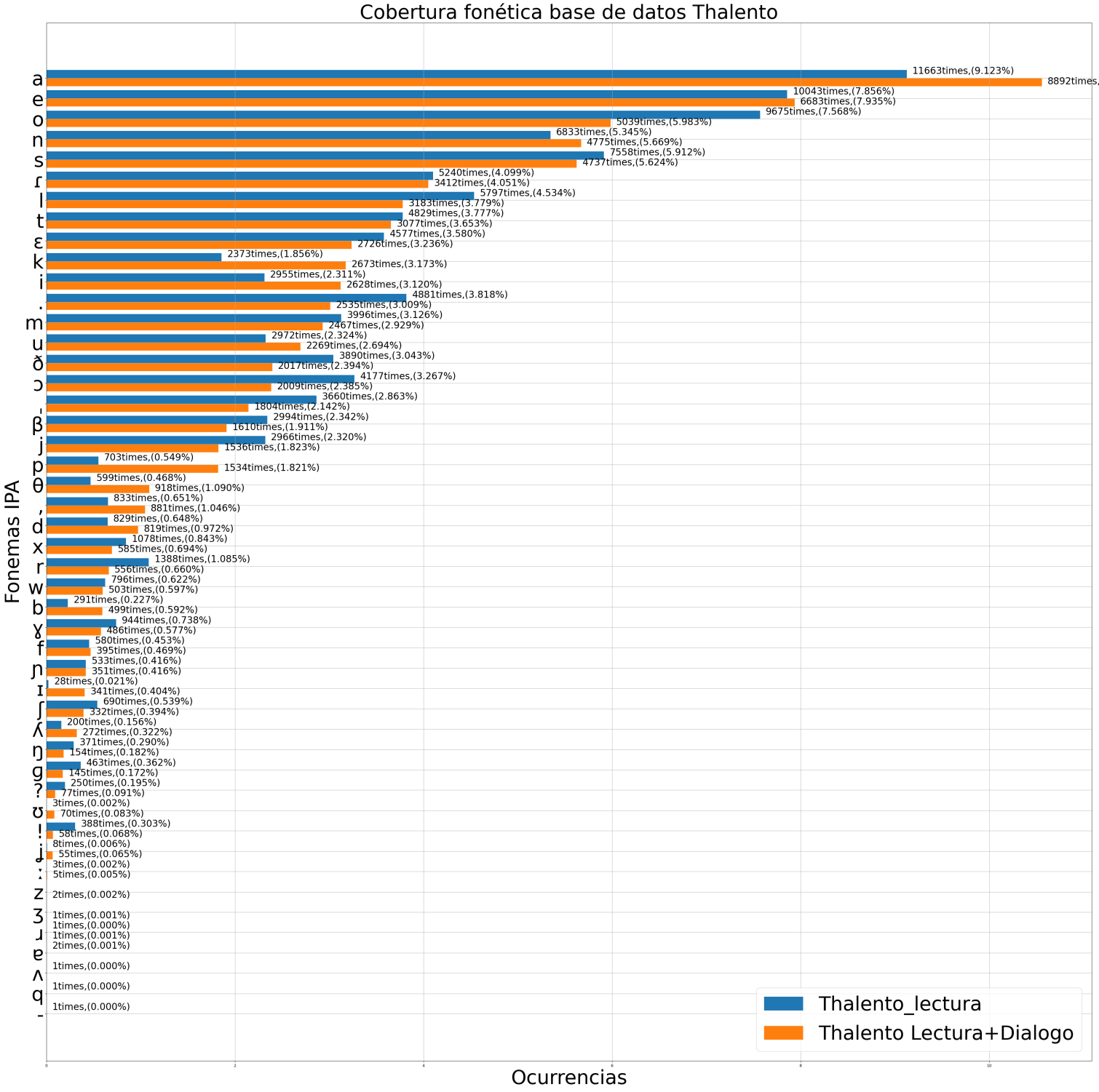


Figura 4.10: Cobertura Fonética Thalento

definirlo a 8, y la partición de datos repartidos a validación sería de 1 % en todas las bases de varios oradores y del 0.1 % en Librivox.

Figura 4.11: Partición de los datos

Respecto a los datos del segmento de prueba se tuvo distinto punto de vista dependiendo la base de datos que se usará. En la base de datos **Librivox** para TTS, están recogidas 50 horas de audio de las 100 horas grabadas por *tux753*. Como la base de datos no contenía todos los libros del usuario, se entrenó el modelo con toda la base de datos y luego se descargaron audios del libro *"Zaragoza"* de Benito Pérez Galdós para usarlos como datos de prueba.

En la base de datos Albayzin, al ser pequeña la cantidad de audio en gran parte de los oradores, se decidió dividir en dos etapas. La primera donde solo se entrenaría con los 4 oradores que tienen mayor cantidad de audio, y una segunda donde se entrenará con toda la base de datos. En la primera etapa, como todas las personas tienen una cantidad de 250 frases, se quitan las 20 primeras frases del entrenamiento y se usan en prueba. Mientras que en la segunda etapa, como la mayor parte de los usuarios tienen tres o menos minutos de audio, se ha entrenado el modelo con todo el audio disponible.

Por último en Thalento, se tiene el mismo problema que en la segunda etapa de Albayzin, que el audio en todas las personas es menor de 4 minutos, llegando a estar alrededor de 40 segundos en la segunda etapa. Por lo tanto, también se entrenó el modelo con todo el audio disponible.

Base de datos	Datos entrenamiento	Datos validación	Datos test
Librivox	53,81 h	0,054 h	1 capítulo
Albayzin 4 oradores	0,80 h	0,008h	20 frases
Albayzin	8,40 h	0,0848 h	No test
Thalento	1,50 h	0,0151 h	No test
Thalento sin restricciones	2,2 h	0,022 h	No test

Tabla 4.3: Resumen de bases de datos

Capítulo 5

Fases y Métodos

5.1. Pérdidas

Con la combinación del entrenamiento del VAE y de la GAN, la pérdida total del entrenamiento del sistema puede ser expresada como la suma ponderada de varias pérdidas individuales:

$$L_{vae} = \alpha_{mel} \cdot L_{recon} + \alpha_{KL} \cdot L_{kl} + \alpha_{dur} \cdot L_{dur} + \alpha_{adv} \cdot L_{adv}(G) + \alpha_{fm} \cdot L_{fm}(G)$$

con:

- $\alpha_{mel} = 45$.
- $\alpha_{KL} = \alpha_{dur} = \alpha_{adv} = \alpha_{fm} = 1$.

(5.1)

Se observa como a la pérdida que mayor importancia se le da es la pérdida de reconstrucción.

donde:

- L_{recon}: Es la pérdida por reconstrucción que mide la diferencia entre el audio objetivo y el audio de referencia.
- L_{kl}: Es la pérdida por la divergencia KL, mide la diferencia entre la distribución a posteriori y a priori.
- L_{dur}: Es la pérdida de la duración de los fonemas introducida por el bloque de Predicción Estocástica de la Duración.
- $L_{adv}(G)$: Es la pérdida adversarial del generador, que se trata de la función de error cuadrático del generador. Este mide las muestras sintetizadas consideradas por el discriminador como correctas.
- $L_{fm}(G)$: Es la pérdida de las características del generador, que mide la diferencia entre las características del audio referencia y las características intermedias generadas.

Además hay que tener en cuenta la pérdida del discriminador que se obtiene como la suma de todas las pérdidas de los subdiscriminadores:

$$L_{disc} = L_{disc0} + L_{disc1} + L_{disc2} + L_{disc3} + L_{disc4} \tag{5.2}$$

donde:

- L_{disc}: Es la pérdida adversarial del discriminador, que se trata de la función de error cuadrático del discriminador. Esta pérdida evalúa la capacidad de distinguir las muestras del discriminador.

Para una explicación en más detalle de cada una de las pérdidas vea Anexo B.6.

5.2. Entrenamiento

El proceso de entrenamiento de una red es un proceso iterativo a partir del cual una red ajusta sus pesos y parámetros internos para aprender a realizar una tarea específica.

5.2.1. *Zero-Shot Adaption*

El *Zero-Shot Learning* es una técnica de aprendizaje profundo con el objetivo de capacitar a una red para clasificar datos que no ha visto en el proceso de entrenamiento. Para esto, en lugar de aprender los ejemplos de entrenamiento, el modelo llega generalizar y encontrar patrones en los datos.

En este trabajo es fundamental tener datos de la voz a clonar, por lo que el *Zero-Shot Learning* no es factible. Por esta razón, en tareas de generación de datos se recurre a una extensión llamada *Zero Shot Adaption*[29]. En este caso el modelo no tiene acceso a datos de la tarea objetivo durante el entrenamiento, sino que los ve por primera vez en la inferencia. El modelo es capaz de adaptar sus pesos según la entrada que recibe en la inferencia.

En el contexto del TTS, los modelos basados en *Zero Shot Adaption*, utilizan el *Speaker Encoder* para extraer el *embedding* del nuevo orador a partir de audios de referencia de unos pocos segundos. Estos *embeddings* actuan como condición a la red que conseguirá generar audio con características similares al audio de referencia. Este enfoque parece ideal, ya que el modelo no necesita ninguna adaptación de los pesos. Sin embargo, en la práctica, la adaptación de los pesos es complicada y no produce resultados de alta calidad, especialmente cuando la voz de referencia es muy diferente a la voz de origen.

5.2.2. *Fine-Tuning* y *Few-Data Adaption*

El ***Fine-tuning*** y ***Few-Data Adaption*** son dos enfoques comunes para aprovechar modelos ya pre-entrenados. Ambas técnicas se basan en la idea de reutilizar el conocimiento previo de un modelo.

El *Fine-Tuning* se refiere a una técnica que toma un modelo pre-entrenado con un conjunto de datos amplio y a partir de este punto se continua el entrenamiento con un conjunto de datos específicos a la tarea. Su objetivo es aprovechar el conocimiento previo del modelo y abordarlo en una tarea de la que se dispone una menor cantidad de datos y que no sería posible abordarlo desde cero.

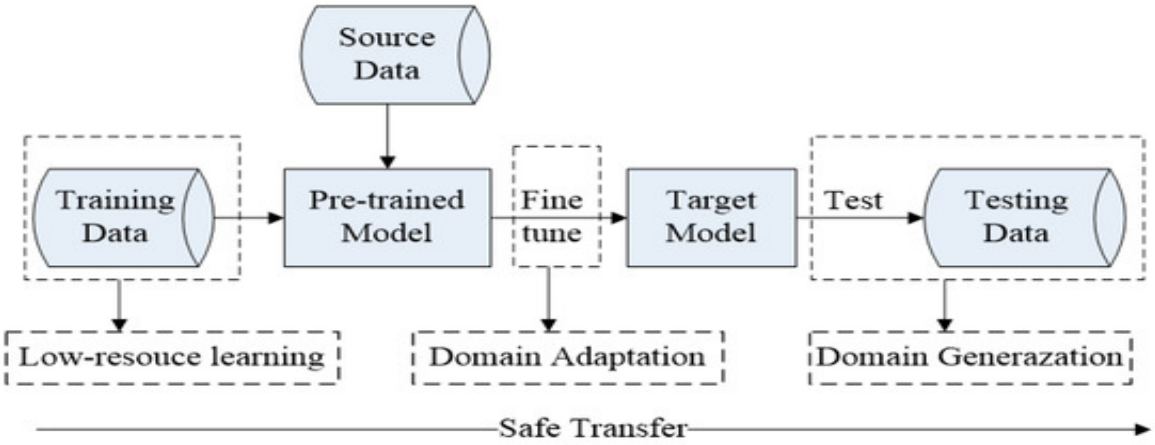


Figura 5.1: Esquema *Fine-Tuning*[42]

Respecto a *Few-Data Adaption*, o también llamado *Few-Shot Adaption*, se centra en adaptar el modelo ya entrenado previamente para trabajar con una cantidad limitada de datos disponibles. En esta técnica no se desea continuar el entrenamiento del modelo pre-entrenado, sino que se busca ajustar sus pesos para que se adapte lo máximo posible a la tarea.

En TTS, es muy común tener una cantidad limitada de audio para entrenar el modelo, por lo que ambas técnicas son muy usadas. Las técnicas de *Few-shot adaption*[43] solo utilizan pocos minutos de audio de la persona objetivo. De estos audios se obtiene el *embedding* de referencia y se ajustan los pesos para la tarea requerida.

Las técnicas de *Fine-Tuning* implican tener que adaptar todos los parámetros de la red a la nueva tarea. Al realizarlo se generan audios de mejor la calidad, pero también aumentaría el coste computacional. Para mejorar esto, en *Adaspeech*[45] se propuso solo realizar la adaptación en los bloques que fueran relativos a las capas condicionales al nuevo audio o *speaker embedding*, generando buena calidad y mejorando el coste computacional.

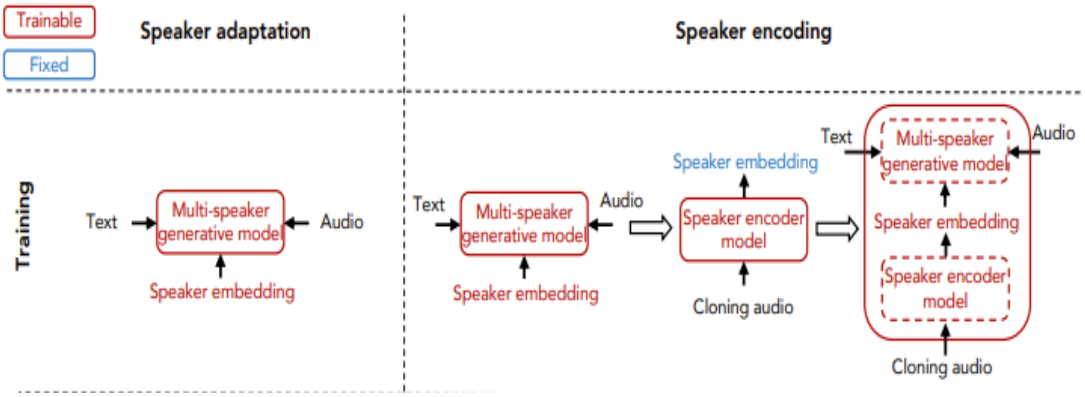


Figura 5.2: Esquema *Few Data Adaption*[44]

5.2.3. Resultados Entrenamiento y Validación

Para entrenar el modelo en las diferentes etapas hay que definir algunos hiperparámetros que caracterizan el entrenamiento

1. **Tamaño de batch** o ***batch size***: Número de muestras que se utilizan en cada iteración del entrenamiento, donde los datos se dividen en lotes para la actualización de los pesos. Un tamaño de batch pequeño conlleva un gradiente con un menor número de muestras, produciendo mayor variabilidad en la actualización de los pesos. Por contra, un tamaño de batch grande implica una mayor demanda de memoria y una convergencia más lenta. En consecuencia, para encontrar el tamaño óptimo, se debe encontrar un equilibrio entre el rendimiento del sistema y los recursos disponibles.

2. **Épocas** o ***Epochs*** del entrenamiento: Número de veces que se recorre el dataset de entrenamiento. Como las bases de datos disponibles son pequeñas, se tiene que recorrer varias veces, por lo que el número de épocas será alto, teniendo cuidado de evitar sobreajustes.

3. **Iteracciones** o **Steps** de entrenamiento: Son los pasos en los cuáles la red neuronal actualiza sus pesos y avanza hacia el intento de convergencia de la red. El número de steps tiene una relación inversa con el tamaño de batch y una directa con el número de épocas.

4. **Técnica de optimización (Adam)**: El algoritmo Adam[46] es el método de optimización utilizado para ajustar los pesos de la red neuronal. Adam adapta automáticamente las tasas de aprendizaje para cada parámetro en función de estimaciones de primer y segundo orden de los gradientes ayudando a acelerar la

convergencia (Vea Anexo E.1.6).

Como se ha visto en el apartado 5.1, este modelo está optimizado por varias funciones de pérdidas y es difícil llegar a una optimización de todas. De aquí las siguientes interrogantes:

¿Cuántas iteracciones se debe realizar para llegar a esta en cada escenario? Es muy importante tener cuidado debido a que un número alto de iteracciones pueden llevar a un sobreajuste de los datos u *overfitting*. En un modelo generativo el *overfitting* puede ser visto como muestras sintéticas muy parecidas a los datos de entrenamiento, careciendo de diversidad y generalización, generalmente cosa no deseada. Para evitar este suceso, se comprobarán las pérdidas en entrenamiento y en validación, una disminución de las pérdidas de entrenamiento con aumento de los pérdidas de validación, será un síntoma de *overfitting* en el modelo.

¿Cuándo se considera que el modelo es óptimo para el trabajo? Se guardan *checkpoints* del modelo cada 2000 iteracciones y cada 10000 iteracciomes se escoge el modelo con menor pérdida total. Así, al final se tienen los últimos *checkpoints* y el modelo de menor pérdidas y se determina cuál es mejor modelo dependiendo de la calidad del audio de salida, considerando que frecuantemente una menor función de pérdidas no implica una mayor calidad de los audios. En estos casos los hiperparámetros definidos están limitados por el coste computacional y se definen en la tabla 5.1.

Base de datos	Batch	Épocas	Steps	Entrenamiento
Librivox	32	400	125k	99.9 %
Albayzin 4 oradores	64	4000	53k	99 %
Albayzin 216 *Speakers*	64	400	57k	99 %
Thalento	48	2500	128k	99 %
Thalento sin restricciones	32	800	120k	99 %

Tabla 5.1: Resumen de hiperparámetros en entrenamiento

Para monitorizar la evolución de las pérdidas del sistema y poder escuchar la evolución de la calidad de los audios sintetizados, se utiliza la interfaz de **"wandb"**[1], donde se exportan los resultados, permitiendo observarlos de una manera sencilla. De esta interfaz se obtuvo la evolución de las pérdidas en todos los escenarios como se muestra en las figuras 5.3 para entrenamiento y 5.4 para validación.

Con respecto a la pérdida KL (Figura 5.3a y 5.4a), se observa un patrón común en todos los escenarios del entrenamiento. Inicialmente disminuye este valor en los

[1]Para ver todos los resultados obtenidos en las simulaciones entrar al enlace: https://wandb.ai/santi-cabila/TFG_809622?workspace=user-santi-cabila

primeros *steps* y luego se estabiliza en un valor constante. En contraste, en la fase de validación, se observa que hay escenarios, como puede ser Librivox y Albayzin 216 *Speakers*, donde está pérdida se queda casi constante. Sin embargo, en los otros tres escenarios se registra un aumento en la pérdida KL durante la validación pero no en el entrenamiento. Inicialmente puede parecer un síntoma de sobreajuste afectando de manera negativa al sistema, pero en realidad interesa seguir entrenando el modelo. Lo que ocurre es que en modelos de gran complejidad pueden surgir fenómenos inesperados. El aumento de la pérdida KL puede ser debido a que el modelo está tratando de ajustar las características generadas a una distribución latente específica, lo cual no necesariamente indica que el audio sea menos natural. Más bien, puede estar relacionado con la capacidad del modelo para capturar la variabilidad en las características acústicas. Este incremento en la pérdida KL se manifiesta en escenarios con una cantidad limitada de datos de audio, y aunque se produzca este aumento, no implica que la calidad del audio empeore. Por este mismo razonamiento, en la Figura 5.3d y 5.4d se observa la pérdida por *Feature Matching Loss*. que mide la diferencia entre las características a la salida del generador y el original aumenta en estos tres escenarios, aunque no tenga como consecuencia una peor calidad del audio.

Este *overfitting* es bueno en algunas situaciones de genración de datos sintéticos donde interesa generar datos sintéticos que sean indistinguibles de los datos reales. En tales casos, un cierto grado de *overfitting* podría ser necesario para que el modelo capture todas las sutilezas y detalles de los datos originales. Los modelos se dejaron de entrenar cuando se consideró que no mejoraba la calidad auditiva de la validación.

Si se analiza la pérdida de Mel (Figura 5.3b y 5.4b), se observa una reducción en todos los escenarios de validación y entrenamiento. La pérdida Mel es una de las más importante a la hora de estudiar la evolución de los sistemas TTS, ya que introduce una relación directa entre el audio original y el sintetizado. Esto hace que al mejorar esta pérdida se consiga una mayor similitud entre el audio original y el sintetizado.

Si se tiene en cuenta el predictor de duración (Figura 5.3c y 5.4c), se observa como este valor en el entrenamiento se queda constante en unos valores bajos en bases de datos de corta duración. Esta pérdida tan baja en bases de datos patológicas será muy importante en la inferencia porque llega a modelar la gran mayoría de pausas y ritmo de habla de los pacientes, haciendo que estos audios suenen más naturales.

Por último, se observa las pérdidas del generador y del discriminador (Figuras 5.3f, 5.4f, 5.4e y 5.3e) que suelen estar relacionadas de manera que el aumento de una implica la disminución de la otra. En este caso, hay que tener en cuenta que un aumento de la pérdida del generador no implica que el audio sea de peor calidad, simplemente indica que el discriminador está aprendiendo más rápido que el generador y discrimina mejor

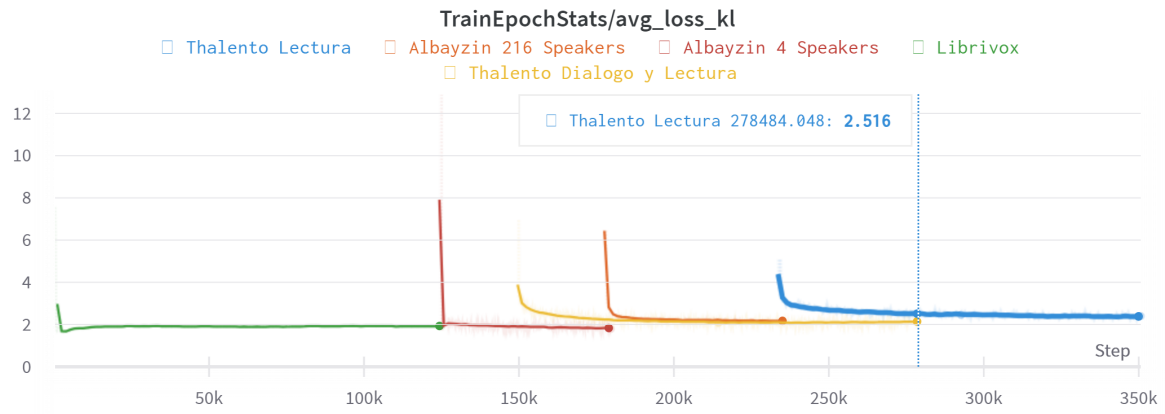
la salida, a este proceso se le denomina **modo de colapso**.

Al observar las funcion de pérdida total en el discriminador (Figuras 5.3g y 5.4g) se calcula como la suma de la pérdida de los cinco subdiscriminadores y tiene un comportamiento bastante constante en validación, implicando que este se optimiza debido a que la calidad del audio es cada vez superior.

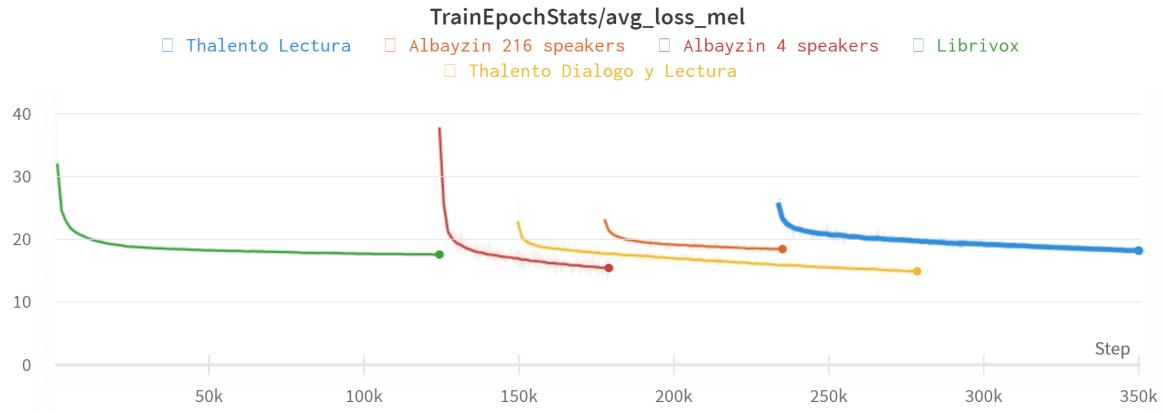
Respecto a la pérdida total del sistema (Figuras 5.3h y 5.4h), se aprecia como en **Librivox** al tener una gran cantidad de audio los valores de pérdidas de quedan estabilizados sin *overfitting* en la validación. Respecto al modelo **Albayzin 4 *Speakers*** se ve como la pérdida en validación empieza a aumentar, síntoma de *overfitting*, pero se frena pocas iteracciones después porque la calidad del audio generado ya era alta. En **Albayzin 216 *Speakers*** se observa como el modelo sigue disminuyendo sus pérdidas en validación[2], pero se consideró que la calidad era suficiente en los audios generados y detuvo a las pocas iteracciones. En **Thalento Lectura y Diálogo** se aprecia como el modelo se sobreajusta y las pérdidas empiezan a aumentar de manera considerable. Esto no es lo ideal pero los audios generados eran de muy baja calidad y conforme continuaba las iteracciones aumentaba la calidad del audio aunque aumentará las pérdidas. Cuando se consiguió un modelo con de buena calidad se paró el entrenamiento y se eligió el último *checkpoint* en vez del mejor en pérdidas. Por último, en **Thalento Lectura** se aprecia como el modelo sigue el mismo comportamiento que la anterior etapa. A diferencia del anterior escenario, los audios generados cada vez eran de peor calidad y solo se consigue obtener audio de buena calidad con las palabras vistas en el entrenamiento, siendo el principal síntoma de sobreajuste que se debe evitar. Esto ocurre debido a que la cantidad de audio es tan limitada que el modelo no es capaz de extrapolar las características fonéticas necesarias para formar nuevas palabras no vistas en el entrenamiento[3].

[2]Tras redactar esto se siguió entrenando el modelo y se conseguieron unos resultados ligeramente mejores a los usados y no incluidos en el proyecto `https://wandb.ai/santi-cabila/TFG_TTS_multispeaker_Albayzin`

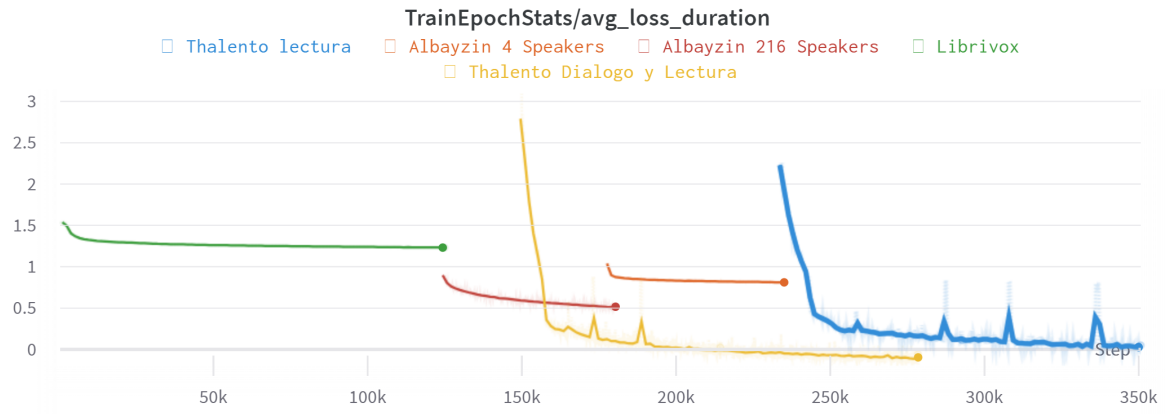
[3]En el mismo enlace se intentó seguir entrenando el modelo pero no se consiguieron resultados de buena calidad



