UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE POSTGRADO Y EDUCACIÓN CONTINUA
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

REDUCCIÓN DE TIEMPOS DE INFERENCIA EN MODELOS DE EFECTOS DE AUDIO MEDIANTE TÉCNICAS DE *KNOWLEDGE DISTILLATION*

TESIS PARA OPTAR AL GRADO DE
MAGÍSTER EN CIENCIA DE DATOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL ELÉCTRICO

DIEGO ALONSO CANALES RODRÍGUEZ

PROFESOR GUÍA:
FELIPE TOBAR HENRÍQUEZ

MIEMBROS DE LA COMISIÓN:
JORGE SILVA SÁNCHEZ
JAVIER RUÍZ DEL SOLAR SAN MARTÍN

Este trabajo ha sido parcialmente financiado por Fondecyt Regular N° 1210606

SANTIAGO DE CHILE
2023

RESUMEN DE LA TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIA DE DATOS Y AL TÍTULO PROFESIONAL DE INGENIERO CIVIL ELÉCTRICO
POR: DIEGO ALONSO CANALES RODRÍGUEZ
FECHA: 2023
PROF. GUÍA: FELIPE TOBAR HENRÍQUEZ

REDUCCIÓN DE TIEMPOS DE INFERENCIA EN MODELOS DE EFECTOS DE AUDIO MEDIANTE TÉCNICAS DE KNOWLEDGE DISTILLATION

En la música popular, los dispositivos que amplifican y/o alteran las señales de audio obtenidas desde los instrumentos son fundamentales y entendidos como parte de la creación musical. Por ejemplo, en el caso de la guitarra eléctrica, existe una gran variedad de efectos de audio y amplificadores que "colorean" y "transforman" el sonido para dar un carácter único al artista. Estos dispositivos, implementados en la forma de circuitos electrónicos, suelen ser costosos, delicados y difíciles de transportar. Esta investigación se enmarca en el modelamiento de estos dispositivos sobre señales de audio.

Efectos no lineales de distorsión como el overdrive y el *fuzz* son difíciles de modelar mediante ecuaciones explícitas, pero en los últimos años se han desarrollado modelos basados en *Deep Learning* que han mostrado resultados prometedores, pero a un costo computacional que complica el poder implementar los modelos como una herramienta en tiempo real. El objetivo de esta investigación es reducir los tiempos de inferencia de los métodos anteriores, entendiendo que una baja latencia de respuesta es fundamental para su uso en escenarios realistas. Se emplean técnicas de *Knowledge Distillation* (KD), que apuntan a obtener modelos más simples y rápidos a partir de modelos complejos ya entrenados. Se considera tanto el escenario en que se dispone de los datos de entrenamiento como en el que no.

Se realiza una serie de experimentos que considera los modelos del estado del arte y los dos casos posibles recién enunciados. Los resultados obtenidos muestran que sí es posible obtener modelos que introducen una menor latencia en inferencia aplicando KD desde modelos más grandes. Si bien los modelos destilados presentan un desempeño inferior al modelo completo, obtienen mejores resultados que los que se logran entrenando la arquitectura simple sin las técnicas de KD. Cabe destacar que en uno de los experimentos realizados se logra obtener un modelo más rápido, liviano y preciso que el modelo completo desde el cual se le aplica KD, proponiendo un nuevo estado del arte para el modelamiento de efectos no lineales de audio bajo los 3 criterios mencionados.

"Holding out for something greater
Holding out for something else"
♪
Parcels

Agradecimientos

Por encima de todo, doy las gracias a mi familia. A mis padres, Alejandra y Claudio, por siempre asegurarse de entregarme las mejores condiciones para mi desarrollo como estudiante y como persona. A sus esfuerzos y sacrificios individuales en favor de nuestra familia. Por el apoyo incondicional en los momentos en que las ganas de seguir hacían falta. A Bárbara, la Yaya, mi *ernita*, por escucharme y por ser mi principal apoyo. Por tantos momentos inolvidables juntos y por cada risa y sonrisa que me entregas cada día con tus ocurrencias.

A Alexandra, la Pepi, por tantos años de amistad y cariño, y por siempre confiar en mí y en cada cosa que hago. Al Team, Xabi, Javier y Toto, por el apoyo que cada uno me ha dado y por acompañarme a despejar la mente con algún jueguito, con una junta o practicando algún deporte. A Mica, por el amor que entregas, por iluminar mis días y por darme el empujoncito que necesitaba para recuperar las ganas de seguir progresando.

A los amigos de la universidad, Víctor, Bárbara, Javi, Pollo, Coon, Guerra, Óscar y Nibs, por todos los momentos vividos tanto fuera como dentro de la U y por todo lo que me enseñaron, cada uno a su manera. A las personas que ya no están, pero que en su momento fueron un aporte en mi vida.

Agradezco al profesor Felipe Tobar por su excelente labor como guía durante el desarrollo de este trabajo y por preocuparse de generar un ambiente de buena onda y colaboración con todos los tesistas. Al profesor Javier Ruiz del Solar, por formar parte de esta comisión y por la valiosa retroalimentación entregada. Al profesor Jorge Silva por su motivación e interés en el tema de la tesis y por ser parte de la comisión.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

La invención de variantes eléctricas o electrónicas de distintos tipos de instrumentos musicales, a mediados del siglo pasado, también trajo consigo el desarrollo de dispositivos que se encargan de amplificar y/o alterar las señales de audio obtenidas desde dichos instrumentos. Esto ha motivado el desarrollo de dispositivos de modificación del audio que se adecúan, específicamente, a las señales de audio provenientes de cada instrumento particular. Un caso específico de lo anterior ha ocurrido para instrumentos que durante varias décadas han sido de comercialización masiva, como la guitarra eléctrica y el bajo eléctrico. Existe una gran variedad de efectos de audio y amplificadores que "colorean" y "transforman" el sonido, y los artistas los utilizan para dar un carácter único a sus composiciones. Estos dispositivos, implementados en la forma de circuitos electrónicos, son costosos, delicados y en muchos casos difíciles de transportar debido a su gran tamaño y peso [1]. El problema que se aborda en esta tesis se enmarca en el modelamiento del efecto que los dispositivos mencionados tienen sobre una señal de audio, con el propósito de reemplazarlos por el modelo obtenido y así facilitar la tarea de traslado del equipamiento utilizado para la modificación de las señales de audio de instrumentos musicales. Eso se representa esquemáticamente en la Figura 1.1:

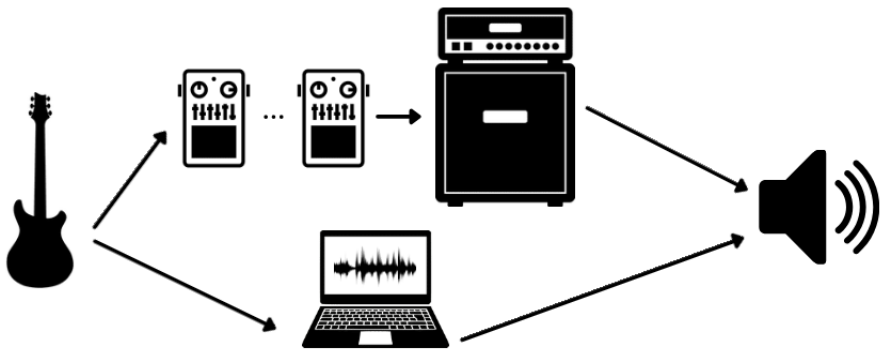


Figura 1.1: Esquema del modelamiento de efectos de audio. Elaboración propia.

Se han propuesto variadas estrategias para modelar tanto amplificadores de guitarras como efectos de audio. Un conjunto de estas estrategias consiste en utilizar ecuaciones que modelan explícitamente el comportamiento de los componentes electrónicos de los circuitos que constituyen a los dispositivos. No obstante, soluciones de este tipo son limitadas en el sentido de que cada efecto o amplificador debe modelarse a partir de conocimiento experto sobre cómo son implementados, lo cual limita su capacidad de generalización para distintos dispositivos. Con los avances publicados en las últimas décadas en el área de la Inteligencia Artificial y *Machine Learning* (ML), ha surgido una nueva corriente de desarrollo de modelos para el problema a tratar que, tal como se ha visto en una amplia gama de problemas resueltos mediante técnicas de estas características, ha ido logrando progresivamente mejores resultados en términos de ajuste a la señal modelada y capacidad de generalización a diferentes dispositivos. De forma más específica, modelos al nivel del estado del arte utilizan combinaciones de algoritmos de *Deep Learning* (DL) como Redes Neuronales Artificiales de tipo *Multilayer Perceptron* (MLP), *Convolutional Neural Networks* (CNN) y *Recurrent Neural Networks* (RNN), considerando variaciones de cada uno de estos.

Si bien los trabajos que establecen el estado del arte logran resultados suficientemente buenos en términos del ajuste a las señales objetivo, existe espacio para investigar acerca de la reducción de los tiempos de inferencia de los modelos, entendiendo que una baja latencia de respuesta es fundamental para su utilidad. Lo anterior ocurre porque al utilizar los modelos para modificar el sonido de una guitarra, por ejemplo, lo ideal es que al tocar el instrumento musical el modelo entregue una señal de salida, con el sonido modificado, de forma instantánea o en tiempo real. Por lo anterior, esta tesis se enfoca en la reducción de los tiempos de inferencia de los modelos del estado del arte. Para ello se emplean técnicas de *Knowledge Distillation* (KD), que se utilizan para la compresión de modelos o, dicho de otra forma, para obtener modelos más simples y rápidos a partir de modelos complejos ya entrenados.

1.1. Hipótesis

Teniendo en cuenta las motivaciones ya expuestas, se plantea que la hipótesis de este trabajo puede separarse en dos partes: en primer lugar, se propone que sí es posible disminuir los tiempos de inferencia de los modelos del estado del arte en la tarea del modelamiento de efectos de audio, entrenando variantes mediante técnicas de *Knowledge Distillation*, con una baja reducción en el desempeño de los modelos obtenidos. En segundo lugar, se postula que es posible superar el desempeño que se obtendría entrenando las mismas variantes de los modelos sin considerar la metodología presentada en esta tesis.

1.2. Objetivo General

El objetivo general de este trabajo consiste en reducir los tiempos de inferencia de modelos del estado del arte de amplificadores y efectos de audio, en el sentido de obtener modelos menos complejos, más livianos en memoria y rápidos en inferencia, que cumplan la misma tarea que los modelos originales con el menor sacrificio posible en su desempeño.

1.3. Objetivos Específicos

Para este trabajo, los objetivos específicos contemplados son los siguientes:

- Proponer criterios de evaluación que permitan comparar los modelos obtenidos de manera justa.
- Comparar el desempeño de arquitecturas ya existentes en la literatura para modelos de efectos de audio y amplificadores.
- Evaluar la efectividad de técnicas de *Knowledge Distillation* existentes en la literatura sobre un problema más complejo que los estudiados por sus respectivos autores.
- Entrenar modelos que apunten a mejorar el estado del arte en términos de reducir el tiempo de inferencia con miras de obtener una respuesta en tiempo real del modelo frente a una señal de entrada.

1.4. Contribución

Una vez desarrollado el trabajo, las contribuciones que esta tesis aporta son las siguientes. En primer lugar, se contribuye con un criterio de comparación de modelos, que considera múltiples factores relevantes al problema a resolver y que evita introducir sesgos hacia un modelo u otro en su definición. En segundo lugar, se aporta con una comparación justa de modelos representantes de dos de líneas de desarrollo que constituyen el estado del arte para esta tarea. Por último, se contribuye con la demostración de la aplicabilidad de *Knowledge Distillation* para el problema particular de modelamiento de efectos de audio no lineales. Esto lleva consigo el aporte de nuevos modelos, correspondientes a una versión comprimida de los modelos que establecen el estado del arte, con una menor cantidad de parámetros por almacenar y con menores tiempos de inferencia al procesar los datos de entrada. Lo anterior es un aporte directo al empleo de dichos modelos para aplicaciones en tiempo real, considerando que una baja latencia en el procesamiento de las señales de entrada es un factor fundamental para que los artistas puedan utilizar los modelos de efectos de audio obtenidos en sus flujos de creación y trabajo.

De manera más específica las contribuciones de esta tesis se enumeran a continuación:

- Determinación de cuatro criterios de comparación para modelos de efectos de audio.
- Implementación de dos modelos representantes del estado del arte para el modelamiento de efectos de audio no lineales.
- Comparación justa, sobre un mismo dataset, de los dos modelos representantes del estado del arte.
- Demostración de la aplicabilidad de técnicas de *Knowledge Distillation* sobre el modelamiento de efectos de audio no lineales.

- Obtención de modelos más precisos, livianos y/o rápidos en inferencia que los representantes del estado del arte.
- Modelo candidato a establecer nuevo estado del arte para el modelamiento de efecto de audio no lineal tipo *fuzz*.
- Aporte a la aplicabilidad de modelos de efectos de audio en contextos reales, al reducir la latencia en el procesamiento de las señales de entrada.

1.5. Estructura

Los capítulos restantes de esta tesis se organizan según lo explicado a continuación: el Capítulo 2 corresponde al Marco Teórico, en el que se exponen conceptos acerca de sonido y audio, de Señales y Sistemas, y sobre *Machine* y *Deep Learning* necesarios para una mejor comprensión del trabajo realizado. En dicho capítulo también se exponen 2 líneas de desarrollo de modelos que establecen el Estado del Arte en la tarea a resolver. El Capítulo 3, de Metodología, describe cómo se trata el problema a resolver, los datos utilizados y se enumeran y justifican los experimentos a realizar. Los resultados obtenidos en los experimentos realizados, junto al análisis correspondiente para cada uno se exponen en el Capítulo 4. Finalmente, el Capítulo 5 expone las conclusiones obtenidas una vez realizado el trabajo, en cuanto al cumplimiento de los objetivos y acerca de posibilidades para un trabajo futuro.

Capítulo 2

Marco Teórico

2.1. Sonido y Señal de Audio

El sonido puede definirse como una perturbación física de un medio que propaga ondas mecánicas a través de él. Las propiedades de las ondas propagadas dependen tanto de la perturbación en sí como de las características del medio a través del cual se propagan. Dichas propiedades dan lugar a diferentes tipos de sonidos que pueden explicarse a grandes rasgos a través de la frecuencia, amplitud y forma de onda. Distintas frecuencias dan lugar a distintos tonos perceptibles por los seres humanos como diferentes notas musicales. Cabe destacar que un único sonido puede estar compuesto por un conjunto de varias frecuencias, de las cuales la más baja observada es llamada frecuencia fundamental. Por su parte, la amplitud de la onda define la percepción de la intensidad del sonido. La forma de onda, entendiéndola como un concepto que engloba tanto a las variaciones en amplitud y frecuencia de la onda, define distintos timbres, siendo el timbre la característica que permite distinguir fuentes de sonido distintas cuando vibran en una misma frecuencia fundamental.

Las ondas sonoras pueden ser capturadas mediante transductores como micrófonos y cápsulas o pastillas en el caso de guitarras eléctricas. Los transductores permiten transformar las ondas de sonido en señales de energía eléctrica que pueden ser transmitidas mediante cables hacia circuitos electrónicos de amplificación o modificación de las señales, o pueden ser discretizadas y almacenadas digitalmente. Estas señales son conocidas como señales de audio y permiten representar las ondas sonoras capturadas. En la Figura 2.1 se observan ejemplos de señales de audio con distinta forma de onda que, por lo tanto, representan diferentes sonidos pero que tienen la misma frecuencia fundamental y la misma amplitud máxima:

2.2. Sistemas

En los campos de Procesamiento de Señales y de Señales y Sistemas, un sistema se define como una relación que mapea una función de entrada a una de salida. En general,

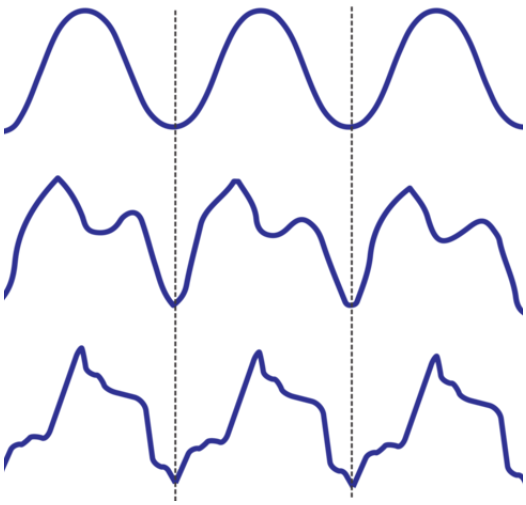


Figura 2.1: Ejemplos de señales de audio de igual frecuencia y amplitud pero con distinta forma de onda.[1]. Las líneas verticales demarcan 1 período de la señal.

las funciones mencionadas corresponden a señales que pueden ser discretas o continuas y el sistema corresponde a aquello que transforma la señal de entrada en la señal de salida [2]. En el problema abordado en este trabajo, tanto los efectos de audio como los amplificadores pueden entenderse como sistemas, siendo la señal de entrada la señal de audio capturada desde un instrumento y la señal de salida la señal de audio modificada por el sistema.

Los sistemas son clasificables de acuerdo a sus propiedades, siendo un grupo relevante los llamados sistemas lineales e invariantes en el tiempo o *Linear Time-Invariant*, ya que permiten conocer, de forma relativamente sencilla, la salida correspondiente para cualquier señal arbitraria de entrada, lo que facilita su modelamiento [2].

2.2.1. *Linear Time-Invariant* (LTI)

Se dice que un sistema es lineal e invariante en el tiempo o LTI cuando cumple las siguientes propiedades:

- Linealidad: Un sistema es lineal si cumple en conjunto con las propiedades de aditividad y proporcionalidad, las cuales en conjunto pueden ser explicadas por las siguientes ecuaciones:

$$R\{f_1(t)\} = y_1(t) \tag{2.1}$$

$$R\{f_2(t)\} = y_2(t) \tag{2.2}$$

$$R\{\alpha f_1(t) + \beta f_2(t)\} = \alpha y_1(t) + \beta y_2(t) \tag{2.3}$$

[1]Tomada de `https://www.themusictelegraph.com/427`.

Considerando un sistema R, funciones de entrada f_1,f_2, funciones de salida y_1,y_2 definidas como $y_i = R(f_i)$, y α, β constantes, un sistema es lineal si cumple que para una función de entrada que es combinación lineal de otras funciones, la salida corresponde a la combinación lineal de las funciones de salida respectivas, tal como se representa en la ecuación (2.3).

- Invarianza en el tiempo: Si se considera una función de entrada $f(t)$ con una función de salida $y(t)$, se dice que un sistema R es invariante en el tiempo si cumple que al desplazar temporalmente la función de entrada en un tiempo t_0, la función de salida del sistema es la función de salida con el mismo desplazamiento temporal. Esto se representa en las ecuaciones (2.4) y (2.5).

$$R\{f(t)\} = y(t) \tag{2.4}$$

$$R\{f(t - t_0)\} = y(t - t_0) \tag{2.5}$$

Los sistemas que cumplen con ambas propiedades pueden ser modelados con cierta facilidad, ya que su salida se puede expresar como una convolución entre la señal de entrada y la respuesta al impulso del sistema, siendo esta última la señal de salida que entrega el sistema cuando la función de entrada es un impulso unitario [2].

Por el contrario, aquellos sistemas que no cumplen la primera propiedad son llamados sistemas no lineales y los que no cumplen la segunda son llamados sistemas variantes en el tiempo. Para este tipo de sistemas no se cumple ninguna propiedad particular que facilite conocer su salida para entradas arbitrarias, por lo que obtener modelos que puedan generalizar para distintos tipos de sistemas no lineales y variantes en el tiempo sigue siendo un campo de investigación abierto.

2.3. Efectos de Audio

Los efectos de audio consisten en dispositivos, generalmente implementados en circuitos electrónicos, que modifican señales de audio. Como ya se dijo, estos dispositivos pueden entenderse como sistemas en que la señal de entrada es una señal de audio y la señal de salida es la señal de audio modificada por el sistema. Lo anterior permite clasificar los distintos tipos de efectos de audio como sistemas lineales o no lineales y variantes o invariantes en el tiempo.

Como los efectos de audio pueden modificar las características de la señal de entrada, se producen alteraciones que son percibidas por los oyentes. A continuación, se entrega un ejemplo de efecto de audio según las dos categorías de sistemas mencionadas, además de una breve descripción subjetiva de las sensaciones que provoca el escuchar cada efecto:

- No lineal: *Fuzz*.
  El efecto de *fuzz* modifica una señal de audio de entrada saturándola en cierto nivel de amplitud, lo que puede entenderse como un recorte de la señal, entregando como salida una versión de la entrada con *peaks* de amplitud, como se observa en el ejemplo de la

Figura 2.2. Esto produce un sonido que podría describirse como ruidoso o sucio, que se utiliza típicamente en guitarras de géneros musicales como el *rock*, *metal* y subgéneros asociados.

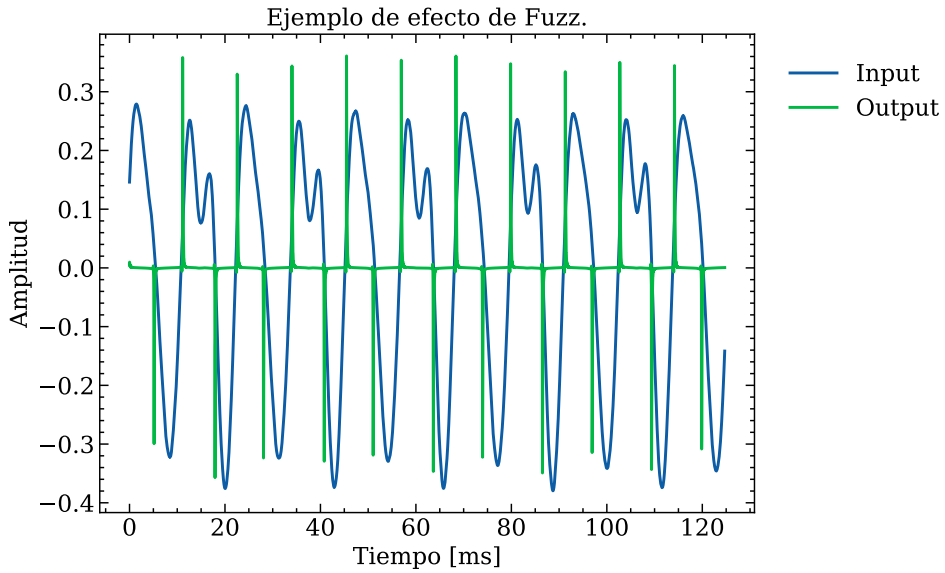


Figura 2.2: Ejemplo de efecto de *fuzz*. Elaboración propia.

- Variante en el tiempo: *Tremolo*.
  El efecto llamado *tremolo* altera la señal de entrada aplicando un proceso de modulación en amplitud, como se observa en la Figura 2.3. Este efecto produce la sensación de un sonido vibrante, debido a las modificaciones en la amplitud efectuadas sobre la señal original.

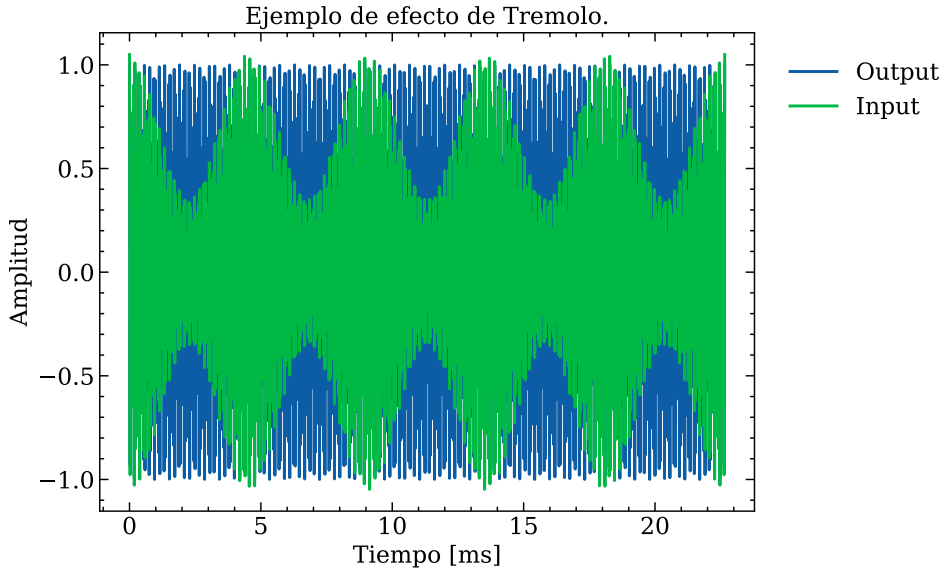


Figura 2.3: Ejemplo de efecto de *tremolo*. Elaboración propia.

2.4. Amplificadores de Audio

Los amplificadores son dispositivos electrónicos que tienen como finalidad incrementar la amplitud de una señal de audio de entrada. Por esto, también pueden ser entendidos como sistemas y su funcionalidad se representa en el esquema de la Figura 2.4:

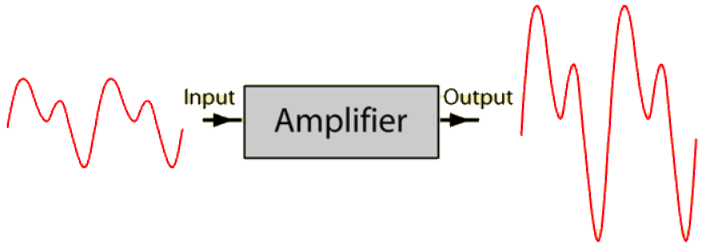


Figura 2.4: Esquema de funcionalidad de un amplificador de audio.[2]

Para el caso particular de los amplificadores de guitarra, los dispositivos suelen introducir modificaciones no lineales en la señal de salida. Esto se debe al comportamiento de sus componentes electrónicos como los tubos de vacío en amplificadores antiguos o transistores en amplificadores de diseño más reciente. En la comunidad musical es ampliamente conocido que un gran número de guitarristas valoran perceptivamente las características de ciertos amplificadores, más aún en el caso de amplificadores que funcionan con tubos de vacío. Coincidentemente, estos amplificadores son en particular muy delicados y difíciles de transportar, por lo que también es interesante modelarlos.

2.5. *Machine Learning* (ML)

El Aprendizaje de Máquinas o *Machine Learning* (ML) es una disciplina de la Inteligencia Artificial que engloba a distintos algoritmos que permiten que un sistema aprenda a realizar una tarea de forma autónoma o sin ser explícitamente programado para aquello. Típicamente, las tareas aprendidas mediante algoritmos de ML son las de clasificación, regresión, *clustering*, entre otras. Dentro de la disciplina existen distintos enfoques de aprendizaje, correspondientes al aprendizaje supervisado, no supervisado, semi-supervisado y reforzado. En este trabajo se utiliza el enfoque de aprendizaje supervisado, el cual puede definirse como el aprendizaje a partir de datos de entrada y datos de salida deseados, de forma que es posible monitorear las diferencias entre salidas obtenidas y salidas objetivo y aprender el comportamiento deseado. En el problema que se busca resolver en esta tesis, se cuenta con datos de entrada que son las muestras de una señal de audio sin procesar (*dry signal*) y la salida deseada son las muestras correspondientes de una señal de audio modificada (*wet signal*), ya sea por un efecto o por un amplificador.

[2]Tomada de `http://hyperphysics.phy-astr.gsu.edu/hbase/Audio/amp.html`.

2.6. *Deep Learning* (DL)

El Aprendizaje Profundo o *Deep Learning* (DL) corresponde a un subconjunto de algoritmos de ML que se basa en ANN de varias capas. En los últimos años se han desarrollado distintas variantes de algoritmos de ML que se ajustan a las tareas específicas que se busca resolver con esta clase de algoritmos. A continuación, se entrega una explicación de los algoritmos que son mencionados y utilizados en este trabajo:

2.6.1. Perceptrón

El Perceptrón [3] es un algoritmo de ML que consiste en la unidad básica de las Redes Neuronales Artificiales (ANN) que se explican más adelante en este reporte. Su definición se inspira en el funcionamiento de las neuronas biológicas, las cuales reciben una señal eléctrica proveniente de otras neuronas, con diferente intensidad, y si la señal es suficientemente intensa para estimularlas, estas transmiten una señal eléctrica a las neuronas siguientes.

Cada Perceptrón puede entenderse como un discriminador lineal, que permite aprender la tarea de clasificación binaria. Como se puede observar en la representación de la Figura 2.5, para una entrada x, se calcula un producto escalar con un conjunto de pesos w, obteniendo un valor y. Sobre este último se aplica una "función de activación", que para la tarea de clasificación suele ser una función no lineal que permite discriminar entre dos valores de salida posibles (σ en la Figura 2.5). No obstante, utilizando funciones de activación lineales es posible obtener un valor de salida que puede ser utilizado para el aprendizaje de tareas de regresión. Mediante algoritmos de optimización, comparando las salidas deseadas con las obtenidas para distintos datos de entrada, es posible actualizar los pesos y aprender progresivamente a resovler las tareas mencionadas.

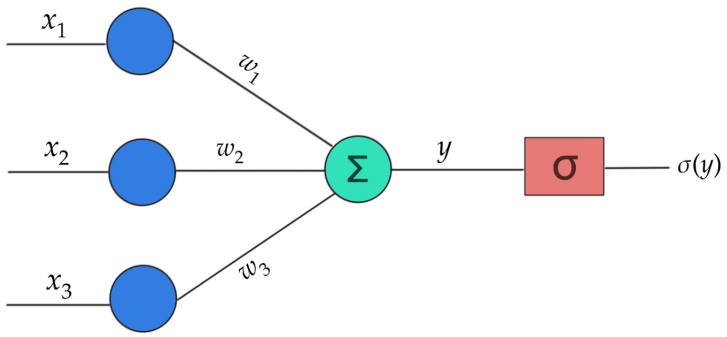


Figura 2.5: Representación de un Perceptrón.[3]

[3]Tomada de `https://becominghuman.ai/a-brief-introduction-to-perceptron-f3b9bade8f67`.

2.6.2. *Multilayer Perceptron* (MLP)

El Perceptrón Multicapas o *Multilayer Perceptron* (MLP), puede definirse, tal como lo dice su nombre, como un conjunto de perceptrones interconectados que forman múltiples capas. Una representación de esto puede verse en la Figura 2.6, que corresponde a un MLP con una capa para los datos de entrada, dos capas ocultas y una capa de salida, entendiendo cada capa como un conjunto de nodos o perceptrones que están a un mismo nivel de profundidad, que no tienen conexiones directas entre sí y que están conectados con los nodos de la capa anterior y la siguiente. Tanto el número de capas como la cantidad de nodos por capa pueden variar de acuerdo al diseño de la arquitectura de la red. Cabe destacar que en cada una de las capas se aplica una función de activación sobre la salida de los nodos que la componen. La ventaja que esta arquitectura muestra frente a un Perceptrón radica en que este último puede aprender la solución de problemas linealmente separables, mientras que la red MLP puede aprender patrones más complejos.

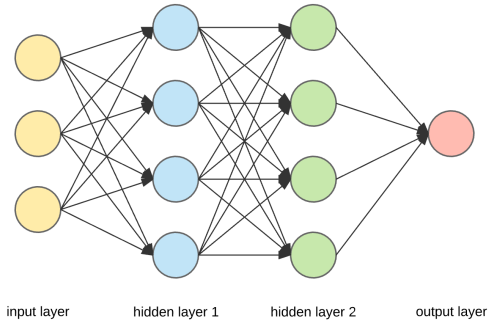


Figura 2.6: Representación de una red MLP.[4]

2.6.3. *Convolutional Neural Networks* (CNN)

Las *Convolutional Neural Networks* (CNN) [4] o Redes Neuronales Convolucionales fueron propuestas inicialmente para el procesamiento de imágenes y basan su funcionamiento en el aprendizaje de filtros o *kernels* que son aplicados sobre la imagen mediante la operación de convolución. Esta operación se define como un producto escalar que se desplaza sobre la imagen obteniendo un valor para cada una de las posiciones posibles. Esto se muestra en el ejemplo de la Figura 2.7, en la que la imagen de entrada y el filtro se representan como un arreglo bidimensional, siendo el segundo de menor tamaño. Las dimensiones del *kernel* definen el campo receptivo de este sobre la imagen.

[4]Tomada de `https://gupta-anika.github.io/Machine%20learning%20primer.pdf`.

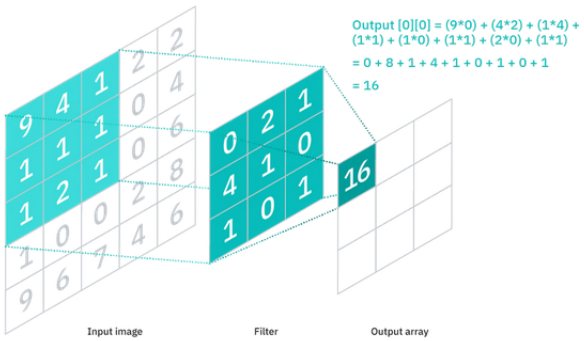


Figura 2.7: Ejemplo de la operación de convolución entre un *kernel* y una imagen.[5]

Si bien esta idea fue propuesta para trabajar con imágenes, también es posible aplicarla sobre datos secuenciales o unidimensionales. Así como en el ejemplo mostrado, tanto la entrada como el *kernel* son bidimensionales, en este caso ambos elementos cuentan con sólo una dimensión. Un ejemplo se muestra en la Figura 2.8, en la que se aplica la convolución con un filtro de largo 3, desplazándolo a lo largo de una secuencia de entrada. Es necesario destacar que el filtro aplicado en las distintas posiciones es el mismo.

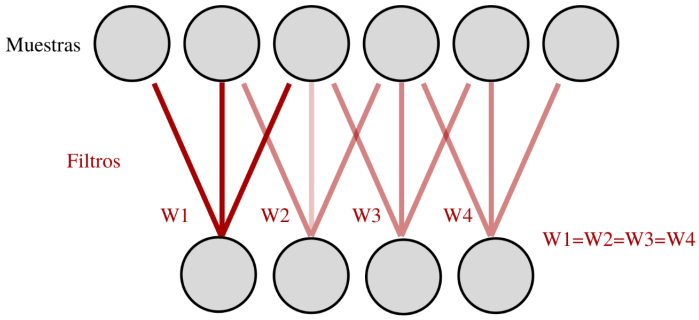


Figura 2.8: Convolución sobre datos unidimensionales. Elaboración propia.

Las aplicación de redes CNN sobre datos de entrada permite extraer *features* o características relevantes. Esto ocurre en la medida de que se configure un algoritmo de aprendizaje utilizándolas. Por ejemplo, pueden usarse para extraer características que posteriormente serán utilizadas por algún algoritmo de clasificación. Lo anterior permite implementar un aprendizaje *end-to-end* en el que las CNN aprenden los *kernels* aplicados sobre la entrada.

En la literatura se han propuesto variaciones sobre las redes CNN, siendo algunas de ellas relevantes en este trabajo. Estas se explican en las siguientes subsecciones.

2.6.3.1. *Dilated Convolution*

Las Convoluciones Dilatadas o *Dilated Convolutions* [5] fueron propuestas en el contexto del análisis de señales de audio y buscan incrementar el campo receptivo de los *kernels*

[5]Tomada de `https://chowdera.com/2022/148/202205280853244711.html`.

aprendidos en una red CNN, sin necesariamente aumentar el tamaño del filtro, evitando de esta forma requerir más recursos computacionales. En este contexto, el campo receptivo puede entenderse como la cantidad de muestras de la señal que influyen en el aprendizaje de un *kernel*. Para lograr lo anterior los autores proponen ignorar ciertas muestras intermedias, tal como se muestra en la Figura 2.9 a la derecha, en contraste a una convolución sin dilatación como la que se muestra en la Figura 2.9 a la izquierda. También pueden observarse distintos valores para la dilatación, que se traducen en cantidad de muestras no consideradas para calcular la convolución. Es importante notar que la cantidad de parámetros aprendidos en ambos ejemplos mostrados es la misma, pero el caso con dilatación permite alcanzar un campo receptivo mucho más extenso.

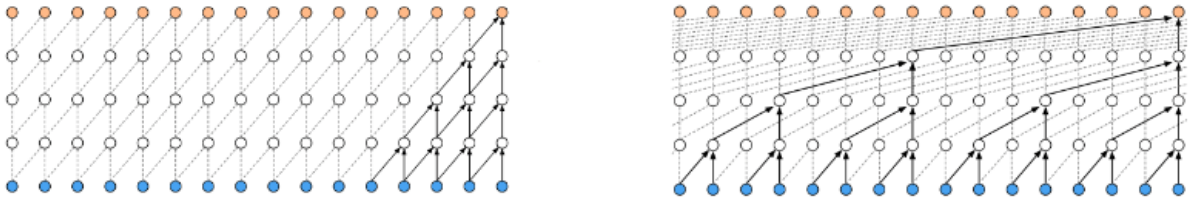


Figura 2.9: Ejemplo de Convolución (izq.) y Convolución Dilatada (der.) [6].

2.6.3.2. *Locally Connected Convolution*

Otra modificación propuesta sobre las convoluciones calculadas en las CNN es la llamada Conexión Local, dando lugar a las *Locally Connected Convolutions* [7]. Esto consiste en que para cada capa convolucional no se aprende un único *kernel* que se aplica sobre la entrada. La propuesta consiste en aprender un filtro independiente para cada una de las posiciones sobre las que se aplica la convolución. La diferencia entre una convolución sin conexiones locales y una que sí las tiene se observa en la Figura 2.10. Esto permite aprender un banco de filtros que se aplica sobre la entrada, obteniendo filtros para distintas secciones de una señal de entrada, en lugar de una serie de filtros de distintas capas que se aplican sobre la señal completa.

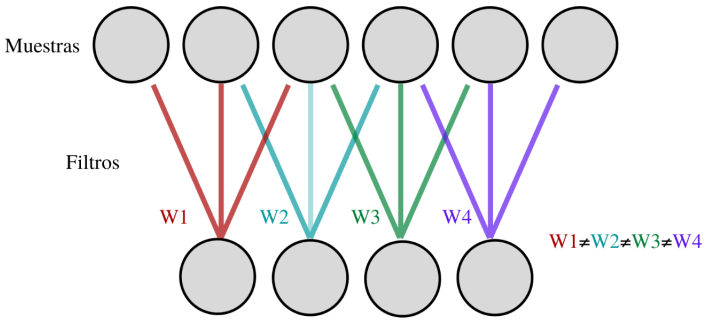


Figura 2.10: Ejemplo *Locally Connected Convolution*. Elaboración propia.

2.6.4. *Recurrent Neural Networks* (RNN)

Las Redes Neuronales Recurrentes o *Recurrent Neural Networks* (RNN) [8] corresponden a un grupo de ANN diseñadas para procesar datos secuenciales con dependencias temporales. Su principio de funcionamiento se basa en incorporar información obtenida con datos de entrada de un determinado instante de tiempo al procesamiento de entradas siguientes. Esto se logra conectando la salida de un nodo de la red como entrada de sí mismo, como se observa en la Figura 2.11 a la izquierda. Considerando que los datos de entrada son secuenciales, el funcionamiento de una RNN puede visualizarse según la representación mostrada en la Figura 2.11 a la derecha, en la que se observa que cada dato de entrada es procesado utilizando la información obtenida con el dato anterior de la secuencia.

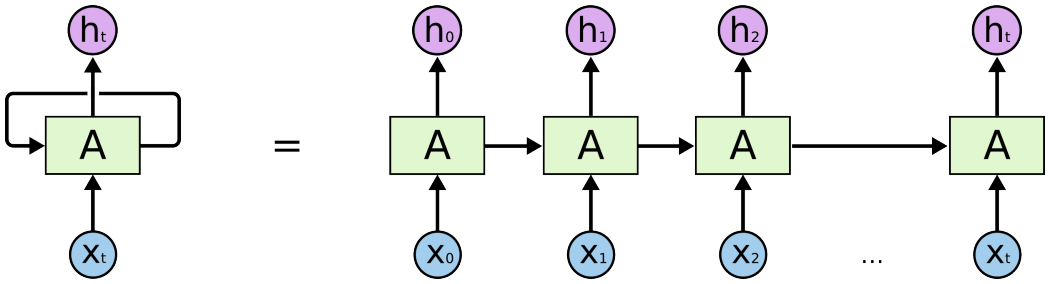


Figura 2.11: Representación de la arquitectura de una RNN.[6]

Este tipo de redes permite aprender dependencias temporales sobre datos de texto, audio, video, etc. No obstante, este tipo de redes, al igual que gran parte de las ANN, es entrenado mediante algoritmos de aprendizaje basados en gradiente para actualizar los pesos aprendidos y sufre del llamado *vanishing/exploding gradient problem* [9] al trabajar con secuencias con dependencias de largo plazo. La actualización de los pesos de una ANN en general se realiza de forma proporcional a la derivada parcial de una función de error entre la salida obtenida y la deseada, con respecto a cada peso actualizado. El problema mencionado ocurre cuando el valor de dicha derivada se vuelve infinitesimalmente pequeño, o grande en caso contrario, lo cual hace imposible un entrenamiento adecuado de la red.

Frente a dicha problemática han surgido modificaciones de las RNN que buscan resolverla. Una de ellas corresponde a las redes *Long Short-Term Memory* que se explican a continuación:

2.6.4.1. *Long Short-Term Memory* (LSTM)

Las redes *Long Short Term-Memory* (LSTM) [10], como ya se mencionó, corresponden a una modificación sobre las RNN que permite abordar la problemática del *vanishing/exploding gradient*. Esta incorpora un conjunto de *gates* o compuertas que permiten que la red pueda aprender cual información conserva y cual descarta para propagarla hacia el procesamiento de datos de entrada siguientes en la secuencia. Los *gates* corresponden a operaciones de combinaciones lineales con parámetros aprendibles y aplicación de funciones de activación.

[6]Tomada de `http://nmi-lab.org/PSYCH239-NNML19/slides_8.html?print-pdf#/`.

Una celda LSTM define su estado a partir de dos tensores llamados *cell state* o c y *hidden state* o h. Para un instante de tiempo n, el cálculo de los estados $h(n)$ y $c(n)$ recibe como información de entrada los estados del tiempo anterior, $h(n-1)$ y $c(n-1)$, y los datos de entrada $x(n)$. Esta actualización se realiza según las ecuaciones mostradas a continuación, donde $i(n)$ es llamada *input gate*, $f(n)$ es la *forget gate*, $\tilde{c}(n)$ es candidata a a $c(n)$. Además, tanh y σ son las funciones de activación tangente hiperbólica y sigmoide. Los tensores W y b corresponden a los parámetros aprendidos en el entrenamiento de la red.

$$i(n) = \sigma(W_{ii}x(n) + b_{ii} + W_{hi}h(n-1) + b_{hi}) \tag{2.6}$$

$$f(n) = \sigma(W_{if}x(n) + b_{if} + W_{hf}h(n-1) + b_{hf}) \tag{2.7}$$

$$\tilde{c}(n) = \tanh(W_{ic}x(n) + b_{ic} + W_{hc}h(n-1) + b_{hc}) \tag{2.8}$$

$$o(n) = \sigma(W_{io}x(n) + b_{io} + W_{ho}h(n-1) + b_{ho}) \tag{2.9}$$

$$c(n) = f(n)c(n-1) + i(n)\tilde{c}(n) \tag{2.10}$$

$$h(n) = o(n)\tanh(c(n)) \tag{2.11}$$

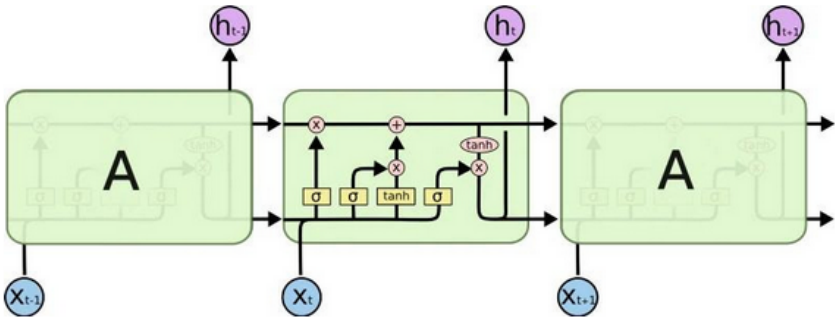


Figura 2.12: Representación de la arquitectura de una celda LSTM.[7]

2.6.4.2. *Gated Recurrent Unit* (GRU)

Las redes *Gated Recurrent Unit* o GRU [11] son otra variante de las RNN que también aborda la problemática del *vanishing/exploding gradient* mediante el uso de compuertas. En comparación a las LSTM son de arquitectura más simple, lo que reduce sus costos computacionales. No obstante, suelen mostrar desempeños más bajos para problemas que involucran dependencias temporales largas [12].

Una celda GRU define su estado a partir de un único tensor, llamado *hidden state* o h. Para un instante de tiempo n, el cálculo del estado $h(n)$ recibe como información de entrada el estado en el tiempo anterior $h(n-1)$ y los datos de entrada $x(n)$. El cálculo del estado se lleva a cabo según las ecuaciones mostradas en seguida. En ellas, $r(n)$ es la *reset gate*, $z(n)$ es llamada *update gate* y $\tilde{h}(n)$ es la cantidata a $h(n)$. Al igual que en las LSTM, W y b corresponden a los parámetros aprendidos en el entrenamiento.

[7] Tomada de `http://nmi-lab.org/PSYCH239-NNML19/slides_8.html?print-pdf#/`.

$$r(n) = \sigma(W_{ir}x(n) + b_{ir} + W_{hr}h(n-1) + b_{hr}) \tag{2.12}$$

$$z(n) = \sigma(W_{iz}x(n) + b_{iz} + W_{hz}h(n-1) + b_{hz}) \tag{2.13}$$

$$\tilde{h}(n) = \tanh(W_{ih}x(n) + b_{ih} + r(n)(W_{hh}h(n-1) + b_{hh})) \tag{2.14}$$

$$h(n) = (1 - z(n))\tilde{h}(n) + z(n)h(n-1) \tag{2.15}$$

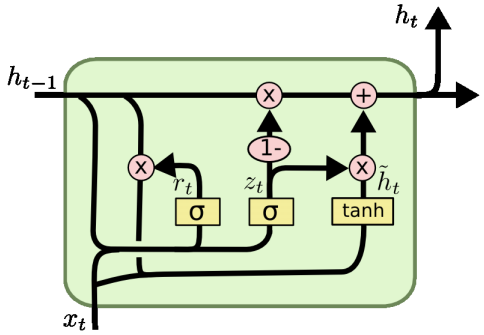


Figura 2.13: Representación de la arquitectura de una celda GRU.[8]

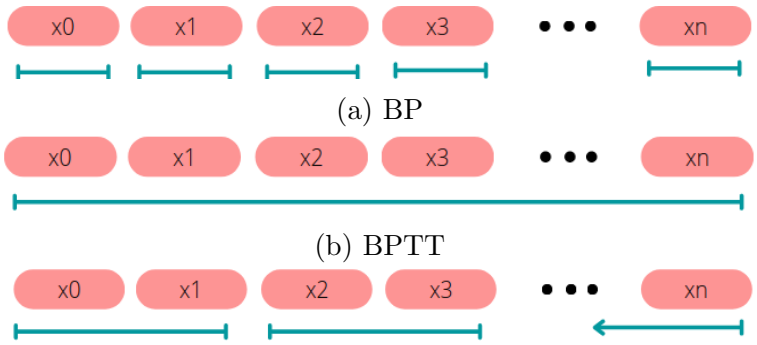
2.6.4.3. *Truncated Backpropagation Through Time* (TBPTT)

Truncated Backpropagation Through Time o TBPTT es una variante del algoritmo de entrenamiento de *Backpropagation* (BP), utilizado típicamente en el entrenamiento de redes neuronales recurrentes para reducir los costos computacionales.

Al procesar, por ejemplo, una serie de tiempo, BP realiza actualizaciones de los parámetros de la red neuronal entrenada luego de procesar cada muestra de la serie, lo que pueed ser caro computacionalmente. Otro enfoque propuesto en la literatura es el llamado *Backpropagation Through Time* (BPTT) [13], en el cual, en lugar de actualizar los parámetros para cada muestra procesada, se actualizan luego de haber procesado una serie completa de datos. Lo anterior puede no ser conveniente, sobre todo en el caso en que se trabaja con series de tiempo con muchas muestras, puesto que la actualización de parámetros es poco frecuente, lo que lleva a que el entrenamiento de la red sea muy lento.

TBPTT [14] propone un enfoque intermedio entre BP y BPTT, actualizando los parámetros luego de procesar una cantidad fija de muestras de la serie de tiempo. Esto reduce los costos computacionales del entrenamiento de la red y también permite que el aprendizaje sea más rápido. En la Figura 2.14 se busca ilustrar de manera comparativa la idea detrás de BP, BPTT y TBPTT. En ella, los bloques $x_1, x_2, ..., x_n$ representan muestras de la serie de tiempo procesada, mientras que los *brackets* debajo de ellos indican cada cuantas muestras se realiza la actualización de parámetros de la red neuronal. En el caso mostrado, TBPTT utiliza un *step* de actualización igual a 2.

[8]Tomada de `https://primo.ai/index.php?title=Gated_Recurrent_Unit_(GRU)`.



(a) BP

(b) BPTT

(c) TBPTT con *step*=2

Figura 2.14: Ilustración comparativa entre BP, BPTT y TBPTT. Elaboración Propia.

2.6.5. *Encoder-Decoder*

El término *Encoder-Decoder* es utilizado para definir una arquitectura particular de ANN que es ampliamente utilizada en tareas de tipo *sequence-to-sequence* como la traducción automática. Se compone de dos etapas, una de codificación, en la que los datos de entrada son procesados hasta obtener una representación de ellos en un espacio latente que es típicamente de menor dimensionalidad que la entrada, llamada *code* o código, y una segunda etapa de decodificación que, a partir de dicho código, realiza una reconstrucción de los datos [15]. Un caso particular de esto son los *Autoencoders*, en los que se busca reconstruir idénticamente la señal de entrada luego de la decodificación. Un ejemplo de arquitectura se muestra en la Figura 2.15, donde las etapas de *encoding* y *decoding* pueden estar compuestas por distintos tipos de ANN y, exceptuando el caso de los *Autoencoders*, no necesariamente siguen una misma estructura.

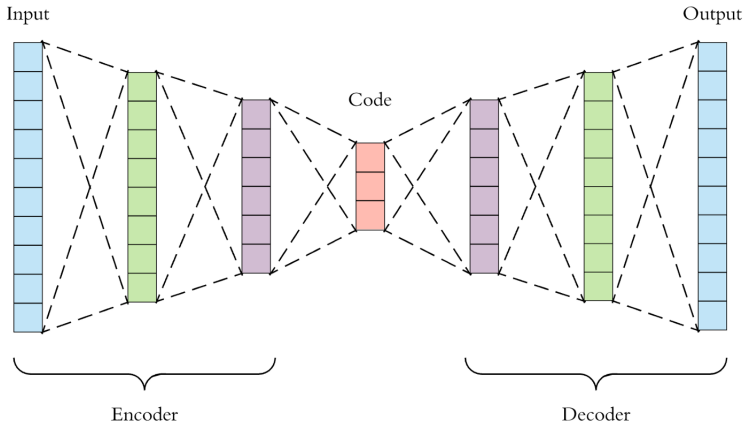


Figura 2.15: Representación de la arquitectura de un *Encoder-Decoder*.[9]

[9]Tomada de `https://tomfletcher.github.io/FoDA/lectures/L22-AE.pdf`.

2.7. Funciones de Activación

Como ya se mencionó en la Sección 2.6.1, la unidad básica de las Redes Neuronales Artificiales basa su funcionamiento en el de las neuronas biológicas. En estas últimas, parte fundamental de su operación consiste en determinar qué tan intensa es la señal eléctrica de salida de una neurona cuando la señal de entrada es suficientemente fuerte como para activarla. El análogo matemático de este comportamiento lo definen las llamadas funciones de activación, las cuales permiten definir el valor de salida para una neurona artificial a partir del valor de entrada [16]. En esta sección se describen brevemente las funciones de activación involucradas en las Redes Neuronales Artificiales utilizadas en esta tesis:

2.7.1. Función Sigmoide (σ)

La función sigmoide es una de las funciones de activación más comúnmente utilizadas en la definición de Redes Neuronales Artificiales. Si bien las funciones sigmoides corresponden a una familia de funciones de comportamiento similar, en el contexto de *Deep Learning* generalmente se llama sigmoide a la función logística. Esta función mapea valores desde $\mathbb{R}$ al conjunto $(0, 1)$ y se define según la ecuación 2.16. Su forma es la mostrada en la Figura 2.16 [16]:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.16}$$

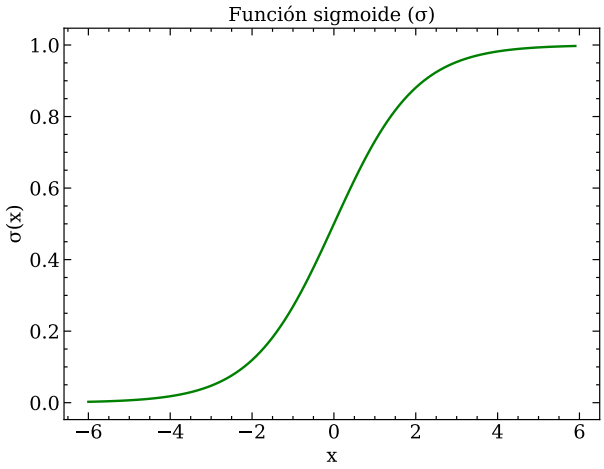


Figura 2.16: Función de activación sigmoide (σ). Elaboración propia.