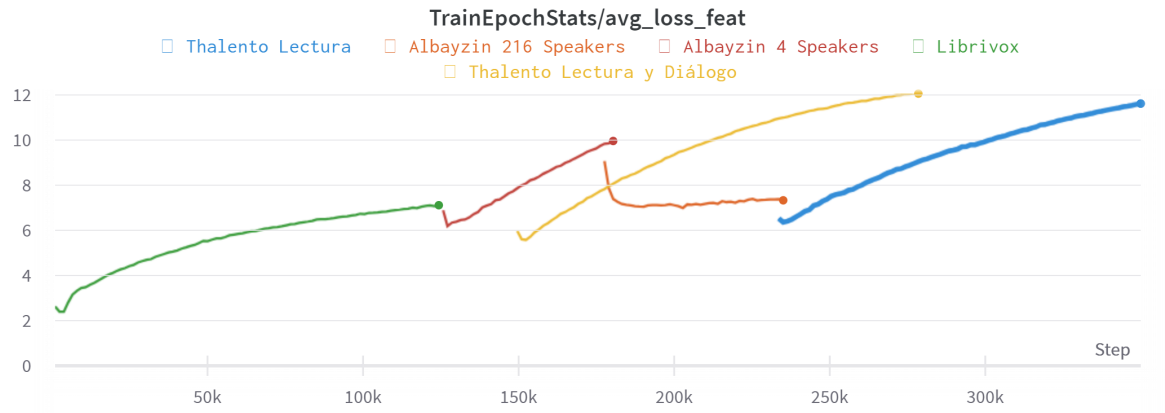
(a) Pérdida KL



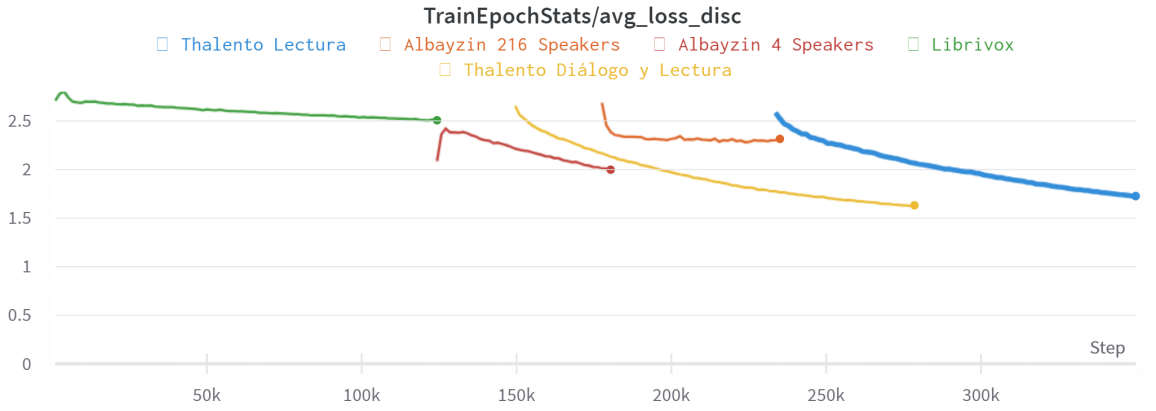
(b) Pérdida Mel



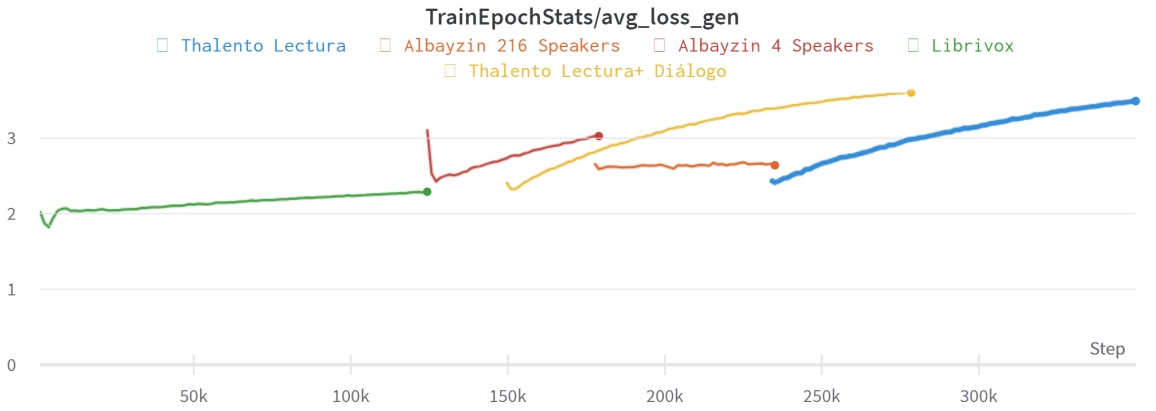
(c) Pérdida Duración



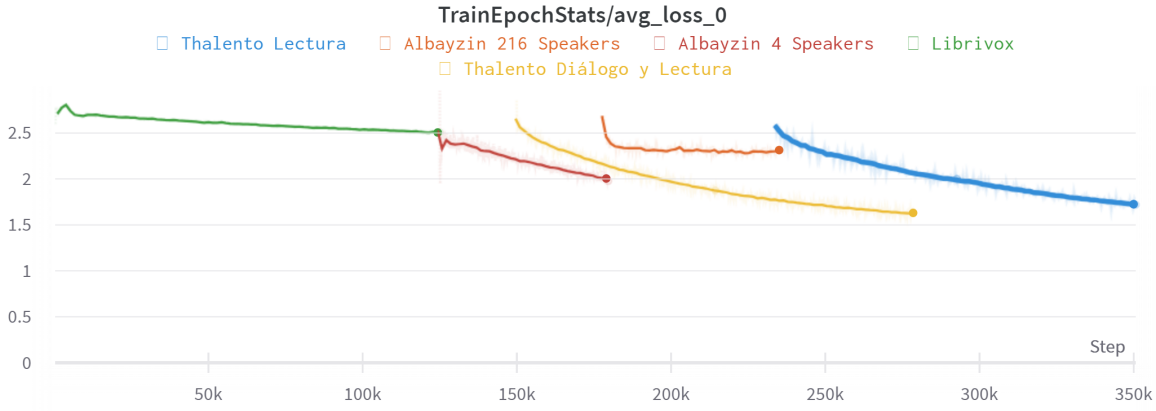
(d) *Feature Matching Loss*



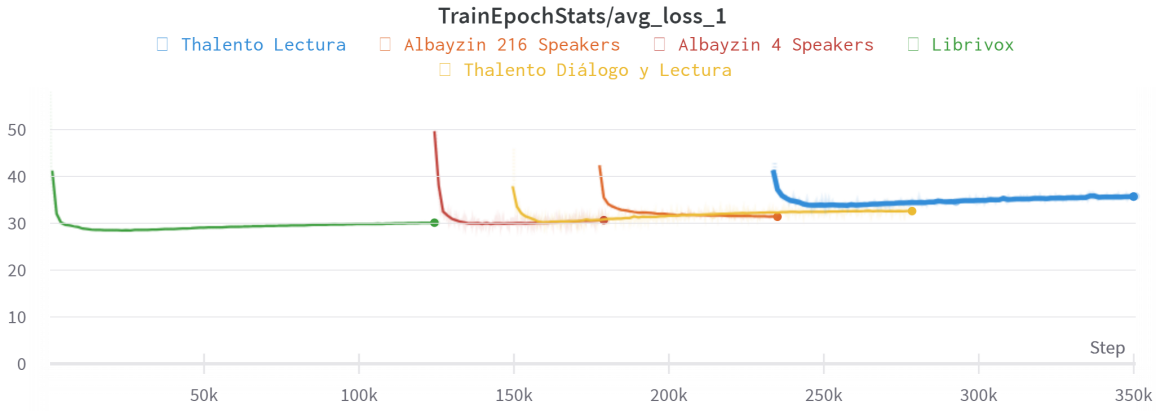
(e) Pérdida en el Discriminador



(f) Pérdida en el Generador

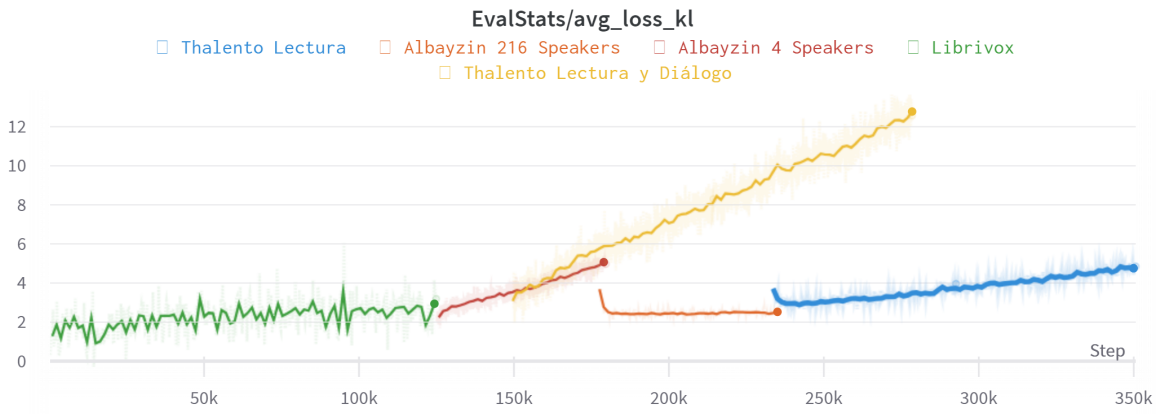


(g) Pérdida total en el Discriminador

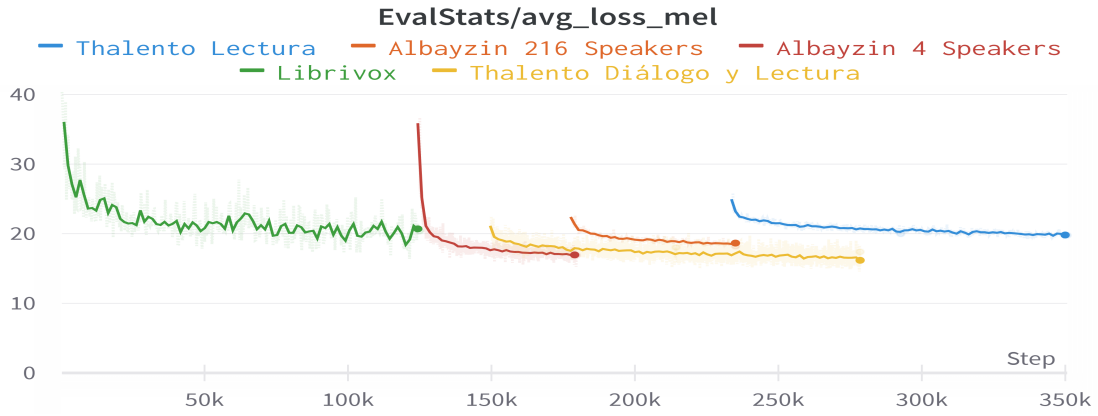


(h) Pérdida total en el sistema

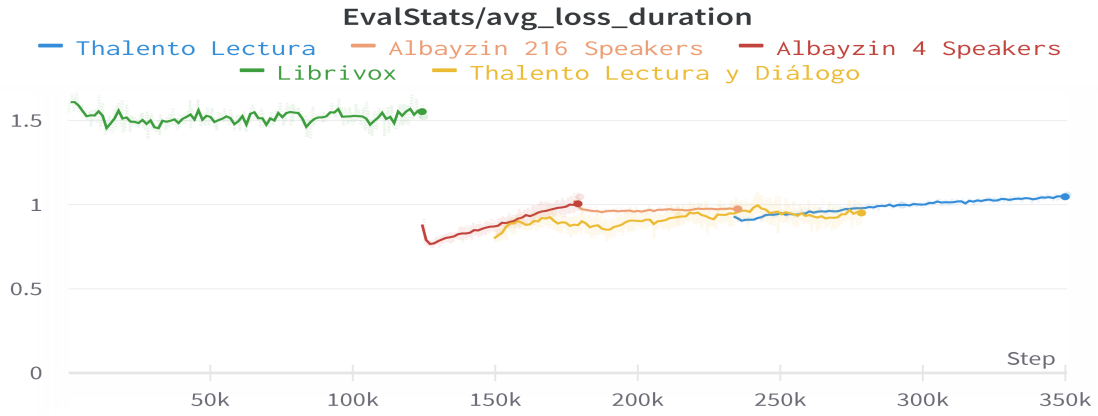
Figura 5.3: Pérdidas en el Entrenamiento en todas las etapas



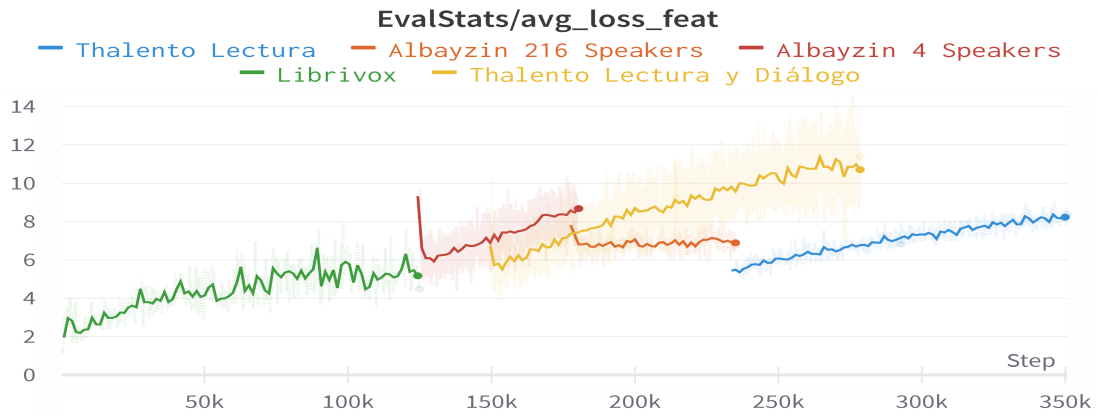
(a) Pérdida KL



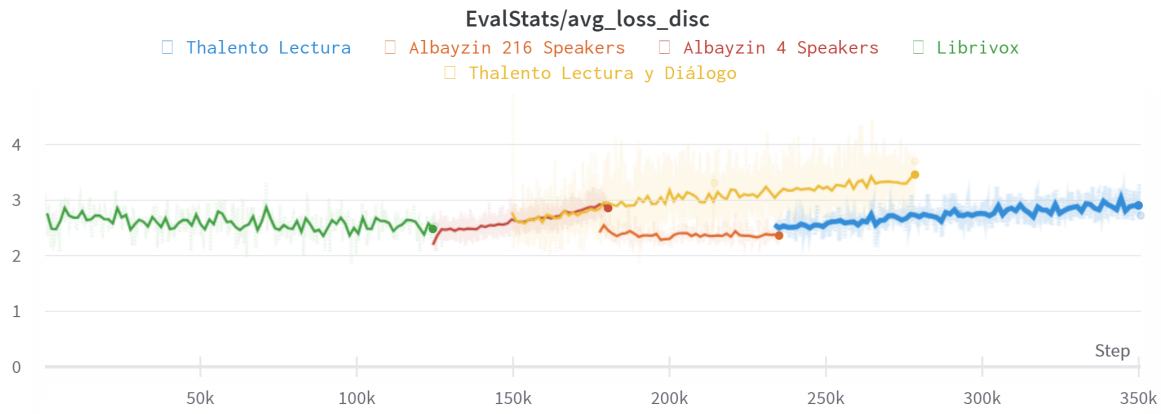
(b) Pérdida Mel



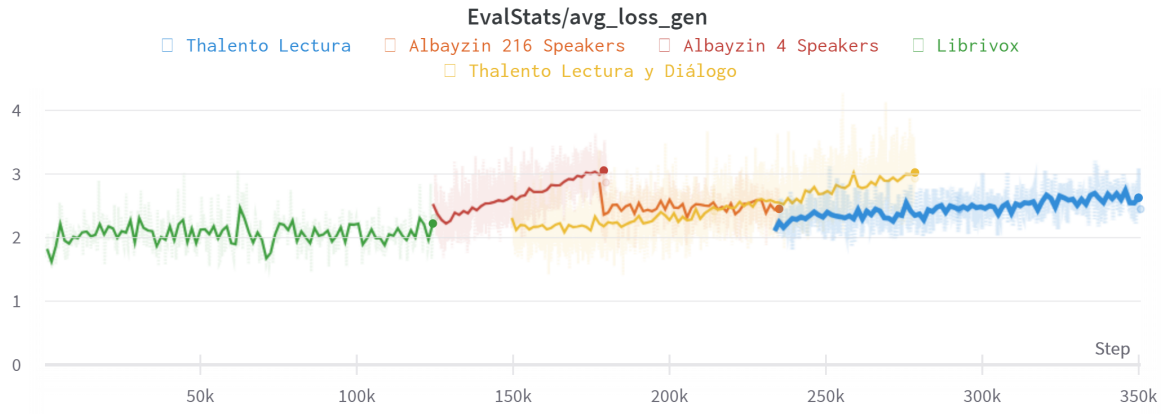
(c) Pérdida Duración



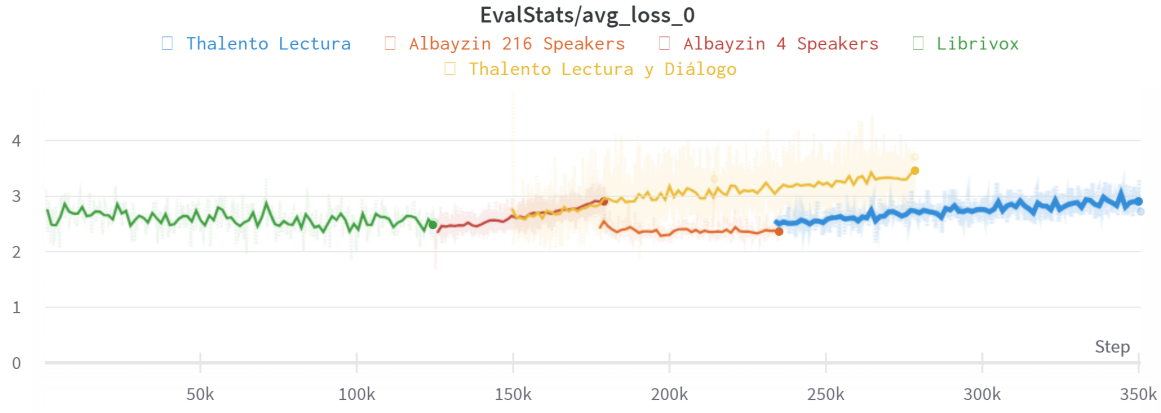
(d) *Feature Matching Loss*



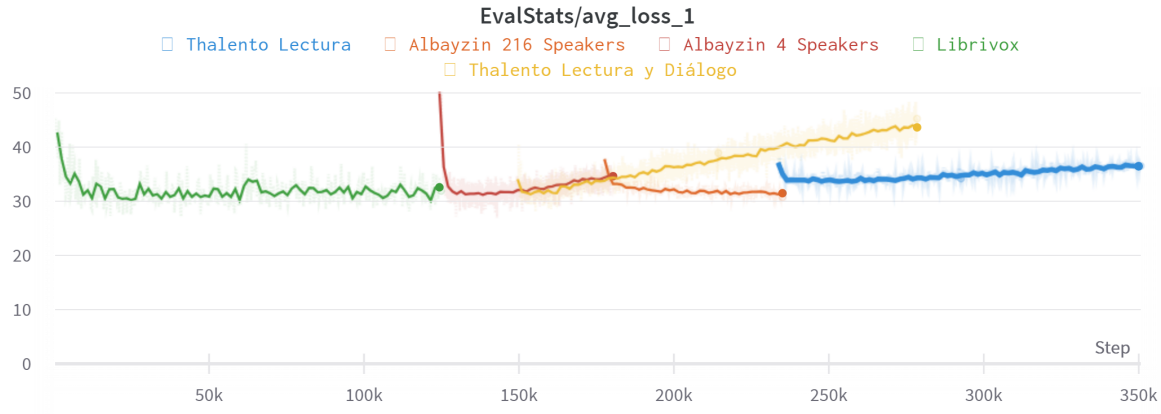
(e) Pérdida en Discriminador



(f) Pérdida en Generador



(g) Pérdida total en el Discriminador



(h) Pérdida total en el sistema

Figura 5.4: Pérdidas en validación en todas las etapas

5.3. Métricas

Para evaluar la calidad de la voz sintetizada con un TTS se utilizan métricas objetivas y subjetivas. Una métrica objetiva es una médida cuantitativa e imparcial que permite medir el rendimiento del sistema. Dentro de las métricas objetivas se encuentran la Distorsión Mel-Cepstrum (MCD)[47], la distorsión señal a ruido (SDR), la evaluación de la percepción de la calidad de habla (PESQ)[48] y la inteligibilidad objetiva a corto plazo (STOI)[49]. La MCD mide la diferencia entre dos audios en términos de sus MFCC, a menor MCD mayor similitud en los audios. Mientras que las métricas PESQ y STOI miden la calidad del habla sintetizada en términos de calidad del audio e inteligibilidad. El problema de estas métricas es la necesidad de contar con un audio de referencia, por lo que no podrían ser usadas en las etapas donde no hay disponibilidad de audios de prueba. Además, las métricas obtenidas no aportan tanto al no tener un *"baseline"* para compararlas.

Las métricas subjetivas son medidas cualitativas que se realizan preguntando la opinion de personas sobre la calidad, similitud o inteligibilidad del audio sintetizado. Se evaluan en la escala MOS que es una puntuación entre 0 y 5 de la opinión de las personas que valoran los audios. Si las preguntas se refieren a la comparativa de calidad sonora entre dos audios la métrica se denomina CMOS. Si se mide la similitud entre dos audios la métrica se denomina SMOS y si se mide la inteligibilidad se conoce por IS. La ventaja de estos sistemas es que se pueden obtener métricas acorde a la percepción humana que indican directamente la calidad del proceso de síntesis. Sin embargo, realizar una encuesta sobre la opinión de las personas requiere reunir a varios voluntarios dispuestos que van a expresar opiniones subjetivas y que suele ser costoso en tiempo.

En este trabajo se utiliza una métrica objetiva similar al STOI, que se basa en el cálculo de la inteligibilidad a través de un modelo de **reconocimiento fonético** basado en WavLM[50]. Además, se utiliza la función ***sclite*** del software SCTK[51], para lograr la mejor alineación entre los fonemas originales y la predicción del sistema. Respecto al *baseline*, se usa el reconocedor fonético en los audios originales y posteriormente en los sintetizados, para compararlos y medir la pérdida de calidad en los audios sintetizados. Finalmente, para representar estos resultados se genera una matriz de confusión con los fonemas del alfabeto fonético SAMPA[31] y con la letra "*S*"simulando los puntos. Además, se diseñó una encuesta[4] para realizar una evaluación subjetiva en escala MOS, en la que participaron 23 personas evaluando una selección de los audios sintetizados (Vea Anexo F).

[4] https://santirf01.github.io/TFG_TTS_page.io/

Capítulo 6

Simulación y Análisis de los Resultados

En este capítulo se explica como se implementaron las métricas de calidad y los resultados obtenidos de ellas. Consiste en una métrica objetiva de inteligibilidad y una métrica subjetiva de calidad y similitud que se obtiene a través de la encuesta.

Para calcular la métrica objetiva de inteligibilidad se utiliza un reconocedor fonético que permite obtener la transcripción del audio sintetizado para luego compararlo con el texto original que se indica al hacer la síntesis. A continuación se describe el procedimiento para obtener esta métrica: 1) Transformar el texto transcrito en formato ASCII[1] para ser acorde al diccionario SAMPA; 2) Pasar los audios originales por el reconocedor fonético para que devuelva los fonemas detectados en formato SAMPA. Al hacer esto surge un problema es que el reconocedor fonético devuelve el fonema detectado en ventanas de duración inferior al fonema, por lo que en la salida se tiene los fonemas repetidos; 3) Filtrar está salida para eliminar los fonemas repetidos. Para conseguirlo, se eliminan los casos donde hay dos fonemas iguales seguidos, prestando atención en casos donde la palabra acabe en la misma letra que empiece la siguiente palabra; 4) Pasar los resultados por la función *sclite* para obtener el mejor alineamiento y detectar los errores clasificandolos en omisión, introducidos y sustituidos

```
sclite -i wsj -r SAMPA_ori.txt -h SAMPA_Rec_Fon.txt -O dir_out -o all -s;
```

5) Leer estos errores y representar la matriz de confusión; 6) Repetir el proceso con los audios sintetizados.

Para calcular la métrica subjetica de calidad y similitud se diseñó una encuesta para solicitar la opinión de las personas voluntarias sobre los audios sintetizados. Uno de los dilemas principales es la elección de los audios a evaluar. Para seleccionar los audios que van a formar parte de la encuesta se utilizó un compromiso entre lo ideal y lo objetivo. En otras palabras, se quiere evaluar subjetivamente la mayor cantidad de

[1] El formato ASCII no admite ni "ñ"ni tildes.

audios pero solo se cuenta con una cantidad limitada de voluntarios. Por lo tanto se aseguró que la encuesta fuera lo más amena posible y realizable en un tiempo razonable, no más de 15 minutos. Así que se tuvieron que seguir una serie de criterios a la hora de elegir los audios para que fueran seleccionados. Estos criterios fueron:

1. *Seleccionar una sola base de datos de voces sanas y la de voces patológicas*: Decidimos eliminar la evaluación de una de las bases de datos de voces sanas. Debido a que Librivox es de uso más frecuente y tiene una mayor cantidad de audio, se decidió utilizar la base de datos más restrictiva que era Albayzin.

2. *Seleccionar las personas y los audios a evaluar*: Para hacer esto se decidió hacer un **balance de género en las voces sanas**, es decir, intentar incluir una equidad entre las voces femeninas y masculinas. Mientras que en las voces patológicas se hace un balance entre el género de la voz y la severidad de la patología.

Siguiendo estos criterios, en las voces sanas se decidió eliminar la evaluación de la calidad de la voz generada de la persona de Librivox, debido a que la principal función de esta base de datos era el pre-entrenamiento el modelo para tener una base más estable. Después, en Albayzin se tuvo que volver a filtrar los audios utilizados. En primer lugar, se eligieron las cuatro personas que disponían de 250 oraciones, teniendo así dos voces masculinas y dos femeninas con audios de prueba. Además, de esta base de datos se decidió elegir a tres personas aleatorias del resto de la base de datos con 25, 50 y 75 frases.

Respecto a los pacientes de Thalento se utilizaron los audios 'TVD-D-0001': mujer con Edema de Reinke con gran severidad, 'TVD-D-0002': hombre con Edema de Reinke con una severidad alta y 'TVD-D-0005': mujer con Edema de Reinke leve y poco detectable. Así se consigue generar una encuesta breve y diversa. Respecto al modelo de Thalento solo Lectura, no se consiguió obtener ningún audio que se considerara con la suficiente calidad e inteligibilidad para enfrentarse a una prueba. La falta de audio, acompañada de la poca riqueza fonética y la alta variabilidad en el timbre de las voces hizo que el modelo no llegará a converger a una solución efectiva. Se realizaron varios intentos para hacer converger el modelo, como aumentar el número iteracciones en el entrenamiento o variar algunos hiperparámetros del modelo. Sin embargo, la escasez de datos provocaba un sobreajuste, consiguiendo capturar el timbre de algunos locutores pero con demasiado ruido y sin claridad en los sonidos. El sobreajuste hizo que solo se obtuvieran claramente las palabras vistas en el entrenamiento, haciendo imposible su evaluación.

6.1. Resultados Albayzin

6.1.1. Albayzin 4 *Speakers*

En primer lugar, se necesita el *baseline* para comparar los resultados obtenidos. En Albayzin, se obtuvo la matriz de confusión de cada persona original por separado y la de la base de datos total. Al no existir una diferencia relevante en la matriz entre las distintas personas, se decidió usar como *baseline* la matriz de confusión total de Albayzin 4 *Speakers* (Figura 6.1). En esta matriz, se aprecia como el reconocedor genera pocos errores. Un error que comete el reconocedor es confundir el fonema 'i' y el fonema 'j',lo que es muy común en español ya que suenan muy parecido. Otro error que suele cometer es la omisión de fonemas que no llega a detectar, pudiendo ser causa de una mala pronunciación del audio. Un último error que comete, es detectar fonemas que no aparecen. Esto pasa debido a que un error en una ventana pequeña del fonema, el sistema lo interpreta como un fonema nuevo, aunque en realidad es que no existe, conllevando a la frecuente aparición de este error en la matriz de confusión. Tras esto, se realiza la matriz de confusión con los audios sintetizados (Figura 6.2) donde se aprecia que los errores son prácticamente los mismos que en el *baseline*, considerando que en términos de inteligibilidad no hay pérdida entre los audios sintetizados y los originales.

Respecto a las métricas MOS obtenidas en la encuesta, lo primero que se hace es escuchar el contenido de las oraciones:

- Audio 0001: Francia, Suiza y Hungría ya hicieron causa común.
- Audio 0002: Mi primer profesor de Lengua fue Lopez García.
- Audio 0003: Guillermo y Yolanda practicaban ciclismo con Jaime.

En la tabla 6.1 que mide el CMOS y calidad de los audios, se observa que la mayor correlación en los resultados viene determinada por el contenido del audio y no por el orador. Se ve que el primer audio tiene una mayor calidad que el segundo, que a su vez tiene mejor calidad que el tercero. Con estos resultados y otros propios sacados de varias simulaciones, se llega a la conclusión que el primer audio es el de mayor calidad debido a que tiene más pausas. Esto es un factor principal en la calidad de la sintesis geenrada, ya que los sistemas TTS suelen producir audios más acelerados de lo natural. Respecto al segundo audio se tiene también buena calidad de escucha, pero el sistema falla la acentuación de la última palabra, haciendo que pierda calidad. El último, es el de menor calidad debido a que los fonemas pronunciados son menos comunes y aumenta la probabilidad de fallo. Un error en un fonema, afecta a la percepción humana del audio haciendo que se propague el error al resto del audio y parezca que tenga una calidad

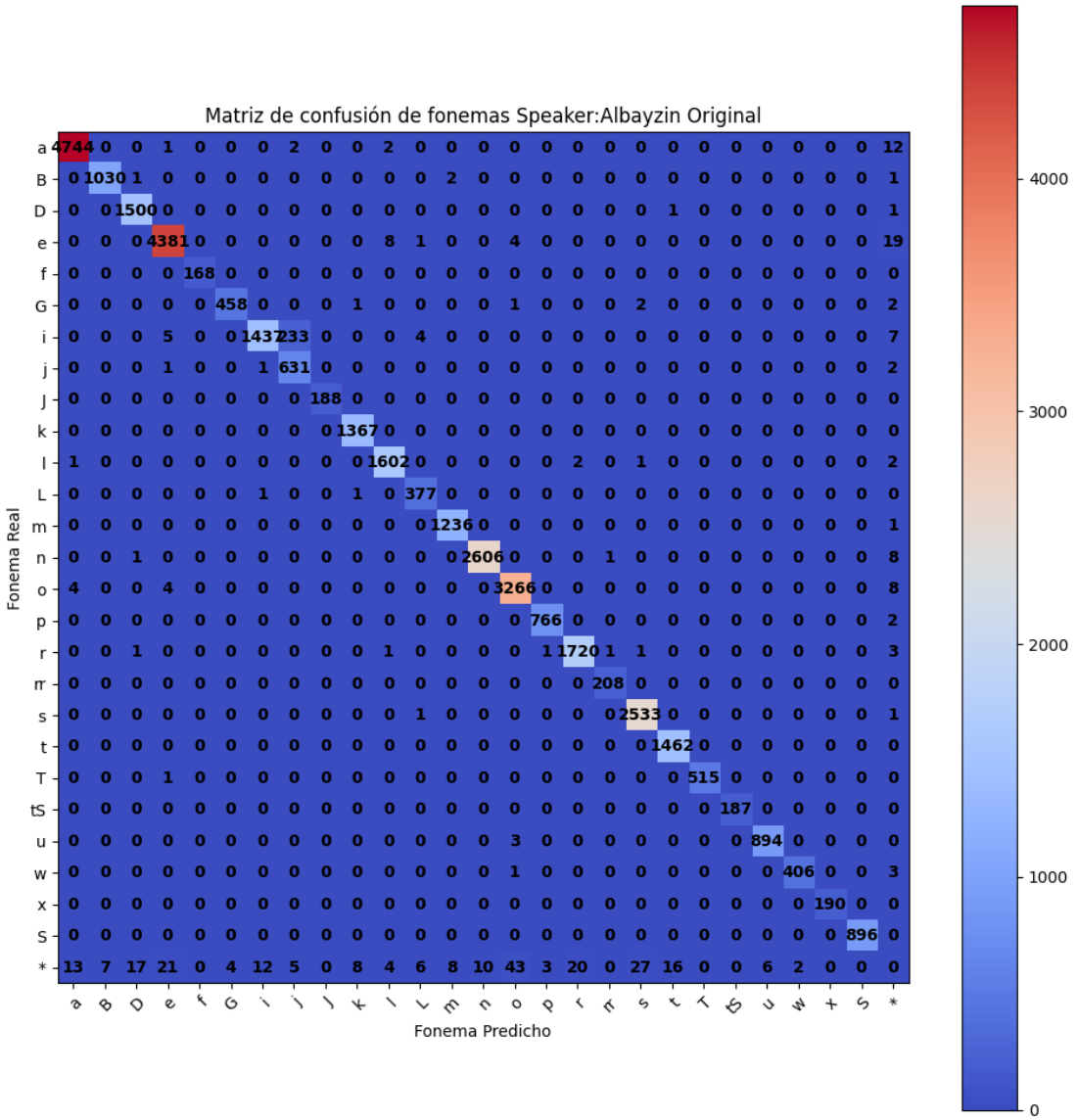


Figura 6.1: Matriz de confusión en Albayzin original

peor. No obstante, todos los audios tienen una calidad buena con resultados superiores a CMOS=3.85/5. Respecto a los resultados de similitud representados en la tabla 6.2, se aprecia una mayor correlación con la persona del audio y no con el contenido. Indicando que la persona que asimila mejor el sistema es la voz masculina de **mb**. Seguido de muy cerca, están las dos siguientes voces femeninas **ab** y **aa**. Se concluye que con esta cantidad de audio el sistema modela correctamente ambas voces, existiendo poca diferencia entre la calidad de las voces sintetizadas y modelando ligeramente peor las voces con una más dificultades al vocalizar. En esta encuesta, también se aprecia una fuerte correlación entre los resultados de calidad y similitud.