2.7.2. Tangente Hiperbólica (tanh)

La tangente hiperbólica es otra de las funciones de activación más utilizadas. Se define como el cociente entre el seno y el coseno hiperbólico, según la ecuación 2.17. Si bien su

forma, mostrada en la Figura 2.17, es similar a la de la función sigmoide, esta mapea valores desde $\mathbb{R}$ al conjunto $(-1, 1)$ [16].

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.17}$$

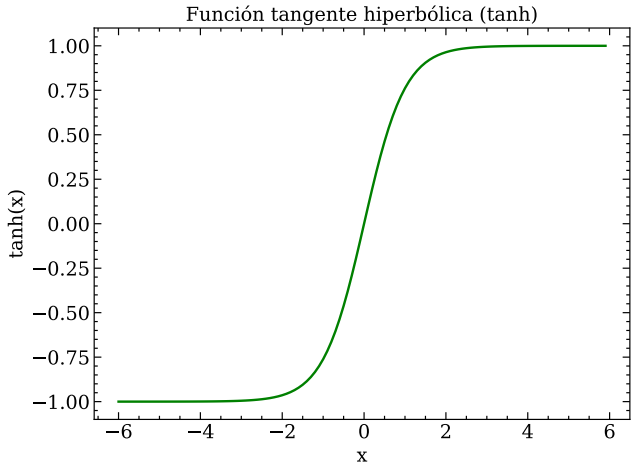


Figura 2.17: Función de activación tangente hiperbólica (tanh). Elaboración propia.

2.7.3. *Smooth Adaptive Activation Functions* (SAAF)

Usualmente, las funciones de activación que se utilizan dentro de una ANN son hiperparámetros, es decir, decisiones de diseño de la arquitectura de las redes. Esto puede provocar que se incurra en incrementar el sesgo de los resultados obtenidos. Las Funciones de Activación Adaptativas o *Adaptive Activation Functions* (AAF) [17] han sido utilizadas exitosamente con el fin de reducir dicho sesgo para tareas de clasificación. Sin embargo, al utilizarlas para tareas de regresión se ha visto un incremento en el sobreajuste de las redes a los datos de entrenamiento. Las Funciones de Activación Adaptativas Suaves o *Smooth Adaptive Activation Functions* (SAAF) [18] se proponen con el objetivo de reducir dicho sobreajuste. Estas consisten en polinomios definidos por partes, donde para cada parte se aprende un peso durante el entrenamiento. Los pesos aprendidos determinan la forma final de la función de activación. Un ejemplo de SAAF se muestra en la Figura 2.18, en la que los pesos aprendidos determinan la segunda derivada de la función de activación en los tramos respectivos.

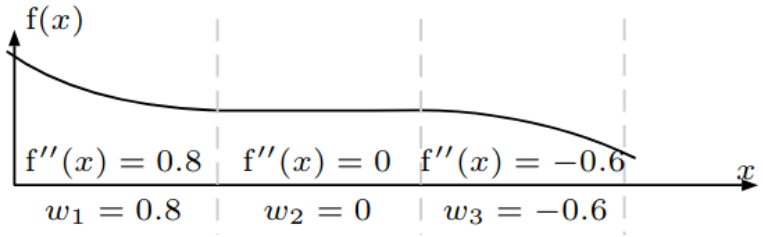


Figura 2.18: Ejemplo de función de activación SAAF [18].

2.8. Knowledge Distillation (KD)

En los últimos años, junto al creciente interés por el uso de algoritmos de *Deep Learning* tanto en el ámbito académico como fuera de este, ha aparecido el desafío relacionado con el despliegue de los modelos obtenidos en dispositivos portátiles y/o para aplicaciones en tiempo real. Al obtenerse, en general, mejores resultados con modelos de mayor escala, ha surgido la necesidad de comprimir los modelos para ampliar la cantidad de escenarios sobre los cuales es factible utilizarlos. Un enfoque de solución adoptado por los investigadores ha sido la Destilación de Conocimiento o *Knowledge Distillation* (KD). Esto consiste en aprender modelos pequeños, transfiriendo conocimiento adquirido en el entrenamiento de modelos grandes que se busca comprimir [19].

La arquitectura más utilizada para lograr lo anterior es llamada *Teacher-Student* y pretende hacer la transferencia del conocimiento desde el modelo complejo o *Teacher* hacia el modelo más simple o *Student*, siguiendo una estructura como la mostrada en la Figura 2.19:

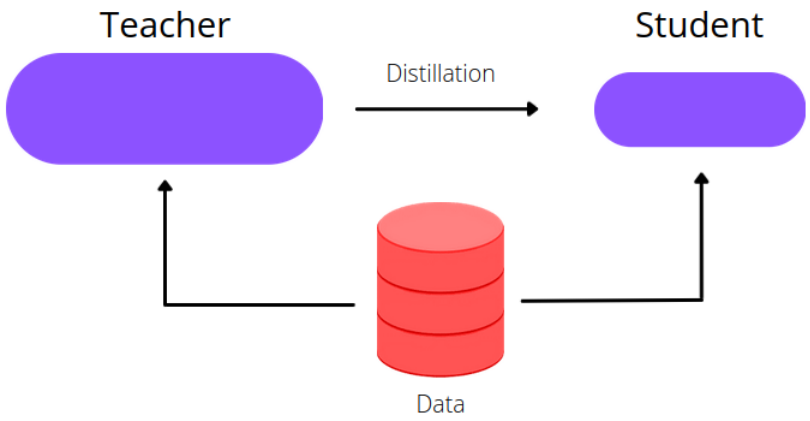


Figura 2.19: Esquema de arquitectura *Teacher-Student* para *Knowledge Distillation*. Elaboración Propia.

No obstante, la mayoría de los avances alcanzados en KD están concentrados en la resolución de tareas de clasificación, siendo tales avances no aplicables de forma directa a problemas de regresión como el que se aborda en este trabajo. Investigaciones recientes [20][21] han propuesto enfoques de solución, que buscan lidiar con el hecho de que los valores de salida no acotados en un modelo de regresión pueden ser perjudiciales para una correcta transferencia de conocimiento desde un modelo preentrenado a uno con menor cantidad de parámetros. Dos enfoques propuestos se explican a continuación:

2.8.1. Enfoque *Teacher Outlier Rejection* (TOR)

Un primer enfoque de KD para regresión es el presentado en [20] que sigue una arquitectura *Teacher-Student* y que, para lidiar con valores de salida no acotados, considera una etapa de detección de *outliers* o de valores no deseados para el entrenamiento del modelo

Student. Para lo anterior, utiliza como parte de la función de *loss* para el entrenamiento de este último modelo una función de *Teacher Outlier Rejection* (TOR), definida según la ecuación (2.18):

$$L_{TOR} = \begin{cases} ||R_s - t||_2^2, & \text{if } |t - R_t| \leq \varepsilon_{outlier} \\ f(R_s - R_t), & \text{if } |t - R_t| > \varepsilon_{outlier} \end{cases} \tag{2.18}$$

En la ecuación (6) t corresponde al valor *target* o *ground-truth* para regresión, R_t es la salida del modelo *Teacher* para el input asociado a t y R_s es la salida obtenida con el modelo *Student*. La función está diseñada para que cuando la salida del *Teacher* sea suficientemente cercana al valor *target*, según $\varepsilon_{outlier}$, se considere como *loss* el cuadrado de la norma L_2 de la diferencia entre la salida del *Student* y el valor objetivo. En caso contrario, se utiliza simplemente una función de la diferencia entre las salidas de los modelos *Teacher* y *Student*. Los autores proponen que el caso más simple consiste en utilizar $f(x) = 0$, pero experimentalmente logran buenos resultados con $f(x) = \sqrt{(}x)$. La función de costos utilizada, además de L_{TOR} utiliza la distancia L_1 entre R_t y R_s, llamada L_D, de manera que la *loss* usada es una ponderación entre ambas funciones definidas, con coeficientes c_{TOR} y c_D respectivamente, como se observa en la ecuación (2.19). Los resultados muestran que este método entrega mejores resultados que el caso en que el modelo *Student* es entrenado sin considerar KD, para un mismo tiempo de entrenamiento.

$$L = c_{TOR}L_{TOR} + c_D L_D. \tag{2.19}$$

La idea recién planteada muestra ser una metodología útil para KD, construyéndose sobre la concepción de que las Redes Neuronales Artificiales tienden a aprender en primer lugar los datos del conjunto de aprendizaje que están más limpios y luego aprenden a modelar los datos ruidosos [22].

2.8.2. Enfoque *Data-Free*

Un segundo enfoque de KD para regresión, presentado en [21], también se basa en una arquitectura *Teacher-Student* y propone además una solución para el caso en que no se dispone de los datos originales con que se entrenó el modelo, lo que es conocido como un enfoque *Data-Free*. Esto corresponde a la primera metodología de KD *Data-Free* para problemas de regresión. Para lo anterior, considera además de las redes *Teacher* (T) y *Student* (S) un modelo *Generator* (G) para la generación de datos de entrenamiento, asumiendo una distribución normal $\mathcal{N}(0, 1)$ para los datos. Para lograr generar datos útiles para el entrenamiento, las redes G y S se entrenan de manera adversaria en cada iteración según lo siguiente: al entrenar G, se congelan los parámetros de S y se optimiza con una función de *loss* (ecuación (2.20)) que busca maximizar al diferencia entre la salida de T y la de S. Lo anterior se esquematiza en la Figura 2.20:

$$L_G(z) = E_{x_g \sim G(z)}[-(T(x_g) - S(x_g))^2] \tag{2.20}$$

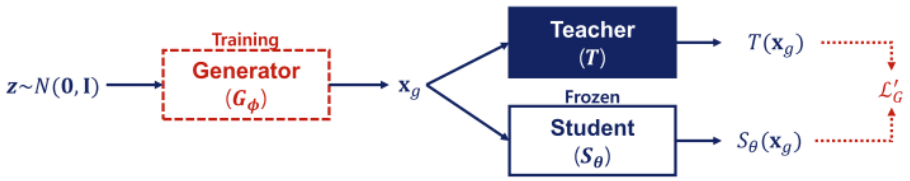


Figura 2.20: Entrenamiento de *Generation* en KD *Data-Free* [21].

Por su parte, S se entrena con la función de *loss* de la ecuación (2.21), diseñada para minimizar las diferencias entre la salida de T y S, según el esquema mostrado en la Figura 2.21. Cabe destacar que adicionalmente al entrenamiento con los datos generados por G, se entrena con datos muestreados desde una distribución normal $\mathcal{N}(0, 1)$, teniendo en cuenta que en este trabajo se asume dicha distribución para los datos de entrada.

$$L_S(x) = (T(x) - S(x))^2 \tag{2.21}$$

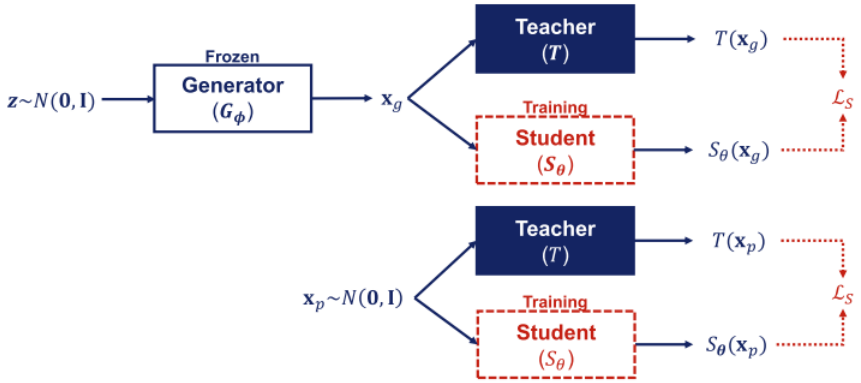


Figura 2.21: Entrenamiento de *Student* en KD *Data-Free* [21].

El entrenamiento conjunto de G y S, de manera adversaria, hace posible transferir conocimiento desde T a S y obtener buenos resultados, aún cuando los datos con que se entrenó a T originalmente no están disponibles para el entrenamiento de S.

Si bien las funciones de *loss* presentadas en las ecuaciones (2.20) y (2.21) constituyen la idea fundamental para el método de entrenamiento que se propone, los autores sugieren añadir términos de regularización en ambos casos, para que G no genere datos demasiado alejados de la distribución asumida y que S no aprenda desde ellos, lo que podría ser perjudicial para el entrenamiento de las redes involucradas. La *loss* utilizada para G se observa en la ecuación (2.22), donde se observan 3 términos, siendo el primero el ya presentado en la ecuación (2.20) y los dos restantes términos de regularización para los datos generados x_g y para la estimación de S sobre dichos datos, regulados por hiperparámetros β y γ respectivamente:

$$L'_G(z) = E_{x_g \sim G(z)}[-(T(x_g) - S(x_g))^2 + \beta ||x_g||^2 + \gamma S_\theta(x_g)^2]. \tag{2.22}$$

Para el entrenamiento de S, además de calcular la *loss* sobre los datos generados, se incorpora el cálculo de la *loss* sobre datos sampleados directamente desde una distribución normal $\mathcal{N}(0,1)$. Ambos términos se ponderan mediante un hiperparámetro α, según lo mostrado en la ecuación (2.23), considerando que m es la cantidad de datos utilizados para el cálculo de la función de *loss* y que $L_S(x)$ es la mostrada en la ecuación (2.21):

$$L'_S(z) = \frac{\alpha}{m}\sum_{i=1}^{m} E_{z\sim\mathcal{N}(0,1)}E_{x_g\sim G_\phi}[L_S(x_g)] + \frac{1-\alpha}{m}\sum_{i=1}^{m} E_{x_p\sim\mathcal{N}(0,1)}[L_S(x_p)]. \tag{2.23}$$

2.9. Métricas de Error para Señales

2.9.1. *Error-to-signal Ratio* (ESR)

El error llamado *Error-to-signal Ratio* se define como el error cuadrático dividido por la energía de la señal *target*. Fue introducido al problema de modelamiento de efectos de audio en [23], argumentando que sin aplicar la normalización por la energía, los fragmentos de señales del conjunto de entrenamiento que muestren una mayor energía predominarían en el valor del error, llevando a que no se valore adecuadamente el error existente en las señales de menor energía. Se define según la ecuación (2.24), donde y_i es la i-ésima muestra de la señal *target* y $\hat{y}_i$ es la muestra correspondiente pero de la señal estimada por el modelo o *output*. En la literatura, el ESR predomina en la evaluación de los modelos de una de las principales líneas de desarrollo, que en esta tesis es llamada línea B [23][24][25][26] y que es explicada en mayor detalle en la Sección 2.12.2 de este mismo capítulo.

$$\mathcal{E}_{ESR} = \frac{\sum_{i=0}^{N-1}|y_i - \hat{y}_i|^2}{\sum_{i=0}^{N-1}|y_i|^2} \tag{2.24}$$

2.9.2. *Mean Absolute Error* (MAE)

El *Mean Absolute Error* o Error Absoluto Medio, tal como su nombre sugiere, corresponde al promedio de los valores absolutos de las diferencias entre muestras de dos series de datos, que en este caso corresponden a dos señales de audio. De la misma forma que en la notación utilizada para el ESR, en la ecuación (2.25) y_i y $\hat{y}_i$ son la i-ésima muestra de la señal *target* y *output* respectivamente.

$$\mathcal{E}_{MAE} = \frac{1}{N}\sum_{i=0}^{N-1}|y_i - \hat{y}_i| \tag{2.25}$$

2.10. Enfoques de Modelamiento

En la literatura relacionada al modelamiento de efectos de audio y amplificadores pueden identificarse tres enfoques de desarrollo, correspondientes a los modelos *White-Box*, *Black-Box* y *Gray-Box* [1]. Cada uno de ellos se explica a continuación:

- *White-Box*: en este enfoque se modelan los efectos o amplificadores mediante el modelamiento de sus componentes electrónicos utilizando ecuaciones explícitas que buscan ajustarse a su comportamiento frente a las señales de entrada. Este enfoque no es generalizable, pues depende de la composición de cada uno de los dispositivos en términos de las interconexiones entre sus elementos y por lo mismo, requiere un alto nivel de conocimiento experto.
- *Black-Box*: se modela a partir de las señales de entrada y de salida, principalmente mediante algoritmos de ML. No requiere mayor conocimiento experto y permite que una única arquitectura de modelo generalice para señales obtenidas con distintos tipos de efectos y amplificadores.
- *Gray-Box*: este enfoque consiste en una combinación de los dos enfoques anteriores, incorporando conocimiento experto sobre los dispositivos con modelamiento explícito y algoritmos de aprendizaje.

Como se verá más adelante en este trabajo, los resultados que establecen el estado del arte en esta tarea se han alcanzado con modelos de tipo *Black-Box*, en particular, utilizando algoritmos de *Deep Learning*.

2.11. Tiempo Real

La latencia que introducen los modelos de efectos de audio al procesar datos de entrada es relevante a esta investigación, puesto que estos son de mayor utilidad cuando son capaces de funcionar en tiempo real. Se dice que un modelo funciona en tiempo real cuando es capaz de producir salidas para sus respectivas entradas, a medida que estas últimas evolucionan en el tiempo y bajo restricciones temporales [1]. Dichas restricciones dependen de la aplicación para la cual se utiliza el modelo, pero suelen ser tiempos reducidos.

El caso más común en el que se utilizan efectos de audio es al ejecutar instrumentos musicales. En particular, al tocar guitarra eléctrica con efectos de audio, es fundamental que el músico pueda monitorear la señal de audio de salida mientras toca, porque esto afecta la forma en que interpreta su instrumento [1], por lo que se requieren latencias muy bajas.

En [27] se encontró que para el caso específico de la guitarra, la restricción temporal de tiempo real requiere que la latencia sea menor a 12 ms. Sin embargo, esto no se puede monitorear directamente en este trabajo, ya que los modelos no son incrustados en un sistema que permita realizar pruebas con un instrumento directamente. Esto último quiere decir que

los modelos obtenidos no son incorporados, en esta tesis, como parte de un sistema que haga posible conectar un instrumento y evaluar tocándolo si el modelo es capaz de procesar la señal de audio de entrada en tiempo real.

Otro enfoque es propuesto en [25], donde se plantea que un modelo es capaz de funcionar en tiempo real si es capaz de procesar 1 segundo de audio de entrada en menos de 1 segundo. Considerando la restricción mencionada en el párrafo anterior, en este trabajo se considera esta definición, catalogando como modelo en tiempo real a aquellos que logren una latencia menor a 1000 ms, pero teniendo presente que mientras más baja sea la latencia lograda, más probable será cumplir con el criterio de los 12 ms.

2.12. Estado del Arte en emulación de efectos de audio mediante *Deep Learning*

El estado del arte para la tarea de modelar amplificadores y efectos de audio se ha alcanzado mediante la utilización de algoritmos de *Deep Learning*. En la literatura es posible identificar dos líneas de desarrollo de las cuales puede decirse que establecen el estado del arte según distintos criterios. De manera más específica, la línea de desarrollo que será llamada **línea A** [28][29][30][31] alcanza una mayor capacidad de generalización a distintos tipos de amplificadores y efectos, ya que ha desarrollado variantes de modelos para efectos no lineales y también variantes en el tiempo. Por su parte, la segunda línea, que será llamada **línea B** [23][24][25][26], logra modelos con tiempos de inferencia más bajos, lo que permite acercarse a implementaciones en tiempo real, pero únicamente para efectos no lineales. Es preciso enfatizar que la nomenclatura utilizada para referirse a los modelos del estado del arte como líneas A y B es propuesta en este trabajo, ya que en la literatura ambos modelos no han sido comparados previamente. En las secciones siguientes se explican ambas líneas de desarrollo en mayor detalle.

2.12.1. Línea A

Los modelos de la **línea A** fueron propuestos para modelar ecualizaciones [28], efectos de audio no lineales [29] y efectos de audio variantes en el tiempo [30][31], por lo que también son útiles para modelar amplificadores. Su funcionamiento se basa en una arquitectura de tipo *Encoder-Decoder* con una conexión residual o *skip connection*. El diseño de dicha arquitectura está pensado, a grandes rasgos, para que en la etapa de *encoding* se aprendan las alteraciones que los efectos y amplificadores tienen sobre la señal y luego, utilizando la conexión residual, para que en la etapa de *decoding* se apliquen dichas alteraciones sobre una representación de la señal original. Finalmente, se reconstruye la señal de salida. Esta estructura, para el caso en que se considera el modelamiento de efectos de audio variantes en el tiempo, será llamada en esta tesis como modelo A1. Este último puede observarse en la representación mostrada en la Figura 2.22:

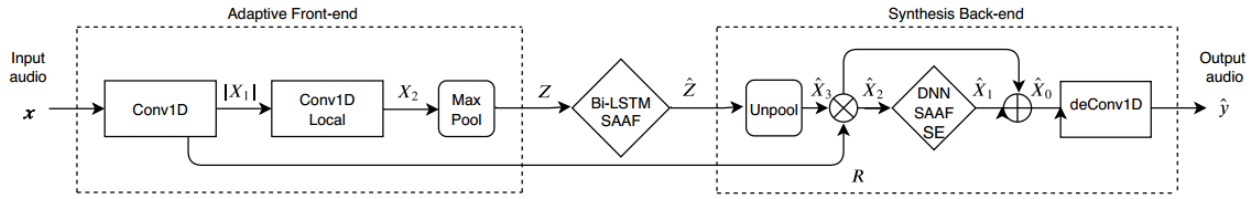


Figura 2.22: Arquitectura de modelos A1 de la línea A [30].

La etapa de *encoding* es llamada por los autores como *Adaptive Front-end* y está compuesta por una red CNN, una red CNN con *Locally Connected Convolutions* (CNN local) [7], lo que permite aprender un banco de filtros sobre la señal. Después aplica una operación de *max pooling*, que consiste en recorrer la señal resultante con una ventana móvil de tamaño fijo y conservar únicamente el valor máximo dentro de esa ventana. Esta primera cadena de procesamiento entrega una representación de la señal en un espacio latente que es entregado como entrada a una red RNN tipo LSTM con bidireccionalidad, es decir, que procesa la secuencia en ambas direcciones. Esta red pretende aprender las dependencias temporales de los efectos variantes en el tiempo. Además se incluyen funciones de activación tipo SAAF.

Luego, la etapa de *decoding* llamada *Synthesis Back-end* consta de una capa de *unpooling*, y luego calcula el producto entre la señal obtenida y la representación proveniente desde la conexión residual que es la salida de la primera red CNN. Sobre esta señal se aplica una red DNN con SAAF localmente conectadas, las que permiten aprender un "banco de funciones de activación" con el objetivo de modelar las no linealidades de los efectos y amplificadores. La señal obtenida se suma con una conexión residual proveniente del producto ya mencionado, para luego ser procesada por una capa de deconvolución, que no es entrenable, y consiste en una transposición de los filtros aprendidos en la primera CNN. Esto concluye la etapa de reconstrucción y genera una señal de salida de las mismas dimensiones que la señal de audio de entrada.

Una versión previa del modelo recién explicado fue propuesta en [29]. Este modelo fue diseñado para modelar específicamente efectos no lineales, sin modulaciones que incorporen variabilidad en el tiempo. La arquitectura de este modelo es en gran parte idéntica a la del modelo descrito en los párrafos anteriores. Las diferencias ocurren en la etapa de *Synthesis Back-end*, existiendo una conexión residual menos, y en el espacio latente entre las etapas de *encoding* y *decoding*, utilizando una red DNN en lugar de una LSTM bidireccional. Esta última diferencia es relevante, ya que el modelo A1, que utiliza una red recurrente bidireccional, introduce inevitablemente cierta latencia al procesamiento de una señal de entrada, ya que para procesar una muestra requiere disponer de muestras anteriores y posteriores a ella, pero permite modelar la variabilidad en el tiempo de los efectos de mejor manera. La versión previa que no incorpora variabilidad en el tiempo será llamada modelo A2 en este trabajo. La arquitectura de este modelo se muestra en el esquema de la Figura 2.23:

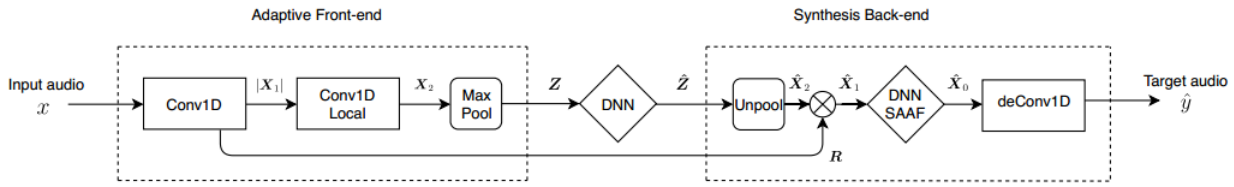


Figura 2.23: Arquitectura de modelos A2 de la línea A [29].

Cabe destacar que al estar esta investigación enfocada principalmente en efectos de audio no lineales, y también en reducir la latencia de los modelos en inferencia, se utiliza el modelo A2 como representante de la línea de desarrollo A para los experimentos a realizar.

2.12.2. Línea B

Los modelos de la **línea B** se proponen para modelar efectos no lineales [23][24][25], como amplificadores a tubos de vacío y efectos de distorsión. Su estructura es menos compleja que la de los modelos de la línea A y está compuesta, en su versión original (que en esta tesis es llamada modelo B1) [23], por una serie de operaciones de convolución de tipo *Dilated Convolution*, seguida por una red que los autores llaman *Post-Processing Module*, que consiste en un MLP de 3 capas. La arquitectura de este tipo de modelos se muestra en la Figura 2.24. El modelo B1 permite además incorporar parámetros de usuario para regular, por ejemplo, la ganancia de los efectos, lo cual se representa en la parte superior de la Figura 2.24. No obstante, al no ser esto relevante a esta investigación, no se entregan más detalles sobre su mecanismo de funcionamiento.

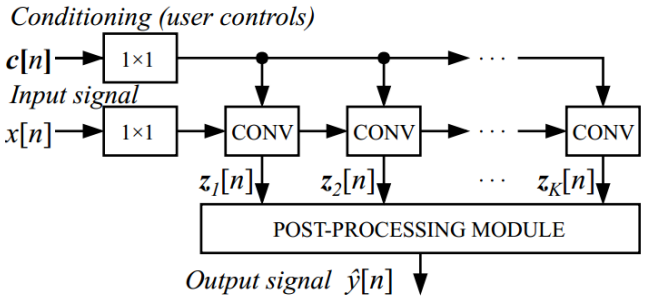


Figura 2.24: Arquitectura de modelos B1 de la línea B [23].