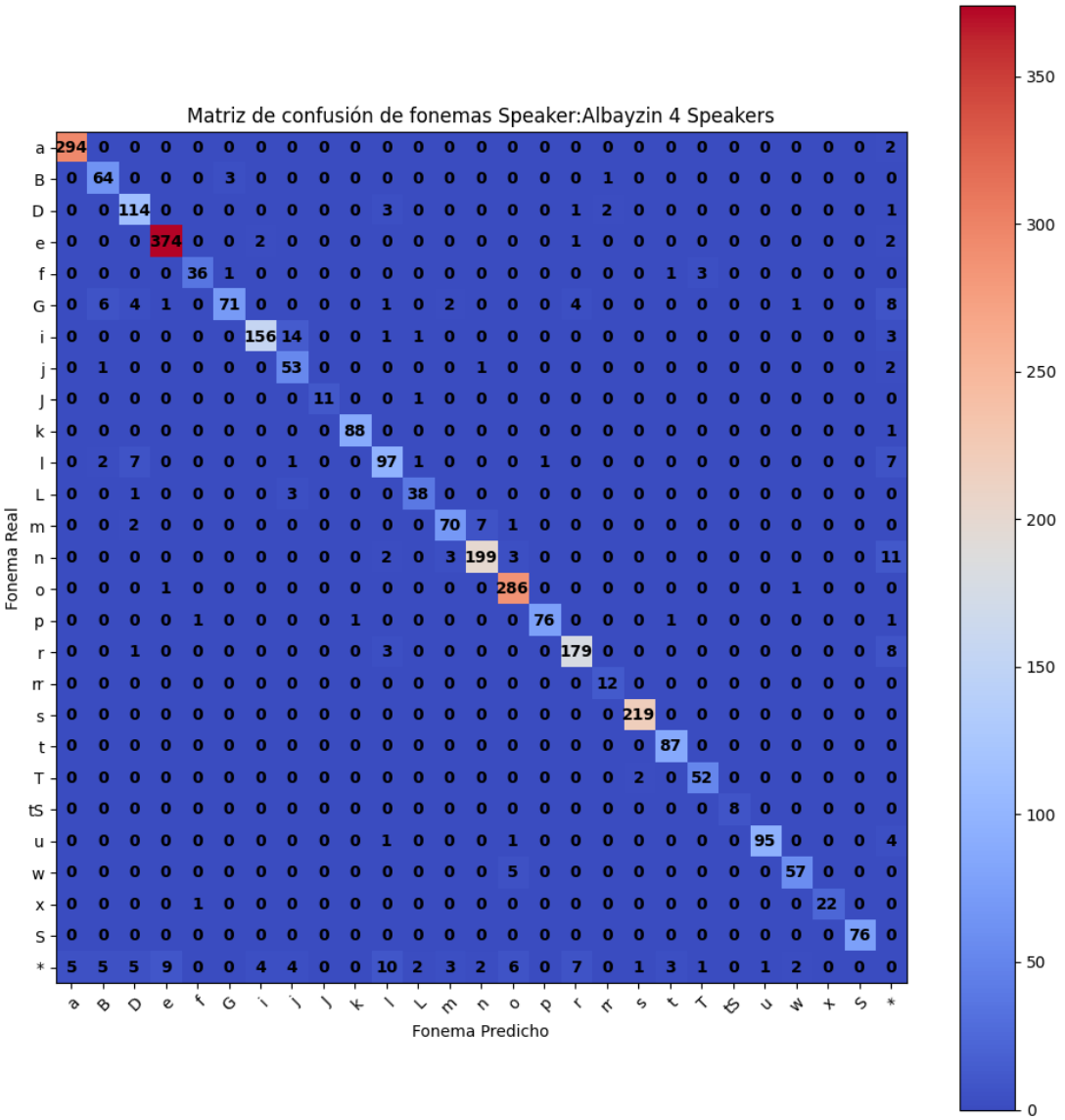


Figura 6.2: Matriz de confusión en Albayzin sintetizado

Calidad				
Audios/Persona	**aa (F)**	**ab (F)**	**ma (M)**	**mb (M)**
Audio 0001	4.56	4.40	4.36	4.47
Audio 0002	4.04	4.15	4.07	4.41
Audio 0003	4.04	3.90	3.87	4.00

Tabla 6.1: Resultados de calidad de la encuesta (Género: Femenino(F)/Masculino(M))

6.1.2. Albayzin 216 *Speakers*

Para hacer el estudio de inteligibilidad, en Albayzin 216 *Speakers* se usará el mismo *baseline* que en Albayzin 4 *Speakers*. Esto se debe a que el fonetizador tenía errores similares en todos las personas y calcular el *baseline* con toda la base de datos iba a ser muy costoso. Tras definir el *baseline*, se generan 10 frases con gran cobertura fonética,

Similitud				
Audios/Persona	**aa (F)**	**ab (F)**	**ma (M)**	**mb (M)**
Audio 0001	4.22	4.35	4.07	4.35
Audio 0002	4.00	4.00	4.13	4.53
Audio 0003	4.05	4.10	3.87	4.00

Tabla 6.2: Resultados de similitud de la encuesta (Género: Femenino(F)/Masculino(M))

para comprobar las prestaciones de este modelo. Estas frases son:

- El sol resplandece sobre el mar sereno.
- Bajo el manto estrellado, susurra el viento al pasar.
- Cascadas de cristal caen en el valle encantado.
- Las olas danzan en la orilla dorada de la playa.
- El canto del ruiseñor ilumina la noche callada.
- Susurros de seda flotan en el aire perfumado.
- En la selva exuberante, la vida late con fuerza.
- El arcoíris brilla en el cielo despejado de colores.
- Mariposas de colores danzan entre las flores silvestres.
- Las palabras fluyen como melodías en la lengua vibrante.

Siguiendo el mismo procedimiento, se obtiene la matriz de confusión de los audios sintetizados. En esta situación, se tiene el problema de que el número de fonemas usados es grande y la figura de la matriz queda sobrecargada. Para solucionar esto, se plantea normalizar cada fila frente al número total de veces que el fonetizador predice ese fonema (Figura 6.3). En los resultados se observa una tendencia similar a lo ocurrido en el caso de Albayzin. Se tienen los errores entre el fonema ‘i’ y ‘j’, aparición de fonemas que no aparecen realmente y omisión de algunos fonemas. Además, también se ve una gran degradación del sistema respecto al fonema ’G’. Esto fonema aparece 432 veces, dos veces por persona, y la mitad de veces no lo detecta y lo considera como ruido o no llega a escucharlo, siendo este fonema el que menor porcentaje de acierto tiene. Por último, aparecen problemas con los fonemas ‘r’ y ‘rr’. El modelo tiende a no pronunciar este fonema con la potencia necesaria y en muchas situaciones no son detectado por el fonetizador. Esto vino anotado por varios voluntarios en la encuesta, donde en anotaciones nos indican que el fonema ’r’ se dice con poca potencia

y muchas veces lo omite. Este fallo se piensa que es debido a la variabilidad del sonido 'r' dependiendo del acento y la pronunciación de cada persona.

En este escenario, se observa una pérdida de calidad en la inteligibilidad al entrenar el modelo con una cantidad mayor de persondas (216 personas) y una cantidad menor de audio. Aun así, los resultados obtenidos son de buena calidad, teniendo fallos muy puntuales y en unos pocos fonemas. Los resultados son cercanos al 90 % en el resto de fonemas no comentados, por lo que la degradación en inteligibilidad es apreciable pero asumible.

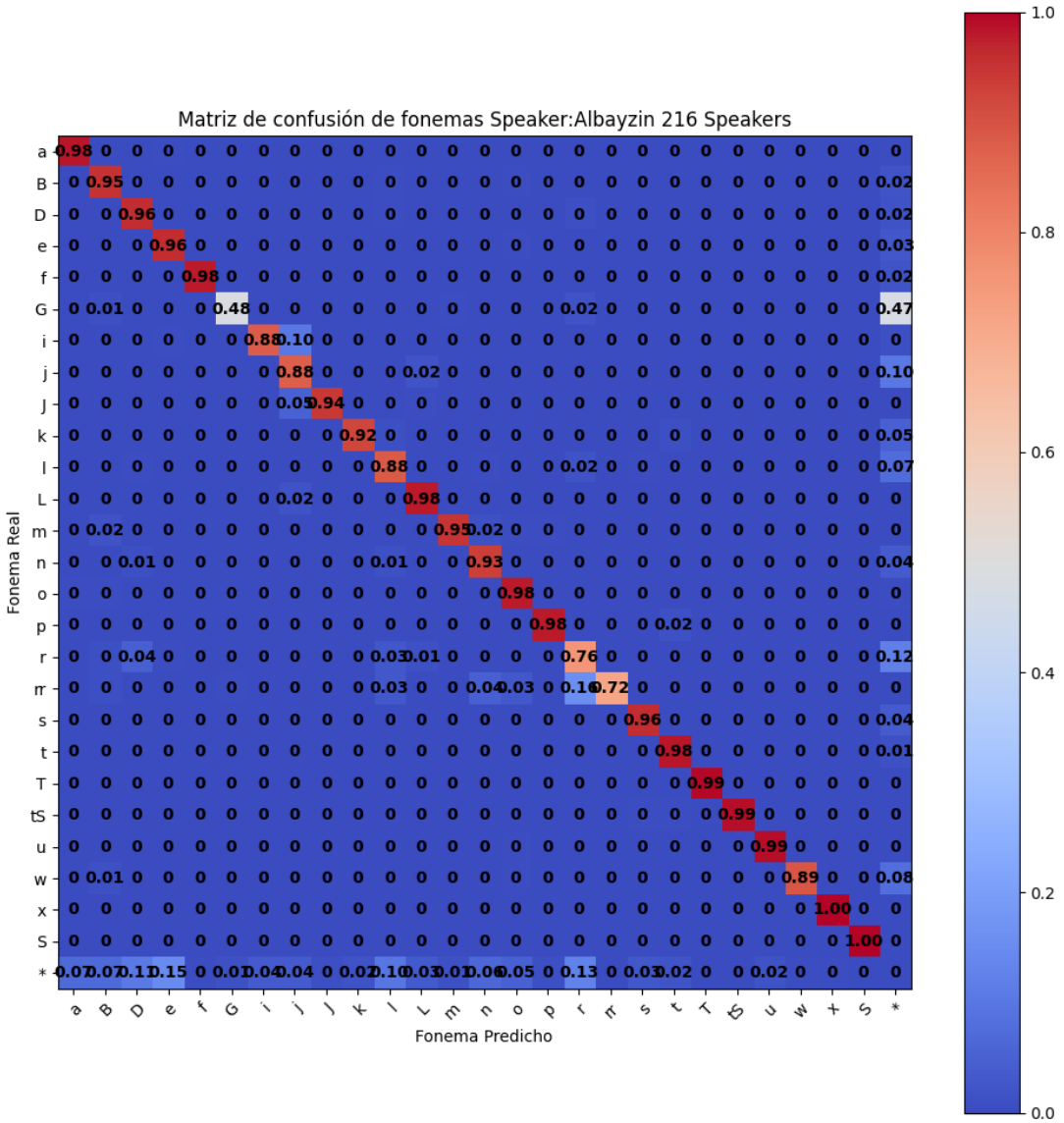


Figura 6.3: Matriz de confusión en Albayzin 216 *Speakers*

Para la realización de la encuesta Albayzin 216 *Speakers* no se tenían datos de prueba, así que las métricas obtenidas son el SMOS y una evaluación de calidad/inteligibilidad, pero ya no sería CMOS al no comparar entre dos audios con

el mismo contenido. En la tabla 6.3, se observa como los valores obtenidos tanto en calidad como similitud son inferiores a los de la encuesta Albayzin 4 *Speakers*. En calidad se tienen resultados cercanos a 3/5 en las dos primeras personas y de 3.39/5 en la tercera persona. Aquí se ve como en calidad de la voz no se obtienen resultados satisfactorios en ninguna de las tres voces.

Comentarios de la encuesta indicaron que los audios sintetizados iban demasiado rápido, siendo difícil de seguir en algunas situaciones. Además, a veces no se vocalizan correctamente los fonemas como la 'r', que junto a la velocidad de habla hace que los audios sean de baja calidad.

En la métrica SMOS se obtienen resultados más altos y con mayor variación dependiendo de la persona. Se ve como el 'ts' es el que mayor similitud obtiene con un resultado cercano a SMOS=3.9/5, después el 'oi' con un SMOS= 3.25/5, y en último lugar 'dh' con SMOS=3.13/5. Aquí se observa como la voz masculina 'ts' con 75 oraciones es la que mayor similitud tiene entre la sintetizada y la original. Pero luego se ve como obtiene mejor similitud en 'oi' con 25 oraciones que en 'dh' con 50 oraciones. Esto se debe a que la voz 'dh' es una voz femenina, mientras que 'oi' es una voz masculina. En síntesis de voz con poca cantidad de audio, usando un modelo pre-entrenado con una voz masculina, es más difícil modelar esta voz femenina por el tiembre y el rango vocal de estas.

Albayzin 216 Speakers				
Persona	**Oraciones**	**Género**	**Calidad**	**Similitud**
oi	Masculino	25	3.00	3.25
dh	Femenino	50	2.94	3.13
ts	Masculino	75	3.39	3.87

Tabla 6.3: Resultados de la encuesta Albayzin 216 *Speakers*

6.2. Resultados Thalento Diálogo y Lectura

Para obtener el *baseline* en Thalento ocurre un problema con el archivo de diálogo. Este diálogo se trata de habla espontánea, es decir, no se sabe exactamente el audio transcrito en cada persona. Aunque se pueda hacer una aproximación con la transcripción ofrecida por *Whisper*, los errores que este cometa se sumarían a los del fonetizador, por lo que no sería asumible. Para solucionar esto, solo se tiene en cuenta los audios de lectura con la transcripción de la fábula (Figura 6.4). En este *baseline* se observa un comportamiento similar a la Figura *baseline* de Albayzin 6.1, pero con algún fallo más como la confusión entre 'r' y 'rr' o por ejemplo la cantidad de sonidos 'x' no detectados. Respecto a los audios sintetizados se generan en cada persona las 250 frases

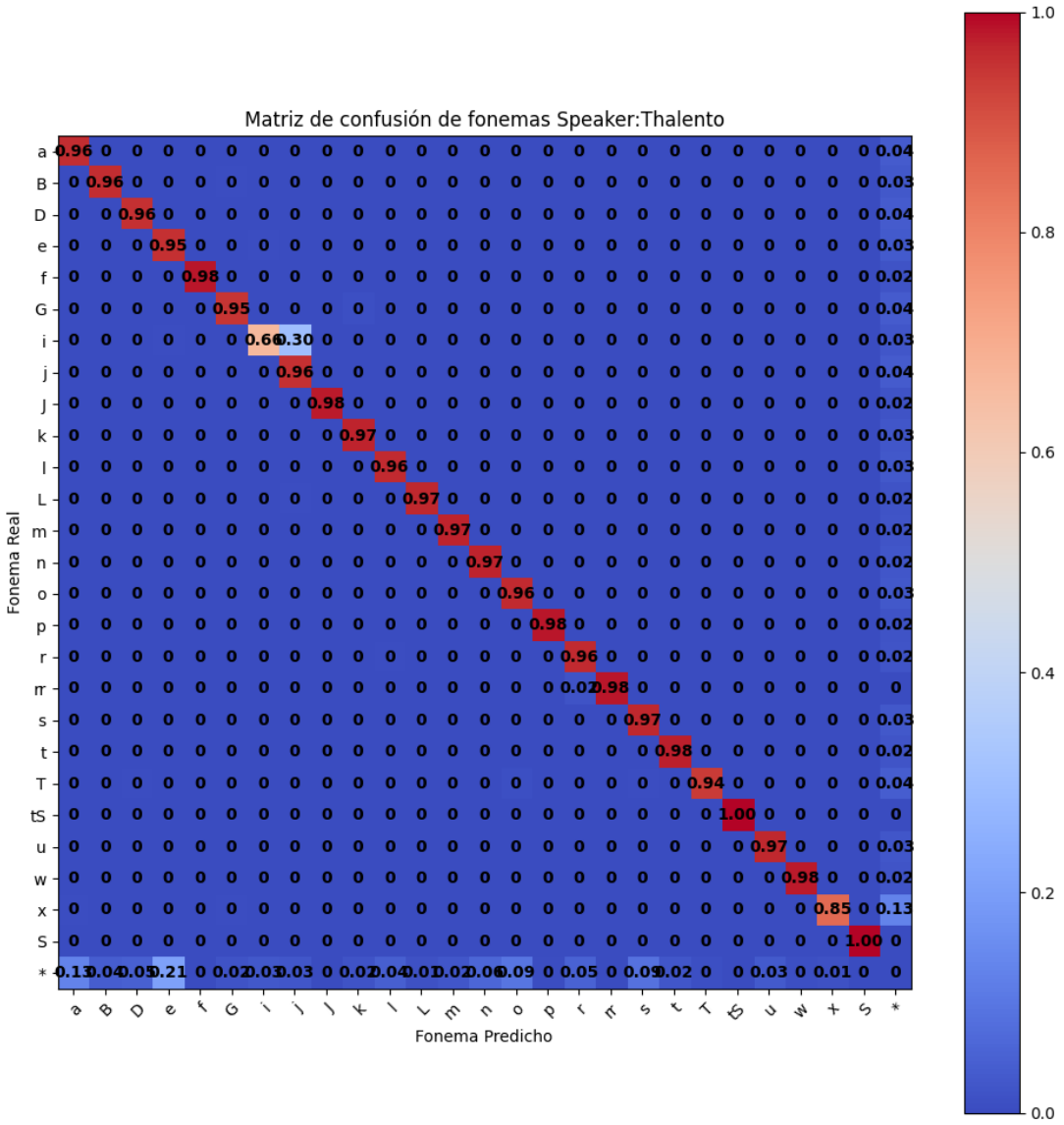


Figura 6.4: Matriz de confusión en Thalento Lectura original

de Albayzin. Los resultados obtenidos demuestran una mayor confusión entre fonemas de manera aleatoria y no entre fonemas similares. Esto no es buen síntoma debido a que estos errores son fallos graves del sistema dónde no llega a representar ni de manera similar el fonema. Al escuchar los audios, se aprecia como hay algunas oraciones donde el sistema colapsa. Esto le ocurre cuando aparece un fonema que prácticamente no ha visto en entrenamiento, lo que luego se propaga al resto del audio.

Aunque la degradación de la calidad de los audios es notable, el reconocedor fonético acierta en más del 80 % en todos los fonemas menos la 'G', lo que se traduce en una inteligibilidad media. Respecto al fonema 'G' se nota en todos los sistemas que tiene una tasa de error bastante superior al resto. Esto puede ser debido a que este fonema es muy similar al ruido y un mal modelado produce altas tasas de error.

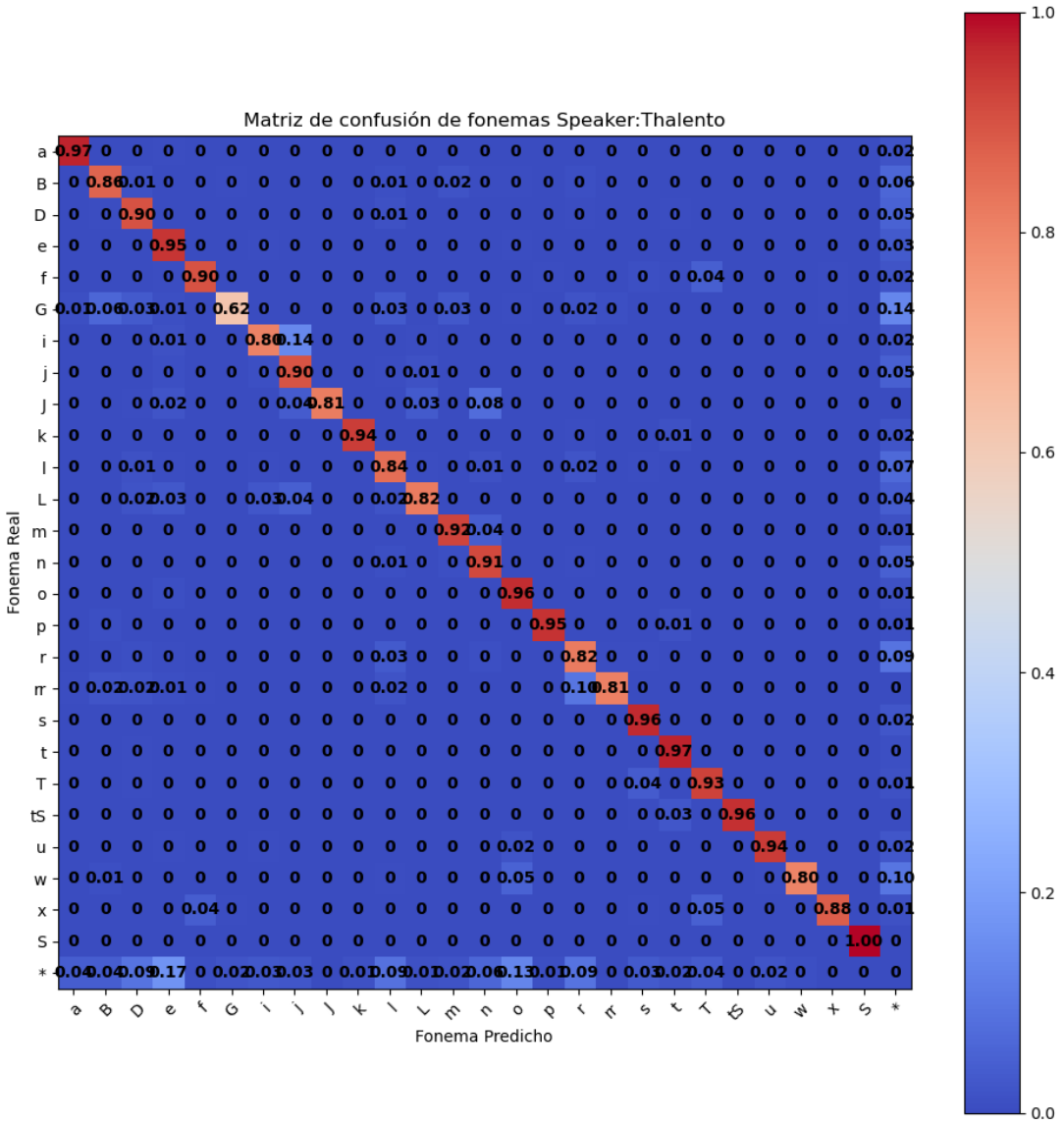


Figura 6.5: Matriz de confusión en Thalento sintetizado

En la encuesta, se observa como en los pacientes 'TVD-D-0001' y 'TVD-D-0002' la respuesta de SMOS ≈4/5. Estos dos pacientes eran los que tenían una patología más severa y obtienen mejores resultados que la persona con patología leve. Esto es debido a que al realizar el sobreajuste del modelo, algunas voces de los pacientes de las bases de datos fueron modelados de manera muy similar, mientras que otras tenían demasiado ruido. La calidad del modelo de estos pacientes está relacionada con la ronquera de la voz. Las voces más roncas, se escuchan mejor que las voces agudas que tienen mucho ruido. Esto es debido a que al tener una voz ronca, el ruido generado por el sistema se enmascara mejor que en voces más agudas. Además, estos resultados de similitud, vienen dados por la correcta adaptación del sistema a la prosodia de los pacientes con una patología severa, reproduciendo adecuadamente las pausas entre palabras, las veces

que se traba, etc.

La calidad de las voces de los pacientes está condicionada al tipo de habla inicial de estos. Es decir, personas con una voz claramente inteligible se entienden mejor y tendrán mejor calidad aunque el modelo no modele bien la voz. Por el contrario, personas con menos inteligibilidad tendrán peores resultados en calidad. En la tabla 6.4, se aprecia como 'TVD-D-0002', que tiene la mejor respuesta en SMOS, logra valores calidad de 3,67/5, estando muy cerca de 'TVD-D-0005' que es el paciente que peor modela el sistema y tiene un resultado de calidad de 3,6/5.

Thalento Lectura y Diálogo				
Paciente	**Género**	**Severidad**	**Calidad**	**Similitud**
TVD-D-0001	Femenino	Severa	3.83	3.94
TVD-D-0002	Masculino	Severa	3.67	4.00
TVD-D-0005	Femenino	Leve	3.60	3.40

Tabla 6.4: Resultados de la encuesta Thalento Lectura y Diálogo

Capítulo 7

Conclusiones y Líneas Futuras

7.1. Conclusiones

A lo largo de esta trabajo se ha explorado en detalle el modelo de síntesis de voz VITS. Se evaluaron dos escenarios: 1) **Albayzin 4/216 *Speakers***: Base de datos de audios de personas sanas con frases leídas de corta duración. 2) **Thalento**: Base de datos de voces patológicas con audio segmentado en dos archivos de larga duración, uno de lectura y otro de habla espontánea. En estos escenarios, se comprobó la gran versatilidad del modelo y la calidad de los resultados obtenidos con bases de datos pequeñas. También, se consigue una velocidad de procesado superior a tiempo real, generando audio a una tasa de 0.3-0.6 veces la duración del audio.

De los escenarios de Albayzin, se ha demostrado que con una cantidad de audio cercana a 10 minutos se puede conseguir desarrollar voces con inteligibilidad cercana a la original en un tiempo de procesado de inferencia menor al tiempo real. De aquí que se proponga este modelo como una solución factible para la clonación de voces por la poca necesidad de audio requerido.

En este escenario, también se ha observado como el sistema tiene la capacidad de adaptarse a bases de datos con varias voces, optimizando su espacio vectorial para permitir la clasificación de hasta 216 personas sin la necesidad de tener un modelo propio para cada uno. Esto es un gran ahorro computacional debido a que cada modelo de un solo orador tiene aproximadamente 80 millones de parámetros y 800 Mb.

Del escenario de voces patológicas Thalento, se llega a la conclusión que estos sistemas pueden modelar voces con patología si se tuviera la cantidad de audio de calidad suficiente. Se ha observado que con pacientes con menos de cuatro minutos de audio se han conseguido sintetizar con una calidad muy alta en la prosodia y adecuada en la voz en general, aunque se haya permitido *overfitting* moderado del modelo. Aún así lo ideal para futuras tareas sería contar con mayor cantidad de audio para evitar el sobreajuste del modelo que ha provocado que con algunos pacientes de Thalento no se

lograran audios sintéticos de calidad.

Por último, se concluye que para lograr clonar una voz de calidad es necesario tener un modelo de partida consistente y una base de datos con una gran cobertura fonética y audios de duración total cercana a 10 minutos en archivos bien fragmentados y limpios.

7.2. Líneas Futuras

Como líneas futuras del trabajo se propone el uso de los resultados obtenidos en tareas de clasificación y detección de patologías para comprobar si el aumento de datos mejoran las prestaciones de estos sistemas.

Uno de las principales areas en las que trabajar es en la obtención de métricas de calidad del audio sintetizado. Enfatizar en el diseño de la encuesta subjetiva de opinión sobre la calidad y similitud del audio sintético y el original, reajustando las preguntas y ampliando la muestra de audios a evaluar. Además, incluir nuevas métricas objetivas de inteligibilidad con las versiones más recientes de métodos de estimación de MOS tomando como referencia el *"Challenge VoiceMOS"*[52].

En cuanto a la tarea de búsqueda de mejora en la calidad de las voces de los pacientes, se propone dar un enfoque distinto a las bases de datos patológicas. En lugar de optar por lecturas de fábulas o habla espontánea, en estas tareas se considera más eficaz la lectura de frases de corta duración fonéticamente balanceadas.

Con respecto al modelo de síntesis se propone explorar la idea de modificar la obtención de las características de los oradores a través del *Speaker Encoder*. Este componente ha demostrado muy buenas prestaciones en la proyección *UMAP* de la base de datos Albayzin, mientras que en la proyección *UMAP* de Thalento se observa una mayor proximidad entre estos. Así, se cree que si se entrenará un nuevo *encoder* con voces patológicas similares a las de Thalento, se podría ver reflejado en una mejora de la representación vectorial de las voces de los pacientes, pudiendo implicar una mayor calidad en los audios sintetizados.

Capítulo 8

Bibliografía

[1] serpwatch.io. Voice search statistics 2022. https://serpwatch.io/blog/voice-search-statistics/, 2023.

[2] Statista. Asistentes virtuales de voz más usados en España. https://es.statista.com/estadisticas/1012695/asistentes-virtuales-de-voz-mas-usados-en-espana/, 2023.

[3] Xu Tan. *Neural Text-to-Speech Synthesis*. Springer, 2023.

[4] Chengyi Wang, Sanyuan Chen, Yu Wu, Ziqiang Zhang, Long Zhou, Shujie Liu, Zhuo Chen, Yanqing Liu, Huaming Wang, Jinyu Li, Lei He, Sheng Zhao, and Furu Wei. Neural codec language models are zero-shot text to speech synthesizers. 2023.

[5] Matthew Le, Apoorv Vyas, Bowen Shi, Brian Karrer, Leda Sari, Rashel Moritz, Mary Williamson, Vimal Manohar, Yossi Adi, Jay Mahadeokar, and Wei-Ning Hsu. Voicebox: Text-guided multilingual universal speech generation at scale. *Meta AI*, 2023.

[6] Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A. Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8342–8360, Online, July 2020. Association for Computational Linguistics.

[7] Francisco Casacuberta, Rafael García, Joaquim Llisterri, Climent Nadeu, Juan M. Pardo, and Antonio Rubio. Desarrollo de corpus para investigación en tecnologías del habla (albayzín). *Procesamiento del Lenguaje Natural*, 12:35–42, 1992.