Dentro de esta misma línea de desarrollo, en [25] se propone una variación de la arquitectura que reemplaza las *Dilated Convolutions* por una red RNN tipo LSTM. La arquitectura resultante, llamada modelo B2 en este trabajo, sigue la estructura mostrada en la Figura 2.25, donde también puede notarse la inclusión de una conexión residual que agrega las muestras de entrada a las muestras de salida ya procesadas por las redes neuronales involucradas. Con estas modificaciones, los autores logran mejorar los tiempos de inferencia acercándose a una implementación en tiempo real, pero con el costo de una disminución en el desempeño del modelo. Teniendo en cuenta que este modelo presenta tiempos de inferencia más bajos, se define como representante de la línea de desarrollo B.

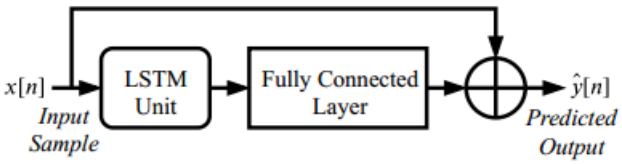


Figura 2.25: Arquitectura de modelos B2 de la línea B [25].

Capítulo 3

Metodología

3.1. Metodología Desarrollada

Teniendo en consideración la hipótesis y los objetivos planteados, en esta sección se detalla la metodología desarrollada en esta tesis. Se trabaja a partir de 2 modelos representantes del estado del arte, los modelos A y B, y se apunta a obtener modelos que sean más rápidos que ellos en inferencia, con el menor sacrificio posible en el desempeño en la tarea de modelar efectos de audio no lineales. En primer lugar, se implementan ambos modelos, para poder compararlos con los nuevos modelos entrenados. Esta comparación se realiza en base a métricas de error como ESR y MAE, y a tiempos de inferencia y cantidad de parámetros de los modelos.

La propuesta que se realiza en este tesis es que los nuevos modelos, que buscan mejorar los tiempos de inferencia, se entrenen como modelos *Student* utilizando técnicas de *Knowledge Distillation*, empleando los modelos del estado del arte como modelos *Teacher*. Para lo anterior se consideran dos escenarios: cuando se dispone de los datos con los que fueron entrenados los modelos *Teacher* y el caso contrario, en que dichos datos no están disponibles. Para el primer caso, se propone utilizar KD mediante *Teacher Outlier Rejection* o KD TOR (Sección 2.8.1). En el segundo caso, se aplica KD mediante el enfoque *Data-Free* o KD DF (Sección 2.8.2), considerando además de los modelos *Teacher* y *Student*, al modelo *Generator* que se encarga de la generación de datos para el entrenamiento. Para evaluar resultados se utilizan los mismos cuatro criterios de comparación propuestos para contrastar el desempeño de los modelos del estado del arte, además de la comparación visual de ejemplos de señales de salida para cada modelo con respecto a la señal *target* correspondiente. Todo lo anterior se realiza sobre un conjunto de datos de *testing*. Una vez obtenidos todos los modelos resultantes de los experimentos, que se explican en detalle más adelante, se comparan los desempeños según los criterios definidos, para determinar cuál modelo entrega un mejor rendimiento. Con esto, se espera obtener modelos que permitan confirmar la hipótesis de que sí es posible entrenar modelos que resulten ser rápidos en inferencia, a partir de los modelos del estado del arte para efectos de audio, con una baja disminución en el desempeño alcanzado.

En la Figura 3.1 se muestra un diagrama de flujo que representa la metodología recién explicada:

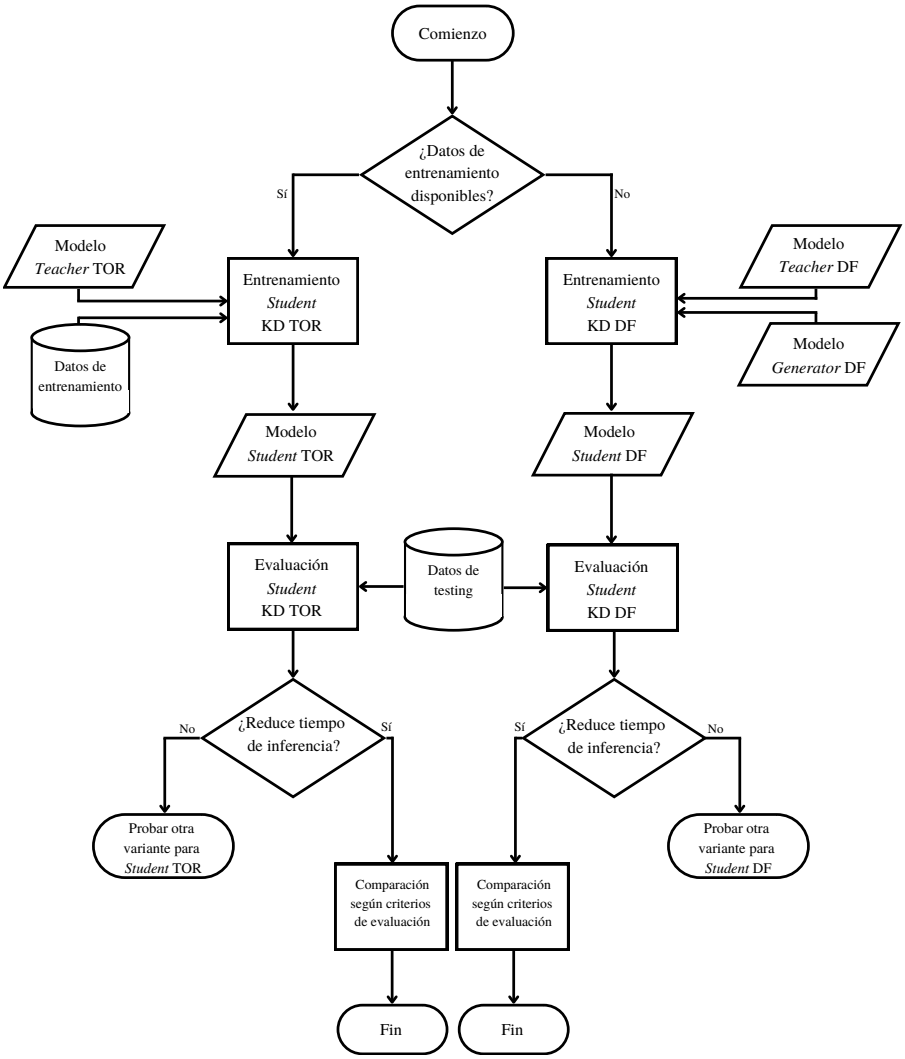


Figura 3.1: Diagrama de flujo de la metodología propuesta.

Cabe señalar que lo explicado en este párrafo corresponde a la metodología general propuesta y que aspectos más puntuales sobre los datos utilizados, la implementación de modelos, los experimentos realizados y lo que se espera mostrar con cada uno de ellos, se detallan a lo largo de este capítulo. Las siguientes secciones siguen la estructura explicada a continuación. Los datos utilizados para el entrenamiento y evaluación de modelos se describen en la Sección 3.2. Los experimentos realizados según la metodología propuesta se explican con mayor detalle en la Sección 3.3. En la Sección 3.4 se comentan aspectos sobre la implementación de tanto los modelos del estado del arte como de las técnicas de *Knowledge Distillation* propuestas para el entrenamiento de los nuevos modelos. En la Sección 3.5 se explican los cuatro criterios de evaluación considerados en la metodología y, por último, en la Sección 3.6 se presenta un resumen de la metodología propuesta en este capítulo.

3.2. Dataset

Para el entrenamiento y evaluación de los modelos propuestos se hace uso del mismo dataset utilizado en [25], compuesto por señales de audio sin procesar, que son las señales de *input* para los modelos, y por señales de audio procesadas por algún dispositivo de efecto o amplificación, que son las señales de salida deseada o *target*. En la literatura y en la jerga musical, también se habla de señales *dry* para hacer referencia a señales sin procesar y señales *wet* para aquellas procesadas. En los párrafos siguientes se describe el proceso mediante el cual fue generado el dataset utilizado.

Inicialmente se construye el dataset con señales de *input*, el cual es procesado por los dispositivos de interés, capturando en un computador las señales de salida mediante una interfaz de audio, construyendo así el dataset con señales *target*. Los archivos disponibles corresponden a un dispositivo de efectos de audio no lineal de tipo *fuzz* llamado *Big Muff Pi* de la marca *Electro-Harmonix* y a un amplificador a tubos de vacío *Blackstar HT-1*. Cabe destacar que el amplificador es grabado utilizando una configuración que introduce no linealidades de tipo *overdrive* a la señal de entrada. Ambos dispositivos se muestran en la Figura 3.2:

(a) *Electro-Harmonix Big Muff Pi* (b) Blackstar HT-1

Figura 3.2: Dispositivos capturados en señales *target*.

El dataset de *input* está conformado por 8 minutos y 10 segundos de audio capturados desde un bajo eléctrico y una guitarra eléctrica. Los audios utilizados son extraídos desde los datasets *IDMT-SMT-Bass-Single-Track* [32] y *IDMT-SMT-Guitar* [33] respectivamente. Ambos fueron publicados por el *Fraunhofer Institute for Digital Media Technology IDMT* y contienen archivos de audio en formato *.wav*, sampleados a 44100 kHz, correspondientes a muestras del sonido de cada instrumento. Estos archivos contienen sonidos monofónicos y polifónicos, es decir, con sólo una nota sonando y con dos o más sonando de forma simultánea. El archivo de audio utilizado se construye teniendo cuidado de incluir en proporciones similares señales de guitarra y bajo, y de sonidos monofónicos y polifónicos.

Una vez construídos los datasets de *input* y *target*, se procede a realizar una partición en conjuntos de entrenamiento, validación y test. De manera respectiva, cada conjunto corresponde a aproximadamente un 70 %, 17 % y 13 % del dataset. Según lo anterior, la duración en formato [mm:ss] de cada conjunto es la mostrada en la Tabla 3.1:

Tabla 3.1: Duración de cada conjunto del Dataset.

Conjunto	Duración [mm:ss]
Entrenamiento	5:42
Validación	1:24
Test	1:04

Un ejemplo de señales de *input* y *target* se observa en las Figuras 3.3 y 3.4. Estas señales pertenecen al dataset del dispositivo *Big Muff Pi*:

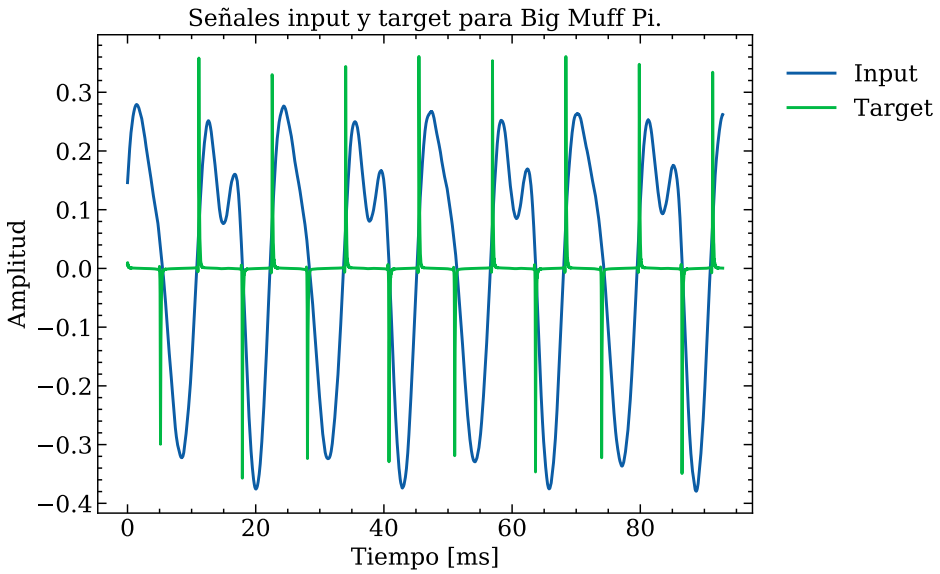


Figura 3.3: Ejemplo de señales *input* y *target* para *Big Muff Pi*.

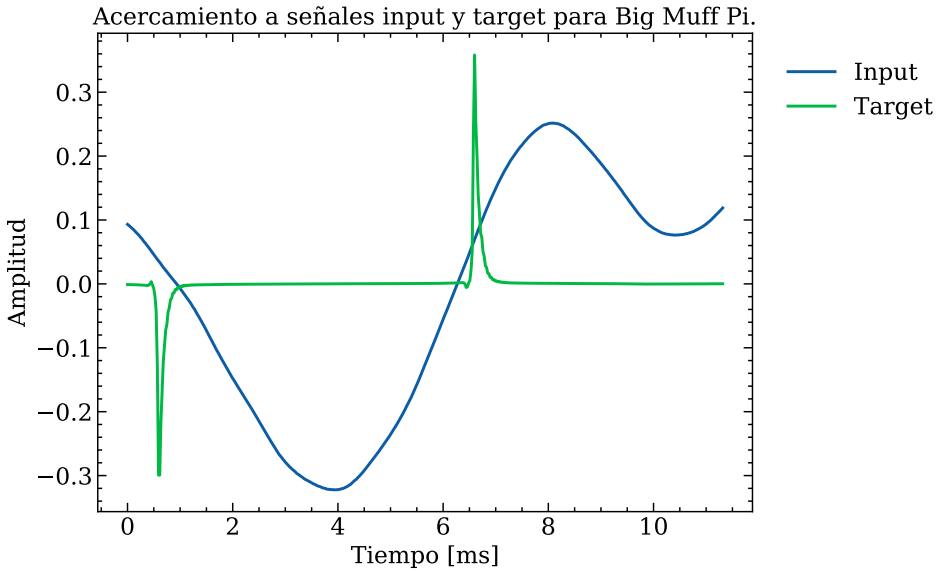


Figura 3.4: Acercamiento a ejemplo de señales *input* y *target* para *Big Muff Pi*.

3.3. Experimentos

En esta sección se enumeran los experimentos a realizar, describiendo el procedimiento a seguir y enunciando las interrogantes que se apunta a resolver con su ejecución. En líneas generales, los experimentos consisten en la aplicación de las técnicas de *Knowledge Distillation* introducidas en la Sección 2.8 de esta tesis, sobre los modelos del estado del arte, presentados en la Sección 2.12. Las técnicas de KD se aplican sobre los modelos representantes de las líneas A y B con el objetivo de obtener modelos que sean más rápidos en inferencia que los modelos completos, con la menor disminución posible en su desempeño. Se considera el escenario en que se cuenta con los mismos datos de entrenamiento con que se entrenó a los modelos originales, para el cual se aplica el enfoque *Teacher Outlier Rejection* (Sección 2.8.1), y el escenario en que los datos no están disponibles, para el que se usa el enfoque *Data-Free* (Sección 2.8.2). En general, todos los experimentos apuntan a responder la siguiente interrogante: ¿es posible obtener modelos más rápidos sin sacrificar el desempeño a partir del estado del arte?

En primer lugar, antes de los experimentos que involucran la aplicación de KD, se realiza una comparación de los desempeños de los modelos A y B sobre un mismo dataset, según criterios de evaluación enunciados más adelante en este capítulo. Esto se realiza debido a que no existe actualmente una comparación justa entre ambos métodos, ya que utilizan distintos conjuntos de datos y diferentes formas de evaluar el desempeño de los modelos obtenidos. Cabe destacar que para esto es necesario implementar ambos modelos utilizando un *framework* común, facilitando así la aplicación de los experimentos que se exponen más adelante en esta sección. Con este primer experimento se busca responder a: ¿cuál de los dos modelos representantes del estado del arte logra un mejor desempeño sobre un mismo dataset?

En cuanto a la aplicación de KD, se propone un conjunto de experimentos que es común para ambas líneas, en términos de lo que se busca probar. Esto consiste en aplicar ambas técnicas de KD sobre modelos que conserven la misma arquitectura, pero variando ciertos hiperparámetros de las redes neuronales, reduciendo la cantidad de parámetros de cada modelo. Se estudian las siguientes preguntas: ¿es factible modelar efectos de audio sin recurrir a arquitecturas complejas como las de la línea A?. Para que un modelo presente un buen desempeño, ¿es más influyente la cantidad de parámetros o la arquitectura utilizada?

En tercer lugar, teniendo en cuenta que el primer experimento compara el desempeño de los dos modelos del estado del arte, se estudia si es posible mejorar el desempeño del peor de los dos modelos aplicando KD desde el mejor de ellos.

Luego, con el objetivo de analizar qué ocurre al aplicar KD sobre arquitecturas aún más pequeñas y rápidas, se introduce una tercera arquitectura, también estudiada en [25], que es casi idéntica a la ya estudiada como modelo representante de la línea B, pero que en lugar de usar una celda recurrente LSTM, utiliza una celda de tipo GRU. Cabe destacar que esto se realiza únicamente desde el mejor modelo encontrado en el primer experimento propuesto hacia esta nueva arquitectura. Por simplicidad, se hará referencia a esta arquitectura como modelo C.

Por último, se selecciona el modelo que entrega mejores resultados, comparando todos los experimentos realizados, y sobre este se repite el entrenamiento usando diferentes semillas aleatorias para la inicialización de parámetros de las redes neuronales utilizadas. Esto se hace con el objetivo de reportar la media y desviación estándar de los errores obtenidos, para así evaluar si los resultados son estadísticamente consistentes o si corresponden a resultados fortuitos. En esta tesis, el entrenamiento del mejor modelo es repetido 5 veces.

Habiendo descrito los experimentos a realizar, a continuación se explican aspectos más específicos para las pruebas a realizar sobre cada modelo, en el caso en que se entrenan modelos con la misma arquitectura pero con una menor cantidad de parámetros:

3.3.1. Modelo A

Considerando que los modelos de esta línea son más lentos de entrenar que los de la línea B, se propone una única arquitectura reducida en cantidad de parámetros sobre la cual realizar las pruebas. Cabe señalar que al realizar esta modificación es esperable un decremento en el rendimiento del modelo, pero se hace énfasis en que esta tesis busca obtener modelos más rápidos en inferencia, intentando afectar lo menos posible el desempeño. La arquitectura reducida propuesta se obtiene llevando a cabo las siguientes modificaciones:

- Se reduce el tamaño del *kernel* aplicado en las capas convolucionales en un 50 %, de 64 a 32.
- Se reduce la cantidad de *break points* de la función SAAF en aproximadamente un 50 %, de 25 a 12.
- Se disminuye la cantidad de capas densas en el bloque DNN SAAF de 5 a 1.

La disminución efectiva de la cantidad de parámetros de la red se muestra más adelante en el Capítulo 4 de este documento. El modelo modificado será llamado modelo A*v2*.

3.3.2. Modelo B

En este caso, para conservar la arquitectura pero reducir la cantidad de parámetros se entrenan modelos que utilizan un tamaño de *hidden state* menor para la celda LSTM que forma parte de la estructura del modelo B. Las arquitecturas propuestas son:

- Arquitectura 1: Utilizando *hidden size* = 16 (25 % del tamaño original).
- Arquitectura 2: Utilizando *hidden size* = 8 (12,5 % del tamaño original).

El cálculo en la reducción de parámetros no es directo, porque al variar el *hidden size* de la celda LSTM también se modifica la dimensión de los datos de entrada a la capa *fully connected* del modelo. Al igual que en el caso anterior, el cambio efectivo en la cantidad de parámetros se presenta en el Capítulo 4.

3.3.3. Modelo C

Como se mencionó previamente, el modelo C sigue la misma arquitectura que el modelo B, pero reemplaza la celda LSTM por una celda GRU. Esta arquitectura se ha utilizado anteriormente en [25], obteniendo mejores resultados utilizando la variante con LSTM. Como esta tesis se enfoca en la reducción de tiempos de inferencia, la opción con GRU es interesante para las pruebas a realizar. Para los experimentos se consideran 3 variantes según el hiperparámetro *hidden size* utilizado. Al igual que para el modelo B, los valores empleados son 64, 16 y 8. Los experimentos descritos que involucran al modelo C son aplicados sobre estas 3 variantes.

3.4. Implementación

Previo al desarrollo de los experimentos explicados en la Sección 3.3, es necesario implementar los modelos del estado del arte para ambas líneas de desarrollo. De esta forma, es posible entrenarlos con su arquitectura original y evaluarlos para definir un punto de comparación para el resto de los modelos propuestos en esta investigación. Además, es necesario implementar las metodologías de KD a utilizar en los experimentos.

Cabe destacar que tanto la implementación de los modelos del estado del arte como la de los modelos propuestos en los experimentos se realiza en *Python*, utilizando la librería *Pytorch* [34] para el trabajo con redes neuronales. En lo que sigue se comentan detalles sobre la implementación de los modelos A y B y de las técnicas de KD TOR y DF:

3.4.1. Modelo A

El modelo de la línea A está compuesto, como ya se mencionó en el Capítulo 2.12, por un bloque de *encoding* llamado *Adaptive Front-end*, un espacio latente y un bloque de *decoding* llamado *Synthesis Back-end*. A su vez, cada uno de estos bloques puede descomponerse en subbloques, según lo mostrado en la Figura 2.23. En esta sección se hará referencia a los nombres de dichos subbloques para describir aspectos específicos como los hiperparámetros utilizados para implementar la red.

De los subbloques que componen la arquitectura, los llamados Conv1D Local, DNN SAAF, el espacio latente DNN y deConv1D requieren implementar capas personalizadas en *Pytorch*, ya que incorporan funcionalidades que no están disponibles directamente en la librería.

En particular, Conv1D Local implementa una capa de *Locally Connected Convolutions* (Sección 2.6.3.2). El espacio latente DNN requiere implementar una capa que aplica una capa *fully connected* sobre cada canal del tensor de entrada. DNN SAAF necesita una función de activación SAAF (Sección 2.7.3). Por último, el subbloque llamado deConv1D implementa una capa que depende de otra, siendo en este caso una capa no entrenable que aplica una convolución transpuesta de los pesos aprendidos por el subbloque Conv1D del comienzo del bloque *Adaptive Front-end* de la arquitectura. Para las implementaciones recién mencionadas,

se usa como referencia un código escrito utilizando la librería *Keras*, que fue facilitado por el primer autor de [29], Marco Martínez.

El modelo procesa los datos en ventanas de 4096 muestras de señal de audio. Las capas que aplican convoluciones utilizan 128 filtros con un tamaño de *kernel* igual a 64. Se entrena durante 1500 épocas con un *batch size* de 64 y un optimizador *Adam* con *learning rate* igual a 0.0001. Durante el entrenamiento, se evalúa el desempeño del modelo sobre el conjunto de validación, calculando el valor de la *loss* sobre dicho conjunto cada 6 épocas de entrenamiento.

En la Tabla 3.2 se muestran los cambios en las dimensiones del tensor de datos procesados por el modelo a medida que pasa por cada una de las capas que lo conforman. Los valores mostrados son los obtenidos utilizando los hiperparámetros descritos en el párrafo anterior:

Tabla 3.2: Dimensiones de datos a través de la arquitectura del Modelo A.

Bloque	**Subbloque**	**Dimensión (Muestras, *Batches*, Canales)**
Input	-	(4096, 64, 1)
Adaptive Front-end	Conv1D	(4096, 64, 128)
Adaptive Front-end	Conv1DLocal	(4096, 64, 128)
Adaptive Front-end	MaxPool	(64, 64, 128)
Espacio latente DNN	DNN	(64, 64, 128)
Synthesis Back-end	Unpool	(4096, 64, 128)
Synthesis Back-end	Producto residual	(4096, 64, 128)
Synthesis Back-end	DNN SAAF	(4096, 64, 128)
Synthesis Back-end	deConv1D	(4096, 64, 1)
Output	-	(4096, 64, 1)

3.4.2. Modelo B

El modelo representante de la línea B tiene una arquitectura compuesta una celda LSTM, un capa *fully connected* y una conexión residual. La celda LSTM se construye utilizando un tamaño de *hidden state* o *hidden size* de 64, y la capa *fully connected* realiza una transformación lineal que en este caso recibe como entrada 64 *features* y entrega 1. Esto último es porque el modelo procesa los datos de entrada muestra a muestra.

Si bien la arquitectura del modelo B parece simple de implementar, presenta particularidades en la forma de entrenar la red. Estas están relacionadas con el algoritmo de aprendizaje y con la función de *loss* utilizada.

Con respecto al algortimo de aprendizaje utilizando en el entrenamiento de la red, se utiliza una variante de la retropropagación del error o *Backpropagation* llamada *Truncated Backpropagation Through Time* (TBPTT), explicada en la Sección 2.6.4.3. Se utiliza un *step* de actualización de 2048, tal como proponen los autores [25].

En cuanto a la función de pérdida utilizada, esta considera 2 términos. El primero es el ESR, según la definición presentada en 2.9.1, entre las señales *output* y *target*. El segundo corresponde a un término propuesto en [25] para reducir un posible *offset* DC presente entre

las señales de *output* y *target*, llamado *DC error* o $\mathcal{E}_{DC}$. Este término se define según la ecuación (3.1), donde y_i es la señal *target* y $\hat{y}_i$ es la señal *output*:

$$\mathcal{E}_{DC} = \frac{|\frac{1}{N}\sum_{i=0}^{N-1}(y_i - \hat{y}_i)|^2}{\frac{1}{N}\sum_{i=0}^{N-1}|y_i|^2}. \tag{3.1}$$

Además, antes de calcular el valor de la función de pérdida, se aplica un filtro de "pre-énfasis" de ciertas frecuencias para asignarle más relevancia dentro del cálculo de las pérdidas. Se utiliza un filtro pasa altos de primer orden con la función de transferencia presentada en la ecuación (3.2):

$$H(Z) = 1 - 0{,}95z^{-1}. \tag{3.2}$$

Una vez aplicado el filtro anterior sobre las señales *output* y *target*, se calculan los términos asociados el error ESR y DC. La función de *loss* utilizada para el entrenamiento es la suma de ambos términos, tal como se muestra en la ecuación (A.1a):

$$\mathcal{E}_B = \mathcal{E}_{ESR} + \mathcal{E}_{DC}. \tag{3.3}$$

Para la implementación de TBPTT y de las funciones de *loss* recién explicadas, se utiliza como guía un código programado en *Pytorch* proporcionado por el primer autor de [25], Alex Wright.

Este modelo se entrena durante 700 épocas, con un *batch size* de 40 y un optimizador *Adam* con *learning rate* igual a 0.0005. Durante el entrenamiento, se evalúa el desempeño del modelo sobre el conjunto de validación, calculando el valor de la *loss* sobre dicho conjunto cada 6 épocas de entrenamiento.

3.4.3. *Knowledge Distillation* con enfoque *Teacher Outlier Rejection* (KD TOR)

Luego de realizar pruebas preliminares de entrenamiento de modelos aplicando la técnica de KD con *Teacher Outlier Rejection*, con la función de *loss* propuesta por los autores, se opta por realizar algunas modificaciones. Esto se decide porque en dichas pruebas no se apreciaba una disminución de la función de *loss* a través de las épocas de entrenamiento. En particular, se reemplaza el primer caso mostrado en la ecuación (2.18) por la aplicación de la función de *loss* utilizada en los modelos A y B según corresponda. Además, en lugar de utilizar $f(x) = \sqrt{x}$ para el segundo caso de la ecuación, se usa $f(x) = 0$, tal como los autores proponen para mayor simplicidad. De esta forma, la *loss* utilizada en KD TOR es la presentada en la ecuación (3.4):

$$L_{TOR} = \begin{cases} \mathcal{E}(R_s, t), & \text{if } |t - R_t| \leq \varepsilon_{outlier} \\ 0, & \text{if } |t - R_t| > \varepsilon_{outlier} \end{cases} \tag{3.4}$$

En la ecuación anterior, $\mathcal{E} = \mathcal{E}_{MAE}$ (ecuación (2.25)) para el modelo A y $\mathcal{E} = \mathcal{E}_B$ (ecuación (A.1a)) para el modelo B.

3.4.4. *Knowledge Distillation* con enfoque *Data-Free* (KD DF)

Teniendo en cuenta que el enfoque de KD *Data-Free* o DF asume que los datos de entrada tienen una distribución normal $\mathcal{N}(0, 1)$, para realizar una comparación más justa con los modelos del estado del arte, estos se vuelven a entrenar considerando una estandarización de los datos de entrenamiento, particularmente para los datos de *input*. Es decir, para cada dato de entrada x se utiliza su versión z estandarizada según la ecuación (3.5), donde μ_{train} y σ_{train} son el promedio y la desviación estándar de los datos de entrenamiento, respectivamente:

$$z = \frac{x - \mu_{train}}{\sigma_{train}}. \tag{3.5}$$

Es necesario destacar que, si bien para este caso fue posible reentrenar los modelos del estado del arte con datos estandarizados, la aplicación de KD DF realiza el supuesto de que los modelos *teacher* fueron entrenados de dicha manera, lo cual no se cumple necesariamente al trabajar con modelos ya entrenados y que no tengan disponibles los datos originales de entrenamiento.

3.5. Evaluación de Modelos

Los modelos del estado del arte implementados y los modelos entrenados en los experimentos propuestos son comparados en base a los criterios presentados en esta sección. Estos corresponden a 2 métricas de error, *Error-to-signal Ratio* y *Mean Absolute Error*, de las señales de *output* con respecto a las señales *target* de los datos del conjunto de test. Además, se evalúa según los tiempos de inferencia y cantidad de parámetros de cada modelo. Cabe señalar que además de estos criterios, en el análisis comparativo se contempla explorar visualmente las señales obtenidas y hacer pruebas de escucha, para poder detectar casos de borde en que las métricas obtenidas puedan tomar buenos valores sin que eso se refleje en las señales de audio obtenidas. Los 4 criterios se explican en mayor detalle a continuación:

3.5.1. Métricas de Error

Se utilizan 2 criterios de evaluación basados en métricas de error. El primero es *Error-to-signal Ratio* o ESR. El segundo corresponde al *Mean Absolute Error* o MAE. Ambos fueron explicados previamente en el Capítulo 2, en las Secciones 2.9.1 y 2.9.2 respectivamente.

3.5.2. Tiempo de Inferencia

Los modelos también son comparados en base a su tiempo de inferencia, ya que para el problema tratado es relevante que los modelos introduzcan la menor latencia posible al procesar los datos de entrada. Cabe recordar que el objetivo general de esta tesis consiste en reducir los tiempos de inferencia de los modelos del estado del arte, por lo que es fundamental contar con este criterio de comparación.

Para calcular los tiempos de inferencia, se hace uso de la herramienta *Profiler* provista por la librería *Pytorch*, la cual se utiliza para medir tiempos de ejecución y uso de memoria en los modelos implementados con dicha librería. El cálculo se realiza en la CPU provista por *Google Colaboratory*, de marca *Intel* y modelo *Xeon*, con frecuencia de reloj de 2.00 GHz. Se mide el tiempo que tardan los modelos en procesar 1 segundo de audio y se registran 10 ejecuciones, reportando la media y la desviación estándar obtenidas para cada caso.

3.5.3. Cantidad de Parámetros

El último criterio a considerar es la cantidad de parámetros entrenables de cada modelo. Para esto se registrará directamente la cantidad reportada por la librería *Pytorch*. Este criterio se tendrá en cuenta para analizar la relevancia de la cantidad de parámetros de un modelo frente a otros factores, como la arquitectura elegida.

3.6. Resumen de la Metodología

Teniendo en consideración todo lo comentado en el presente capítulo, se presenta una síntesis de la metodología a seguir en esta investigación. Se trabaja a partir de 2 modelos representantes del estado del arte, los modelos A y B, y se apunta a obtener modelos que sean más rápidos que ellos en inferencia, con el menor sacrificio posible en el desempeño en la tarea de modelar efectos de audio. En primer lugar, se implementan ambos modelos, para poder compararlos con los nuevos modelos entrenados. Esta comparación se realiza en base a métricas de error como ESR y MAE, y a tiempos de inferencia y cantidad de parámetros de los modelos. Los entrenamientos se realizan utilizando los datos descritos en la Sección 3.2.

Los nuevos modelos se entrenan utilizando técnicas de KD, considerando dos escenarios: KD mediante TOR cuando se dispone de los datos de entrenamiento y KD mediante DF en caso contrario. Los experimentos a realizar se enumeran a continuación, teniendo en cuenta que cada uno contempla un análisis de resultados según las interrogantes planteadas en la Sección 3.3:

- Comparación Modelos A y B.
  - Se compara desempeño de ambos modelos sobre un mismo dataset.
- KD sobre Modelo A.

  - Aplicación de KD TOR desde modelo A a modelo A*v2* (Sección 3.3.1).
  - Comparación con modelo A*v2* entrenado sin KD.
  - Aplicación de KD DF desde modelo A estandarizado a modelo A*v2*.
  - Comparación con modelo A*v2* entrenado con estandarización y sin KD.
- KD sobre Modelo B.
  - Aplicación de KD TOR desde modelo B a 2 variaciones del modelo (Sección 3.3.2).
  - Comparación con 2 variantes entrenadas sin KD.
  - Aplicación de KD DF desde modelo B estandarizado a 2 variaciones del modelo.
  - Comparación con 2 variantes entrenadas con estandarización y sin KD.
- KD entre Modelos A y B.
  - Aplicación de KD TOR desde el mejor modelo al peor, según el primer experimento.
  - Aplicación de KD DF desde el mejor modelo al peor, pero con datos estandarizados.
- KD hacia Modelo C.
  - Aplicación de KD TOR desde mejor modelo (según primer experimento) a 3 variantes del modelo C (Sección 3.3.3).
  - Comparación con 3 variantes entrenadas sin KD.
  - Aplicación de KD DF desde mejor modelo estandarizado a 3 variantes de C.
  - Comparación con 3 variantes entrenadas con estandarización y sin KD.
- Validación del Mejor Modelo
  - Repetir entrenamiento de modelo 5 veces y reportar media y desviación estándar de los errores obtenidos.

Capítulo 4

Resultados y Análisis

En esta sección se presentan los resultados obtenidos en los distintos experimentos propuestos en el Capítulo 3. Si bien el dataset utilizado en este trabajo cuenta con pares de señales *input/target* para 2 dispositivos, los resultados presentados en este capítulo sólo contemplan a uno de ellos. Se usan los datos del conjunto de *test* asociados al pedal *Electro-Harmonix Big Muff Pi*, de efecto no lineal de tipo *fuzz*, porque entre los 2 dispositivos disponibles este produce un efecto que incorpora no linealidades más acentuadas en la señal, lo que provoca que este efecto sea más difícil de modelar. De esta manera, las comparaciones se realizan sobre un caso no trivial en cuanto al modelamiento de las señales.

En este capítulo, cada una de las secciones corresponde a un ítem en la enumeración de experimentos presentada en la Sección 3.6. Cabe señalar que las funciones de *loss* obtenidas en cada entrenamiento realizado, se muestran en el Apéndice 5. Los resultados obtenidos en cada caso, junto al análisis respectivo, se presentan en lo que sigue:

4.1. Comparación Modelos A y B

Los resultados obtenidos para la comparación de desempeño de los modelos representantes del estado del arte, según los criterios de evaluación definidos, se muestran el la Tabla 4.1. En esta tabla y en las siguientes, se destaca en negrita los mejores valores obtenidos para cada criterio de evaluación:

Tabla 4.1: Evaluación de Modelos A y B.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
A	0.0099	0.5818	1202 ± 76	72,065
B	**0.0066**	**0.2261**	**311±42**	**17,217**

En la Tabla 4.1 puede observarse que para el conjunto de datos utilizados en la evaluación de los modelos, el modelo B muestra un mejor desempeño que el modelo A, además de presentar un tiempo de inferencia más bajo y una menor cantidad de parámetros.

En particular, los errores MAE y ESR son un 33.3 % y un 61.1 % más bajos en el modelo B con respecto al A. El modelo B es en promedio 4 veces más rápido que el A, con un 23.9 % de sus parámetros.

Si bien el modelo B incluye en su arquitectura una red LSTM, la cual fue diseñada para modelar dependencias temporales, el modelo A fue propuesto incorporando intuiciones adquiridas desde el conocimiento sobre el funcionamiento de los efectos de audio, incluyendo etapas de filtrado de la señal antes de introducir no linealidades. Además el modelo A posee más de 4 veces más parámetros que B. Por lo anterior, una hipótesis razonable sería esperar que el modelo A presentara un mejor desempeño, lo cual no se cumple según las pruebas realizadas.

Teniendo en cuenta que, hasta donde se ha estudiado, la recién presentada es la primera comparación formal entre ambas arquitecturas frente a un mismo dataset, el resultado obtenido es significativo, puesto que sugiere que para modelar efectos no lineales es más relevante utilizar una arquitectura especializada en modelar dependencias temporales que una arquitectura más compleja inspirada en las implementaciones en circuitos electrónicos de los efectos modelados, aún cuando la primera aprenda menos parámetros que la segunda.

En las Figuras 4.1 y 4.2 se muestran ejemplos de las señales obtenidas como *output* de cada uno de los modelos, comparando también con la señal *target*. En el primer caso, se muestra una ventana de 4096 muestras, equivalentes a 93 milisegundos. En el segundo caso, se muestra un fragmento de 100 muestras de la ventana anterior, que corresponde a poco más de 2 milisegundos:

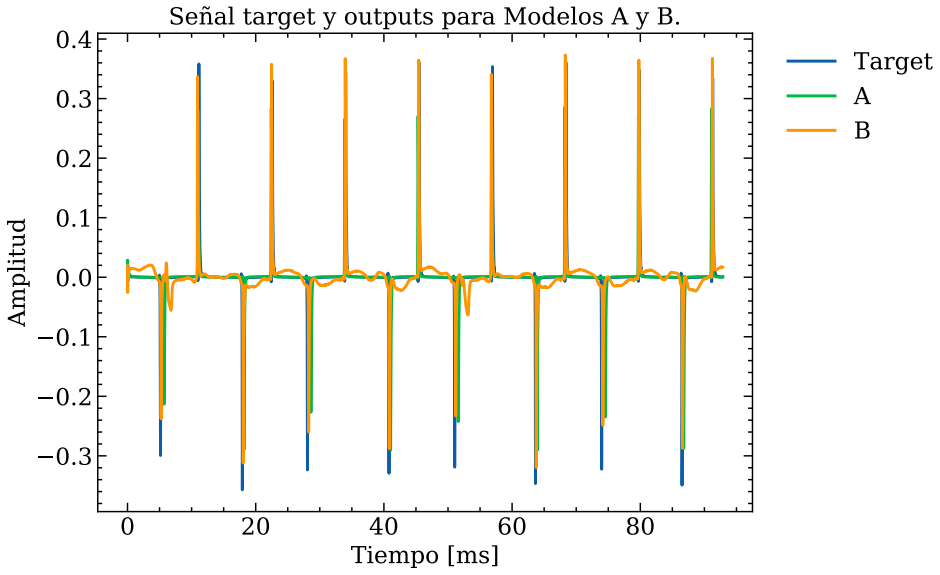


Figura 4.1: Señales *target* y *output* para comparación de Modelos A y B.

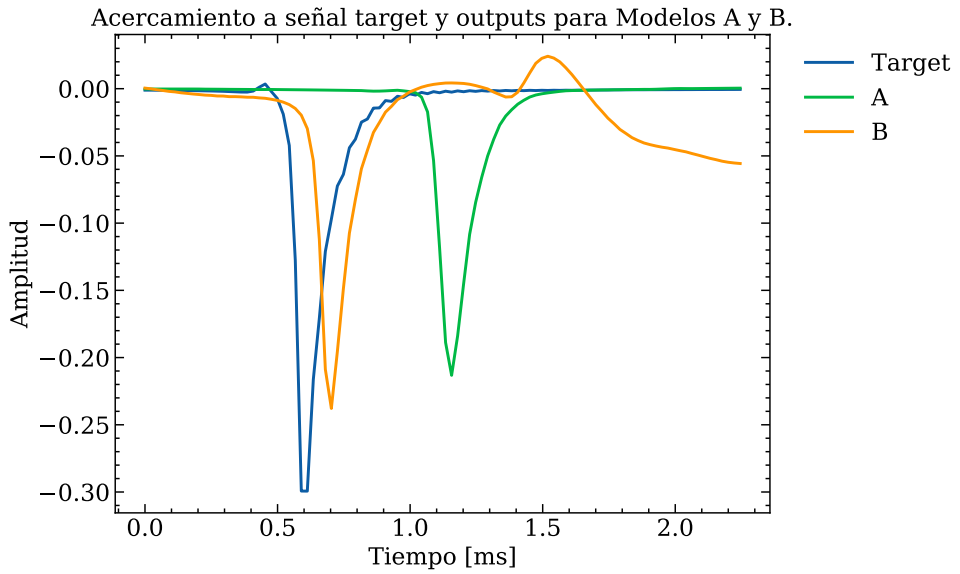


Figura 4.2: Acercamiento a señales *target* y *output* para comparación de Modelos A y B.

En la Figura 4.1 se aprecia que ambos modelos se ajustan de buena manera a la señal *target*, pero con ciertas diferencias. El modelo A se comporta mejor que el B fuera de los *peaks* de señal observados, pero en ellos el modelo B se acerca más a modelar correctamente su amplitud. Aún cuando en los valles de la señal el modelo B tiende a fallar, la diferencia de amplitud en dichas zonas con respecto a la señal objetivo es menor que la que existe entre la salida del modelo A y los *peaks*, lo que se traduce en que las métricas de error de B sean menores que las de A.

Lo recién explicado se visualiza con mayor claridad en la Figura 4.2. También se observa que ambos modelos introducen una fracción de milisegundo de retraso en la aparición del *peak* y que ninguno de los dos modela exactamente la forma de onda del recorte producido por la saturación del efecto de *fuzz*. No obstante, esto último difícilmente es perceptible o significativo al realizar pruebas de escucha, notando que la saturación de cada *peak* particular ocurre a una escala temporal muy pequeña.

4.2. KD sobre Modelo A

A partir de esta sección se presentan los resultados para la aplicación de las técnicas de KD sobre los modelos. En primer lugar, se expone lo obtenido para el empleo de KD TOR y DF sobre el Modelo A:

4.2.1. TOR sobre Modelo A

Lo obtenido para la aplicación de KD TOR en el modelo A se muestra en la Tabla 4.2, donde se compara el desempeño del modelo A, su arquitectura modificada A*v2* y este último

entrenado utilizando TOR, referido como TOR A*v2*. Se destacan en negrita los mejores valores para cada criterio de evaluación, pero esta vez sin considerar el modelo de referencia de la primera fila de la tabla:

Tabla 4.2: Evaluación de aplicación de KD TOR sobre Modelo A.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
A	0.0099	0.5818	1202 ± 76	72,065
A*v2*	**0.0129**	**0.6017**	**831±28**	**34,113**
TOR A*v2*	0.0143	0.7019	833 ± 41	**34,113**

De la Tabla 4.2 se desprende que el modelo TOR A*v2* logra aprender desde el modelo A, ya que las métricas de error no son demasiado lejanas a las obtenidas con el modelo A*v2* entrenado desde cero. Sin embargo, no logra superarlo en *performance*. Esto va en contra de lo esperado al aplicar KD, ya que se supone que esto signifique un beneficio en términos de obtener un mejor desempeño que lo que se lograría con la misma arquitectura sin considerar esta técnica. Antes de enunciar análisis generales sobre la aplicación de este método, se evaluará su empleo sobre el resto de los experimentos.

En las Figuras 4.3 y 4.4 se muestran ejemplos de señales *target* y *output* obtenidas desde los modelos entrenados en este experimento:

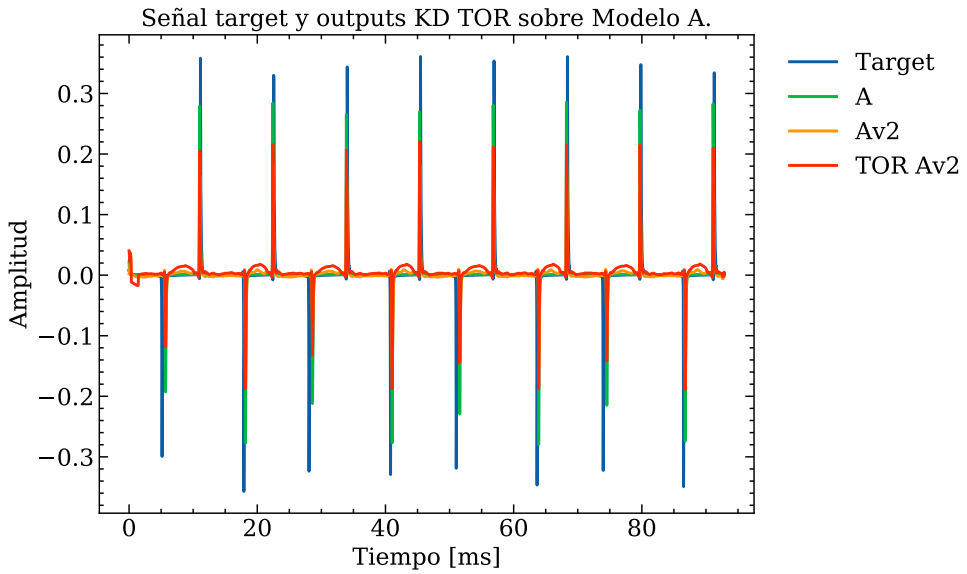


Figura 4.3: Señales *target* y *output* para aplicación de KD TOR sobre Modelo A.

A partir de lo mostrado en la Figura 4.3 puede decirse que A*v2* y TOR A*v2* parecen ajustarse a la señal *target*, pero con notables diferencias en la amplitud de los *peaks* provocados por el efecto *fuzz*. Al realizar pruebas de escucha de las señales resultantes, las diferencias más notorias entre los casos evaluados ocurre en el volumen del sonido, lo cual está directamente relacionado con la amplitud de la señal. Para este trabajo, esta propiedad del sonido también debe ser modelada correctamente por los modelos candidatos. Considerando esto, y

observando la Figura, el modelo TOR A*v2* puede producir un sonido ligeramente más fiel al *target* que A*v2*, pero es poco preciso en los valles de la señal, lo que aumenta sus métricas de error. Aún así, la aplicación de TOR no cumple su objetivo en este caso.

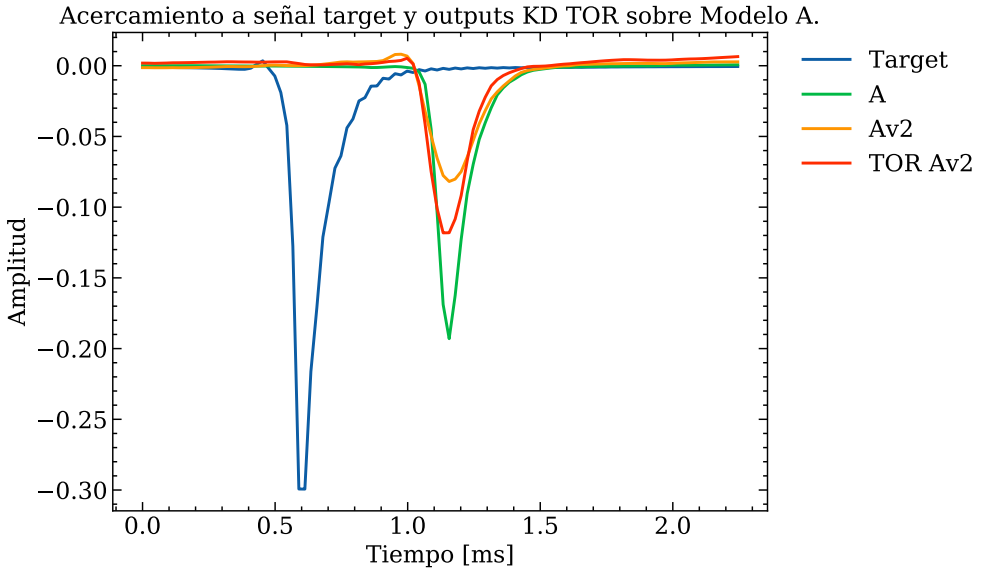


Figura 4.4: Acercamiento a señales *target* y *output* para aplicación de KD TOR sobre Modelo A.

En la Figura 4.4 se puede apreciar con mayor detalle que el modelo TOR A*v2* entrega una señal de amplitud más parecida a la de A y, por lo tanto, a la de la señal *target* que el modelo A*v2*, pero es menos preciso en los valles. Esto se condice con lo comentado en el párrafo anterior.

4.2.2. DF sobre Modelo A

La Tabla 4.3 presenta los resultados de la evaluación de modelos al aplicar KD DF sobre el modelo A. En ella se muestran los criterios de evaluación para los modelos A std, A*v2* std y DF A*v2*, donde "std" hace referencia a que los modelos fueron entrenados con datos de entrada estandarizados, para cumplir con los supuestos de KD DF. DF A*v2* es el modelo A*v2* entrenado utilizando el ya mencionado método de KD. Cabe mencionar que, si bien los modelos son entrenados con datos estandarizados, sus *outputs* no lo están, por lo que los errores obtenidos son comparables con los mostrados en el resto de los experimentos.

Tabla 4.3: Evaluación de aplicación de KD DF sobre Modelo A.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
A std	0.0094	0.5399	722 ± 21	72,065
A*v2* std	0.0491	4.3987	686 ± 28	**34,113**
DF A*v2*	**0.0168**	**0.8491**	**685±4**	**34,113**

En este caso se observa que DF A*v2* entrega mejores métricas de error que A*v2*, pero como se propone en la Sección 3.5, es necesario hacer una revisión y comparación de las señales obtenidas y/o pruebas de escucha.

Ejemplos de las señales obtenidas se muestran en las Figuras 4.5 y 4.6, para la misma ventana usada en los experimentos anteriores:

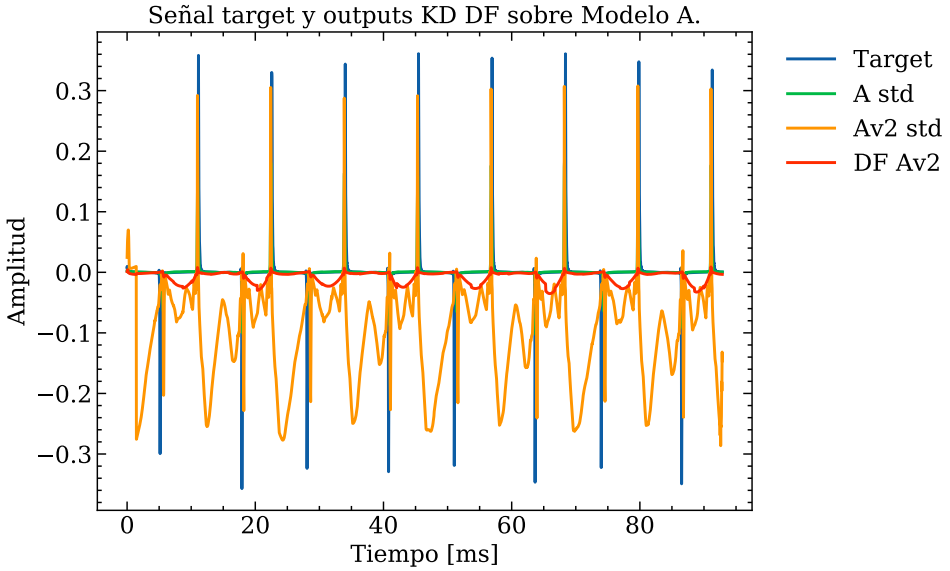


Figura 4.5: Señales *target* y *output* para aplicación de KD DF sobre Modelo A.

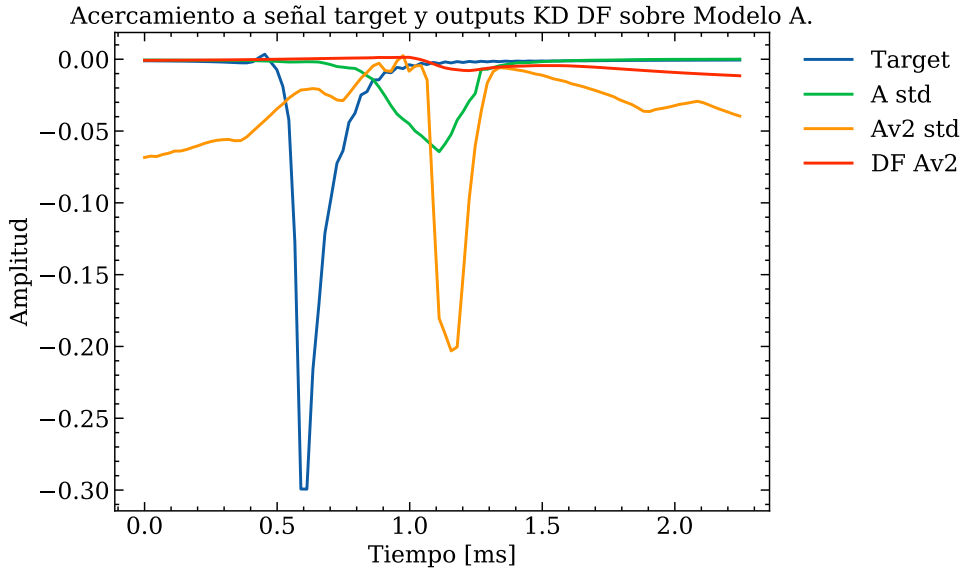


Figura 4.6: Acercamiento a señales *target* y *output* para aplicación de KD DF sobre Modelo A.