[8] H. W. Dudley. The vocoder. *Bell Labs Rec.*, 18:122–126, 1939.

[9] Christine H. Shadle and Robert I. Damper. Prospects for articulatory synthesis: a position paper. In *4th ISCA tutorial and research workshop (ITRW) on speech synthesis*, 2001.

[10] P. Seeviour, J. Holmes, and M. Judd. Automatic generation of control signals for a parallel formant speech synthesizer. In *ICASSP '76. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 690–693, 1976.

[11] A.J. Hunt and A.W. Black. Unit selection in a concatenative speech synthesis system using a large speech database. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 1, pages 373–376 vol. 1, 1996.

[12] Paul Taylor. *Text-to-Speech Synthesis*. University of Cambridge, 2009.

[13] Keiichi Tokuda, Yoshihiko Nankaku, Tomoki Toda, Heiga Zen, Junichi Yamagishi, and Keiichiro Oura. Speech synthesis based on hidden markov models. *Proceedings of the IEEE*, 101(5):1234–1252, 2013.

[14] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. In *Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9)*, page 125, 2016.

[15] Sercan Arik, Gregory Diamos, Andrew Gibiansky, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep voice 2: Multi-speaker neural text-to-speech. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, pages 2966—-2974, 2017.

[16] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif A. Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783, 2018.

[17] Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan O. Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep voice 3: 2000-speaker neural text-to-speech. In *International Conference on Learning Representations*, 2018.

[18] Yi Ren, Yangjun Ruan, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. Fastspeech: Fast, robust and controllable text to speech. In *33rd Conference on Neural Information Processing Systems (NeurIPS)*, pages 3171—-3180, 2019.

[19] Jaehyeon Kim, Jungil Kong, and Juhee Son. Conditional Variational Autoencoder with Adversarial Learning for End-to-End Text-to-Speech. In *Proceedings of the 38th International Conference on Machine Learning (PMLR)*, 2021.

[20] Xu Tan, Jiawei Chen, Haohe Liu, Jian Cong, Chen Zhang, Yanqing Liu, Xi Wang, Yichong Leng, Yuanhao Yi, Lei He, Frank K. Soong, Tao Qin, Sheng Zhao, and Tie-Yan Liu. NaturalSpeech: End-to-End Text to Speech Synthesis with Human-Level Quality. *ArXiv*, abs/2205.04421, 2022.

[21] Yi Ren, Chenxu Hu, Xu Tan, Tao Qin, Sheng Zhao, Zhou Zhao, and Tie-Yan Liu. FastSpeech 2: Fast and High-Quality End-to-End Text to Speech. In *International Conference on Learning Representations (ICLR)*, 2021.

[22] B. M. Halpern. *Making speech technology accessible for pathological speakers.* Thesis, fully internal, Universiteit van Amsterdam, 2022.

[23] Mohammad Soleymanpour. *Synthesizing Dysarthric Speech Using Multi-Speaker TTS for Dysarthric Speech Recognition.* PhD thesis, University of Kentucky, 2022. Author ORCID Identifier: https://orcid.org/0000-0002-3023-4284.

[24] Marc Illa Bello. Dysarthric speech synthesis via non-parallel voice conversion. Master's thesis, Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona, Universitat Politècnica de Catalunya, Barcelona, Spain, June 2021.

[25] Mohammad Soleymanpour, Michael T. Johnson, Rahim Soleymanpour, and Jeffrey Berry. Synthesizing Dysarthric Speech Using Multi-Speaker TTS for Dysarthric Speech Recognition. *Electrical and Computer Engineering, University of Kentucky, Lexington*, 2021.

[26] Frank Rudzicz, Aravind K. Namasivayam, and Tycho Wolff. The TORGO database of acoustic and articulatory speech from speakers with dysarthria. *Language Resources and Evaluation*, 46(4):523–541, 2012.

[27] X. Menendez-Pidal, J. B. Polikoff, S. M. Peters, J. E. Leonzio, and H. T. Bunnell. The nemours database of dysarthric speech. In *Proceeding of Fourth International Conference on Spoken Language Processing (ICSLP '96)*, pages 1962–1965 vol.3, Philadelphia, PA, USA, 1996.

[28] Heejin Kim, Mark Hasegawa-Johnson, Adrienne Perlman, Jon Gunderson, Thomas S. Huang, Kenneth Watkin, and Simone Frame. Dysarthric speech database for universal access research. In *Ninth Annual Conference of the International Speech Communication Association*, 2008.

[29] Edresson Casanova, Julian Weber, Christopher Dane Shulby, Arnaldo Cândido Júnior, Eren Gölge, and Moacir Antonelli Ponti. YourTTS: Towards Zero-Shot Multi-Speaker TTS and Zero-Shot Voice Conversion for everyone. In *International Conference on Machine Learning (ICML)*, 2021.

[30] Eugenio Martínez-Celdrán, Ana Ma. Fernández-Planas, and Josefina Carrera-Sabaté. Castilian spanish. *Journal of the International Phonetic Association*, 33(2):255–259, 2003.

[31] J. C. Wells. *SAMPA computer readable phonetic alphabet*, pages Part IV, section B. Mouton de Gruyter, Berlin and New York, 1997.

[32] Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. HiFi-GAN: Generative Adversarial Networks for Efficient and High Fidelity Speech Synthesis. In *34th Conference on Neural Information Processing Systems (NeurIPS2020)*, 2020.

[33] Hee Soo Heo, Bong-Jin Lee, Jaesung Huh, and Joon Son Chung. Clova Baseline System for the VoxCeleb Speaker Recognition Challenge 2020, 2020.

[34] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. VoxCeleb2: Deep Speaker Recognition. In *Proc. Interspeech 2018*, pages 1086–1090, 2018.

[35] Joon Son Chung, Jaesung Huh, Seongkyu Mun, Minjae Lee, Hee-Soo Heo, Soyeon Choe, Chiheon Ham, Sunghwan Jung, Bong-Jin Lee, and Icksang Han. In defence of metric learning for speaker recognition. In *Interspeech 2020*. ISCA, oct 2020.

[36] Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, page 4879–4883. IEEE Press, 2018.

[37] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust Speech Recognition via Large-Scale Weak Supervision. In *ICLR2021*.

[38] Kyubyong Park and Thomas Mulc. Css10: A collection of single speaker speech datasets for 10 languages. 2019. https://arxiv.org/pdf/1903.11269.pdf.

[39] Keith Ito and Linda Johnson. The LJ Speech Dataset. https://keithito.com/LJ-Speech-Dataset/, 2017.

[40] Christophe Veaux, Junichi Yamagishi, and Kirsten MacDonald. CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit. The Centre for Speech Technology Research (CSTR), University of Edinburgh, September 2019. Version 0.92.

[41] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018.

[42] Jialin Zhang, Mairidan Wushouer, Gulanbaier Tuerhong, and Hanfang Wang. Semi-supervised learning for robust emotional speech synthesis with limited data. *Applied Sciences*, 13(9):5724, May 2023.

[43] Chung-Ming Chien, Jheng hao Lin, Chien yu Huang, Po chun Hsu, and Hung yi Lee. Investigating on incorporating pretrained and learnable speaker representations for multi-speaker multi-style text-to-speech. *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8588–8592, 2021.

[44] Sercan Arik, Jitong Chen, Kainan Peng, Wei Ping, and Yanqi Zhou. Neural voice cloning with a few samples. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[45] Mingjian Chen, Xu Tan, Bohan Li, Yanqing Liu, Tao Qin, sheng zhao, and Tie-Yan Liu. Adaspeech: Adaptive text to speech for custom voice. In *International Conference on Learning Representations*, 2021.

[46] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[47] Qi Chen, Yuanqing Li, Yuankai Qi, Jiaqiu Zhou, Mingkui Tan, and Qi Wu. V2C: visual voice cloning. *CoRR*, abs/2111.12890, 2021.

[48] A.W. Rix, J.G. Beerends, M.P. Hollier, and A.P. Hekstra. Perceptual evaluation of speech quality (pesq)-a new method for speech quality assessment of telephone networks and codecs. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, volume 2, pages 749–752 vol.2, 2001.

[49] Cees H. Taal, Richard C. Hendriks, Richard Heusdens, and Jesper Jensen. An algorithm for intelligibility prediction of time–frequency weighted noisy speech. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(7):2125–2136, 2011.

[50] Sanyuan Chen, Chengyi Wang, Zhengyang Chen, Yu Wu, Shujie Liu, Zhuo Chen, Jinyu Li, Naoyuki Kanda, Takuya Yoshioka, Xiong Xiao, Jian Wu, Long Zhou, Shuo Ren, Yanmin Qian, Yao Qian, Jian Wu, Michael Zeng, Xiangzhan Yu, and Furu Wei. WavLM: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1505–1518, oct 2022.

[51] A. Martin, M. Przybocki, and M. Wester. The nist 2010 speaker recognition evaluation plan. *NIST Tech. Rep.*, 7:1–29, 2010.

[52] Voicemos challenge 2023. https://voicemos-challenge-2023.github.io/, 2023.

[53] A.V. Oppenheim and R.W. Schafer. From frequency to quefrency: a history of the cepstrum. *IEEE Signal Processing Magazine*, 21(5):95–106, 2004.

[54] Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Neural spline flows. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[55] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621, 2019.

[56] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[57] H. Kameoka, Li Li, Shota Inoue, and Shoji Makino. Semi-blind source separation with multichannel variational autoencoder. *ArXiv*, abs/1808.00892, 2018.

[58] Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C

Courville. Melgan: Generative adversarial networks for conditional waveform synthesis. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[59] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). 2016.

[60] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[61] Lilian Weng. Flow-based deep generative models. *lilianweng.github.io*, 2018.

[62] Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2722–2730. PMLR, 09–15 Jun 2019.

[63] Jianfei Chen, Cheng Lu, Biqi Chenli, Jun Zhu, and Tian Tian. VFlow: More expressive generative flows with variational data augmentation. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1660–1669. PMLR, 13–18 Jul 2020.

[64] Rodolfo Zevallos. Text-To-Speech Data Augmentation for Low Resource Speech Recognition. *arXiv e-prints*, page arXiv:2204.00291, April 2022.

[65] Computer Science Department at Winsconsin Madison University. Htcondor. https://research.cs.wisc.edu/htcondor/.

[66] Python. https://www.python.org/.

[67] Conda. https://docs.conda.io/.

[68] Mathieu Bernard and Hadrien Titeux. Phonemizer: Text to phones transcription for multiple languages in python. *Journal of Open Source Software*, 6(68):3958, 2021.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need.

In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[70] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[71] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[72] Jaehyeon Kim, Sungwon Kim, Jungil Kong, and Sungroh Yoon. Glow-tts: A generative flow for text-to-speech via monotonic alignment search. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 8067–8077. Curran Associates, Inc., 2020.

[73] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Lista de Figuras

Lista de Tablas

Anexos

Anexos A

Bloques del Sistema VITS *SingleSpeaker*

A.1. Conversión de texto a fonema

En este bloque, se lleva a cabo todo el procesamiento, extracción, simplificación, normalización y análisis del texto para adaptarlo a una representación fonética adecuada. Esta etapa es crucial en todos los sistemas TTS, aunque en VITS no se considera como un bloque propio del modelo porque el sistema ya debe recibir los fonemas como entrada.

En primer lugar, se realiza la tarea de **normalización de texto**, que consiste en transformar las palabras que contengan alguna **forma no ortográfica** o **clase semiótica** al formato hablado, de manera que el sistema pueda comprenderlo adecuadamente. Algunas de estas clases semióticas pueden ser abreviaturas, acrónimos, formatos numéricos, etc. Inicialmente esta tarea era realizada con normas descritas, pero en la actualidad, en algunos casos, se implementa a través de redes neuronales como una tarea de comportamiento *sequence-to-sequence*.

Tras la normalización de texto, sigue el **análisis fonético**, una etapa que puede ser muy compleja en algunos idiomas. En esta fase, se realizan dos tareas fundamentales: la **desambiguación polifónica** y la **conversión grafema-fonema**. La primera tarea tiene como objetivo resolver las ambigüedades presentes en palabras que se escriben igual pero tienen diferentes pronunciaciones y significados, por ejemplo en inglés la palabra *"lead"*. Afortunadamente en español, aunque hay palabras homófonas, que se escriben y pronuncian igual, la ambigüedad polifónica no es común.

Posteriormente, se realiza la conversión de **grafemas** (caracteres) a **fonemas** (pronunciación). En español, esta tarea suele ser sencilla en comparación con otros idiomas, ya que puede ser llevada a cabo mediante reglas manuales bien establecidas. Los fonemas obtenidos pueden pertenecer a distintos diccionarios fonéticos con una

notación estandarizada como IPA[30] o SAMPA[31] (Vea Anexo D).

A.2. *Text Encoder* y Proyección Lineal de los *Embeddings*

El *Text Encoder* procesa los fonemas de entrada (c_{text}) y los transforma en una representación compacta dentro de un espacio vectorial que representa al fonema (h_{text}). Esta representación vectorial se denomina ***embedding***.

La estructura del *Text Encoder* se basa en un *encoder* basado en un modelo de tipo **transformer**, que utiliza una representación posicional relativa[73] en lugar de absoluta (Vea Anexo E.2). La diferencia fundamental entre la representación posicional relativa y absoluta radica en la relación entre los elementos de la secuencia de entrada. Si se emplea una representación absoluta, se considera cada elemento de la secuencia de entrada de manera aislada. Por otro lado, con una representación relativa, se tienen en cuenta las relaciones posicionales entre elementos adyacentes de la secuencia de entrada, obteniendo así patrones contextuales más informativos. A la salida del *Text Encoder*, se plantea otro bloque que se encarga de hacer una proyección lineal de los embeddings de texto, generando la media y la varianza de cada embedding, que serán utilizados para crear una distribución de probabilidad a priori ($\mu_\theta, \sigma_\theta$) (Para ver en más detalle la configuración del Text Encoder vea Anexo B.1).

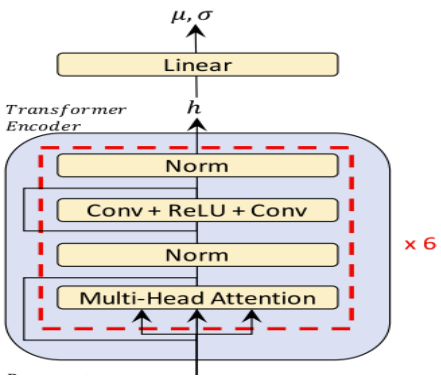


Figura A.1: Esquema *Text Encoder*[72].

A.3. Conversión de voz a Espectrograma y MFCC

En la fase de entrenamiento, es esencial proporcionar al modelo de acceso a las características acústicas de la voz. Estas características se introducen en el sistema en

forma del espectrograma de los audios. Para el cálculo de las pérdidas de reconstrucción se usarán los MFCC, que se calculan a partir del espectrograma Mel.

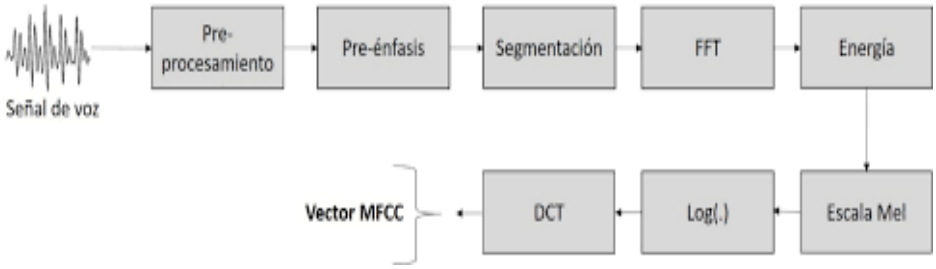


Figura A.2: Cálculo de los Coeficientes MFCC

La representación frecuencial de una señal de audio se obtiene utilizando la Tranformada de Fourier. Primero se aplica un filtro de preénfasis que consiste en un filtrado paso alto que compensa la atenuación de alta frecuencia causada por la radiación de la señal de voz al salir de la boca. Luego se enventana la señal en segmentos temporales solapados que cumplan la propiedad de quasiestacionariedad (en la voz esto se cumple para segmentos mínimos de 20-30 ms). El enventanado permite capturar información localizada en el tiempo preservando la variabilidad temporal de la señal. A continuación se aplica la transformada de fourier a corto término, conocida como "*Short-Term Fourier Transform*"(**STFT**) y se utiliza su parte real, obteniendo una señal en dos dimensiones que muestra cómo evoluciona la energía espectral en función del tiempo y la frecuencia. Esta representación frecuencial lineal se lleva a la escala logaritmica Mel aplicando un banco de filtros de la forma:

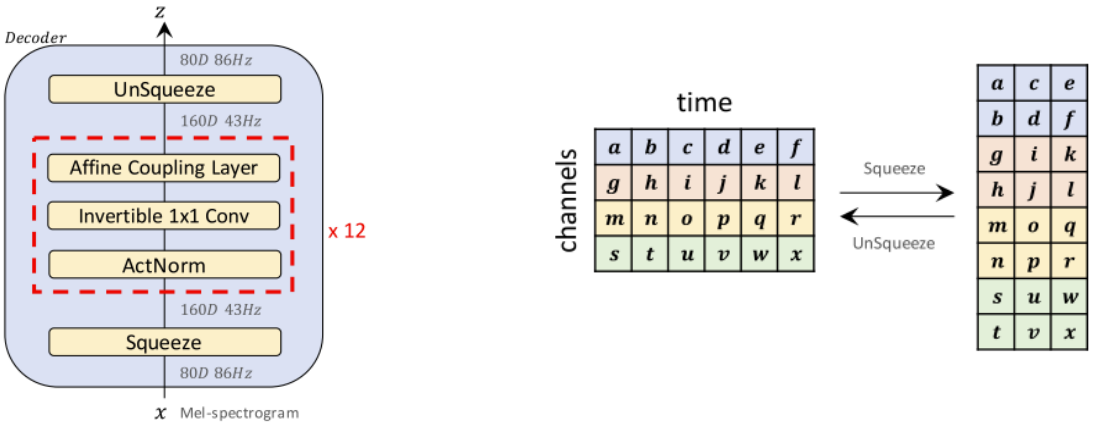
$$Mel(f) = 1127 \cdot ln(1 + \frac{f}{700}).$$

La escala Mel es una adaptación a las frecuencias que procesa el sistema de audición humano. Finalmente se realiza una transformación homomórfica que consiste en aplicar el logaritmo seguido de una Transformada Discreta de Coseno (DCT) para llevar el espectro en escala Mel al dominio cepstral. De esta forma se obtienen los MFCC [53]. Los hiperparámetros escogidos se encuentran explicados en el Anexo B.2.

A.4. *Spectrogram Encoder*

Durante la etapa de entrenamiento, es necesario contar con audio como entrada. Con el fin de interpretar dicho audio de manera eficaz, el sistema recibe la representación frecuencial, a partir de la cual se calcula la representación latente. Anteriores trabajos[72, 16] utilizan los MFCC como entrada, mientras que en este se opta por el espectrograma con el objetivo de conseguir una mayor resolución del audio.

Este bloque, junto al *decoder*, conforman la estructura de un *variational autoencoder* o VAE (Vea Anexo E.3.1). El VAE está condicionado por el factor de alineamiento y por una distribución de probabilidad a priori, obtenida en la sección A.2. Para obtener la distribución de probabilidades a posteriori se utiliza este *encoder* por lo que también se le denomina ***Posterior Encoder***. El esquema del *encoder* se ilustra en la Figura A.3a y se aprecia que al tener capas de acoplamiento afín está basado en la idea de normalización de flujos (Vea Sección A.5).



(a) *Spectrogram Encoder* (b) Redimensión de los datos

Figura A.3: Esquema del *Spectrogram Encoder*. Imágenes tomadas de *GlowTTS*[72]

En primer lugar, para mejorar la eficiencia computacional, el *encoder* recibe como entrada la representación lineal del espectrograma y divide cada canal en dos mitades en distintos canales reduciendo así el tiempo de cómputo del Jacobiano. Así que, cuando la entrada tenga 80 canales, tras este bloque habrán 160 canales de la mitad de duración (Figura A.3b). Si el número de ventanas fuera impar, se debería despreciar la última ventana, eliminando así una cantidad de audio alrededor de 11 ms. Tras redimensionar la entrada, se realizan 12 bloques con capas basadas en modelos de flujo, cuya función es obtener las características acústicas del espectrograma de entrada (Vea Anexo B.3). Por último, a la salida de este bloque se realiza otra conversión de dimensiones para ajustar la dimensión final de 192 canales del espacio latente **z**.

A.5. Normalización de flujo

El bloque de normalización, o ***Normalizing Flows***[56], está formado por cuatro bloques con capas de acoplamiento afin, como las 12 utilizadas en el *Spectrogram Encoder* (Figura A.3a). A diferencia de las capas utilizadas por el *Spectrogram Encoder* que en su primer bloque transformador usaba un bloque residual WaveNet con 16 capas, en este caso, se reduce el número de capas a cuatro con la misma estructura

(Vea Anexo B.3). Este bloque realiza una proyección del espacio latente para mejorar la flexibilidad de la representación latente del *autoencoder* (**z**). A la salida del Encoder se implementa este bloque que corresponde a un modelo generativo que tranforma una función de probabilidad de datos complejos en nuevas distribuciones aplicando transformaciones invertibles. El propósito de estas transformaciones es obtener una nueva representación de los datos más sencilla o flexible, de la cual se pueda volver al resultado inicial aplicando la transformada inversa (Figura A.4).

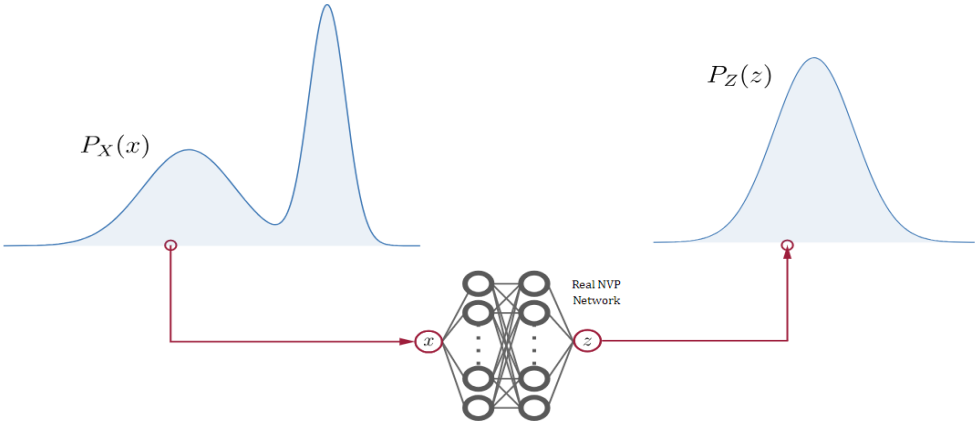


Figura A.4: Ejemplo de Normalización de flujos

VITS propone la incorporación de estos bloques con el propósito de obtener una mayor flexibilidad en la representación del espacio latente de su *autoencoder*. Esta idea en sistemas TTS ya se tiene en cuenta en el modelo *GlowTTS*[72] y se basa en la teoría básica de los ***modelos de flujos***. Para definir estos modelos, se consideran dos variables aleatorias **x** y **z** con una función de distribución de probabilidad p_x y p_z y la transformada invertible **h** donde $z = h(x)$ y $x = g(z)$, siendo $g = h^{-1}$.

Normalmente, se suele considerar p_x como una distribución de entrada simple, como puede ser la Gaussiana, y se le aplica una transformación **h** para modelar una distribución más compleja **z**. En el Anexo E.5.1 se presenta el desarrollo matemático para obtener la función de verosimilitud a través de un cambio de variable. Esto permite relacionar la función de densidad de probabilidad de la entrada con la de salida:

$$\begin{aligned} p_z(z) &= p_x(x)|det\frac{\partial h(x)}{\partial x}|^{-1} \\ log(p_z(z)) &= log(p_x(x)) - log|det(\frac{\partial h(x)}{\partial x})|. \end{aligned} \tag{A.1}$$

El problema es que tanto calcular el **Jacobiano**[1] de funciones de gran-dimensionalidad como los determinantes de grandes matrices, son tareas

[1]En la ecuación A.1 - $det(\frac{\partial h(x)}{\partial x})$ es el Jacobiano

de alto coste computacional. Esto combinado con la necesidad de utilizar funciones biyectivas dificultan la utilización de estos modelos. Así que varios métodos fueron propuestos para satisfacer esta necesidad, siendo las **capas de acoplamiento afín**[56] uno de los más populares (Vea el desarrollo de la demostración de las capas de acoplamiento afín en el Anexo E.5.2). El uso de estas capas en lugar de la inversa permitirá reducir el coste en la inferencia.

Extrapolando este modelo al VITS, se tiene que la distribución $p_z(z)$ se trata del espacio latente del autoencoder condicionado a la distribución a priori ($p_\theta(z|c)$). Respecto a la distribución $p_x(x)$ se considera a la salida del bloque de normalización de flujo denominado $f_\theta(z)$. Además, como sabemos que este bloque proviene de la salida del alineador se va a tener una distribución normal que dependerá de la distribución de entrada, por lo que se puede representar como $N(f_\theta(z); \mu_\theta(c), \sigma_\theta(c))$, donde **c** representa todas las condiciones que tiene el VAE: los fonemas de entrada y la salida del alineador (c=$[c_{text}, A]$). Al extrapolar las fórmulas de la ecuación A.1,se obtiene la función de verosimilitud que se utiliza para entrenar a este modelo y encontrar el valor θ óptimo que se ajuste al modelo:

$$\begin{aligned} p_\theta(z|c) &= N(f_\theta(z); \mu_\theta(c), \sigma_\theta(c))|\frac{\partial f_\theta(z)}{\partial z}| \\ c &= [c_{text}, A] \end{aligned} \tag{A.2}$$

$$log(p_\theta(z|c)) = log(N(f_\theta(z); \mu_\theta(c), \sigma_\theta(c))) + log(|\frac{\partial f_\theta(z)}{\partial z}|) \tag{A.3}$$

A.6. Técnicas de Alineamiento

Una parte de las más importantes dentro de los sistemas de conversión de texto a voz es el alineamiento entre los fonemas y los grafemas. Para estimar el alineamiento **A** entre el texto de entrada y el audio, se adopta el algoritmo denominado "*Monotonic Alignment Search*"(**MAS**), desarrollado en *GlowTTS*[72].

Este método busca el alineamiento más probable entre las variables del espacio latente (z) y las estadísticas de la distribución a priori ($\mu_\theta, \sigma_\theta$). En *GlowTTS* plantean maximizar la función de verosimilitud:

$$\begin{aligned} A &= \arg\max_{\hat{A}} log(p(x|c_{text}, \hat{A})) \\ &= \arg\max_{\hat{A}} log(N(f(x); \mu(c_{text}, \hat{A}), \sigma(c_{text}, \hat{A})) \end{aligned} \tag{A.4}$$

Para esto se propone una solución recursiva sobre los alineamientos parciales para luego buscar el alineamiento final. Así que se plantea una matriz de coste $Q_{i,j}$ definida por

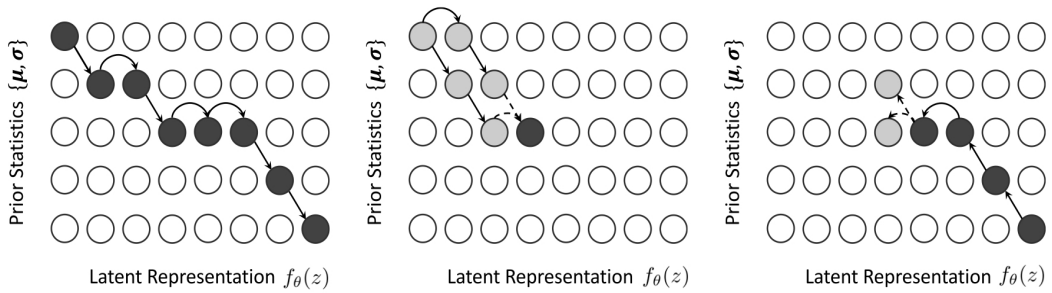
esta función de máxima verosimilitud;

$$Q_{i,j} = \max_{A} \sum_{k=1}^{j} logN(z_k, \mu_{A(k)}, \sigma_{A(k)}) = max(Q_{i-1}, Q_{i,j-1}) + log(N(z_j; \mu_i, \sigma_i)) \quad \text{(A.5)}$$

Luego se itera sobre todos los valores de Q hasta completar la matriz Q de dimensión $[T_{text}, T_{mel}]$. De manera similar, el alineamiento más probable A^* viene determinado por el mejor valor de Q en una relación de recurrencia. De este modo, A^* se obtiene de manera eficiente con programación dinámica almacenando en caché todo los valores de Q, y los valores A^* se obtienen mediante el método *backpropagation* (Figura A.5).

El algoritmo *MAS* es un una elección sólida para identificar el alineamiento óptimo entre texto y voz, debido a que obliga a mantener la monotonía y evita la omisión o repetición de palabras. De esta forma se adapta al proceso de lectura en español que se realiza de izquierda a derecha en una secuencia oredenada, sin saltar ni repetir palabras. Esta idea que parece tan obvia no estaba claramente definida en anteriores sistemas como Tacotron2[16]. Este modelo usaba un mecanismo de *sequence-to-sequence* basado en RNN para alinear. Este mecanismo tenía un comportamiento autorregresivo al que no se le imponían restriciones y suele fallar cuando se le introducían secuencias largas de texto.

El tiempo de cómputo de algoritmo está definido por un orden de complejidad $\mathbf{O}(T_{text}, T_{mel})$. A pesar de que el algoritmo no es susceptible a una paralelización, su ejecución se puede realizar de manera eficiente en la CPU requiriendo menos de 20 ms por cada iteración, lo que representa menos del 2 % de todo el tiempo de entrenamiento[72]. Además, es importante destacar que este algoritmo no se tendrá que ejecutar durante la etapa de inferencia, ya que el alineamiento se calculo usando el Predictor Estocástico de Duración(Vea sección A.7).



(a) An example of monotonic alignments

(b) Calculating the maximum log-likelihood Q.

(c) Backtracking the most probable alignment A^*.

Figura A.5: Esquema Alineamiento[72]

A diferencia de *GlowTTS* que tiene como objetivo maximizar la función de verosimilitud, *VITS* tiene como objetivo maximizar la función del límite inferior

variacional de verosimilitud (**ELBO**) (Vea Anexo E.3.1) y no la función de verosimilitud. Inicialmente parece un problema, pero afortunadamente como vemos en la ecuación A.6, al definir la función de ELBO, solo un parámetro es dependiente del alineamiento A^* y al reescribir la fórmula queda un algoritmo similar al implementado en *GlowTTS*[72] (Figura A.5).

$$\begin{aligned} A &= \arg\max_{\hat{A}} log(p_\theta(x_{mel}|z)) - log\frac{q_\phi(z|x_{lin})}{p_\theta(z|c_{text}, \hat{A})} \\ &= \arg\max_{\hat{A}} log(p_\theta(z|c_{text}, \hat{A})) \\ &= \arg\max_{\hat{A}} log(N(f_\theta(z); \mu_\theta(c_{text}, \hat{A}), \sigma_\theta(c_{text}, \hat{A})) \end{aligned} \tag{A.6}$$

Con la conclusión de la fórmula A.6, ya se podría realizar el algoritmo MAS que se programaría con el pseudocódigo del algoritmo 1.

Algorithm 1 *Monotonic Alignment Search*(**MAS**)[72]

Require: Representación Latente *Spectrogram o Posterior Encoder*: $f_\theta(z)$, Estadísticas de la distribución a priori:μ, σ, Longitud Mel-Espectrograma: T_{mel}, Longitud secuencia texto: T_{text}

Ensure: Monotonic alignment A^*

Initialize Q, con $-\infty$, una matriz de dimensiones (T_{text}, T_{mel}), para almacenar la caché de la función límite inferior de verosimilitud(**ELBO**).
for $j = 1$ to T_{mel} **do**
    $Q_{1,j} \leftarrow \sum_{k=1}^{P_j} \log N(f_\theta(z_k); \mu_1, \sigma_1)$
end for
for $j = 2$ to T_{mel} **do**
    for $i = 2$ to mín(j, T_{text}) **do**
        $Q_{i,j} \leftarrow$ máx$(Q_{i-1,j-1}, Q_{i,j-1}) + \log N(f_\theta(z_j); \mu_i, \sigma_i)$
    end for
end for
Initialize $A^*(T_{mel}) \leftarrow T_{text}$
for $j = T_{mel} - 1$ to 1 **do**
    $A^*(j) \leftarrow \arg\max_{i \in \{A^*(j+1)-1, A^*(j+1)\}} Q_{i,j}$
end for
return A^*

A.7. Predicción Estocástica de la duración

El objetivo principal del **Predictor Estocástico de la duración** es proporcionar una estimación probabilística del tiempo que debe durar cada unidad de habla en la salida. Esta duración viene condicionada por una serie de factores de entrada como puede ser el contexto lingüístico y el estilo de habla.

Para empezar, se puede calcular la duración de cada token d_i sumando todas las columnas de la estimación del alineamiento $\sum_j A_{i,j}$. Esto puede ser usado para calcular la predicción determinista de la duración. Sin embargo, esto no permite modelar la pronunciación de una persona o su ritmo de habla en diferentes contextos. Para conseguir generar estas diferencias en la forma del habla, se debe diseñar un bloque de **Predicción Estocástica de la Duración**(***SDP*** o "*Stochastic Duration Predictor*").

Este predictor es un modelo generativo basado en flujo entrenado a través de la función de estimación de máxima verosimilitud. Aplicar la función de máxima verosimilitud de manera directa es difícil debido a que la duración de cada fonema de entrada está representado mediante un valor entero discreto y por un escalar. El valor entero hace necesario realizar una **decuantificación variacional**[62] de los datos discretos para poder aplicar los modelos de flujo continuo. Esta considera el proceso de cuantización como una distribución continua sobre los datos discretos. Mientras que la representación por escalares evita conseguir una transformada de alta dimensionalidad por la necesidad de que el valor sea invertible. Para solucionar esto se aplica una técnica denominada ***Variational Data Augmentation***[63]. Para aplicar el aumento de datos, se introducen dos variables denominadas **u** y **v**, que tienen la misma resolución y dimensión que la secuencia de duración de los tokens, **d**. Los valores de **u** se encontrarán restringidos en un intervalo [0,1), para que la diferencia de **d-u** sea una secuencia de números reales positivos. Tras esto se concatenan las secuencias **v** y **d** en cada canal para conseguir una representación latente de mayor dimensión. Se representan estas dos variables a través de una distribución a posteriori $q_\phi(u, v|d, c_{text})$, obteniendo como función de pérdidas la función de límite inferior variacional de la función de verosimilitud(**ELBO**) (Para comprender el desarrollo vea Anexo E.3.1).

$$log(p_\theta(d|c_{text})) \geq \mathbb{E}_{q_\phi(u,v|d,c_{text})}[log(\frac{p_\theta(d-u,v|c_{text})}{q_\phi(u,v,d|c_{text})}] \tag{A.7}$$

De la fórmula A.7, se puede obtener la métrica que modela el error del predictor y que permite su optimización. En la Figura A.9a, se observa que una entrada del predictor son los *embeddings* de los fonemas, pero la optimización de este módulo es independiente del resto de módulos que no estén relacionados con el alineador, así que se le aplica un **operador de fin de gradiente** para evitar que la propagación del error por el resto de la red (Vea detalles del funcionamiento del predictor en el Anexo B.4).

A.8. *Decoder*

El decoder es la última parte del sistema que transforma la representación latente de las características acústicas ($\mathbf{z}$) en una señal de audio. El decoder usado es la versión V1 del generador HiFiGAN[32]. Este es un modelo de GAN(*Generative Adversarial Network*) (Vea Anexo E.4), especializada en la generación de audio que está formada por un generador y dos discriminadores.

La estructura de las GANs consiste en el intento de la optimización de dos bloques que compiten entre sí, el generador de datos artificiales y el discriminador que clasifica los datos en reales y artificiales. Esta estructura se optimiza a través de un entrenamiento adversario. HiFiGAN introdujo mejoras en la generación de audio consiguiendo una calidad similar a modelos autorregresivos con una mejor eficiencia computacional.

A.8.1. Generador

El generador adopta la forma de una red convolucional con 192 canales de entrada que recibe como entrada las características acústicas y las interpola hasta que la dimensión coincida con la resolución temporal del audio. Este proceso lo realiza a través de **capas de deconvolución**. Estas capas realizan una transformación inversa a la convolución estándar, tomando una entrada de dimensiones reducidas y expandiendola para obtener una salida de mayor resolución (Figura A.6).

El bloque MRF se trata de un bloque residual con capas convolucionales y una conexión residual. La idea es que esta conexión directa permita que el bloque residual aprenda la diferencia entre la salida deseada y la salida actual de una manera más sencilla. Si la capa aprende a identificar esta diferencia, la transformación que realiza se convierte en un "residuo"que se suma a la entrada original para obtener la salida deseada. Con esto se consigue que el sistema aprenda los patrones de periodicidad de la señal de audio. Los hiperparámetros usados para el bloque V1 están explicados en el Anexo B.5.

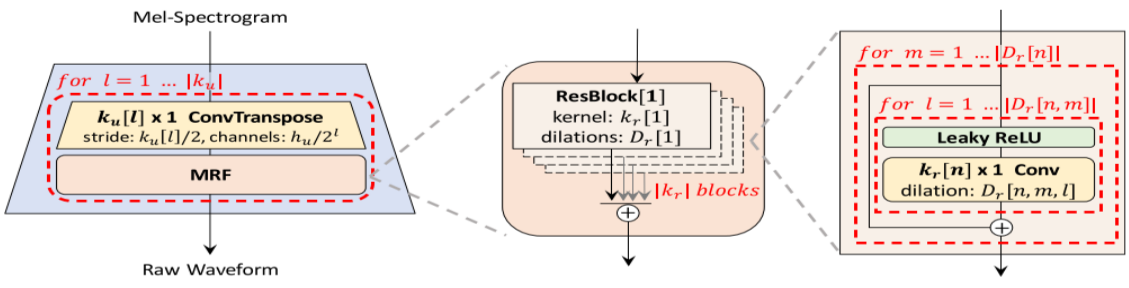


Figura A.6: Generador HiFiGAN

A.8.2. Discriminador

Identificar relaciones a largo plazo es una de las características importantes que debe tener un discriminador en tareas de audio. Esto es debido a que un fonema tiene una duración aproximada de 100 ms, que a una frecuencia de 16 kHz son 1600 muestras que están altamente correladas para representar el fonema. Para solucionar esto, se proponen discriminadores multiperiodo y multiescala.

Los discriminadores multiperiodo son un tipo de discriminadores que solo aceptan muestras equiespaciadas una distancia denominada periodo **p**. Estos subdiscriminadores están diseñados para capturar diferentes estructuras mirando a diferentes partes de audio. Como se ve en la Figura A.7.b, estos disriminadores cambian la estructura de la señal del audio temporal en una dimensión T, a una matriz de datos 2D de dimensiones **[T/p,p]**. En HiFiGAN se eligieron cinco subdiscriminadores que usarán diferentes periodos definidos a [2,3,4,7,11] para intentar evitar solapamientos. Tras esto cada subdiscriminador tiene una entrada de diferente dimensión a la que se le aplican varias capas convolucionales como se ilustra en la Figura A.8a.

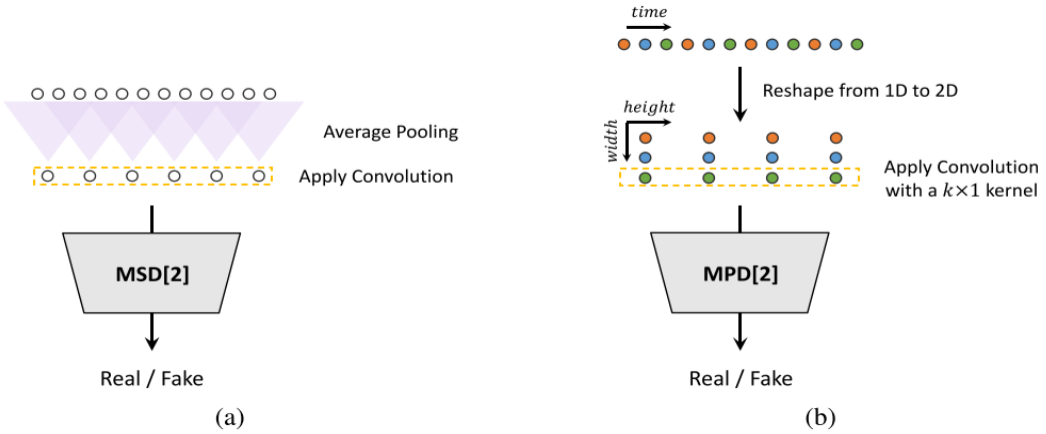
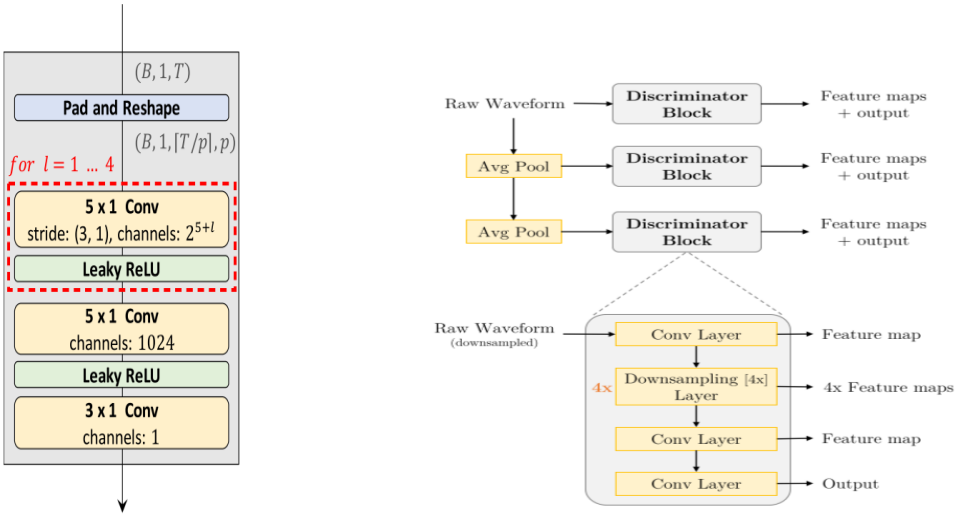


Figura A.7: Estructura MPD y MSD Discriminador[32]

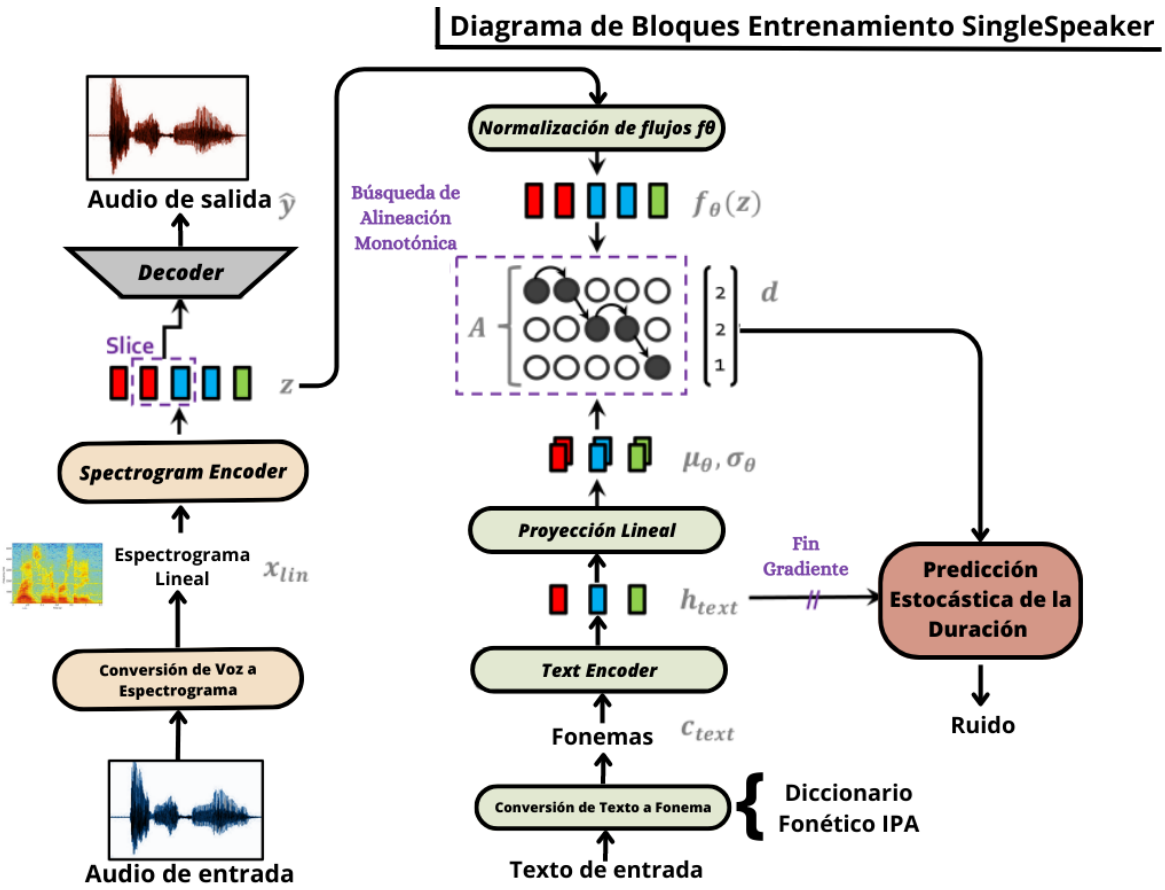
Como el discriminador multiperiodo trabaja con muestras separadas, a cada subdiscriminador se le implementa tambien la estructura de un discriminador multiescala. Este bloque, implementado en MELGAN[58], es una mezcla de tres discriminadores que trabajan a tres escalas: la forma temporal del audio, en un promedio temporal del audio x2 y en el promedio temporal x4, como se aprecia en A.8b. Mezclando estas dos arquitecturas se consigue tener conocimiento de muestras independientes y características del audio suavizado al promediar, obteniendo así una mejor prestación a la hora de discriminar de la veracidad o no del audio.



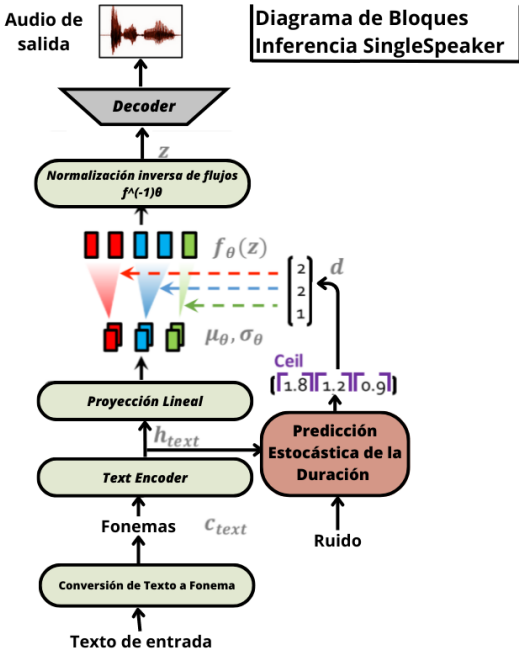
(a) Bloque MPD[32]. Se emplea un kernel 5x1 desplazando a lo ancho, procesando muestras periódicas de manera independiente. Utiliza salto de 3x1 y función de activación ReLU. Finalmente una convolución final clasifica la autenticidad.

(b) Bloque MSD[58]. Compuesto por capas convolucionales con funciones de activación ReLU. El tamaño del discriminador va aumentando reduciendo el salto y aumentando el número de capas. Finalmente se consigue características en las tres escalas

Figura A.8: Bloque MPD y MSD



(a) Entrenamiento.



(b) Inferencia.

Figura A.9: Diagrama en bloques del modelo VITS de un solo orador

Anexos B

Hiperparámetros y pérdidas del modelo

En la Sección 3 se ha explicado el comportamiento de los distintos bloques del modelo. Estos bloques son de gran complejidad y tienen muchos parámetros ajustables que no se aprenden en el entrenamiento. Estos parámetros ajustables, o hiperparámetros, influyen a la hora de entrenar el modelo y tienen un impacto significativo en su rendimiento y capacidad de generalización. De estos hiperparámetros dependerá el número de parámetros que tenga nuestro modelo, llegando a encontrar el valor óptimo entre el rendimiento del modelo y coste computacional. A la hora de escoger los hiperparámetros se consideraron trabajos previos donde se realiza un estudio de los valores óptimos.

B.1. Hiperparámetros *Text Encoder*

El *Text encoder* escogido en el trabajo es una variación del usado en *Glow TTS*[72], que adopta una configuración *Multi-Head-Attention* y representación posicional relativa. La configuración ***Multi-Head-Attention*** es de *"dos cabezas"*, que implica que se implementan dos mecanismos de atención independientes para procesar la información de forma paralela e independiente. La salida final será una combinación de ambas salidas, como indica el esquema de la Figura B.1. Se aplica una técnica de regularización denominada *dropout* para mitigar el riesgo de sobreajuste o *overfitting*. Está técnica consiste en desactivar aleatoriamente un cierto número de unidades de la red durante el entrenamiento.

Respecto a los hiperparámetros definidos será el número de cabezas igual a 2, el número de capas del transformer a 6, la dimensión de los canales ocultos a 256 y tamaño del núcleo o *kernel* de la capa convolucional a 3. Además, se deberá definir el número de unidades desactivadasdurante el entrenamiento por el dropout. A esto se le denomina coeficiente de dropout que se establece a 0.1 indicando que se desactivarán de manera

aleatoria el 10 % de las unidades en cada iteracción de entrenamiento. (Figura B.1).

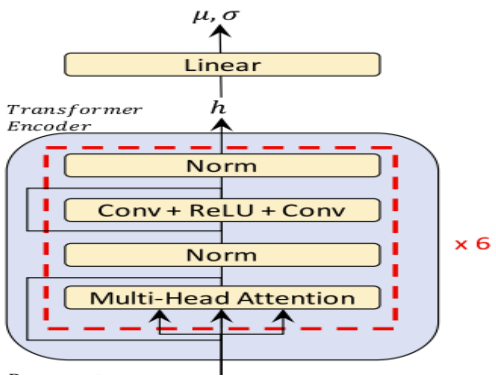


Figura B.1: Esquema *Text Encoder*[72].

B.2. Hiperparámetros *Conversión de Voz a MFCC*

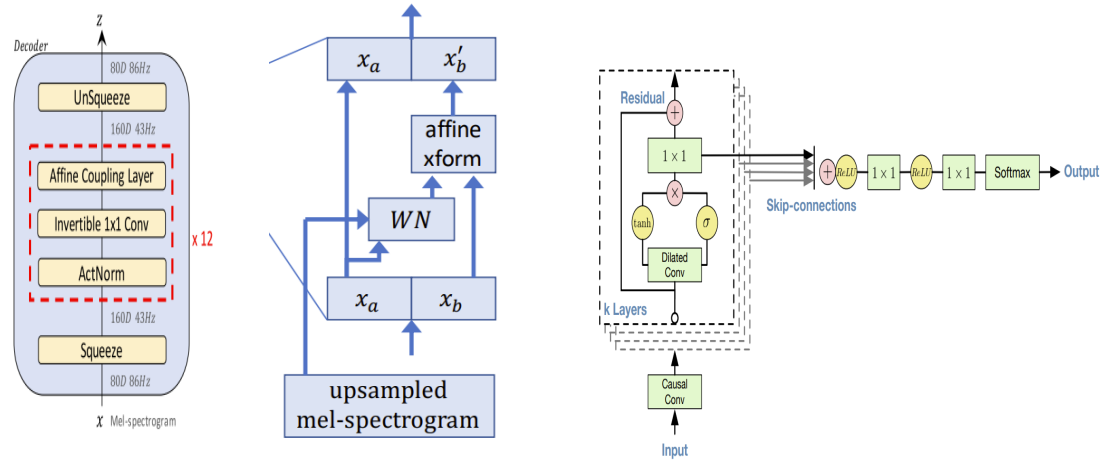
En el bloque de conversión de voz a MFCC se han decidio escoger los siguientes hiperparametros. La dimensión de la Transformada de Coseno Discreta (DCT) se ha establecido en 80 ventanas para capturar características espectrales relevantes. También, se ha elegido un tamaño de ventana FFT de 1024 muestras que coincide con la duración de la ventana. Por último, se ha escogido una longitud de salto de 256 muestras, buscando con estos valores tener un equilibrio entre la resolución espectral y temporal en el espectrograma[55].

Con el número de muestras de la ventana y el salto definidos, se tienen ventanas de alrededor de 50 ms, siendo aproximadamente la mitad de la duración de un fonema, consiguiendo muestras que están altamente correladas. Además, con el número de ventanas definidas a 80, se consigue tener una gran resolución frecuencial que nos permitirá obtener de manera más precisa los MFCC.

B.3. Hiperparámetros *Spectrogram Encoder*

Este *encoder*, está constituido por una estructura de 12 bloques cuya función es representar las características acústicas del audio de entrada. Para empezar el bloque, a la entrada se le aplica una función de activación que la normaliza. Al normalizar la entrada, se consigue una mayor estabilidad del entrenamiento y convergencia más rápida, debido a que las características tienen escalas menores. Tras esto como se aprecia en la Figura B.2e, se divide la entrada en 40 grupos de 4 canales y se le

aplica de manera separa una capa convolucional invertible con un kernel de dimensión 5. Una capa de convolución invertible es un tipo de capa convolucional que permite deshacer sus operaciones y recuperar la entrada original. A la salida de la convolución, se aplica una capa de acoplamiento afín que se trata de una estructura que a través de propiedades matemáticas (Vea Anexo E.5.1) consigue convertir realizar un bloque de flujos eficiente computacionalmente. Esta capa sigue la arquitectura similar a la capa de acoplamiento afín utilizada en WaveGlow [55], representado en la Figura B.2b. La diferencia introducida es que no se aplica ninguna condición local, es decir, el espectrograma no actúa como entrada directa en el bloque transformardor WaveNet. Esta capa a su vez está formada por dos bloques transformadores: 1)Bloques residuales de WaveNet[14] (Figura B.2c), 2) Bloque de transformación afín.



(a) *Spectrogram Encoder*

(b) Capa de acoplamiento afín

(c) Esquema Bloque Residual WaveNet

(d) Esquema *Squeeze y Unsqueeze*

(e) Esquema Convolución 1x1

Figura B.2: Esquema del *Spectrogram Encoder*. Imágenes sacadas de [55, 72, 14]

El primer transformador, está compuesto por el bloque residual de WaveNet. Este bloque se utiliza para capturar patrones a diferentes escalas temporales. Su estructura se basa en 16 series de capas compuestas por convoluciones dilatadas y convoluciones 1x1. Estas capas de convolucion tienen un kernel fijo a 5 y una tasa de dilatación que aumenta un factor 2 en cada capa, para así capturar diferentes escalas temporales.

Es importante indicar que la entrada que entra al modelo es causal, siendo un factor importanta para la generación autoregresiva. Otras propiedades importantes de este modelo es que a cada capa convolucional se conecta a la salida de las capas anteriores, generando una especie de mecanismo de atención que le permite tener una memoria a mayor plazo[1] Por último en la salida se aplica una conexión residual, de manera que La salida de las capas convolucionales se suma a la entrada original antes de pasar por la función de activación. Así se obtiene la salida del primer trasnformador, respecto a la segunda transformación afín se trata de un conjunto de transformadas que conserven la colinealidad y proporción de los datos.

Este bloque también será usado en la normalización de flujos pero cambiando el númeor de capas usadas

B.4. Bloque de Predicción de Duración Estocástica

En la Figura B.3, se observa el funcionamiento del predictor en entrenamiento e inferencia. En el entrenamiento, la secuencia de los tokens de duración de los fonemas y los embeddings de texto son procesados por un bloque denominado *Conditional Encoder*. Este bloque, como se ve en la Figura B.4.a, consiste en dos capas convolucionales 1x1 y un bloque más complejo denominado **Bloque Residual de Convolución Dilatada y Separada en Profundidad**, representado en la Figura B.3.c.

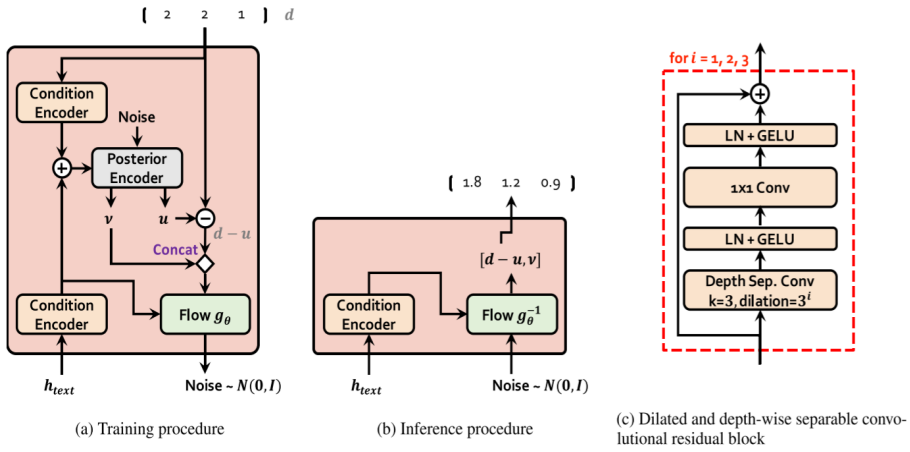
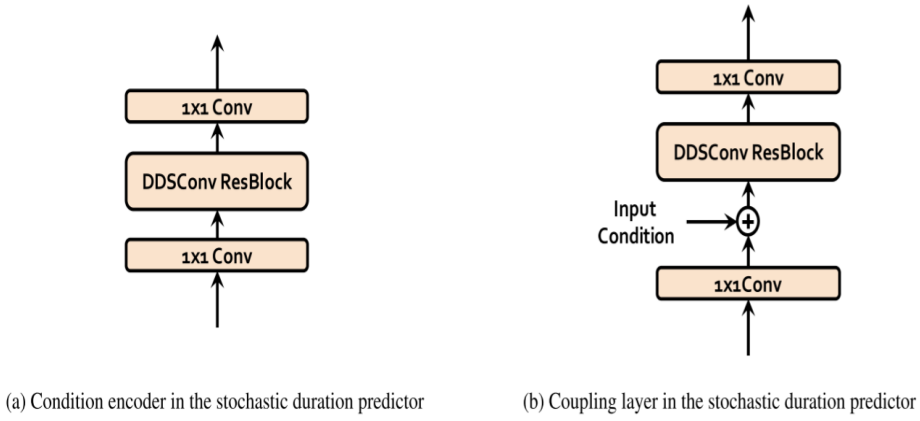


Figura B.3: Bloque de Predicción de Duración Estocástica

Este bloque tiene en primer lugar un subbloque que combina dos variaciones de las capas convolucionales: la **convolución dilatada** y la **convolución separada**

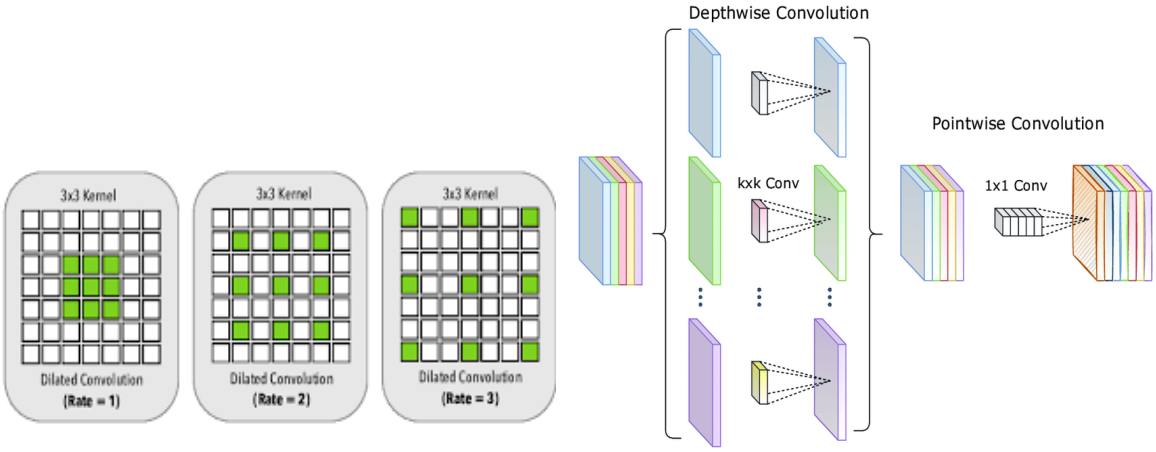
[1]Esta estructura es similar a una *Gated Recurrent Units*, variante de las unidades de memoria a largo plazo (LSTM), que también se utilizan en RNNs para manejar dependencias a largo plazo en secuencias



(a) Condition encoder in the stochastic duration predictor (b) Coupling layer in the stochastic duration predictor

Figura B.4: *Conditional Encoder*

en profundidad. La convolución dilatada es una variación de la convolución en la que se introduce un factor de dilatación en el núcleo, permitiendo que se aumente el campo receptivo del filtro sin aumentar el número de parámetros. Mientrás que la convolución separada en profundidad consta de dos etapas: una primera donde realiza la convolución de cada canal individualmente y otra donde aplica una convolución 1x1 a todos los canales para combinar las características de los canales individuales, permitiendo reducir así el número de operaciones. Tras este bloque se aplica una normalización de las capas con una función de activación no lineal *GELU*[59].