En ambas figuras es notorio que el modelo DF A*v2* no aprendió a modelar el efecto. Su señal de *output* no presenta los *peaks* característicos de la señal *target*, y sólo toma valores cercanos a 0. Por su parte, A*v2* logra hacer un buen *matching* en los valores positivos de la señal, pero no consigue lo mismo para los valores negativos. De cualquier forma, es directo verificar que en pruebas de escucha el modelo A*v2* std tiene un mejor desempeño que DF A*v2*. Esto permite afirmar que la aplicación de KD DF no cumple su finalidad.

Con respecto a los posibles motivos del mal desempeño de este método, se propone que puede deberse a una configuración incorrecta de ciertos hiperparámetros utilizados específicamente en el cálculo de las funciones de *loss* de las ecuaciones 2.22 y 2.23. Al observar la función de *loss* de la red *Student* en función de las épocas de entrenamiento del modelo DF A*v2* (ver Apéndice A.2, Figura A.3), es claro que el proceso de entrenamiento no está siendo llevado a cabo de manera propicia, pues esta muestra un comportamiento inestable y ruidoso según avanzan las épocas.

Para los experimentos realizados, los hiperparámetros α, β y γ se configuran según lo propuesto por los autores del método [21]. Estos valores son evaluados en la resolución de problemas más simples que el abordado en este trabajo, por lo que puede que sea necesaria una puesta a punto más cuidadosa de los hiperparámetros para incrementar el desempeño del modelo. Teniendo en cuenta que este modelo tarda más de 20 horas en entrenar y que se cuenta con recursos limitados, la exploración de otros hiperparámetros se propone como trabajo futuro.

De la misma forma que para la aplicación de KD TOR sobre el modelo A, se evaluará la aplicación de KD DF en el resto de los experimentos para emitir un análisis general sobre este método.

4.3. KD sobre Modelo B

4.3.1. TOR sobre Modelo B

La aplicación de técnicas de KD TOR sobre el Modelo B entrega los resultados mostrados en la Tabla 4.4, mostrando ejemplos de las señales obtenidas en las Figuras 4.7 y 4.8, para los mismos casos sobre los que se evaluó el modelo A.

Tabla 4.4: Evaluación de aplicación de KD TOR sobre Modelo B.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
B	0.0066	0.2261	311 ± 42	17,217
B con *hs*=16	0.0113	0.3936	292 ± 51	1,233
B con *hs*=8	0.0103	0.4161	277 ± 28	**361**
TOR B con *hs*=16	**0.0091**	**0.3128**	290 ± 28	1,233
TOR B con *hs*=8	0.0102	0.3595	**272±29**	**361**

Observando la Tabla 4.4 y las figuras, puede afirmarse que esta vez la aplicación de KD TOR parece haber sido efectiva, ya que los modelos entrenados con TOR muestran métricas de error más bajas que las obtenidas con sus modelos equivalentes entrenados desde cero y todas las señales mostradas parecen modelar en mayor o menor medida los *peaks* del *target*, no existiendo casos como el de DF sobre el modelo A.

El modelo TOR B con *hidden size*=16 logra errores menores que los demás modelos en comparación, pero cabe resaltar que la versión que utiliza un *hidden size* de 8 alcanza errores similares, menores que las variantes de B sin TOR, pero con un tiempo de inferencia de aproximadamente 18 ms menos. Por ello, este último es un buen candidato a ser el mejor modelo en esta comparación.

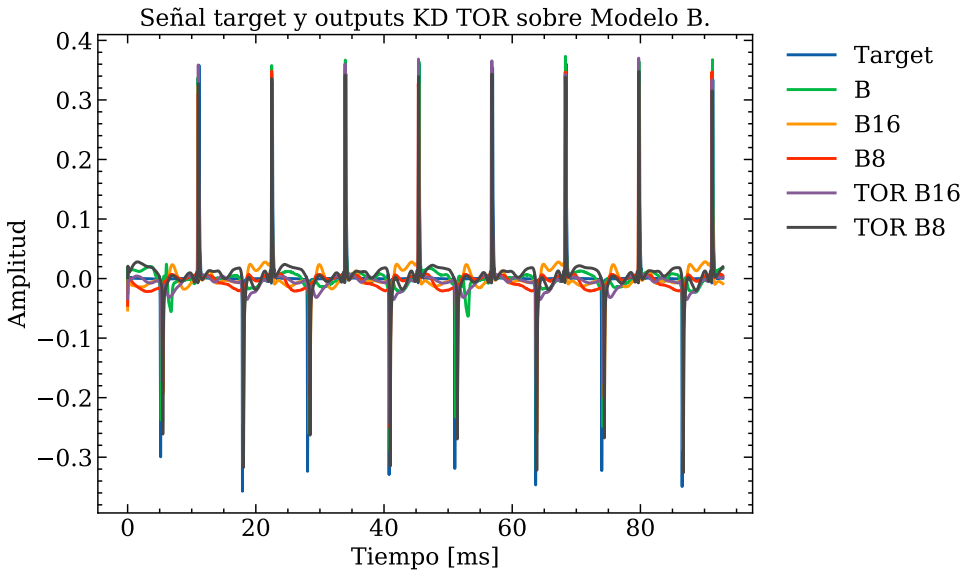


Figura 4.7: Señales *target* y *output* para aplicación de KD TOR sobre Modelo B.

En este caso, los resultados evidencian que sí es posible entrenar modelos utilizando la metodología TOR y que además es viable obtener mejoras en el desempeño de los modelos. En particular, para el modelo con menor error se alcanza una disminución de aproximadamente un 20 % tanto en el error ESR como en el MAE, con respecto a la misma arquitectura entrenada desde cero.

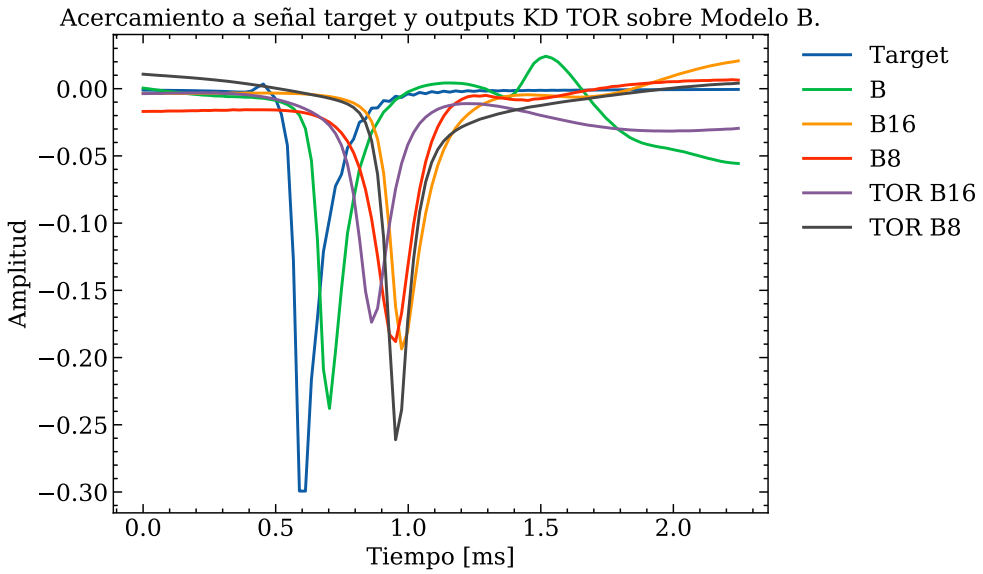


Figura 4.8: Acercamiento a señales *target* y *output* para aplicación de KD TOR sobre Modelo B.

4.3.2. DF sobre Modelo B

Para el empleo de KD DF sobre el modelo B, los valores obtenidos para los criterios de evaluación se observan en la Tabla 4.5. En ellos se aprecia que la técnica utilizada no fue efectiva en la reducción de las métricas de error en ninguno de los dos casos evaluados.

Tabla 4.5: Evaluación de aplicación de KD DF sobre Modelo B.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
B std	0.0131	0.5665	294 ± 35	17,217
B std con hs=16	**0.0139**	**0.5799**	277 ± 37	1,233
B std con hs=8	0.0147	0.6558	267 ± 30	**361**
DF B con hs=16	0.0199	0.8709	272 ± 36	1,233
DF B con hs=8	0.0425	2.3818	**258±32**	**361**

Las Figuras 4.9 y 4.10 muestran ejemplos de las señales obtenidas en el experimento. Observando la primera, se aprecia una tendencia a modelar el efecto por parte de los modelos entrenados con KD DF, pero da la impresión de que no se alcanza una transición completa entre la señal de entrada y la *target*. Esto sugiere que puede que sea necesario entrenar el modelo durante más épocas.

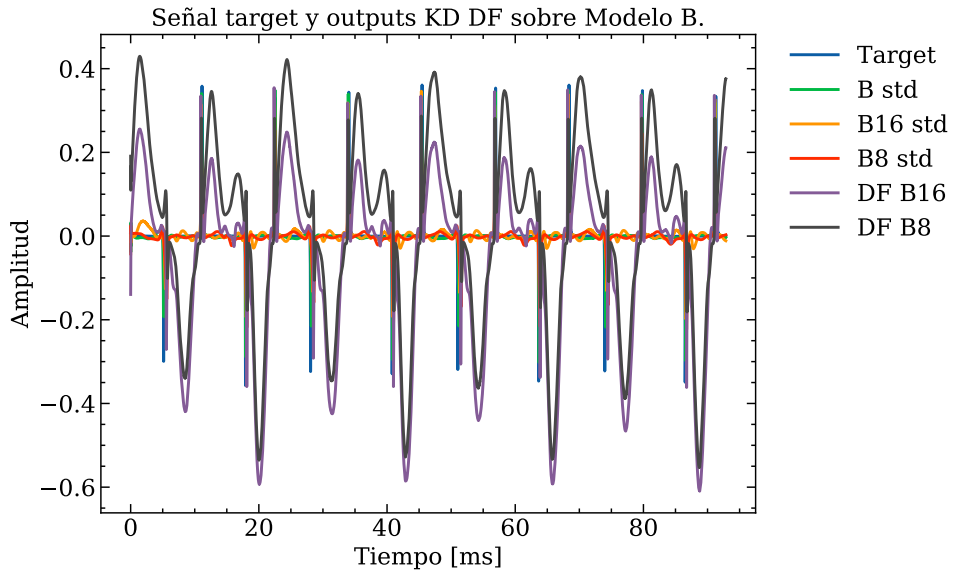


Figura 4.9: Señales *target* y *output* para aplicación de KD DF sobre Modelo B.

Al observar las funciones de *loss* de los modelos en cuestión (ver Apéndice A.3, Figuras A.7a y A.7b) se aprecia que a medida que avanzan las épocas el valor de *loss* comienza a crecer, luego de haber descendido en las primeras épocas. Sin embargo, se observa un leve decrecimiento hacia las últimas épocas, lo que podría significar que si se entrena durante más tiempo la *loss* vuelva a decrecer hacia un valor óptimo. Se pueden probar otras alternativas, como modificar hiperparámetros de la función de pérdidas o el *learning rate* del modelo. Nuevamente, este ajuste de hiperparámetros se propone como trabajo futuro.

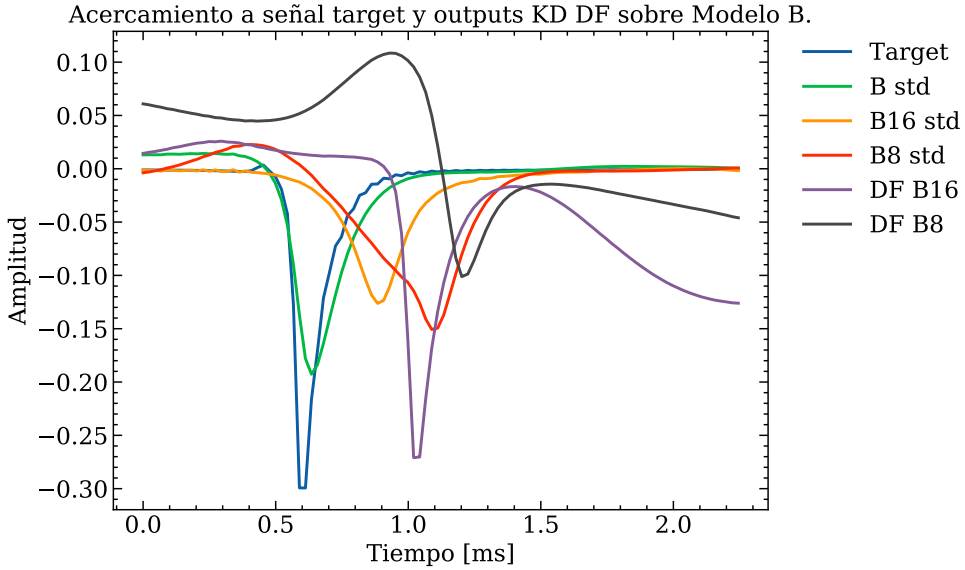


Figura 4.10: Acercamiento a señales *target* y *output* para aplicación de KD DF sobre Modelo B.

En la Figura 4.10 se aprecia como ambos modelos entrenados con DF tienden a modelar el *peak* de la señal *target*, pero no logran un buen ajuste. Además, se observa que en los valles se alejan notablemente de los valores deseados. Por lo anterior, las métricas de error son altas. En este caso, KD DF no cumple su objetivo, pero parece debserse a una falta de ajuste de hiperparámetros.

4.4. KD entre Modelos A y B

4.4.1. TOR desde B a A

Lo obtenido para la utilización de KD TOR desde el modelo B hacia el modelo A se muestra en la Tabla 4.6. También, las señales *output* y *target* obtenidas en las pruebas se muestran en las Figuras 4.11 y 4.12. En la Tabla 4.6, a partir de las métricas de error obtenidas para el modelo TOR B-A puede afirmarse que el experimento no fue exitoso. Los valores de ESR y MAE son notoriamente altos, lo cual permite concluir, aún sin revisar las señales obtenidas, que el modelo no fue capaz de ajustarse a la señal *target*.

Tabla 4.6: Evaluación de aplicación de KD TOR desde el Modelo B al A.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
B	0.0066	0.2261	311 ± 42	17,217
A	0.0099	0.5818	1202 ± 76	72,065
TOR B-A	0.6187	639.1638	1126 ± 65	72,065

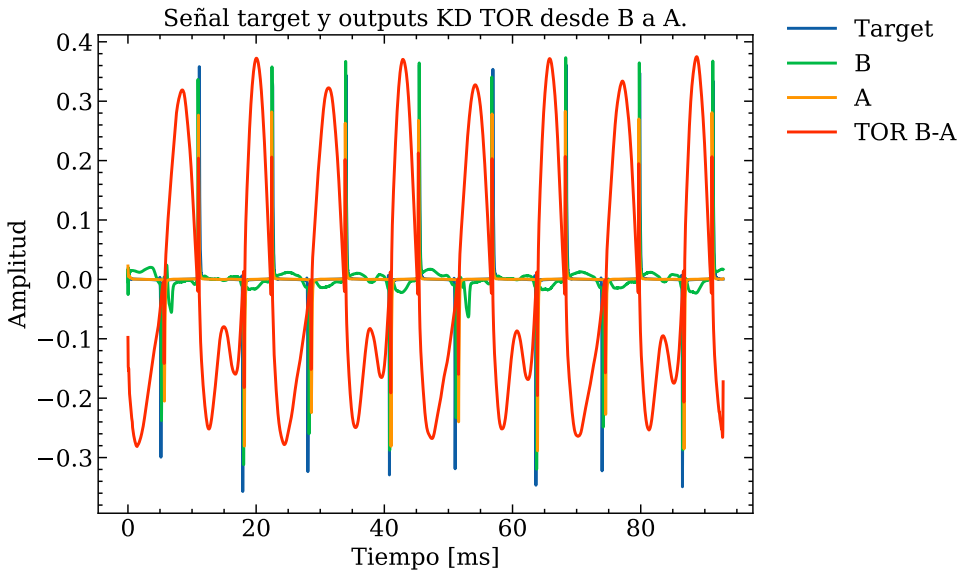


Figura 4.11: Señales *target* y *output* para aplicación de KD TOR desde Modelo B a A.

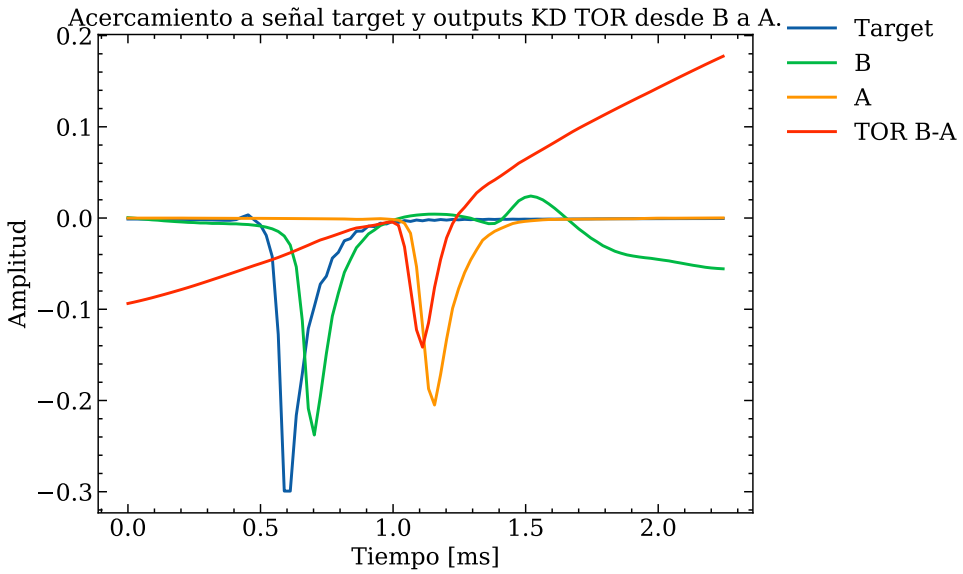


Figura 4.12: Acercamiento a señales *target* y *output* para aplicación de KD TOR desde Modelo B a A.

Luego de observar las Figuras 4.11 y 4.12 se confirma lo planteado: el modelo TOR B-A no se ajusta a la señal *target*. Se puede ver que la señal *output* para TOR B-A es similar a la señal *input*, mostrada previamente en la Figura 3.3. De hecho, parece ser la misma señal reflejada con respecto al eje horizontal. Habiendo revisado el código que en el que se define este experimento en búsqueda de posibles errores de implementación que lleven a obtener los resultados observados, se comenta que no se encontró nada extraño y que lo observado puede ser una mera casualidad.

Al examinar la función de *loss* de este entrenamiento (ver Apéndice A.4, Figura A.8a), se observa que su valor decrece en las primeras épocas para luego estabilizarse, tanto para la *loss* en entrenamiento como en validación. No se observa sobreajuste del modelo a los datos de entrenamiento.

Al tener el modelo TOR B-A la arquitectura del modelo A, su *loss* utiliza el error MAE según lo explicado en la Sección 3.4.3. Se propone como experimento futuro evaluar qué sucede al utilizar la función de *loss* usada por el modelo B para el entrenamiento de este modelo.

4.4.2. DF desde B a A

Para la aplicación de KD DF desde el modelo B hacia el modelo A, los resultados en evaluación se observan en la Tabla 4.7. Las Figuras 4.13 y 4.14 presentan ejemplos de las señales obtenidas en este experimento.

Tabla 4.7: Evaluación de aplicación de KD DF desde el Modelo B al A.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
B std	0.0131	0.5665	294 ± 35	17,217
A std	0.0094	0.5399	722 ± 21	72,065
DF B-A	0.1010	129.8341	726 ± 14	72,065

De manera similar a lo que ocurrió con el caso de TOR B-A, basta con analizar la Tabla 4.7 para notar que el modelo no aprendió a modelar el efecto de audio. Los errores obtenidos para DF B-A son demasiado altos en comparación a los errores de los modelos A y B estandarizados. Al observar las Figuras 4.13 y 4.14 se confirma que la señal de *output* está lejos de ser similar a la de *target*. Esta vez ocurre algo que no había sucedido en los experimentos anteriores: el modelo introduce un *offset* DC a la señal. La señal de *output* de DF B-A parece estar centrada en -0.1 en lugar de 0. Esto refuerza la hipótesis de que utilizar la *loss* del modelo B para este experimento puede mejorar los resultados obtenidos, puesto que esta incorpora un término específicamente enfocado en la reducción de dicho *offset*.

En la misma línea de lo dicho para los experimentos fallidos anteriores, es posible explorar mejores ajustes de hiperparámetros para el entrenamiento utilizando DF, ya que nuevamente se observa un comportamiento extraño en la función de *loss* (ver Apéndice A.4, Figura A.8b). De la misma forma, queda pendiente como trabajo futuro.

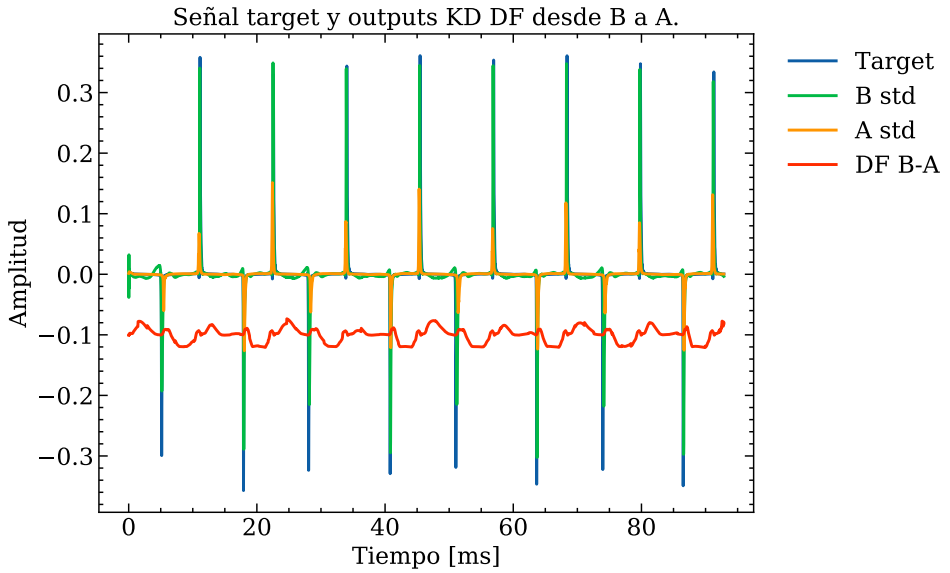


Figura 4.13: Señales *target* y *output* para aplicación de KD DF desde Modelo B a A.

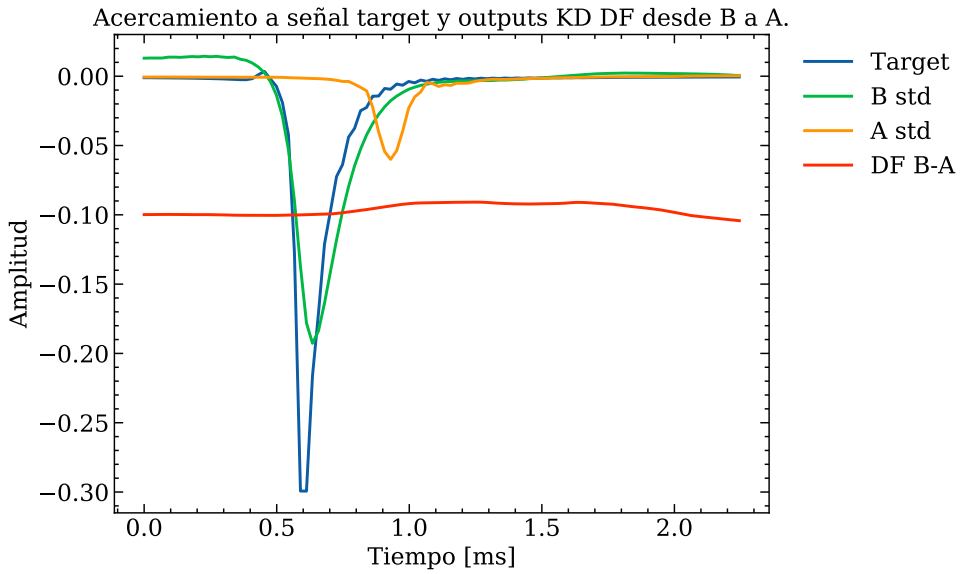


Figura 4.14: Acercamiento a señales *target* y *output* para aplicación de KD DF desde Modelo B a A.

4.5. KD hacia Modelo C

4.5.1. TOR desde B a C

En esta sección se presentan lo logrado al aplicar KD TOR desde el modelo B a la nueva arquitectura propuesta o modelo C. Los valores obtenidos para los criterios de evaluación se exponen en la Tabla 4.8:

Tabla 4.8: Evaluación de aplicación de KD TOR desde el Modelo B al C.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
B	0.0066	0.2261	311 ± 42	17,217
C con *hs*=64	**0.0045**	0.1850	152 ± 16	12,929
C con *hs*=16	0.0081	0.2625	136 ± 13	929
C con *hs*=8	0.0111	0.3335	136 ± 12	**273**
TOR C con *hs*=64	0.0051	**0.1568**	149 ± 15	12,929
TOR C con *hs*=16	0.0071	0.2058	137 ± 18	929
TOR C con *hs*=8	0.0112	0.3373	**135±14**	**273**

Observando las métricas de error, se encuentra que los modelos entrenados utilizando TOR logran obtener mejores resultados con respecto a sus versiones entrenadas desde 0. Un caso puntual ocurre para los modelos que utilizan un *hidden size*=64, ya que la versión TOR entrega un mejor ESR pero un peor MAE. Esto se debe a que ambas métricas enfatizan distintas características de la señal en su cálculo, como se explica en la Sección 2.9. De todas formas, ambos modelos son los que mejores resultados obtienen en este experimento.