(a) Convolución dilatada (b) Convolución separada en Profundidad

Figura B.5: Tipo de Convoluciones

Respecto al *"Posterior Encoder"* y al bloque de flujos ambos son modelos de generación **basados en flujos** con una estructura similar, explicada en la sección A.5. La diferencia entre ellos radica en que el *"Posterior Encoder"* transforma una secuencia de ruido Gaussiano en dos variable aleatorias **v** y **u** para expresarlo como una distribución posterior denominada $q_\phi(u, v|d, c_{text})$, mientras que el módulo de

normalización de flujos convierte la secuencia **d-u** y **v** en una secuencia de ruido Gaussiano utilizada para obtener la función de pérdidas.

Estos bloques están formados por cuatro capas de *"Neural Spline Flow"*[54], modelo de generación basado en las funciones de spline. Estas capas adoptan la forma de transformaciones no lineales invertibles utilizando en las capas de acoplamiento splines racionales cuadráticos y monótonos. Con esto se consigue mejorar la expresividad de las transformaciones frente a la capas de acoplamiento afines utilizadas comúnmente, explicadas en la sección A.5. En la Figura B.4.b, se ve el esquema de esta capa que procesa la entrada y las condiciones de entrada a través del bloque DDSCov y produce parámetros de dimensión de canal 29, que suelen ser usados para construir 10 funciones cuadráticas racionales.

En la Figura B.3.b, se puede observar el predictor en la inferencia. En esta etapa se tiene una secuencia de ruido gaussiano como entrada al bloque de normalización de flujo, el cuál al realizar la transformada inversa, condicionada por el fonema a predecir, debe obtener como salida la secuencia que modela el vector **d-u** concatenado con **v**. Al tener estos vectores concatenados, se puede separar la secuencia **d-u** y como al saber que **u** está limitado entre [0,1) se aproxima el valor de la secuencia mayor consiguiendo así el valor entero aproximado de **d**. Con esto se consigue obtener una duración estocástica que permita variar el ritmo y la pronunciación de la voz y con paralelización del computo.

B.5. Hiperparámetros del Generador

El generador usado en VITS se trata de la versión 1 de HiFiGAN con los Hiperparámetros definidos en la tabla B.1.

Así que en la Figura B.7, el generador consta inicialmente de un proceso de expansión del generador se inicia con el bloque de convolución transpuesta. En las dos primeras etapas, este bloque utiliza un kernel de dimensiones 16x1, y posteriormente se emplea un kernel de 4x1. Posteriormente el MRF contiene bloque residual con capas de convolución dilatada, donde varia el tamaño del kernel utilizando 3,5 y 7, mientrás que la tasa de dilatación valdrá 1,3 y 5. Esto nos permite tener una recepción más amplia, que implica que nuestro *kernel* abarque un mayor número de muestras de manera más eficiente. Todo estos bloques convolucionales están seguidos por una función de

Modelo	h_u	k_u	k_r	D_r
V1	512	[16, 16, 4, 4]	[3, 7, 11]	[[1, 3, 5], [1, 3, 5], [1, 3, 5]]

Tabla B.1: Hiperparámetros Generador V1

activación *Leaky ReLU*.

Finalmente como se aprecia en la Figura B.7, se utilizará una función convolucional de kernel 7 y una función de activación tangente hiperbólica, que nos dará como dimensión de salida un solo canal que será el audio en dominio temporal generado a través del espacio latente que recibe de entrada.

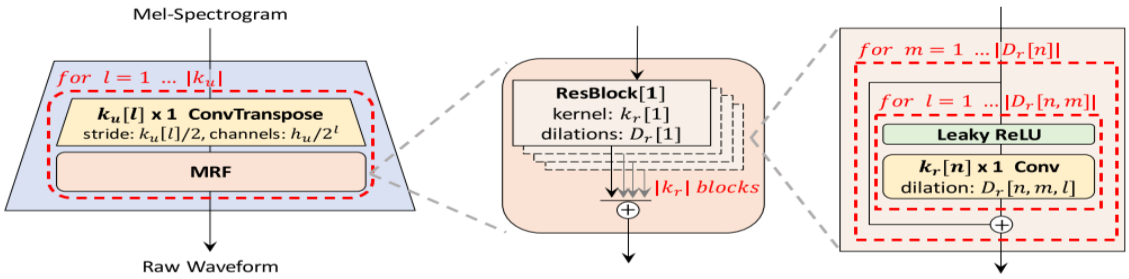


Figura B.6: Esquema del bloque MRF

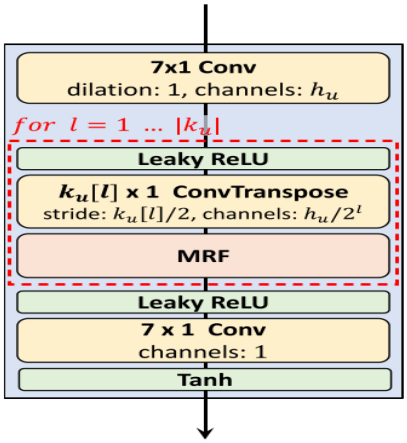


Figura B.7: Generador HiFiGAN

B.6. Pérdidas VITS

B.6.1. Pérdidas de Reconstrucción

Esta pérdida mide la diferencia entre el audio objetivo y el audio de referencia. Para calcularla, se utiliza los MFCC ya que estas representaciones son más acorde a la percepción auditiva humana. Considerando que los MFCC de referencia se denominan como x_{mel} y los reconstruidos a la salida del decoder $\hat{x}_{mel}$, las pérdidas de reconstrucción se calculan como:

$$L_{recon} = ||x_{mel} - \hat{x}_{mel}||_1 \tag{B.1}$$

Esta pérdida mide la distancia entre dos puntos en un espacio euclidiano usando como norma el valor absoluto, también llamada. Esta pérdida es usada en el entrenamiento para optimizar el comportamiento del sistema como se explica en el Anexo E.

B.6.2. Pérdidas por Divergencia KL

La divergencia de Kullback-Leibler (divergencia KL) es una medida positiva y no simétrica de similitud entre dos funciones de distribución de probabilidad, P y Q. Esta divergencia se utiliza comúnmente para evaluar el grado de similitud entre una función de distribución de datos reales u observaciones, P(x), y una distribución teórica o modelo, Q(x). En el caso de que las distribuciones sean discretas, como por ejemplo nuestro caso que tenemos vectores discretizados de una función continua como es la voz, la divergencia KL se calcula como:

$$\begin{aligned} D_{KL}(P||Q) &= \sum_x P(x) log(\frac{P(x)}{Q(x)}) \\ KL(P,Q) &= E_P[log(\frac{P(x)}{Q(x)})] \end{aligned} \tag{B.2}$$

En VITS la divergencia KL se interpreta como la diferencia de una distribución a posteriori frente a una distribución a priori. La distribución a posteriori es definida como la salida del *Posterior Encoder* o *Spectrogram Encoder* (Sección A.4) que se denomina $q_\phi(z|x_{lin})$. Respecto a la distribución a priori se trata de la representación del espacio latente z condicionado por el fonema de entrada y la predicción de alineamiento $p_\theta(z|c_{text}, A)$. Ambas distribuciones son modeladas como proyecciones de distribuciones Gaussianas condicionadas por su entrada, tal y como se aprecia en la fórmula B.3.

$$\begin{aligned} L_{KL} &= log(q_\phi(z|x_{lin})) - log(p_\theta(z|c_{text}, A)) \\ z \sim q_\phi(z|x_{lin}) &= N(z; \mu_\phi(x_{lin}), \sigma_\phi(x_{lin})) \\ p_\theta(z|c_{text}, A) &= N(f_\theta(z); \mu_\theta(c_{text}, A); \sigma_\theta(c_{text}, A)), \end{aligned} \tag{B.3}$$

$$\begin{aligned} L_{KL} &= log(N(z; \mu_\phi(x_{lin}), \sigma_\phi(x_{lin}))) - log(N(f_\theta(z); \mu_\theta(c_{text}, A); \sigma_\theta(c_{text}, A))) \\ \phi^*, \theta^* &= \arg\min_{\phi,\theta} L_{KL} \end{aligned} \tag{B.4}$$

De la ecuación B.4, se busca los parámetros θ y ϕ que consigan minimizar esa pérdida, lo que implica que se propaga el error para optimizar los parámetros del Posterior encoder y del bloque de normalización de flujos para conseguir adaptar la red.

B.6.3. Pérdidas Predicción de la Duración

Esta pérdida viene introducida por el bloque de Predicción de Duración Estocástica (Sección A.7). Tal y como está explicado en la sección A.7, este modelo se basa en un modelo de flujos, por lo tiene una pérdida que permita optimizar su comportamiento. Está perdida se trata del *ELBO* que estima la probabilidad logarítmica de los datos observados, penalizados por la diferencia entre la distribución dada por el modelo y la referencia. Esta pérdida se descompone en dos términos: el término de verosimilitud logarítmico que mide el ajuste a los datos y la divergencia KL entre la distribución posterior y a priori. Así que teniendo la distribución posterior $q_\phi(u, v, d|c_{text})$, la distribución a priori $p_\theta(d-u, v|c_{text})$ y la función de verosimilitud $p_\theta(d|c_{text})$:

$$log(p_\theta(d|c_{text})) \geq \mathbb{E}_{q_\phi(u,v|d,c_{text})}[log(\frac{p_\theta(d-u,v|c_{text})}{q_\phi(u,v,d|c_{text})}] \tag{B.5}$$

Al realizar las simplificaciones matemáticas correspondientes (Vea Anexo E.3.1) , la función de pérdidas para optimizar el predictor estocástico se define como:

$$L(\phi^*, \theta^*) = -ELBO = -\mathbb{E}_{q_\phi}[log(p_\theta(d|c_{text})] + D_{KL}(q_\phi(u, v|d, c_{text}), p_\theta(d-u, v|c_{text})) \tag{B.6}$$

B.6.4. Pérdidas del Entrenamiento Adversario

El entrenamiento del modelo implementa un nuevo componente crucial denominado entrenamiento adversario (Sección E.4.1). Este nuevo enfoque introduce unas nuevas pérdidas con un papel fundamental para guiar a la convergencia de la GAN. En el entrenamiento de HiFiGAN[32], se reemplaza la pérdida por entropía cruzada binaria de [71] por la función de error cuadrático con el fin de evitar desvanecimientos en los gradientes. Esta perdida está definida basado en la idea que la salida del discriminador será 1 para muestras originales y 0 para sintetizadas. Así que considerando $\mathbf{x}$ como los datos reales y $\mathbf{z}$ como la entrada al generador, se definen una pérdida para optimizar cada sistema:

$$\begin{aligned} L_{Adv}(D;G) &= \mathbb{E}_{(x,z)}[(D(x)-1)^2 + (D(G(z)))^2] \\ L_{Adv}(G;D) &= \mathbb{E}_z[(D(G(z))-1)^2] \end{aligned} \tag{B.7}$$

Estas funciones de pérdidas están inversamente relacionadas, lo que se verá reflejado en que a medida que un sistema perfecciona su función de pérdida, en un sentido, la mejora puede manifestarse como un empeoramiento implícito de la otra función de pérdida.

Otra pérdida muy usada es la pérdida de emparejamiento de características o *Feature Matching Loss*. Esta pérdida mide la distancia L1 entre las características del audio de referencia y las características intermedias generadas:

$$L_{FM}(G;D) = \mathbb{E}_{x,z}[\sum_{i=1}^{T} \frac{1}{N_i} ||D^i(x) - D^i(G(z))||_1] \tag{B.8}$$

Donde T define el número de capas que hay en el discriminador, D^i y N_i denota la característica y la dimensión de la característica de la capa i. Además de estas pérdidas se utiliza la pérdida de reconstrucción (Sección B.6.1), a la que denomina como pérdida de Mel.

$$L_{Mel}(G) = \mathbb{E}_{x,z}[||\phi(x) - \phi(G(z))||_1]^2 \tag{B.9}$$

La pérdida final de la GAN se interpreta como una ponderación de todas las pérdidas, dandole una mayor importancia a la pérdida de Mel:

$$\begin{aligned} L_G &= \sum_{k=1}^{K} [\lambda_{adv-gen} \cdot L_{Adv}(G;D_k) + \lambda_{FM} \cdot L_{FM}(G;D_k)] + \lambda_{mel} \cdot L_{Mel} \\ L_D &= \sum_{k=1}^{K} [\lambda_{adv-disc} \cdot L_{Adv}(D_k;G)] \\ \lambda_{adv-gen} &= \lambda_{adv-disc} = \lambda_{FM} = 1, \lambda_{Mel} = 45 \end{aligned} \tag{B.10}$$

Donde K denota el número de discriminadores y D_k el subdiscriminador k del modelo.

[2] ϕ representa la función de conversión de voz a MFCC (Sección A.3)

Anexos C

Entorno de Simulación, Coste y Tiempo Computacional

C.1. Entorno de Simulación

Para la simulación de las tareas de este trabajo se requieren tarjetas gráficas con una alta capacidad de memoria RAM, lo que nos impide trabajar en el ordenador local, teniendo que trabajar en el clúster VOZ de la universidad. El clúster VOZ está compuesto por 4 nodos de almacenamiento, 2 nodos de bases de datos y 13 nodos de cómputo. Los nodos de cómputo van enumerados desde voz01 hasta voz13 incluyendo una o varias tarjetas gráficas. Debido a los altos requerimientos computacionales de algunas de las tareas que se han realizado, especialmente en memoria RAM de las tarjetas gráficas, se ejecutarán las simulaciones en los nodos voz11, voz12 y voz13 que son los mejor dotados en recursos computacionales. El nodo voz11 contiene una tarjeta gráfica RTX3090, el nodo voz12 una gráfica A10 y una TITAN V, y el nodo 13 dos gráficas RTX3090. Tanto la RTX3090 como la A10, son tarjetas gráficas con una memoria RAM de 24 GBs en las que se realizarán los entrenamientos de los modelos.

El clúster se utiliza simultaneamente por decenas de usuarios y especialmente estos nodos potentes tienen una alta ocupación. Para evitar conflictos entre varias tareas lanzadas simultaneamente se hace uso del sistema de colas **Condor**[65]. Condor es un sistema de colas de la Universidad de Wisconsin-Madison utilizado en computación distribuida y de alto rendimiento para la administración y ejecución de trabajos en clústeres de computadoras.

Como lenguaje de programación se ha usado Python[66] y para gestionar las dependencias y entornos de trabajo, se ha utilizado el gestor de paquetes Anaconda[67], que gestionará dos entornos de trabajos, el entorno base versión 3.9, que tendrá las dependencias necesarias para la simulación de la librería TTS, y otro entorno versión

3.10 donde están las dependencias necesarias para el paquete openai-whisper[1].

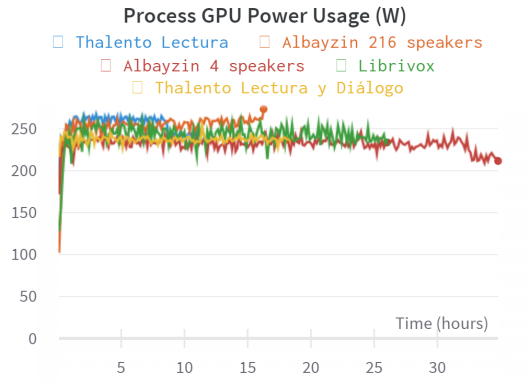
C.2. Tiempo y Coste computacional

Una vez configurado el entorno de simulación, se observará el tiempo y coste de computo de las simulaciones. Para obtener estas gráficas se hará uso de la interfaz ***wandb***[2] donde se podrá observar todas las gráficas e información del consumo, temperatura y memoria de las GPUs y CPUs usadas.

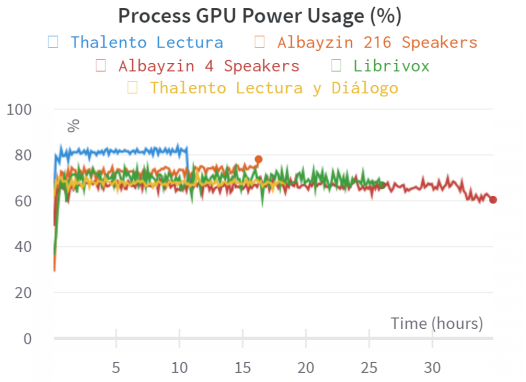
Para el procesamiento se usaron las GPUs con memoria RAM de 24 GBs y una CPU del clúster. La gran mayoría de tareas de procesado fueron desarrolladas en la GPU, por lo que las gráficas que mayor valor tendrán son las de consumo del proceso de la GPU (Figura C.1). En la figura C.1e se ve que el tiempo de acceso a la memoria es menor al 40 % que se considera relativamente bajo, esto es debido a que en el entrenamiento de estos modelos se han usado audios de corta duración, que optimizan el rendimiento del sistema. Respecto a la potencia y memoria usada por el proceso vemos que se ocupa valores medios de cerca del 60 % de la GPU con picos que a veces alcanzan valores cercanos al 85 % de la GPU. Por eso a la hora de entrenar modelos de síntesis de voz se debe tener en cuenta el coste computacional como un importante factor limitante.

Respecto al tiempo de generación de audios en inferencia, no se puede llegar a hacer una estimación exacta debido a que depende de varios factores como puede ser el tamaño del modelo, tráfico en el clúster, longitud del texto de entrada, etc. Aún así, tras la generación de varios audios se observa que el tiempo de inferencia se indica de dos maneras: **tiempo de procesado** y **tiempo real**. El tiempo de procesado se trata del tiempo total que se tarda en generar el audio. Este depende de la longitud de la secuencia de texto de entrada, por lo que a secuencias más largas mayor será este. Por otro lado, el tiempo real se trata del tiempo de procesado entre el tiempo del audio generado. Este último, indica la velocidad de procesado del sistema independientemente de la duración del audio. Este valor suele rondar entre 0.3 y 0.6 veces la duración del audio, apreciando la gran velocidad de procesado del sistema, siendo capaz de generar audio en inferencia en tiempo real con el entorno de simulación adecuado. Esta es una de las grandes de ventajas de VITS frente a anteriores modelos, ya que hace todo el procesamiento en una sola etapa evitando sistemas secuenciales y de esta forma consigue mejorar la velocidad en entrenamiento e inferencia mejor que entrenar de manera secuencial a *GlowTTS* y *HiFiGAN*.

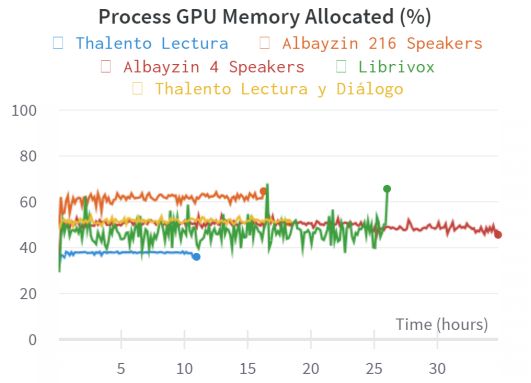
[1] https://github.com/openai/whisper
[2] https://wandb.ai/santi-cabila/TFG_809622?workspace=user-santi-cabila



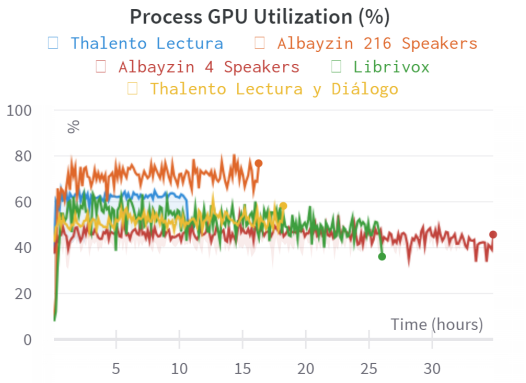
(a) Potencia del proceso en la GPU(W)



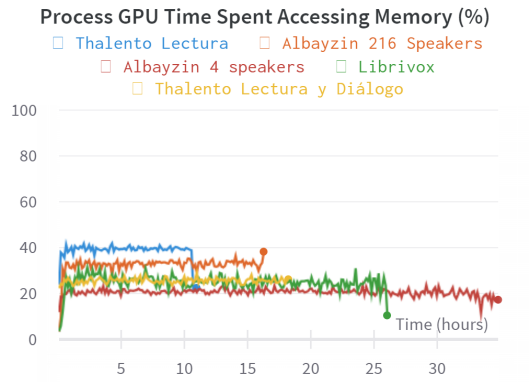
(b) % Potencia de GPU usada proceso



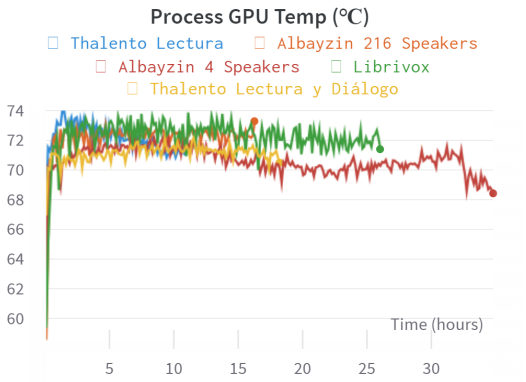
(c) % Memoria de GPU ocupada proceso



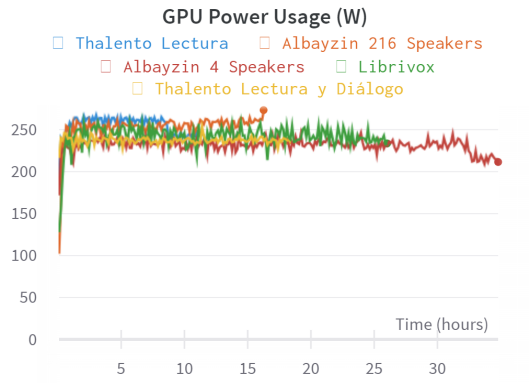
(d) % de la GPU usada por el proceso



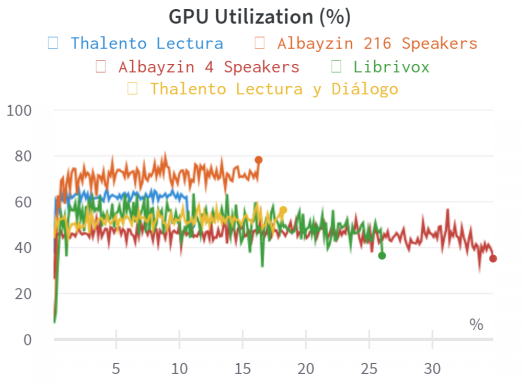
(e) % Tiempo de acceso a la memoria



(f) Temperatura de la GPU



(g) Potencia de la GPU utilizada



(h) % de la GPU utilizada

Figura C.1: Coste computacional de la GPU

Anexos D

Alfabetos fonéticos

D.1. Alfabeto Fonético Internacional

El Alfabeto Fonética Internacional o IPA[30] (*International Phonetic Alphabet*)[1], es un sistema de notación fonética que se utiliza para representar los sonidos de diferentes idiomas del mundo. Debido a la lingüística y en campos relacionados con el procesamiento del habla y el lenguaje.

En el *Phonemizer*[68] usado en el trabajo IPA está implementado indicando que soporta más de 100 idiomas con una alta velocidad de procesado. Por eso fue elegido como Alfabeto fonético durante el entrenamiento del modelo. Además, el alfabeto IPA es ampliamente más detallado que el resto de alfabetos y puede llegar a describir una mayor cantidad de fonemas. La relación de estos sonidos puede ser apreciada en la Figura D.2 y en la Tabla D.1[2].

[1] https://www.ipachart.com/

[2] Tabla obtenida: https://www1.essex.ac.uk/linguistics/external/clmt/latex4ling/tipa/ipa-consonants.tex

Consonantes IPA											
	Bilabial	Lab. dent.	Dental	Alveolar	Post-Alveolar	Retroflex	Palatal	Velar	Uvular	Faring.	Glotal
Plosive	p b			t d		ʈ ɖ	c ɟ	k g	q ɢ		ʔ
Nasal	m	ɱ		n		ɳ	ɲ	ŋ	ɴ		
Trill	ʙ			r					ʀ		
Tap/Flap				ɾ		ɽ					
Fricative	ɸ β	f v	θ ð	s z	ʃ ʒ	ʂ ʐ	ç ʝ	x ɣ	χ ʁ	ħ ʕ	h ɦ
Lat. Fric.				ɬ ɮ							
Approx		ʋ		ɹ		ɻ	j	ɰ			
Lat. appr.				l		ɭ	ʎ	ʟ			

Tabla D.1: Consonantes Alfabeto Fonético Internacional.

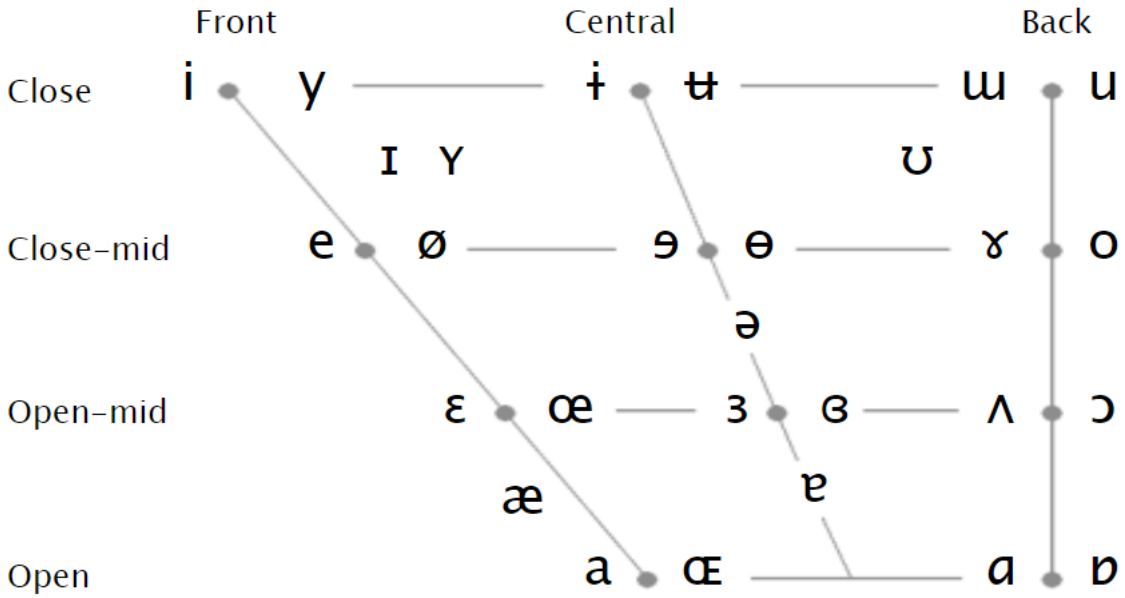


Tabla D.2: Vocales Alfabeto Fonético Internacional

D.2. Alfabeto SAMPA

Aunque el Alfabeto IPA tenga una mayor velocidad de procesado, a la hora de obtener las métricas de calidad las asociación entre letras y fonemas IPA obtenidos no es muy intuitiva. Para solucionar esto, se usará el alfabeto SAMPA.

El alfabeto SAMPA (*Speech Assessment Methods Phonetic Alphabet*) es un sistema de notación fonética basado en letras del alfabeto latino y caracteres ASCII que se utiliza para representar los sonidos del habla en una forma más simple y fácilmente legible que el IPA. El SAMPA se creó para ser compatible con la codificación ASCII, lo que lo hace adecuado para su uso en entornos informáticos y digitales. Aunque es menos detallado que el IPA, el SAMPA sigue siendo capaz de representar los sonidos esenciales del habla de manera efectiva. Una explicación de los fonemas SAMPA con ejemplos puede ser encontrada en la tabla D.3.

SAMPA	Característica	Ejemplo	Transcripción
p	explosiva bilabial sorda	**p**ala	**p**ala
b	explosiva bilabial sonora	**b**ala	**b**ala
t	explosiva dental sorda	**t**ala	**t**ala
d	explosiva dental sonora	**d**ar	**d**ar
k	explosiva velar sorda	**c**ala	**c**ala
g	explosiva velar sonora	**g**ala	**g**ala
m	nasal bilabial sonora	**m**ala	**m**ala
n	nasal alveolar sonora	**n**ada	naDa
N	nasal velar sonora[3]	ho**n**go	o**N**go
J	nasal palatal sonora	ca**ñ**a	ka**J**a
tS	africada palatal sorda	**ch**ico	**tS**iko
B	aproximante bilabial sonora	la**v**a	la**B**a
f	fricativa labiodental sorda	**f**also	**f**also
T	fricativa interdental sorda	**z**ona	**T**ona
D	aproximante dental sonora	ca**d**a	ka**D**a
s	fricativa alveolar sorda	**s**ala	**s**ala
z	fricativa alveolar sonora [4]	de**s**de	de**z**De
jj	fricativa palatal sonora	a**y**er	ajjer
x	fricativa velar sorda	**j**amón	xamon
G	aproximante velar sonora	la**g**o	la**G**o
l	lateral alveolar sonora	**l**a	**l**a
L	lateral palatal sonora	llana	llana
rr	vibrante múltiple alveolar sonora	ca**rr**o	ka**rr**o
r	vibrante simple alveolar sonora	ca**r**o	ka**r**o
i	vocal anterior cerrada	t**i**la	t**i**la
j	semivocal palatal [5]	lab**i**o	laN**j**o
e	vocal anterior media	t**e**la	t**e**la
a	vocal central abierta	t**a**l	t**a**l
o	vocal posterior media redondeada	t**o**do	t**o**d**o**
u	vocal posterior cerrada redondeada	t**u**l	t**u**l
w	semivocal labiodental [6]	ag**u**a	ag**w**a

Tabla D.3: Ejemplos de transcripción SAMPA

Anexos E

Redes Neuronales Artificiales

E.1. Introducción

Las redes neuronales surgen de la idea matemática de emular el comportamiento del procesado de la información del cerebro humano. Estos modelos funcionan mediante la simultaneación de un número elevado de unidades de procesamiento interconectadas entre sí, simulando así el comportamiento de las neuronas.