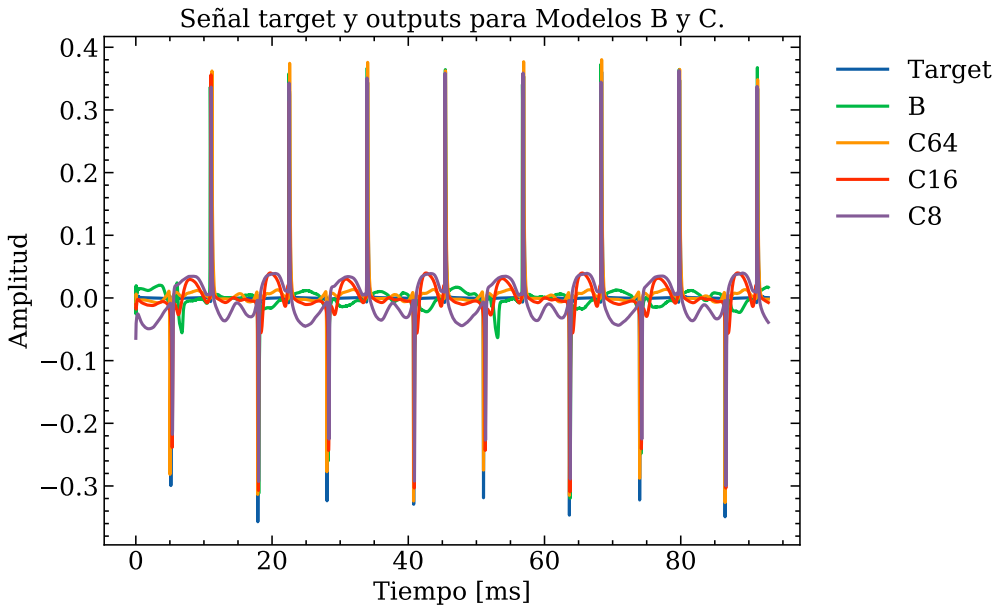
Para los casos en que el *hidden size* utilizado es 16 y 8, ambas métricas son reducidas utilizando KD TOR.

En la Figura 4.15b se observan ejemplos de señales obtenidas con los modelos entrenados. Esta vez, se muestran 2 subfiguras para favorecer la legibilidad de los gráficos. En la Figura 4.15a se observa una comparación entre la señal *target*, el modelo B y las variantes entrenadas del modelo C. En la Figura 4.15b se presenta la comparación entre el *target*, el modelo *teacher* B y los modelos C entrenados utilizando KD TOR. En las Figuras 4.16a y 4.16b se presentan los mismos casos recién explicados, pero con un acercamiento a 100 muestras de la señal, como se ha hecho en experimentos anteriores.

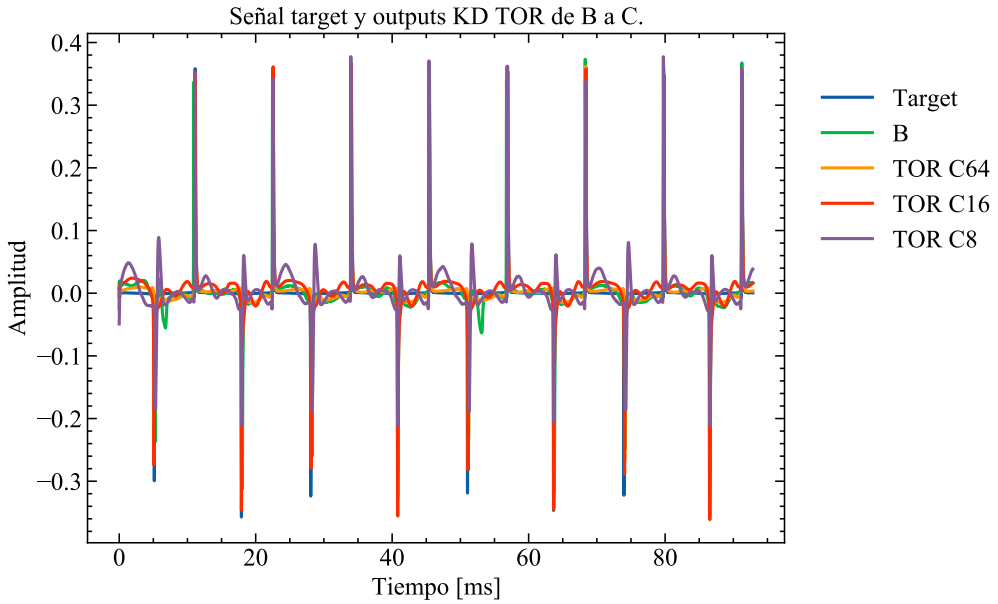
En las Figuras 4.15a y 4.15b se aprecia que todos los modelos probados tienden a modelar adecuadamente los *peaks*, con algunas variaciones de amplitud y artefactos con los valles de la señal. En la primera comparación es posible notar que el modelo que mejor modela los *peaks* y su amplitud es el modelo C con *hidden size*=64, mientras que según la segunda figura el que mejor lo logra es el modelo TOR C con *hidden size*=16. Esto se confirma observando las Figuras 4.16a y 4.16b, donde también es posible notar que los modelos TOR C64 Y TOR C16 muestran *outputs* muy similares, pero el primero se ajusta mejor al *target* en los valles. Lo anterior es verificable notando que ambos modelos con *hidden size*=64 obtienen los errores más bajos.

Hablando puntualmente sobre los dos modelos con *hidden size*=64, en base a las Figuras 4.16a y 4.16b puede decirse que TOR C64 modela de mejor manera el efecto de audio, ya que se aprecia que el desfase temporal que introduce en la ubicación del *peak* de la señal es mínimo comparado con el que introduce el modelo entrenado desde cero. En este experimento se muestra nuevamente que sí es posible aplicar técnicas de KD, en este caso TOR, para comprimir modelos con una pérdida menor en el desempeño que la que se tiene entrenando la misma arquitectura más rápida sin la utilización de KD. En este caso, el modelo TOR C64 es en promedio 162 ms más rápido en inferencia que el modelo *teacher* utilizado.

Cabe destacar que en este experimento ocurre un resultado interesante, no observado previamente en la literatura: el modelo que utiliza GRU logra un mejor desempeño que el que usa LSTM, para un mismo *hidden size*, y mejora al utilizar KD TOR. En [25] se utilizan ambos modelos, pero el que entrega mejores resultados es el que usa LSTM, por lo que se eligió para este trabajo como modelo representante de la línea de desarrollo B. Lo recién explicado es un resultado valioso, puesto que se establecería un nuevo estado del arte para modelos no lineales, puesto que el modelo obtenido es más preciso, liviano y rápido en inferencia que el modelo B.

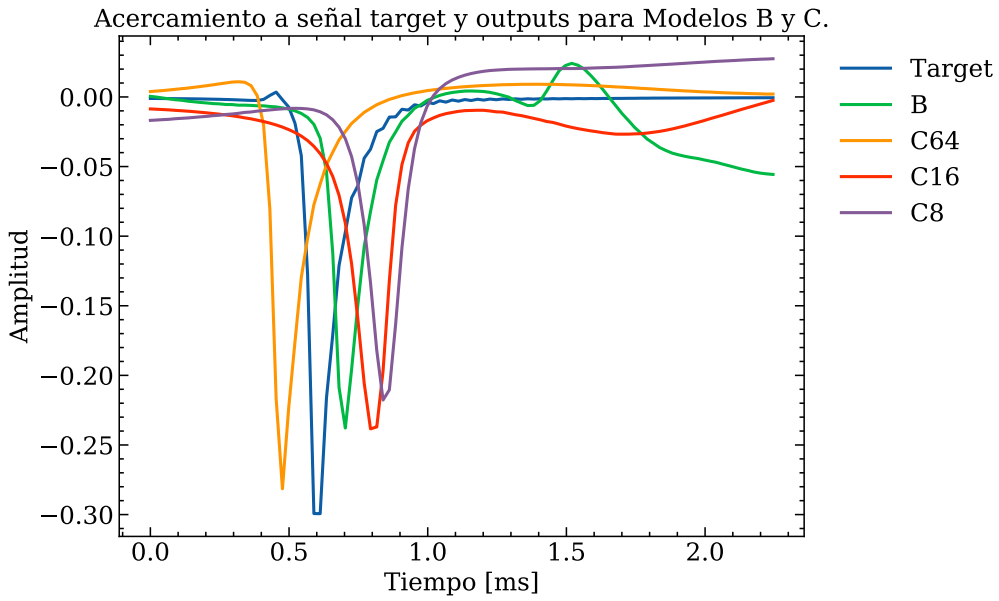


(a) Comparación Modelo B y variantes de Modelo C.

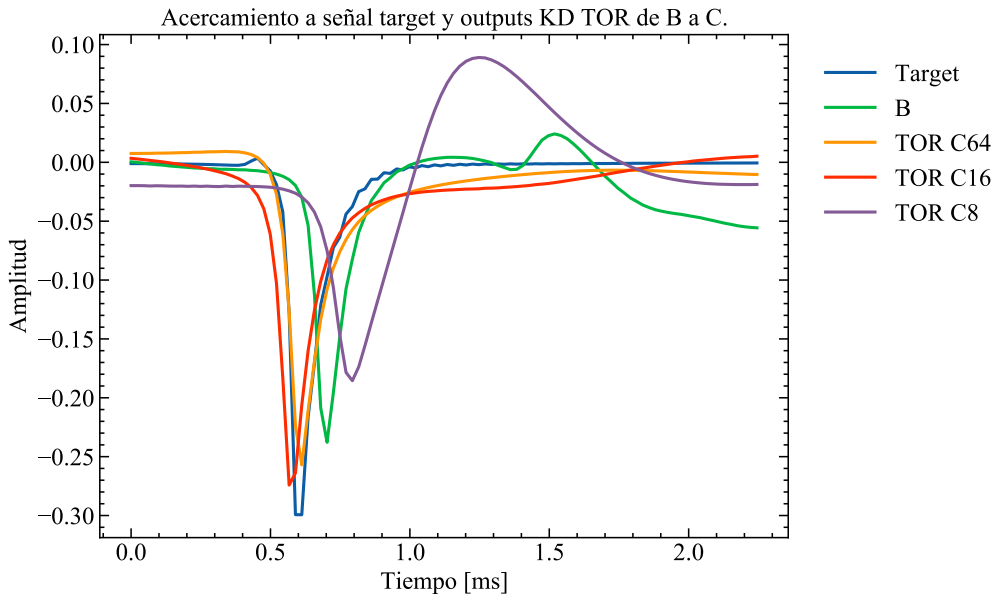


(b) Comparación de Modelo B y variantes de TOR B.

Figura 4.15: Señales *target* y *output* para aplicación de KD TOR desde Modelo B a C.



(a) Comparación Modelo B y variantes de Modelo C.



(b) Comparación de Modelo B y variantes de TOR B.

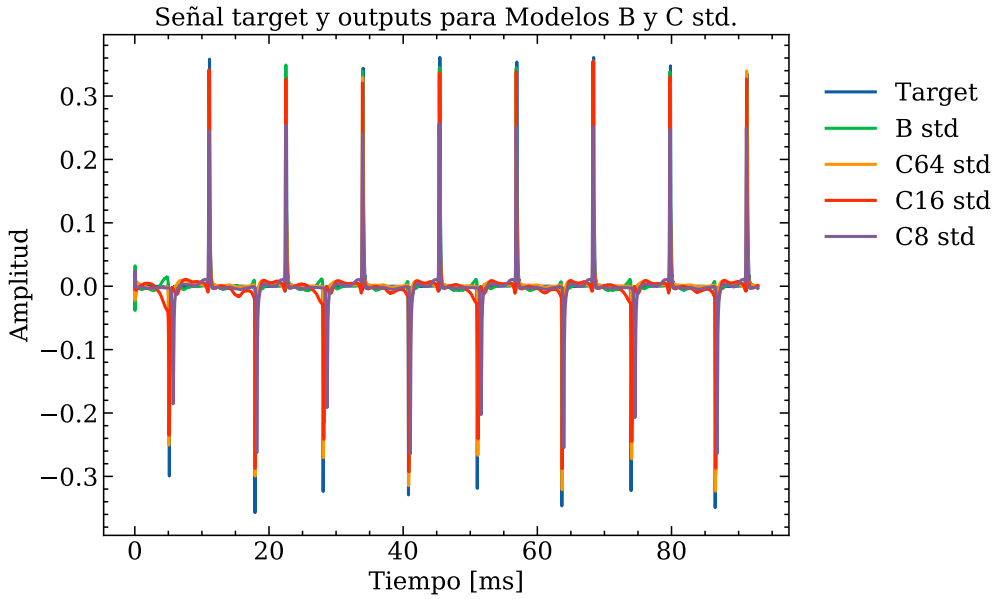
Figura 4.16: Acercamiento a señales *target* y *output* para aplicación de KD TOR desde Modelo B a C.

4.5.2. DF desde B a C

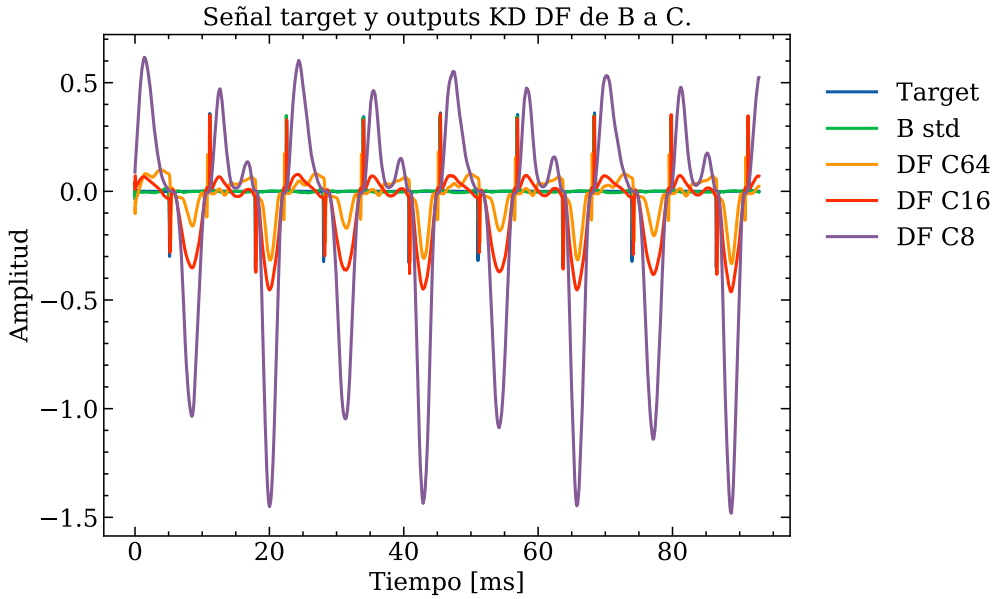
Los resultados obtenidos en el experimento que aplica KD DF desde el modelo B hacia el C se exponen en la Tabla 4.9. Por otra parte, ejemplos de las señales obtenidas se muestran en las Figuras 4.17 y 4.18, siguiendo un formato similar al usado en KD TOR: en 4.17a y 4.18a se comparan el *target*, el modelo B y variantes estandarizadas del modelo C, y en 4.17b y 4.18b se comparan el modelo *teacher* B, el *target* y los modelos C entrenados con KD DF:

Tabla 4.9: Evaluación de aplicación de KD DF desde el Modelo B al C.

Modelo	**MAE**	**ESR**	**Tiempo de inferencia [ms]**	**Parámetros**
B std	0.0131	0.5665	294 ± 35	17,217
C std con hs=64	**0.0123**	**0.5077**	159 ± 28	12,929
C std con hs=16	0.0130	0.5272	147 ± 16	929
C std con hs=8	0.0147	0.6778	**144±20**	**273**
DF C con hs=64	0.0133	0.5731	157 ± 28	12,929
DF C con hs=16	0.0156	0.6117	147 ± 15	929
DF C con hs=8	0.0163	0.9592	145 ± 11	**273**



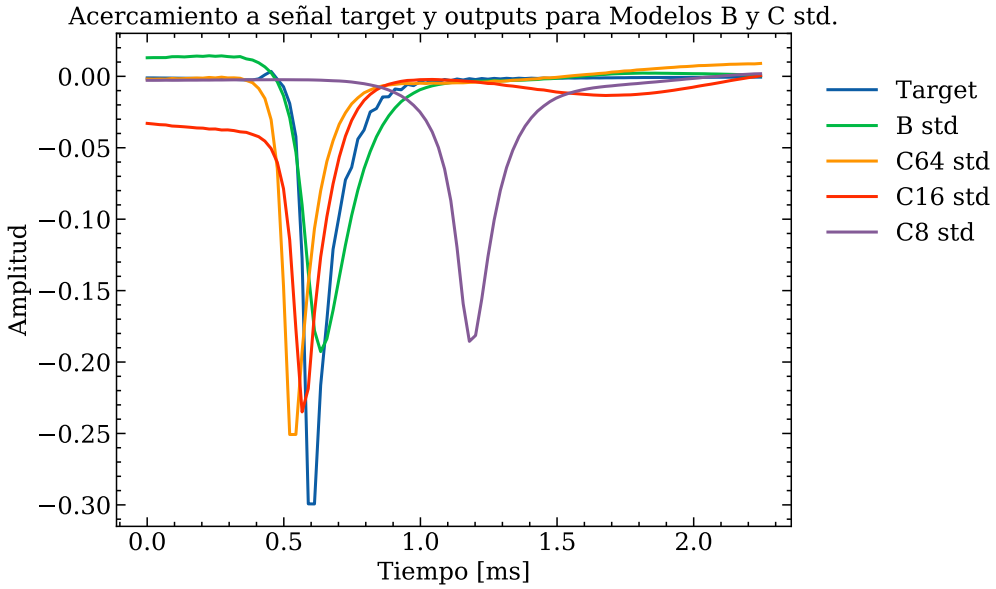
(a) Comparación Modelo B std y variantes std de Modelo C.



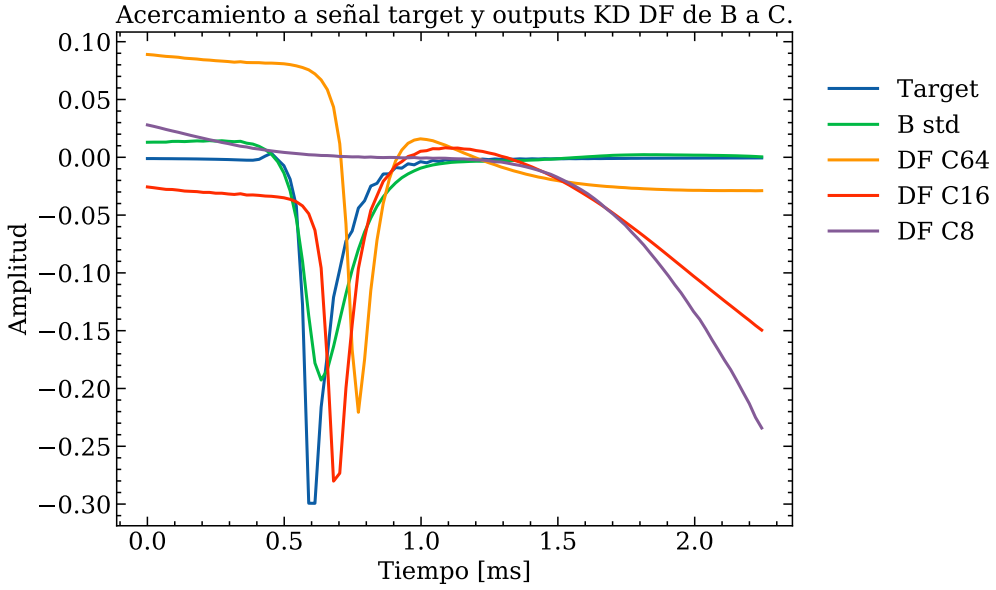
(b) Comparación de Modelo B std y variantes de DF B.

Figura 4.17: Señales *target* y *output* para aplicación de KD DF desde Modelo B a C.

Nuevamente ocurre, pero esta vez con modelos entrenados con datos estandarizados, que el modelo C con *hidden size*=64 entrega errores más bajos que el modelo B, con las mismas unidades en el *hidden state*. Según los resultados observados en la Tabla 4.9, los modelos DF alcanzan un mejor desempeño en comparación a todos los experimentos anteriores que utilizaron esta técnica. Esto se debe verificar mediante la revisión de las señales *output* de los modelos. No obstante, no logran superar en *performance* a los modelos entrenados sin KD DF.



(a) Comparación Modelo B std y variantes std de Modelo C.



(b) Comparación de Modelo B std y variantes de DF B.

Figura 4.18: Acercamiento a señales *target* y *output* para aplicación de KD DF desde Modelo B a C.

De las Figuras 4.17a y 4.18a se desprende que, en general, los artefactos generados en los valles de la señal son más bajos al entrenar los modelos con datos estandarizados. Además, según lo observado en la primera, las variantes C64 std y C16 std, logran un desempeño similar entre ellas y superior al resto. Además, los desfases introducidos en la aparición del *peak* son más bajas que cuando los modelos se entrenan con datos sin estandarizar, pero tienden a fallar en el modelamiento de la amplitud de la señal, lo que aumenta sus métricas de error con respecto al caso no estandarizado.

Si bien no se cumple el objetivo de aplicar KD DF, se obtiene el resultado relevante de que esta vez, al menos para DF C64 y DF C16 la señal *target* tiende a ser modelada de buena manera. La excepción ocurre con DF C8, pero de todas formas el modelo aprende a sincronizar los *peaks* del *output* con los del *target*, lo cual lleva a que las métricas de error sean menores que otros casos vistos, como el de TOR B-A en la Figura 4.11, donde el *output* tiene *peaks* en zonas que la señal *target* tiene valles cercanos a 0.

4.6. Validación del Mejor Resultado

Teniendo en consideración todos los modelos entrenados en los experimentos mostrados en este capítulo, en esta sección se selecciona el mejor de ellos según los criterios de evaluación propuestos y se repite su entrenamiento utilizando diferentes semillas aleatorias para inicializar los parámetros de las redes neuronales involucradas en la arquitectura correspondiente.

En particular, el modelo elegido como el de mejor desempeño es el modelo TOR C64, es decir, aquel que sigue la arquitectura de la línea C, con un *hidden state* de tamaño 64 y entrenado utilizando KD TOR con el modelo representante de la línea B como *teacher*. Esta elección se debe a que entrega un error ESR notablemente más bajo que el resto de los modelos en evaluación, junto con un error MAE también bajo, muy cercano al menor obtenido en los experimentos. Muestra además un tiempo de inferencia que reduce en más de un 50 % al alcanzado por el modelo *teacher* utilizado, con menos parámetros aprendibles. Por otra parte, en base a la evaluación visual de los ejemplos de señales de salida mostrados, el modelo TOR C64 es el que muestra un mejor balance entre el ajuste a la señal *target* en el valle y en el *peak*, observándose un bajo desfase temporal en este último caso.

Como se declara en la Sección 3.3, el entrenamiento se repite 5 veces con semillas aleatorias distintas. Una vez realizado esto, se reportan las métricas de evaluación y señales de salida de ejemplo. En ambos casos se considera el promedio de los valores obtenidos y su desviación estándar respectiva. Dichos resultados se presentan correspondientemente en la Tabla 4.10 y en la Figura 4.19.

Tabla 4.10: Consistencia estadística del modelo TOR C64: promedio y desviación estándar de los errores MAE y ESR sobre 5 repeticiones del entrenamiento.

Modelo	**MAE**	**ESR**
TOR C64	$0{,}0052 \pm 0{,}0003$	$0{,}1595 \pm 0{,}0161$

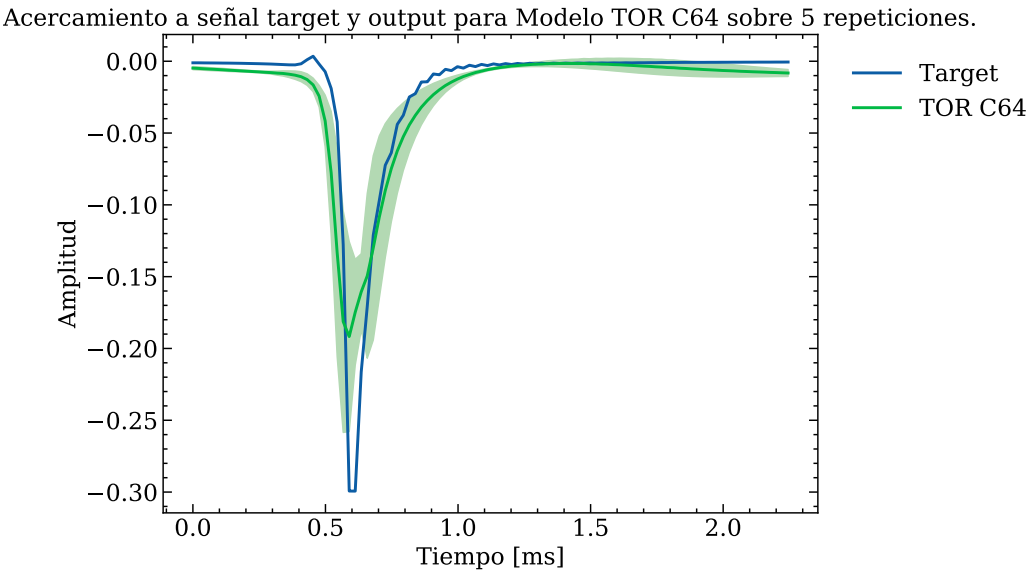


Figura 4.19: Acercamiento a señales *target* y *output* para KD TOR C64. El área sombreada representa el valor promedio ± 1 desviación estándar sobre 5 repeticiones del entrenamiento.

En la Tabla 4.10 se observan desviaciones estándar pequeñas tanto para MAE como ESR. Por otro lado, en la Figura 4.19 se aprecia a partir de la salida para una señal de *test*, que TOR C64 modela de buena forma, en promedio y con baja desviación estándar, la señal *target*. Lo recién comentado permite afirmar que el modelo seleccionado entrega resultados estadísticamente consistentes, que dan muestra de que el buen desempeño se mantiene de forma no condicionada por la inicialización aleatoria de los parámetros de las redes neuronales empleadas.

4.7. Discusión

Una vez desarrollados todos los experimentos propuestos, puede discutirse de forma más general los resultados obtenidos. En primer lugar, se obtuvo que el modelo B supera al modelo A al evaluarlos sobre un mismo dataset, según los criterios de evaluación definidos.

Con respecto a la aplicación de técnicas de KD con el objetivo de comprimir modelos con la menor reducción posible en su desempeño, se afirma que esto sólo se logró aplicando KD TOR sobre el modelo B y desde el modelo B hacia el C. La aplicación de KD DF no fue exitosa, pero muestra resultados prometedores, sobre todo en el experimento de KD desde el modelo B hacia el modelo C. En el análisis particular de cada experimento se sugieren posibles opciones para mejorar los resultados obtenidos, a modo de trabajo futuro, para los casos en que los resultados no fueron satisfactorios. A grandes rasgos, esto último consiste en hacer una búsqueda más exhaustiva de hiperparámetros para implementar los entrenamientos en cada caso y no depender únicamente de los valores propuestos por los autores de los métodos para los problemas particulares tratados en sus investigaciones.