Todas las redes neuronales aritificales organizan estas unidades de procesamiento en capas. Cada red neuronal consta de una capa de entrada, donde se introducen los datos de entrada; una o varias capas ocultas, que son las responsables de procesar y transformar la información; y finalmente, una capa de salida que proporciona los resultados finales de la red.

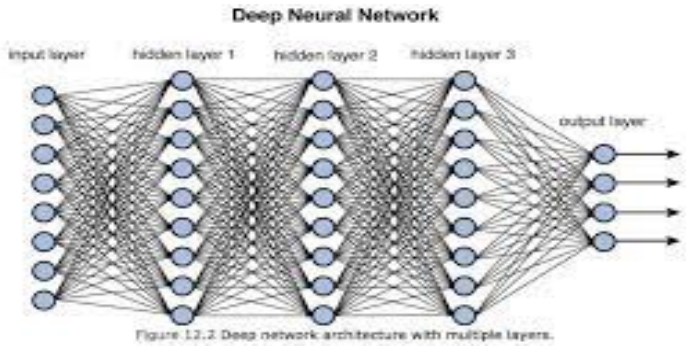


Figura E.1: Esquema de las capas de una red neuronal

E.1.1. Perceptrón

En 1957 Frank Rosenblatt inventó el Perceptrón[70], creando así la primera aplicación práctica de una red neuronal. El perceptrón se trata de la unidad más sencilla de red neuronal que es capaz de efectuar los cálculos necesarios para detectar características o tendencias en los datos de entrada.

El perceptrón se puede explicar como una función matemática donde la salida de la red es la salida del resultado de una función de activación no lineal del producto escalar entre los datos de entrada y los pesos de la red

$$y = \sum_{i=0}^{N} w_i \cdot x_i. \tag{E.1}$$

.

Algorithm 2 Algoritmo ***Perceptrón***

```
P ←inputs with label 1
N ←inputs with label 0
Ensure: Initialise w randomly;
while !converge do
    Pick random x ∈ P ∪ N;
    if x ∈ P and w·x < 0 then
        w = w+x;
    end if
    if x ∈ N and w·x ≥ 0 then
        w = w-x;
    end if
end while
```

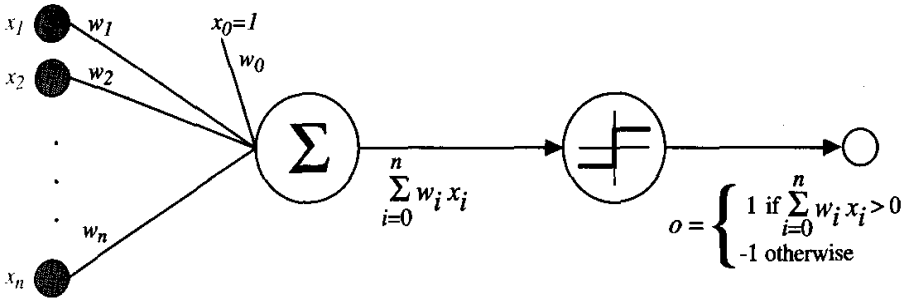


Figura E.2: Esquema del perceptron

E.1.2. Funciones de activación

Una de las bases del éxito de las redes neuronales radica en su capacidad de modelar funciones no lineales. Esto se debe a que, en la salida de una capa de la red, donde se realiza una combinación lineal de los datos de entrada, los pesos de la capa y el sesgo, la salida es transformada por una función de activación no lineal.

Estas funciones de activación son muy importantes porque nos permiten mantener el valor de la salida de la neurona en un cierto límite dependiendo el requisito que tenga nuestra red. La elección de estas funciones es crucial debido a que permiten moldear

la salida de la neurona, controlando como se transmiten las señales a lo largo de la red y permitiendo la captura de patrones y características más complejas en los datos. La elección de la función de activación depende de la naturaleza del problema y los requisitos de la red. Algunas de las funciones de activaciones utilizadas en este modelo son: ReLU, Leaky ReLU, sigmoide, tangente hiperbólica, GeLU, etc.

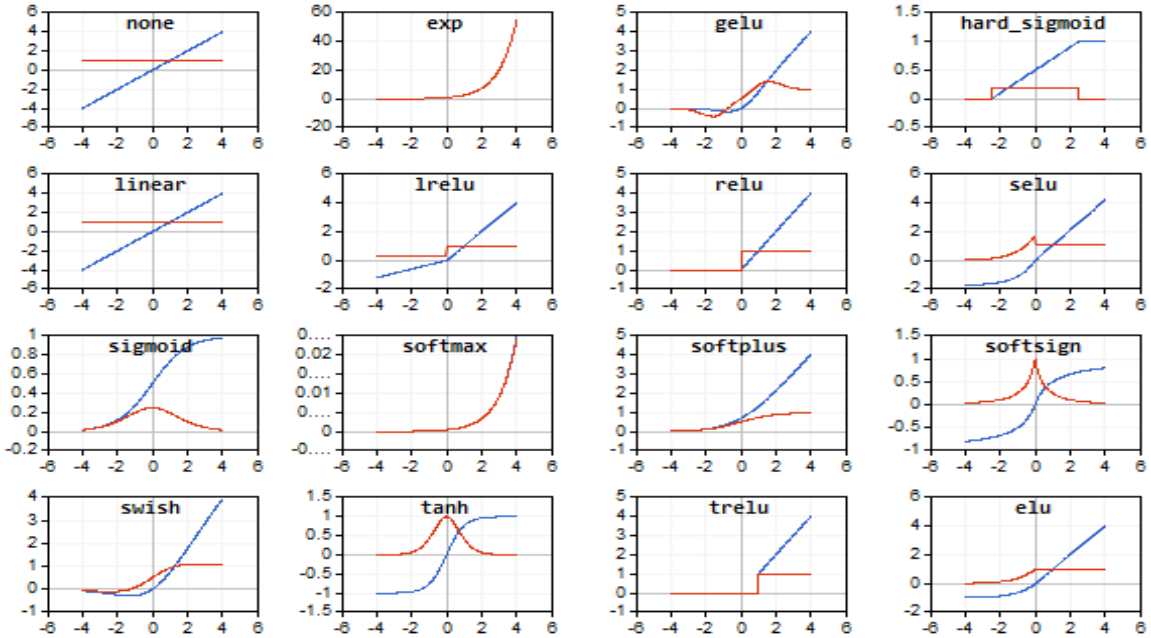


Figura E.3: Funciones de activación (azul: función de activación, rojo: derivada)

E.1.3. RNN

Las redes neuronales recurrentes (RNN) son un tipo de arquitectura diseñada para trabajar con datos secuenciales o temporales. La estructura de estas redes están diseñadas para capturar dependencias temporales y contextuales en los datos.

Como se ve en la Figura E.4, la estructura de las RNN se basan en celdas interconectadas entre si de manera que la salida de una celda estará condicionada por el valor de las anteriores celdas. Esto hace que las redes consigan una propia "memoria" interna o estado que se actualiza con nuevo elemento. El problemas de estas redes es que son bastantes eficaces en memorias a corto plazo, pero conforme tenemos secuencias de entradas más largas empiezan a sufrir problemas de desvanecimiento de los gradientes, haciendo que estén limitado a secuencias de entrada de corta duración.

E.1.4. LSTM

Para mejorar el problema de los desvanecimientos de los gradientes, y conseguir mejorar memoria a largo plazo se diseñaron las redes LSTM, variante de las RNN.

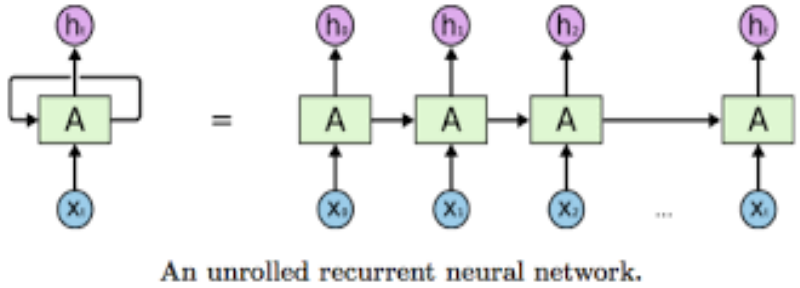


Figura E.4: Esquema de una red neuronal recurrente

Estas redes LSTM introdujeron una estructura de celda que permite a la red mantener información relevante a lo largo de secuencias extensas. Para conseguir esto las redes LSTM definieron tres puertas: 1) la puerta de entrada qie determina la cantidad de información agragada a la celda en función de la entrada y el contexto. 2) la puerta de olvido que controla la cantidad de información a olvidar en lacelda. 3) la puerta de salida que controla la información que se mostrará e la salida dependiendo de la entrada y el contexto. Al introducir esa puerta de olvido se evita los propagación de errores y el desvanecimiento del gradiente en largas dependencias (Figura E.5)

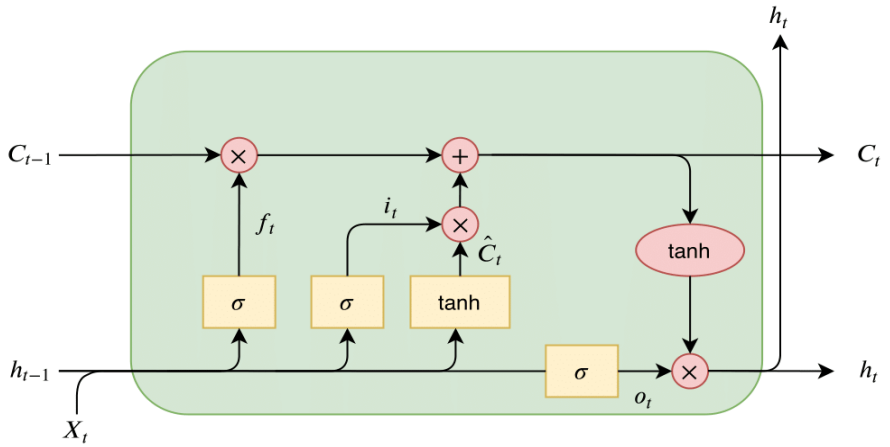


Figura E.5: Esquema de una red LSTM

E.1.5. CNN

Las redes neuronales artificiales convolucionales son un tipo de red neuronal especializada en el procesamiento de datos espaciales. Estas están diseñadas para identificar patrones y características en los datos de entrada, a través de capas de convolución y capas de agrupación (*pooling* o submuestreo) y capas totalmente conectadas. Las capas de convolución deslizan filtros o "*kernels*.ª través de la entrada para realizar operaciones de convolución con el fin de detectar características

específicas. La capa de agrupación reduce el tamaño espacial de la representación y disminuye la cantidad de parámetros en la red. Por último, las capas totalmente conectadas toman las características extraídas y las utilizan para clasificar (FiguraE.6).

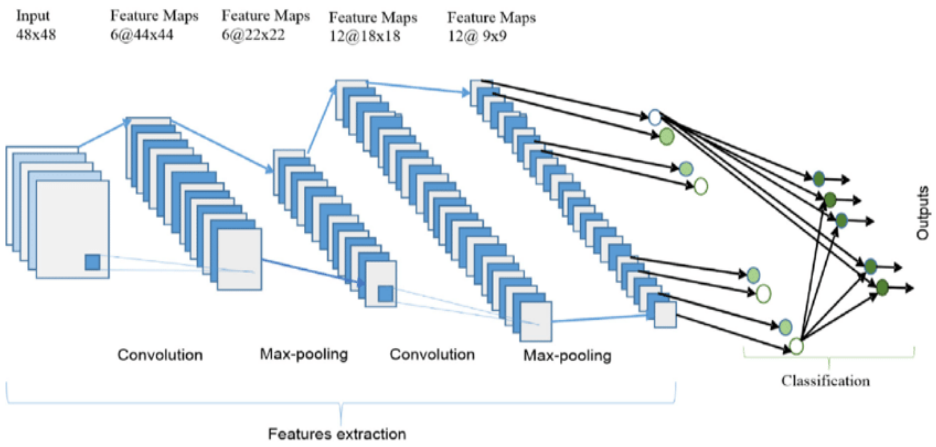


Figura E.6: Esquema de una CNN

E.1.6. ADAM

Adam es un método de optimización utilizado en el entrenamiento que emplea el primer momento (media acumulada) y segundo momento (media acumulada de los gradientes al cuadrado) del gradiente para adaptar la tasa de aprendizaje acelerando la convergencia y evitando problemas asociados con tasas de aprendizaje fijas. Este algoritmo introduce las ventajas de RMSPROP y del Gradiente Estocástico con Momentum. De este primero, introduce un tamaño de paso variable que nos permite acelerar o desacelerar el proceso de entrenamiento dependiento la variación del gradiente, y del segundo utiliza una media móvil para evitar oscilaciones en el entrenamiento.

Para definir este algoritmo hay que definir unos hiperparámetros antes de empezar a entrenar:

- **Tasa de aprendizaje** o *Learning Rate* α: Controla el tamaño del paso que el algoritmo toma en la dirección opuesta al gradiente para actualizar los pesos. Un valor alto puede llevar a oscilaciones y divergencia, mientras que uno bajo ralentiza la convergencia. En el entrenamiento este valor se define a un valor $\alpha = 0{,}001$.

- **Decaimiento del primer momento** β_1: Controla como se calcula y actualiza el primer momento de los gradientes. Este valor toma valores entre 0 y 1, y en este caso está definido a $\beta_1 = 0{,}8$.

– **Decaimiento del segundo momento** β_2: Controla como se calcula y actualiza el segundo momento de los gradientes. Valor entre 0 y 1 y se define a un valor $\beta_2 = 0{,}99$.

– ϵ: Valor definido para evitar una división entre 0 cuando el segundo momento sea 0.

– **Decaimiento de los pesos** λ: Valor definido para evitar gradientes de gran tamaño.

Así nuestro algoritmo se quedará definido a través del pseudocódigo definido en el algoritmo 3, obtenido de [46] y cambiando los hiperparámetros a nuestro entrenamiento.

Algorithm 3 Algoritmo Adam con caída de pesos (AdamW)[46]

Given $\alpha = 0{,}001$, $\beta_1 = 0{,}8$, $\beta_2 = 0{,}99$, $\epsilon = 2 \cdot 10^{-4}$, $\lambda \in \mathbb{R}$
Initialize time step $t \leftarrow 0$, parameter vector $\theta_{t=0} \in \mathbb{R}^n$, first moment vector $m_{t=0} \leftarrow 0$, second moment vector $v_{t=0} \leftarrow 0$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
    $t \leftarrow t + 1$
    $\nabla f_t(\theta_{t-1}) \leftarrow \text{SelectBatch}(\theta_{t-1})$ ▷ Select batch and return gradient
    $g_t \leftarrow \nabla f_t(\theta_{t-1}) + \lambda\theta_{t-1}$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ ▷ β_1 raised to the power of t
    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ ▷ β_2 raised to the power of t
    $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ Can be fixed, decay, or warm restarts
    $\theta_t \leftarrow \theta_{t-1} - \eta_t \left(\frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} + \lambda\theta_{t-1} \right)$
until stopping criterion is met
return optimized parameters θ_t

E.2. Transformer y Self Attention

En el año 2017, Google presnetó la arquitectura del Transformer en su famoso artículo *Attention is all you need*[69], cuya mejora estaba basada en su nuevo mecanismo de atención. En este artículo se proponía una nueva solución al problema de traducción de texto. Unas de las mejoras de los transformers introdujo se trata de reducir las dimensiones de nuestra redes para dependencias de alto rango. Además, permite una mayor paralelización del procesado adaptándose a los nuevos requerimientos de los procesadores modernos.

Los ***transformers*** son modelos basados en grandes bloques de codificación/decodificación que procesan dependencias entre entrada y salida sin necesidad de técnicas recurrentes. Esto permite eliminar la secuencialidad de procesamiento, lo que a su vez conlleva una significativa mejora en la eficiencia y paralelismo en la computación.

La idea básica de la arquitectura de los transformers son dos bloques principales: el *encoder* y el *decoder*. El *encoder* genera un vector *embedding* $Z = (z_1, ..., z_n)$ a partir de una secuencia de entrada $(x_1, .., x_n)$. Este vector pasa al *decoder* con el que consigue la secuencia de salida $(y_1, ..., y_n)$. En el sistema del trabajo, el *encoder* estará formado por un *transformer*, pero el *decoder* no. Este trata de una GAN (AnexoE.4)) que consigue generar audio a partir del espacio latente de entrada. Así que en este apartado se explicará la estructura del *encoder* y los mecanismos de *self attetion* utilizados.

E.2.1. *Encoder Transformer* y *Self Attention*

Para entender el comportamiento del *encoder* se utilizará tres subsecciones donde se explicará cada parte del esquema representado en la Figura E.7a. Estas subsecciones serán: Conversión a *Embeddings*, Mecanismo de *Self-Attention* y Capa FFN.

Conversión a *Embeddings*

En primer lugar, el *encoder* convierte la secuencia de entrada (en este trabajo una secuencia de fonemas) en una secuencia de vectores a través de un diccionario. Esta tarea la realiza en primer lugar mapeando la secuencia fonemas de entrada y convirtiendola a una serie de IDs o *tokens*. Estos IDs pasan por una capa que los convierte en un vector o *embbeding*, siendo esta una representación más rica del fonema. Como el *transformer* trabaja en paralelo, el sistema debe introducir todos los datos de forma paralela, perdiendo toda información posicional. Para solucionar esto debe existir otro bloque de codificación posicional, que aporta información sobre la posición de manera independiente a la secuencia de entrada. Las posiciones son cacluladas a través de forma sinusoidales con la siguiente expresión:

$$
\begin{aligned}
&PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}).\\
&PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}).\\
&\quad \text{donde:}\\
&\quad\quad \bullet \text{ pos: Se trata de la posición en la secuencia.}\\
&\quad\quad \bullet\ d_{model}\text{: Se trata de la longitud del vector del } \textit{embedding}.\\
&\quad\quad \bullet \text{ i : Es el índice del valor.}
\end{aligned}
\tag{E.2}
$$

Mecanismo de *Self-Attention*

Con la representación obtenida en la anterior etapa, se entra al *transformer* y se aplica el Mecanismo de *Self-Attention*. Para esto lo primero que se hace es convertir las entradas en tres representaciones lineales de esta: *"query"*, *"key"* y *"values"*. Con esto tres vectores se obtiene las matrices *'Q'*,*'K'* y *'V'* que serán las entradas al mecanismo:

- ***Matriz "Q"***: Esta matriz Q se utiliza para medir la similitud entre la palabra fonemas y todas los demás fonemas en función de sus características.
- ***Matriz "K"***: La matriz K contiene todas las *"keys"* se utiliza para determinar qué fonemas tienen una relación significativa con la consulta actual.
- ***Matriz "V"***: La matriz V contiene los *values* información rica y detallada sobre cada fonema. Se utiliza para construir la representación resultante.

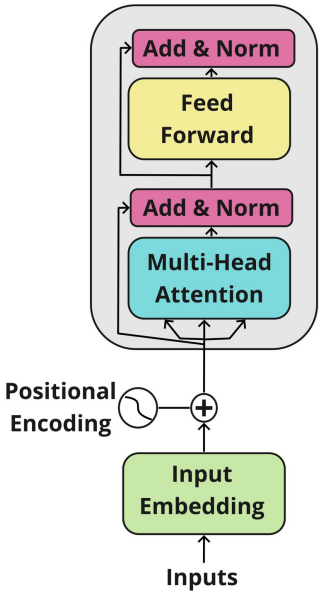
Una vez calculados las matrices de entrada al mecanismo (Q,K,V), se inicia la creación de una matriz de atención ponderada para evaluar la relación entre los diferentes fonemas. La atención se calcula usando la matriz 'Q' y la matriz 'K'. El producto de esta comparación da forma a la matriz de atención, obtenida a través del producto de ambas matrices. Tras esto se le aplicaría una función *softmax*, sin embargo, para evitar que los valores obtenidos sean muy grandes y afecten negativamente al cálculo de esta función, se divide primero el resultado por la raíz cuadrada de la dimensión de Q y K. Tras esto, ya se aplica la función *softmax*. Finalmente, la representación final se obtiene ponderando las matriz V con esta representación normalizado y se suman para la representación final. A esta operación se le denomina *Scaled Dot Product* definida por la ecuación: (Figura E.7b)

$$attention(Q, K, v) = softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V \tag{E.3}$$

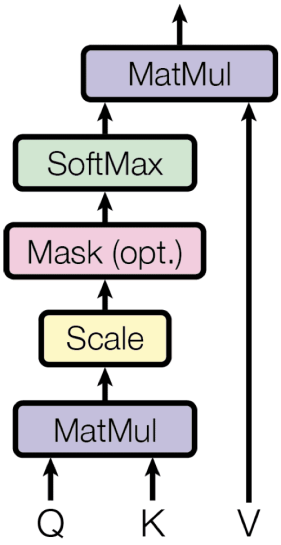
Capa FFN

A la salida del mecanismo de atención, se tiene otra capa denominada **FFN** (*Feed Forward Network*) con una conexión residual (Figura E.7a).

La capa FFN es una red neuronal *fully-conected* que se utiliza para transformar los tokens en las secuencias de entrada. Esta consiste en dos capas lineales, cada una seguida de una función de activación no lineal, típicamente una función $ReLU$. Esta capa es la responsable de aplicar transformaciones no lineales a las representaciones de palabras, ayudando a la red a modelar relaciones más complejas y generar salidas



(a) Arquitectura del *Encoder transformer*



(b) Esquema *Scaled Dot Product*

Figura E.7: *Transformers*[69]

contextualmente más ricas. A la salida de esta capa se tiene la representación latente de los fonemas que tiene como entrada el *encoder* y a los que se le aplicará en este trabajo la proyección lineal explicada en Anexo A.2.

E.2.2. Mecanismo *Multihead-Attention*

Con el mecanismo *Multihead-Attention* el *transformer* proyecta linealmente las matrices 'Q','K' y 'V', utilizando una proyección propia en cada situación. Luego, el mecanismo de *Self-Attention* se aplica a cada una de estas proyecciones en paralelo para producir salidas, las cuales a su vez se concatenan y proyectan nuevamente para generar un resultado final. Con esta idea el *transformer* es capaz de extraer información de varios subespacios vectoriales, cosa imposible utilizando solo un mecanismo.

En este caso tratando el número de cabezas de manera independiente, se puede considerar que cada cabeza se puede calcular como:

$$head_i = attention(Q \cdot W_i^Q, K \cdot W_i^K, V \cdot W_i^V). \tag{E.4}$$

Y el resultado final se puede interpretar como una proyección lineal de la concatenación de cada cabeza:

$$\begin{aligned} multihead(Q,K,V) &= concat(head_1, ..., head_h) \cdot W^O \\ &\text{donde:} \\ &W^O\text{: Es la matriz de pesos que se aplica para realizar} \\ &\text{la proyección lineal final.} \end{aligned} \tag{E.5}$$

Este mecanismo se puede apreciar en la Figura E.8.

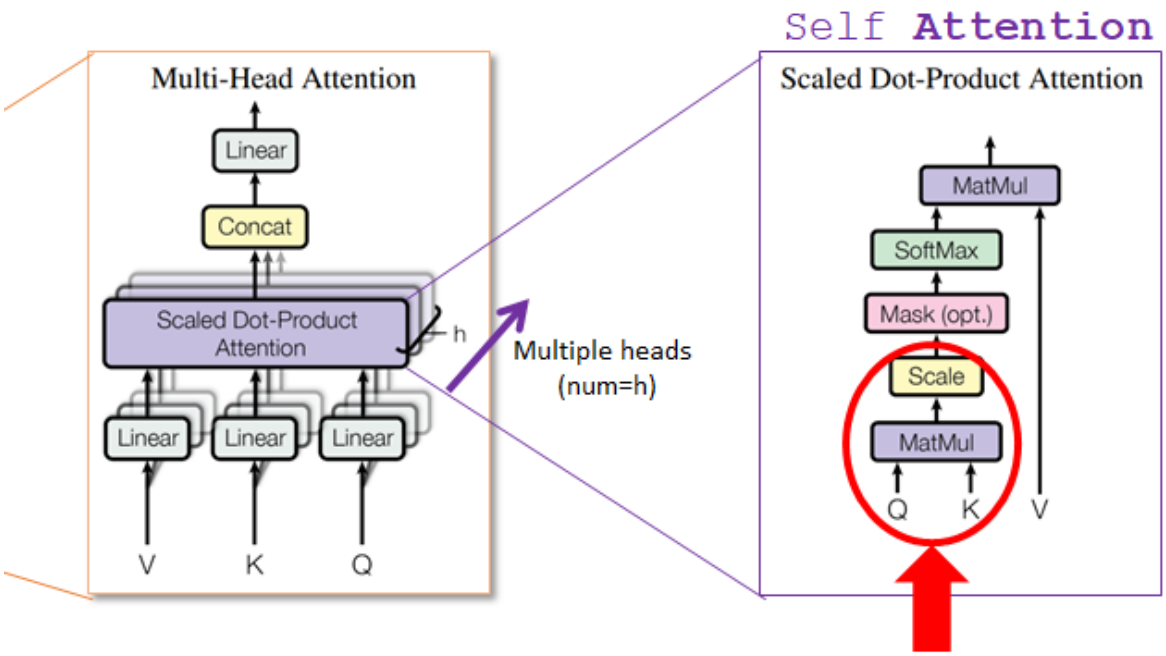


Figura E.8: Mecanismo de *Multihead-Attention*

E.3. Autoencoder

Un autoencoder es un tipo de red neuronal utilziada para la reducción de la dimensionalidad y la extracción de características. El autoencoder mediante un proceso de optimización iterativo intenta aprender una representación comprimida de los datos, denominado **espacio latente**, de manera que pueda reconstruir los datos a partir de ese espacio latente.

En cada iteracción el autoencoder compara la salida generada con la información original, calculando así un error que se propaga por la arquitectura para actualizar los pesos de la red. Asi que lo que se busca es el encoder y el decoder que minimicen el error de la red.

$$Loss = ||x - \hat{x}||^2 = ||x - d(z)||^2 = ||x - d(e(x))||^2 \tag{E.6}$$

A la hora de modelar estos sistemas hay se debe tener en cuenta que una mayor reducción de la dimensión del espacio latente supondrá una menor interpretabilidad y explotación de este, denominanse este problema como **lack of regularity**.

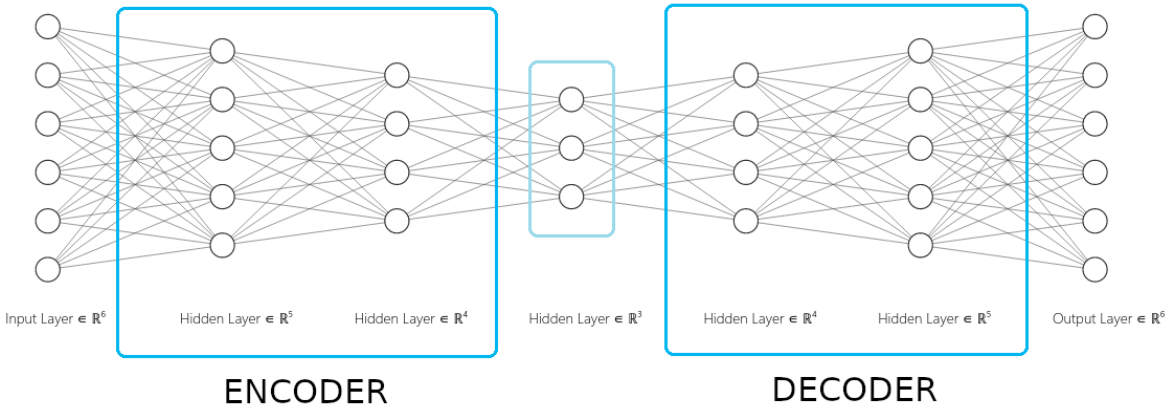


Figura E.9: Arquitectura del Autoencoder

E.3.1. *Variational Autoencoders*

Los Variational Autoencoders (VAEs) son una extensión de los autoencoders que se centran en la generación de datos. Originalmente, los autoencoders solo comprimen datos, pero los VAEs buscan generar nuevos datos al modificar la distribución en el espacio latente. En un autoencoder, la organización del espacio latente no está definida, esta se optimiza a través de la información y la pérdida proporcionada. Los VAEs abordan esto regularizando el espacio latente para evitar sobreajustes. Para conseguir regularizar el espacio latente los VAEs trabajan con modelos probabilísticos en vez de con puntos de datos.

El problema de trabajar con modelos probabilísticos es que dado un conjunto de entrada x = (x^i), estimar la distribución de probabilidad $p(x^i)$ es un problema intratable, por la alta dimensionalidad de los datos de entrada. Para solucionarlo se utiliza una técnica denominada ***"variational inference"*** cuyo objetivo principal es aproximar distribuciones de probabilidad complejas, que a menudo son difíciles de manejar directamente, a distribuciones más simples y fáciles de trabajar.

Así, tambien se define una variable z representadndo al espacio latente con distribución p(z). Igualmente, se denomina $p(z|x^i)$ a la función de densidad del espacio latente generado con la secuencia de entrada x^i, y $p(x^i|z)$ a distribución de probabilidad de los datos reales generados a partir del espacio latente.

También, es muy complejo estimar las $p(z|x^i)$ y $p(x^i|z)$, por lo que estas se aproximan a distribuciones más simples ya conocidas y se estiman los parámetros que mejoren el ajuste. Para ajustar la función a posteriori $p(z|x^i)$ denominamos la función $q_\phi(z|x^i)$ y para la función $p(x^i|z)$ se aproxima como $p_\theta(x^i|z)$, donde ϕ y θ son los parámetros a optimizar para mejorar la aproximación (Figura E.10).

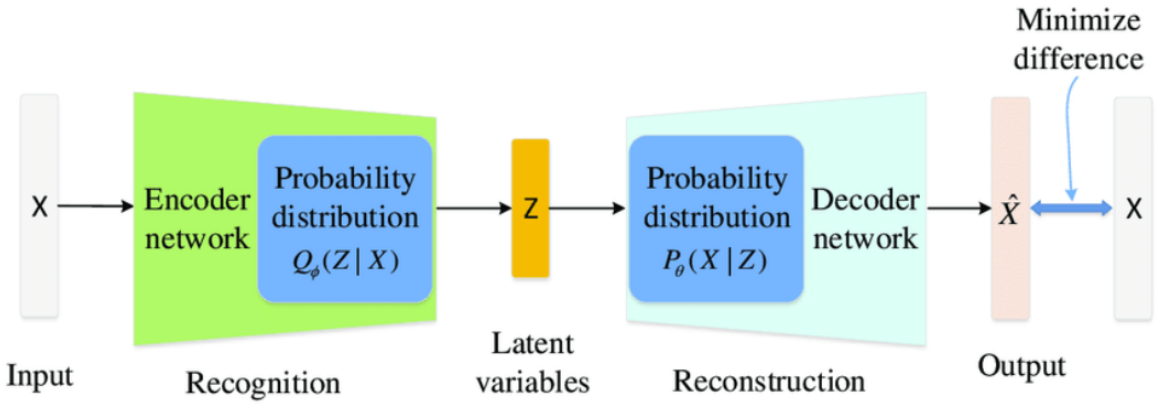


Figura E.10: Arquitectura de Variational autoencoder

Función de Pérdidas

Para llegar a optimizar los parámetros ϕ y θ es necsario definir una función de pérdidas, que vendrá determinada por la idea de disminuir la distancia entre la distribución original y la aproximada. Esta distancia será medida a través de la distancia KL (Sección B.6.2) y se intentará disminuir:

$$\begin{aligned} D_{KL}(q_\phi(z|x^i)||p_(z|x^i)) &= \mathbb{E}_{q_\phi}[log\frac{q_\phi(z|x^i)}{p(z|x^i)}] \\ &= \mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] - \mathbb{E}_{q_\phi}[log(p(z|x^i))] \end{aligned} \tag{E.7}$$

En el ámbito de la teoría de probabilidad, la probabilidad conjunta de dos variables aleatorias se define como el producto de la probabilidad de que ocurra el evento asociado con la primera variable ($p(x^i)$) y la probabilidad de que ocurra el evento relacionado con la segunda variable, condicionado a la ocurrencia del evento asociado con la primera ($p(z|x^i)$). Matemáticamente, esto se expresa como: $p(z, x^i) = p(x^i) \cdot p(z|x^i)$ y se obtiene $p(z|x^i) = \frac{p(z,x^i)}{p(x^i)}$:

$$\begin{aligned} D_{KL}(q_\phi(z|x^i)||p_(z|x^i)) &= \mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] - \mathbb{E}_{q_\phi}[log(\frac{p(z,x^i)}{p(x^i)})] \\ &= \mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] - \mathbb{E}_{q_\phi}[log(p(z,x^i)] + \mathbb{E}_{q_\phi}[log(p(x^i)] \\ &= \mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] - \mathbb{E}_{q_\phi}[log(p(z,x^i)] + \int q_\phi(z|x^i) \cdot log(p(x^i))dz \\ &= \mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] - \mathbb{E}_{q_\phi}[log(p(z,x^i)] + log(p(x^i)) \cdot \int q_\phi(z|x^i)dz \\ &= \mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] - \mathbb{E}_{q_\phi}[log(p(z,x^i))] + log(p(x^i)) \end{aligned} \tag{E.8}$$

donde $\mathbb{E}_x[f(x)] = \int x \cdot f(x)dx \implies \mathbb{E}_{q_\phi}[log(p(x^i)] = \int q_\phi(z|x^i) \cdot log(p(x^i))dz$. La integral de cualquier función de densidad es equivalente a $1 \int_{-\infty}^{\infty} f(x)\,dx = 1$.

Como resultado final de la ecuación , se observa como la función de verosimilitud marginal logarítmica ($log(p(x^i))$), al ser marginal implica que se tiene todas variables observadas, de las cuáles dependen todas las variables y parámetros ocultos, es dependiente de la divergencia KL y la función de verosimilitud logarítmica.

$$log(p(x^i)) = D_{KL}(q_\phi(z|x^i)||p_(z|x^i)) + \mathbb{E}_{q_\phi}[log(\frac{p(z,x^i)}{q_\phi(z|x^i)}] \tag{E.9}$$

Aunque ni la función de verosimilitud marginal logarítmica, ni la distancia KL se puedan calcular de forma analítica, sabemos que la divergencia KL siempre será mayor que cero, por lo que la fórmula E.9, se puede reformular y escribir como:

$$log(p(x^i)) \geq \mathbb{E}_{q_\phi}[log(\frac{p(z,x^i)}{q_\phi(z|x^i)}] \tag{E.10}$$

A esta función se le denomina función del límite inferior de similitud o **"ELBO"**(Ecuación E.11). Aunque no se pueda calcular de forma analítica la función, se sabe que la distancia KL está inversamente relacionado con la función de verosimilitud logarítmica, así que si un término aumenta el otro disminuye. Con esta idea se piensa en la optimización de la VAE haciendo que la función ELBO encuentre el máximo lo que minimizará la divergencia KL de manera implícita.

$$\begin{aligned} ELBO &= \mathbb{E}_{q_\phi}[log(\frac{p(z,x^i)}{q_\phi(z|x^i)}] \\ &= \mathbb{E}_{q_\phi}[log(p(z,x^i)] - \mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] \\ &= -\mathbb{E}_{q_\phi}[log(q_\phi(z|x^i))] + \mathbb{E}_{q_\phi}[log(p_\theta(x^i|z)] + \mathbb{E}_{q_\phi}[log(p(z))] \\ &= \mathbb{E}_{q_\phi}[log(p_\theta(x^i|z)] - \mathbb{E}_{q_\phi}[log(\frac{q_\phi(z|x^i)}{p(z)})] \end{aligned} \tag{E.11}$$

De la ecuación E.11 se observa como el primer término es dependiente de la salida reconstruida a partir del espacio latente y el segundo término es la distancia KL entre la distribución posterior y la distribución a priori, de manera que al disminuir está distancia, aumenta la función ELBO y converge nuestro modelo. Por lo que la función de optimización del modelo será:

$$\phi^*, \theta^* = \arg\max_{(\phi,\theta)} \mathbb{E}_{q_\phi}[log(p_\theta(x^i|z)] - D_{KL}(q_\phi(z|x^i), p(z)) \tag{E.12}$$

. Y las pérdidas

$$L(\phi^*, \theta^*) = -ELBO = -\mathbb{E}_{q_\phi}[log(p_\theta(x^i|z)] + D_{KL}(q_\phi(z|x^i), p(z)) \tag{E.13}$$

E.3.2. Variational Autoencoders Gaussianos

Habituamente, las estas distribuciones son aproximadas a funciones Gaussianas[60], definiéndose como:

1. $p(z) \sim N(0, \mathbb{I})$ es la distribución a priori.

2. $q_\phi(z|x^i) \sim N(z; \mu_\phi(x^i), \sigma_\phi(x^i))$ es la distribución a posteriori.

3. $p_\theta(x^i|z) \sim N(x^i; \mu_\theta(z), \sigma_\theta(z))$ es la versoimilitud.

Así la función de pérdidas ELBO se puede definir como:

$$L(\phi^*, \theta^*) = -ELBO = -\mathbb{E}_{q_\phi}[log(N(x^i; \mu_\theta(z), \sigma_\theta(z))] + D_{KL}(N(z; \mu_\phi(x^i), \sigma_\phi(x^i)), N(0, \mathbb{I})) \tag{E.14}$$

Así que considerando que el espacio latente tiene una dimensión J, la ELBO se puede simplicar sustituendo por las funciones Gaussianas aproximadas:

$$\begin{aligned} -log(p_\theta(x^i|z)) &= -log(\frac{1}{\sigma_\theta(z)\sqrt{2\pi}} \cdot e^{-\frac{(x_j^i - \mu_\theta(z))^2}{2\sigma_\theta(z))^2}}) = \sum_{j=1}^{J} \frac{(x^i - (\mu_\theta(z))_j)^2}{2(\sigma_\theta(z))_j^2} + log(\sqrt{2\pi} \cdot (\sigma_\theta(z)))_j \\ &= \frac{1}{2} \sum_{j=1}^{J} (\frac{(x_j^i - (\mu_\theta(z))_j)}{(\sigma_\theta(z))_j})^2 + log(\sqrt{2\pi} \cdot (\sigma_\theta(z))_j) \end{aligned} \tag{E.15}$$

Además, de [60] se demuestra como se puede simplificar la distancia KL de forma que quede en factor de la media y la varianza de la función a posteriori.

$$\begin{aligned} D_{KL}(q_\phi(z|x^i), p(z)) &= \int q_\phi(z|x^i) \cdot (log(p(z)) - log(q_\phi(z|x^i)))dz = ... \\ &= \frac{1}{2} \cdot \sum_{j=1}^{J} (1 + log((\sigma_\phi(x^i))_j^2) - (\sigma_\phi(x^i))_j^2 - (\mu_\phi(x^i)))_j^2) \end{aligned} \tag{E.16}$$

Al realizar las aproximaciones de las ecuaciones E.15 y E.16 se puede obtener una manera eficiente de computo de la función de pérdidas que nos permita optimizar los párametros ϕ y θ.

Reparametrización de Trick

Inicialmente, el espacio latente z con una entrada x^i es una distribución aleatoria $z \sim q_\phi(z|x^i) = N(\mu_z, \sigma_z)$. Como las variables latentes tienen una distribución determinista se dificulta la propagación y el cálculo de gradientes. Esto puede llevar a una convergencia lenta, inestable o incluso a la imposibilidad de entrenar el modelo de manera efectiva. Para solucionarlo se aplica una técnica denominada Reparametrization Trick para calcular los gradientes de manera eficiente en el proceso de entrenamiento de VAEs. La clave de esta técnica es separar la parte estocástica de la parte determinística en el proceso de generación de las variables latentes, permitiendo que las operaciones de retropropagación se apliquen de manera efectiva permite que las operaciones de

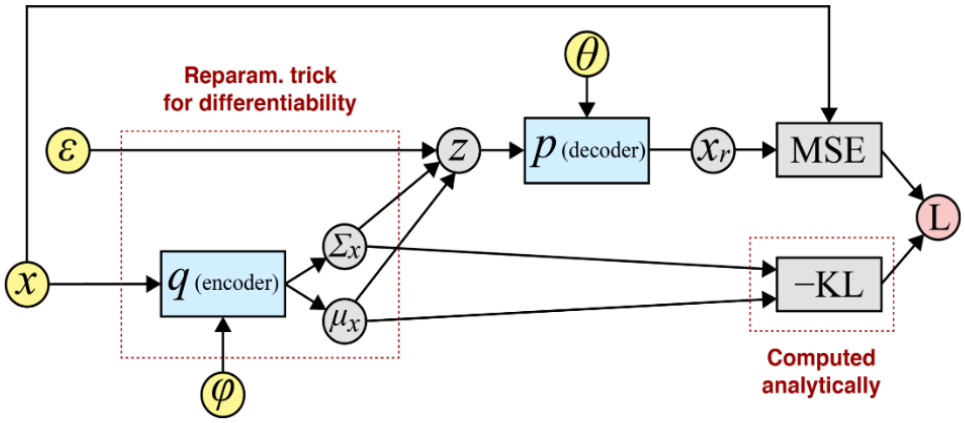


Figura E.11: Reparametrización Trick

retropropagación. Para conseguir esto se considera la distribución del espacio latente como : $z = \mu_z + \sigma_z \odot \epsilon$, siendo $\epsilon \sim N(0, 1)$. Teniendo ya un término independiente que no depende de los parámetros de la red (Figura E.11).

E.3.3. Conditional Variational Autoencoders

Una variación de los VAes son los COnditional Variational Autoencoders(CVAEs) que permiten la generación y manipulación controlada de datos condicionados a información adicional, como etiquetas, atributos u otras variables de control. La variación que se le aplica a la estructura del VAE se basa en introducir una variable condicional c, que afecta a la distribución generada por el encoder y el autoencoder, quedando las nuevas distribuciones como:

1. $p(z) \sim N(0, \mathbb{I})$ es la distribución a priori.
2. $q_\phi(z|x^i, c) \sim N(z; \mu_\phi(x^i, c), \sigma_\phi(x^i, c))$ es la distribución a posteriori.
3. $p_\theta(x^i|z, c) \sim N(x^i; \mu_\theta(z, c), \sigma_\theta(z, c))$ es la versoimilitud.

Con una función de pérdidas equivalente a la fórmula E.13, pero condicionado a la variable c:

$$L(\phi^*, \theta^*) = -ELBO = -\mathbb{E}_{q_\phi}[log(p_\theta(x^i|z, c)] + D_{KL}(q_\phi(z|x^i, c), p(z)) \tag{E.17}$$

E.4. GANs

La idea de las GAN[71] se basa en la competencia entre dos redes neuronales: el generador, que busca producir muestras de audio de alta calidad y realismo, y el

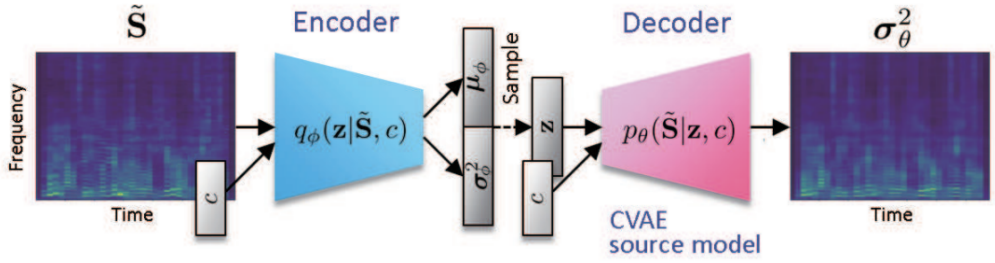


Figura E.12: Esquema Conditional Variational Autoencoder [57]

discriminador, cuya tarea es distinguir entre las muestras generadas por el generador y las muestras reales (Figura E.13). Estos modelos son otro tipo de modelo generativo que a través de un entrenamiento adversario consiguen correlar altamente las muestras de ruido $\mathbf{x}$ con las muestras originales $\mathbf{z}$.

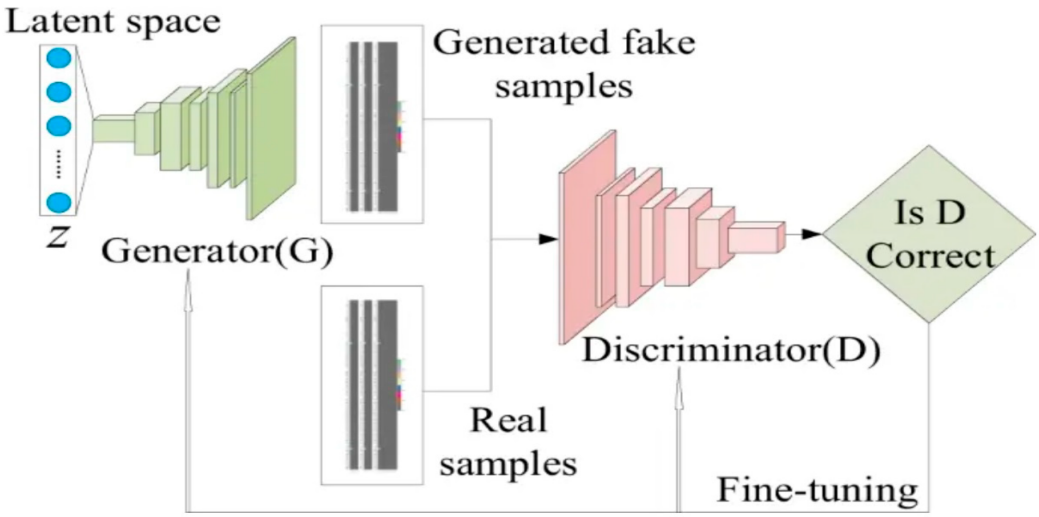


Figura E.13: Estructura de la GAN

E.4.1. Entrenamiento Adversario

El entrenamiento de una GAN tiene un comportamiento propio denominado Entrenamiento Adversario. Esta idea surgió del problema de que las redes neuronales eran susceptibles a un tipo de ejemplos contradictorios los cuáles "parecían naturales y correctos para el ser humano, pero la red fallaba". Así que para corregir esta idea, se empezó a entrenar con estos ejemplos adversos para hacer estas redes robustas frente a estos ejemplos.

Este entrenamiento se basa en una competición ***Minimax*** entre el generador y el discriminador, donde cada uno intenta optimizar su comportamiento. Este comportamiento queda definido a través de una función de pérdida representada en

la ecuación E.18

$$min_G max_D V(D,G) = min_G max_D E_{x\sim P_{data}(x)}[log(D(x)] + E_{z\sim p_z(z)}[log(1-D(G(z))]. \tag{E.18}$$

De esta ecuación se obtiene la idea de que el valor de la pérdida es relativo, es decir, indican como de bien esta comportándose el componente, respecto a su oponente. Así, puede darse el caso que exista aumento del valor de pérdida del generador y esto no implique una pérdida de calidad del dato generado.

Para dotar al generador de pérdidas no relativas del discriminar se intentan implementar otras pérdidas como la *feature loss*. Además, existen múltiples variacioens de las GANs, donde se hace uso de otro tipos de pérdidas como puede el ejemplo de la LSGAN, idea base de HiFiGAN que usa la función de coste cuadrática para así evitar desvanecimientos en los gradientes. Para actualizar los pesos de los componentes de las GANs, primero se actualizará el discriminador disminuyendo el gradiente de la función de pérdidas, y una vez este el discriminador optimizado se actualiza el generador aumentando el gradiente de la función de pérdidas (Algoritmo 4).

Algorithm 4 Entrenamiento Adversario[71]

for Nº epochs **do**
  for k steps **do**
    Genera m muestras de ruido $[z_1, ..., z_m]$ que sigan la distribución $p_g(z)$y transformalas con el generador(G(z))
    Coge m muestras reales de datos reales [$x_1, ..., x_m$]
    Actualiza el **discriminador** aumentando el gradiente de la función de pérdidas
    $\uparrow\uparrow \bigtriangledown_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$
  end for
  Genera m muestras de ruido $[z_1, ..., z_m]$ que sigan la distribución $p_g(z)$y transformalas con el generador(G(z))
  Actualiza el **generador** descendiendo el gradiente de la función de pérdidas.
  $\downarrow\downarrow \bigtriangledown_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [logD(x^{(i)}) + log(1 - D(G(z^{(i)})))]$
  Para minimizar esto suponemos que lo del discriminador es igual a 0 debido a que suponemos que D(x) es 1 que nos dará cero y entonces nos queda como:
  $\downarrow\downarrow \bigtriangledown_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [log(1 - D(G(z^{(i)})))]$
end for

E.5. Modelos de flujos

E.5.1. Normalización de Flujos

Para definir estos modelos, se consideran dos variables aleatorias **x** y **z** con una función de distribución p_X y p_z, respectivamente. Si denotamos **h** como una

transformada invertible donde $z = h(x)$ y $x = g(z)$, siendo $g = h^{-1}$.

Normalmente, se suele considerar p_X como una distribución de entrada simple, como puede ser la Gaussiana, y se le aplica una transformación **h** para modelar una distribución más compleja **Z** Si cogemos esta idea y le aplicamos un cambio de variable a la fórmula se obtiene la fórmula E.19, donde el determinante de la derivada representa el **Jacobiano**[61].

$$\begin{aligned} p_z(z) &= p_x(x)|det\frac{\partial x}{\partial z}| \\ &= p_x(h^{-1}(z))|det\frac{\partial x}{\partial z}| \\ &= p_x(h^{-1}(z))|det\frac{\partial h^{-1}(z)}{\partial z}| \end{aligned} \tag{E.19}$$

Una variación de la fórmula E.19 se puede interpretar como la ecuación E.20:

$$\begin{aligned} p_z(z) &= p_x(x)|det\frac{\partial h^{-1}(z)}{\partial z}| \\ &= p_x(x)|det(\frac{\partial h(x)}{\partial x})^{-1}| \\ &= p_x(x)|det\frac{\partial h(x)}{\partial x}|^{-1} \end{aligned} \tag{E.20}$$

De la fórmula E.20, se llega a la conclusión de que la función de probabilidad de un intervalo del espacio **X** debe permanecer inalterado en el espacio transformado **Z**. El determinante del Jacobiano es un término correctivo que tiene en cuenta la pendiente o la "sensibilidad"de la transformación dada por **h**.

Así que el modelo se puede considerar como una colección de transformadas invertibles anidadas entre sí, $h_1, h_2....h_n$, donde n representa el número de capas del modelo.

Para entender mejor el comportamiento se aplica el logaritmo a la fórmula E.20.

$$log(p_z(z)) = log(p_x(x)) - log|det(\frac{\partial h(x)}{\partial x})| \tag{E.21}$$

Para simplificar la notación de la ecuación E.21[1], vamos a considerar la transformada como una cadena de funciones de densidad de probabilidad de la

[1]En algunos artículos se puede encontrar como: $log(p_z(z)) = log(p_x(x)) + log|det(\frac{\partial g(x)}{\partial x})|$

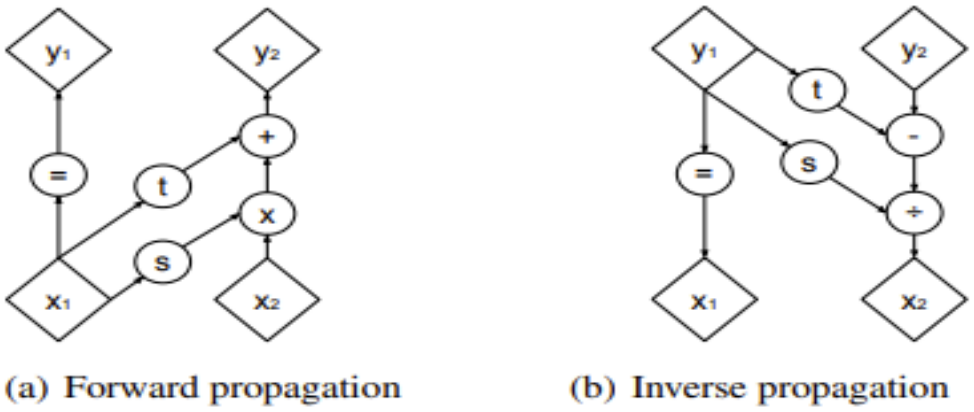
transformada. Así consideramos, $h = h_n, h_{n-1}, ..., h_0$

$$\begin{aligned} log(p_n(x_n)) &= log(p_{n-1}(x_{n-1})) - log|det(\frac{\partial h(x_{n-1})}{\partial x_{n-1}}| \\ &= log(p_{n-2}(x_{n-2})) - log|det(\frac{\partial h(x_{n-1})}{\partial x_{n-1}}| - log|det(\frac{\partial h(x_{n-2})}{\partial x_{n-2}}| \\ &= ... \\ &= log(p_0(x_0)) - \sum_{i=1}^{n} log|det\frac{\partial h(x_i)}{\partial x_i}| \end{aligned} \tag{E.22}$$

De la ecuación E.22, se puede interpretar que el cambio de variables es una manera directa de calcular la función de verosimilitud de una observación de datos complejos, a través de una distribución real p_n mediante una ditribución inicial p_0 y una serie de transformaciones invertibles $h_1, h_2, ...h_n$.

E.5.2. Acoplamiento Afín

Este método aprovecha la propiedad de las matrices triangulares, donde el determinante es igual al producto de sus términos diagonales, mejorando la eficiencia de cómputo. Así que suponiendo que se tiene un espacio vectorial $\mathbb{R}^D$, con una capa con entrada $\mathrm{X} \in \mathbb{R}^D$ y una capa de salida $\mathrm{Y} \in \mathbb{R}^D$. Primero, se elige un valor $d < D$, y se divide la entrada en dos subconjuntos $x_{1:d}, x_{d+1:D}$. Luego se definen dos funciones S y T que son escalables y transladables de dimensión $\mathbb{R}^{D-d}$ y se plantea un esquema como el definido en la Figura E.14. Con este esquema podemos definir la relación entre los datos de entrada y salida a través de la ecuación E.23, donde $\odot$ se trata del producto de matrices de Hadamard, que multiplica los elementos correspondientes de dos matrices de misma dimensión.



(a) Forward propagation (b) Inverse propagation

Figura E.14: Capa de Acoplamiento Afin

$$\begin{cases} y_{1:d} = x_{1:d} \\ y_{d+1:D} = x_{d+1:D} \odot exp(s(x_{1:d})) + t(x_{1:d}) \end{cases} \tag{E.23}$$

Se comprueba que la ecuación E.23[2] cumpla los dos requerimientos necesarios: fácilmente invertible y de fácil computo el determinante del Jacobiano.

$$\begin{cases} x_{1:d} = & y_{1:d} \\ x_{d+1:D} = & (y_{d+1:D} - t(x_{1:d})) \odot exp(-s(x_{1:d})) \end{cases} \tag{E.24}$$

$$\frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y_{1:d+1}}{\partial x_{1:d+1}} & \frac{\partial y_{1:d+1}}{\partial y_{d+1:d}} \\ \frac{\partial y_{d+1:d}}{\partial x_{1:d/2}} & \frac{\partial y_{d+1:D}}{\partial x_{d+1:D}} \end{bmatrix} = \begin{bmatrix} \mathbb{I}_d & 0_{dx(D-d)} \\ \frac{\partial x_{d+1:D}}{\partial z_{1:d}} & \frac{\partial x_{d+1:D}}{\partial z_{d+1:D}} \end{bmatrix} = \begin{bmatrix} \mathbb{I}_d & 0_{dx(D-d)} \\ \frac{\partial x_{d+1:D}}{\partial z_{1:d}} & diag(exp(s(x_{1:d}))) \end{bmatrix}$$

$$det[\frac{\partial y}{\partial x}] = \sum_{j=1}^{D-d} exp(s(x_{1:d}))_j = exp(\sum_{j=1}^{D-d} s(x_{1:d})_j) \tag{E.25}$$

De la ecuación E.24 se observa como para hacer la transformada inversa las funciones **s** y **t** no es necesario calcular su inversa, lo que permite que estas funciones sean de gran complejidad y se obtenga mayor flexibilidad en las representaciones. Estas funciones en muchos casos pueden ser modeladas por redes neuronales.

Mientras que de la ecuación E.25, se ve que el Jacobiano se trata de una matriz triangular inferior, donde el determinante es el producto de la diagonal. Esto hace que se pueda terminar considerando como la exponencial de una sumatorio, que es de fácil computo. Otra propiedad que se obtiene con estas capas, se trata de una equidad de la complejidad en el calculo de la transformada directa e inversa.

[2]Para simplificar el cálculo se hace la exponencial de la transformada aunque en la Figura E.14 no aparezca como tal.

Anexos F

Diseño de la encuesta de evaluación subjetiva de calidad del audio

Para evaluar la calidad de los audios sintetizados se diseñó una encuesta online. Se desarrolló una página web en html usando el dominio de las páginas de github[1], consiguiendo así un servidor capaz de almacenar todos los audios generados. Al crear la página en HTML5, se permite descargar y reproducir audios de manera sencilla y accesible para cualqueira. El problema de las páginas html es que son estáticas, es decir, no se permite la interacción con el usuario para modificar el contenido del documento. Por esto no se pueden realizar llamadas directas para guardar las respuestas en la base de datos. Para solucionar esto, se hizo uso del servidor **Dihana** de la universidad, que almacena la base de datos SQL y las funciones PHP que harán las llamadas a la base de datos para almacenar los datos (Figura F.1).

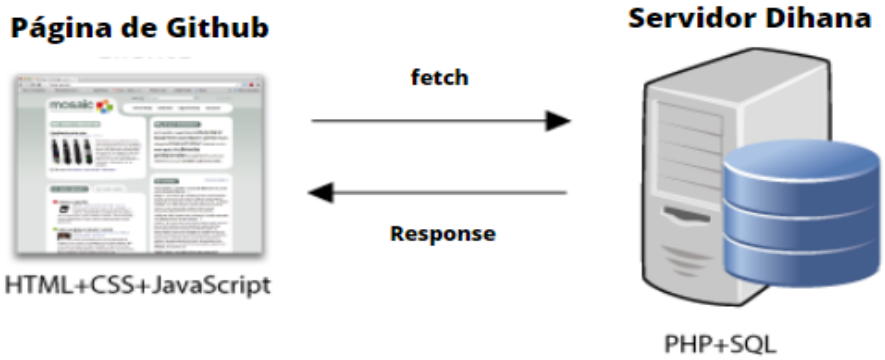


Figura F.1: Esquema la conexión entre la página web y el servidor

Así que en la página web se programó el front-end en html con funciones en Javascript que pudiera leer los datos y realizar un ***fetch*** con las funciones en PHP. Las funciones *fetch* son un tipo de función utilizadas para realizar peticiones de red a un servidor Web.

[1] https://santirf01.github.io/TFG_TTS_page.io/

Se plantea ahora la definición del formato de la encuesta, que se desglosa en dos escenarios distintos:

- **Primer escenario**: En este caso, se presentan audios tanto de prueba como sintetizados, ambos conteniendo las mismas frases. Se pregunta sobre la identificación del audio original, cuya respuesta será almacenada en un booleano. Además, se formulan dos preguntas adicionales: una relacionada con la percepción de la calidad y otra sobre la similitud. Estas dos preguntas recibirán una calificación en una escala del 0 al 5.

- **Segundo escenario**: En esta instancia, no se contarán con audios de prueba. Así que se eliminará la pregunta enfocada a la identificación del original.

Adicionalmente, en el json enviado en ambos casos se envia información que nos permita conocer el caso que evaluado, en el primer escenario este es el audioSeleccionado y en el segundo escenario el SpeakerSeleccionado. Como opción extra, en el segundo escenario se deja un formulario a rellenar con comentarios sobre los audios. El formato de los jsons envíados será:

Listing F.1: JSON primer escenario

```
{
  "audioSeleccionado":"",
  "correcto":True/False ,
  "respuestaSimilitud":0-5,
  "respuestaCalidad":0-5
}
```

Listing F.2: JSON segundo escenario

```
{
  "speakerSeleccionado":"",
  "respuestaSimilitud":0-5,
  "respuestaCalidad":0-5,
  "anotaciones":""
}
```

Además, al momento de almacenar los datos mediante la función PHP[2], se incluye la fecha de envío. Por último se genera en la base de datos las tablas para almacenar información. Así que en primer lugar en dirección *localhost* de ***mysql*** se creó la base de datos llamadas *encuesta_TTS*. Dentro de esta base se genera dos tablas para almacenar estos datos : **"RESULTADOS_Albayzin"** y **"RESULTADOS_Albayzin216"**. Los comandos usados para crear estas tablas son:

[2]Las funciones de php junto a las html usadas se encuentran disponibles también en el github: https://github.com/santirf01/TFG_TTS_page.io

Listing F.3: Tabla Resultados Albayzin 4 Speakers

```
CREATE TABLE RESULTADOS_Albayzin (
    id INT AUTO_INCREMENT PRIMARY KEY,
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    audio_escuchado VARCHAR(255),
    acertado BOOLEAN,
    similitud INT,
    calidad INT
);
```

Listing F.4: Tabla Resultados Albayzin 216 Speakers y Thalento

```
CREATE TABLE RESULTADOS_Albayzin_216 (
    id INT AUTO_INCREMENT PRIMARY KEY,
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    speakerSeleccionado VARCHAR(10),
    Similitud INT,
    Calidad INT,
    anotaciones TEXT
);
```