En cuanto a las interrogantes propuestas en el planteamiento de los experimentos, puede decirse que:

- El modelo B supera al A sobre el dataset *Big Muff Pi*, bajo los criterios de evaluación elegidos.
- Sí es factible modelar efectos no lineales sin recurrir a arquitecturas complejas.
- Es más influyente la arquitectura utilizada que la cantidad de parámetros aprendidos, pues los modelos más pesados no funcionan necesariamente mejor.
- Con la configuración utilizada, no fue posible mejorar el desempeño del modelo A a partir del modelo B.
- Los mejores resultados son alcanzados con el modelo C64 entrenado con KD TOR.

Capítulo 5

Conclusión

Una vez presentados y analizados los resultados obtenidos en los experimentos propuestos, se concluye que en este trabajo se logra cumplir el objetivo general planteado, puesto que se ha mostrado que en al menos dos de los experimentos realizados sí es posible obtener mediante técnicas de *Knowledge Distillation* modelos comprimidos, más rápidos en inferencia, que consiguen un desempeño mejor que el que se obtendría entrenando la misma arquitectura pero sin la aplicación de KD. Esto se mostró para el modelamiento de un efecto de audio no lineal de tipo *fuzz*, que introduce no linealidades más acentuadas que otros efectos de su tipo, las cuales se traducen en sonidos más "ásperos" y en una mayor dificultad de modelar la alteración que produce sobre una señal de audio de entrada.

Dichos resultados se obtuvieron utilizando la variante *Teacher Outlier Rejection* de KD, la cual utiliza el modelo preentrenado o *teacher* como detector y supresor de posibles *outliers* en los datos, lo cual es beneficioso en el caso de que los datos de entrenamiento contengan ruido proveniente desde, por ejemplo, cables defectuosos en el proceso de grabación del audio. En particular, se propone que los resultados obtenidos al aplicar KD TOR sobre el modelo C con *hidden size*=64 establecen un nuevo estado del arte en el modelamiento de efectos no lineales de tipo *fuzz*, ya que corresponde a un modelo más preciso, liviano y rápido en inferencia que el modelo de la línea B.

En cuanto a los objetivos específicos formulados, puede decirse lo siguiente: en primer lugar, sí se cumple con proponer criterios de evaluación para la comparación de los modelos obtenidos. Esto se realizó considerando distintos criterios utilizados previamente en la literatura, formando una batería de criterios que busca no introducir sesgo hacia ninguna arquitectura particular, lo cual puede suceder al evaluar con una única métrica de error, por ejemplo. Los criterios propuestos son 2 métricas de error, ESR y MAE, el tiempo de inferencia de los modelos al procesar 1 segundo de audio de entrada, la comparación de la cantidad de parámetros que conforman a los modelos, y la evaluación visual y auditiva de las señales obtenidas. Cabe destacar que al nivel de desarrollo logrado en este trabajo, este último criterio no es cuantificable.

El objetivo específico de comparar el desempeño de las arquitecturas existentes en la literatura también se logra, específicamente en el primer experimento realizado, obteniendo que el modelo representante de la línea de desarrollo B supera al de la línea A. En tercer lugar, también se logra evaluar la efectividad de KD sobre un problema más complejo que los estudiados en la literatura, ya que estos últimos no involucran sistemas no lineales. Al ser más complejo el problema, en ciertos experimentos se obtienen resultados deficientes, pero se proponen formas de abordarlo en un trabajo futuro y, por otra parte, en los casos ya mencionados se obtienen buenos resultados.

Por último, también se cumple con el entrenamiento de modelos que apunten a mejorar el estado del arte en términos de reducir tiempos de inferencia, puesto que para el entrenamiento con KD TOR del modelo C con *hidden size*=64 se alcanzan resultados que superan el desempeño del modelo B, proponiéndose un nuevo estado del arte para el modelamiento de efectos no lineales, al menos para el caso del *fuzz*. Además, para este modelo en particular se realizan experimentos de repetición del entrenamiento y evaluación con diferentes semillas aleatorias, permitiendo concluir que los resultados obtenidos son consistentes.

Cabe recalcar que los experimentos permiten concluir que es más influyente en un buen resultado utilizar una arquitectura diseñada para modelar dependencias temporales, como las RNN, por sobre una arquitectura más compleja, aún cuando sea un modelo que aprenda más parámetros. Se observa que las arquitecturas de la línea B bastan para alcanzar buenos resultados y que puede seguir investigándose para lograr comprimir aún más los modelos sin afectar su desempeño.

Los resultados logrados, descritos en los párrafos anteriores, permiten concluir que se confirma la hipótesis postulada al comienzo de esta investigación. Esto se afirma porque las variantes de modelos obtenidos en los experimentos exitosos sí reducen los tiempos de inferencia de los modelos del estado del arte en la tarea del modelamiento de efectos de audio, superando el desempeño que se obtendría entrenando las variantes sin considerar la aplicación de *Knowledge Distillation* propuesta en esta tesis. Los resultados, para el caso del modelo TOR C64, incluso muestran un desempeño que supera al obtenido con el modelo del estado del arte utilizado como *Teacher*.

Como trabajo futuro se propone revisitar los experimentos que no entregaron resultados satisfactorios y establecer una metodología para evaluar candidatos a hiperparámetros que definen la construcción de las técnicas de KD propuestas. Un caso específico ocurre con la elección de hiperparámetros que ponderan los términos de regularización en la función de *loss* de KD DF. En este trabajo, los experimentos se llevaron a cabo utilizando los mismos valores utilizados por los autores en sus investigaciones, en problemas de diferente dificultad, y sin mayor fundamentación para su elección, por lo que se postula que es necesario analizar con más cuidado qué valores se utilizan. Por otra parte, se propone evaluar el enfoque propuesto en esta tesis sobre otros tipos de efectos no lineales e incluso sobre efectos variantes en el tiempo.

Bibliografía

[1] T. Schmitz, "Nonlinear modeling of the guitar signal chain enabling its real-time emulation: Effects, amplifiers and loudspeakers," Ph.D. thesis, University of Liège, Liège, Belgium, 2019.

[2] B. Boulet, "*Fundamentals of Signals and Systems*", 1st ed., Boston, MA: Charles River Media, 2006.

[3] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain," in *Psychological Review*, Vol. 65, No. 6, 1958.

[4] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, Vol. 86, 1998.

[5] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," *International Conference on Learning Representations* (ICLR), 2016.

[6] A. van den Oord et al., "WaveNet: A Generative Model for Raw Audio", arXiv:1609:03499, 2016.

[7] Y. Chen and I. Lopez-Moreno, "Locally-connected and convolutional neural networks for small footprint speaker recognition," *Interspeech*, 2015.

[8] D. Rumelhart, G. Hinton and R. Williams, "Learning internal representations by error propagation" Tech. rep. ICS 8504. San Diego, California: Institute for Cognitive Science, University of California, 1985.

[9] Y. Bengio, P. Simard and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," in *IEEE Transactions on Neural Networks*, Vol. 5, No. 2, 1994.

[10] S. Hochreiter, J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, 1997.

[11] K. Cho et al., "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," *Proceedings of the Empirical Methods in Natural Language Processing*, pp. 1724–1734, 2014.

[12] R. Cahuantzi, X. Chen and S. Güttel, "A comparison of LSTM and GRU networks for learning symbolic sequences," arXiv.2107.02248, 2021.

[13] P. J. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, 1990.

[14] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179-211, 1990.

[15] I. Sutskever, O. Vinyals and Q. Le, "Sequence to Sequence Learning with Neural Networks," *Advances in Neural Information Processing Systems*, Vol. 27, 2014.

[16] B. Ding, H. Qian and J. Zhou, "Activation functions and their characteristics in deep neural networks," *Chinese Control And Decision Conference (CCDC)*, pp. 1836-1841, 2018.

[17] F. Agostinelli, M. Hoffman, P. Sadowski and P. Baldi, "Learning activation func- tions to improve deep neural networks," *International Conference on Learning Representations* (ICLR), 2015.

[18] L. Hou, D. Samaras, T. Kurc, Yi Gao and J. Saltz, "ConvNets with Smooth Adaptive Activation Functions for Regression," *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, Vol. 54, pp. 430-439, 2017.

[19] J. Gou, B. Yu, S. Maybank and D. Tao, "Knowledge Distillation: A Survey", *International Journal of Computer Vision*, vol. 129, pp. 1789-1819, 2021.

[20] M. Takamoto, Y. Morishita, and H. Imaoka, "An Efficient Method of Training Small Models for Regression Problems with Knowledge Distillation," *Multimedia Information Processing and Retrieval (MIPR)*, 2020.

[21] M. Kang and S. Kang, "Data-free knowledge distillation in neural networks for regression," *Expert Systems with Applications*, Vol. 175, 2021.

[22] C. Zhang, S. Bengio, M. Hardt, B. Recht and O. Vinyals, "Understanding Deep Learning (Still) Requires Rethinking Generalization," *Communications of the ACM*, vol. 64, no. 3, 2021.

[23] E. Damskägg, L. Juvela, E. Thuillier and V. Välimäki, "Deep Learning for Tube Amplifier Emulation," *International Conference on Acoustics, Speech and Signal Processing* (ICASSP), pp. 471-475, 2019.

[24] A. Wright, E. Damskägg, L. Juvela and V. Välimäki, "Real-Time Guitar Amplifier Emulation with Deep Learning," *International Sound and Music Computing Conference* (SMC), 2020.

[25] A. Wright, E. Damskägg, and V. Välimäki, "Real-Time Black-Box Modelling with Recurrent Neural Networks," *International Conference on Digital Audio Effects* (DAFx), 2019.

[26] A. Wright and V. Välimäki, "Neural Modelling of Periodically Modulated Time-Varying Effects," *International Conference on Digital Audio Effects* (DAFx), 2020.

[27] M. Lester and J. Boley, "The Effects of Latency on Live Sound Monitoring," *Audio Engineering Society Convention* (AES), 2007.

[28] M. Martínez and J. Reiss, "End-to-end Equalization with Convolutional Neural Networks," *International Conference on Digital Audio Effects* (DAFx), 2018.

[29] M. Martínez and J. Reiss, "Modeling Nonlinear Audio Effects with End-to-end Deep Neural Networks," *International Conference on Acoustics, Speech and Signal Processing* (ICASSP), pp. 171-175, 2019.

[30] M. Martínez, E. Benetos and J. Reiss, "A General-Purpose Deep Learning Approach to Model Time-Varying Audio Effects," *International Conference on Digital Audio Effects* (DAFx), 2019.

[31] M. Martínez, E. Benetos and J. Reiss, "Modeling Plate and Spring Reverberation using a DSP-Informed Deep Neural Network," *International Conference on Acoustics, Speech and Signal Processing* (ICASSP), pp. 241-245, 2020.

[32] J. Abeßer, C. Dittmar, P. Kramer and G. Schuller, "Parametric Audio Coding of Bass Guitar Recordings using a Tuned Physical Modeling Algorithm," *International Conference on Digital Audio Effects* (DAFx), 2013.

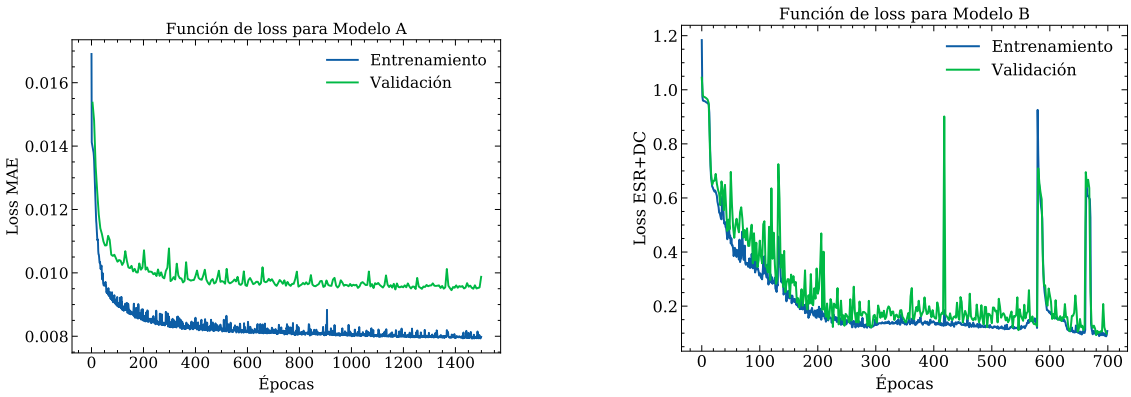
[33] C. Kehling, J. Abeßer, C. Dittmar and G. Schuller, "Automatic Tablature Transcription of Electric Guitar Recordings by Estimation of Score- and Instrument-related Parameters," *International Conference on Digital Audio Effects* (DAFx), 2014.

[34] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, Vol. 32, pp. 8024–8035, 2019.

ANEXO

Funciones de *loss*

A.1. Comparación Modelos A y B

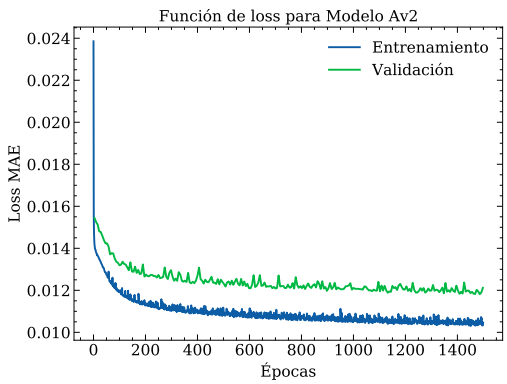


(a) Función de *loss* de modelo A.

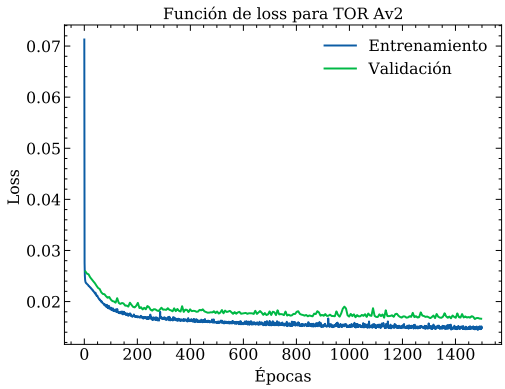
(b) Función de *loss* de modelo B.

Figura A.1: Funciones de *loss* en entrenamiento y validación de modelos A y B.

A.2. KD sobre Modelo A

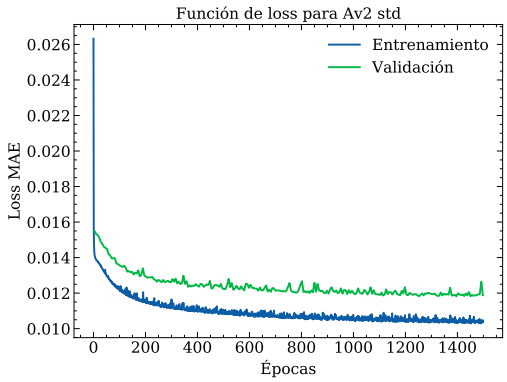


(a) Función de *loss* de modelo A*v2*.

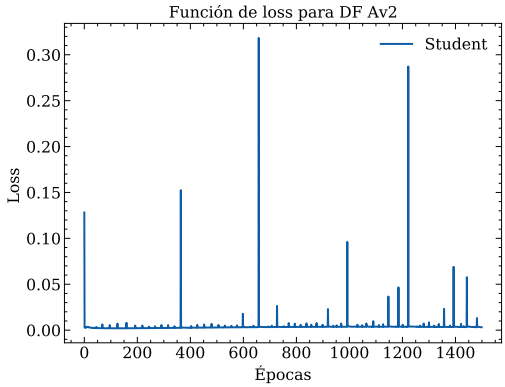


(b) Función de *loss* de modelo TOR A*v2*.

Figura A.2: Funciones de *loss* en entrenamiento y validación para KD TOR sobre modelo A.



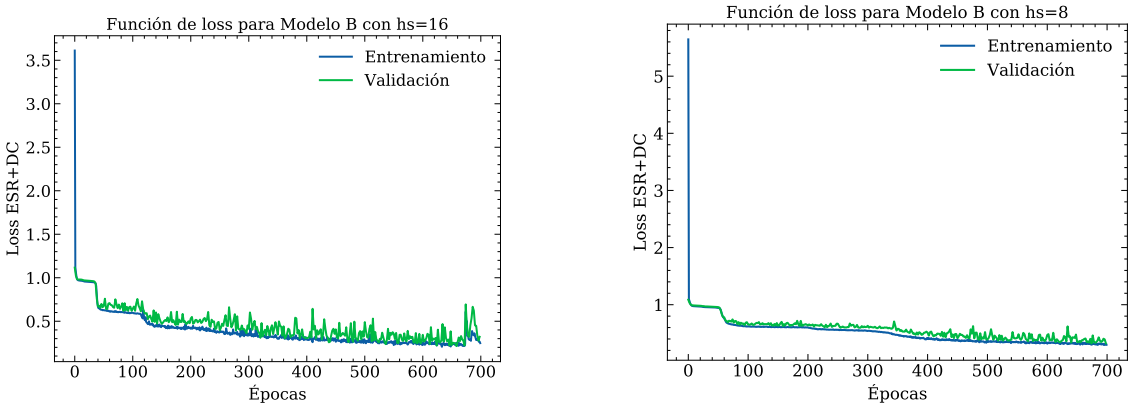
(a) Función de *loss* de modelo A*v2* std.



(b) Función de *loss* de modelo DF A*v2*.

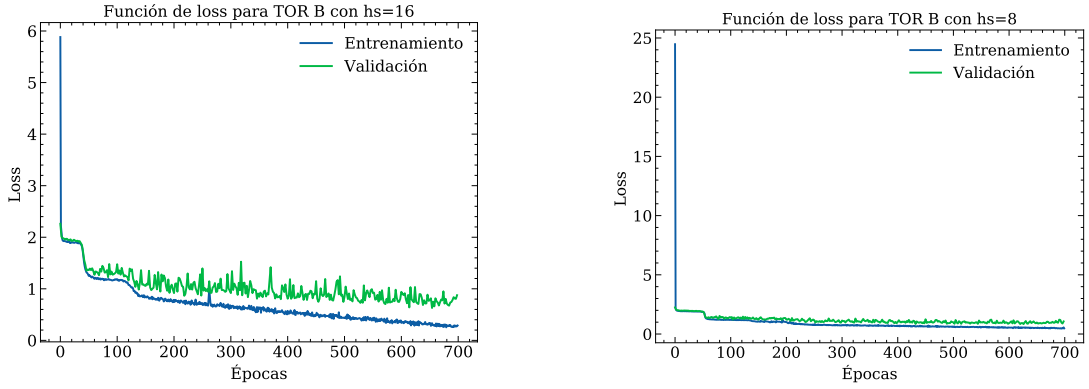
Figura A.3: Funciones de *loss* en entrenamiento y validación para KD DF sobre modelo A.

A.3. KD sobre Modelo B



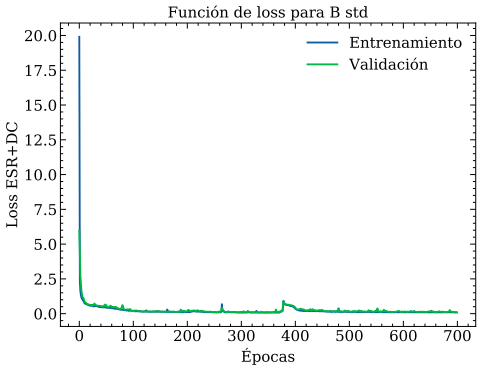
(a) Función de *loss* para modelo B con hs=16. (b) Función de *loss* para modelo B con hs=8.

Figura A.4: Funciones de *loss* en entrenamiento y validación para variantes de modelo B.

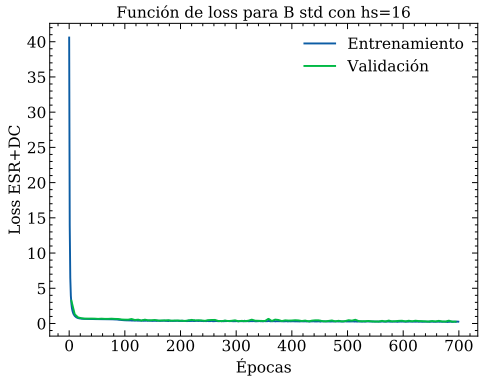


(a) Función de *loss* para TOR B con hs=16. (b) Función de *loss* para TOR B con hs=8.

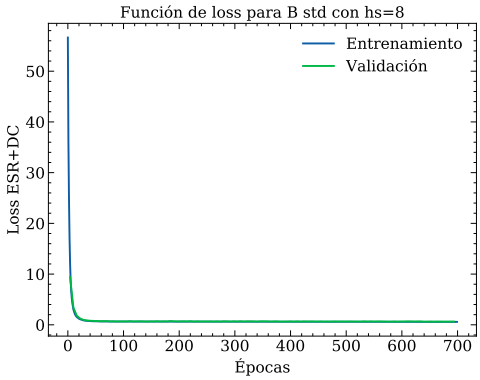
Figura A.5: Funciones de *loss* en entrenamiento y validación para KD TOR sobre modelo B.



(a) Función de *loss* para modelo B std.

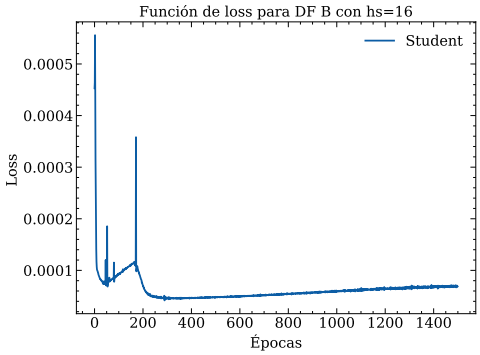


(b) Función de *loss* para B std con hs=16.

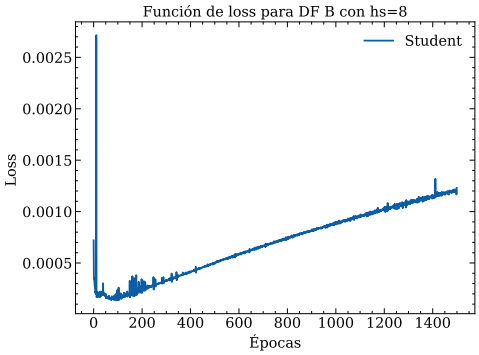


(c) Función de *loss* para B std con hs=8.

Figura A.6: Funciones de *loss* en entrenamiento y validación para variantes de modelo B estandarizado.



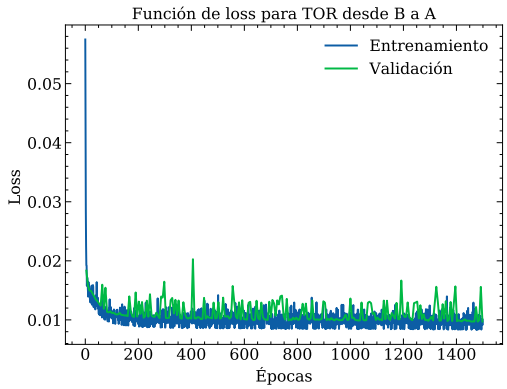
(a) Función de *loss* para DF B con hs=16.



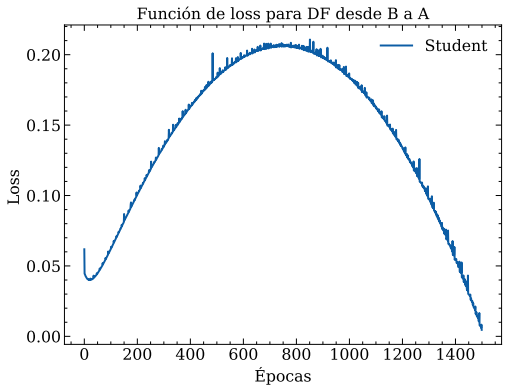
(b) Función de *loss* para DF B con hs=8.

Figura A.7: Funciones de *loss* en entrenamiento y validación para KD DF sobre modelo B.

A.4. KD entre Modelos A y B



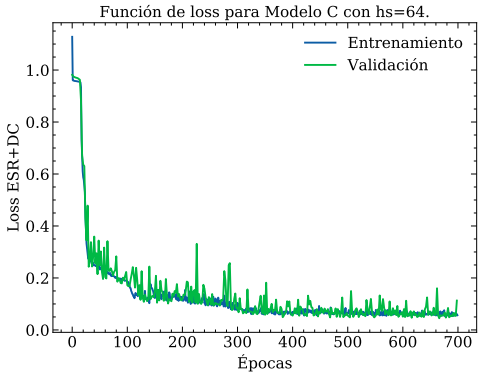
(a) Función de *loss* para TOR desde B a A.



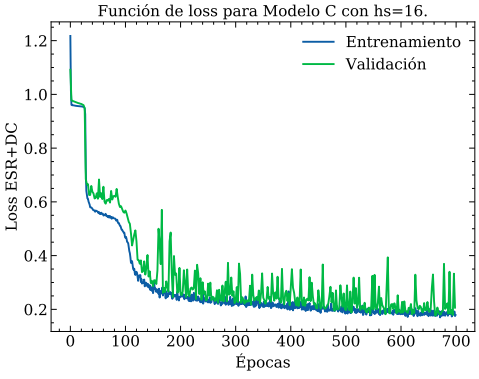
(b) Función de *loss* para DF desde B a A.

Figura A.8: Funciones de *loss* en entrenamiento y validación para KD desde modelo B a A.

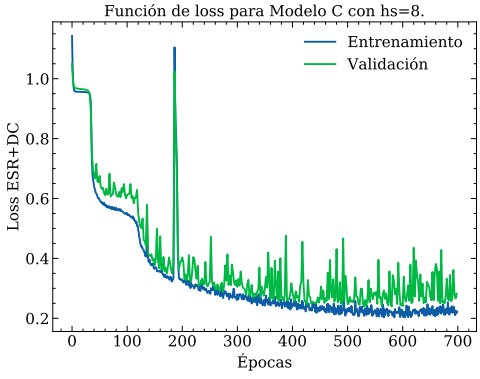
A.5. KD hacia Modelo C



(a) Función de *loss* para C con *hs*=64.

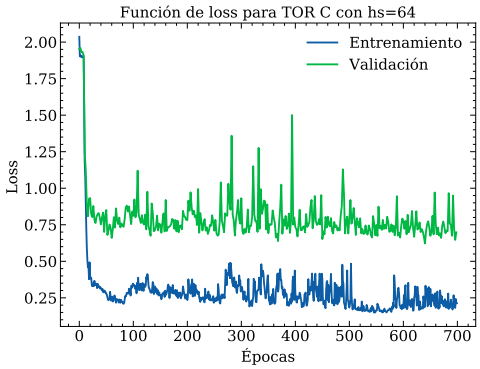


(b) Función de *loss* para C con *hs*=16.

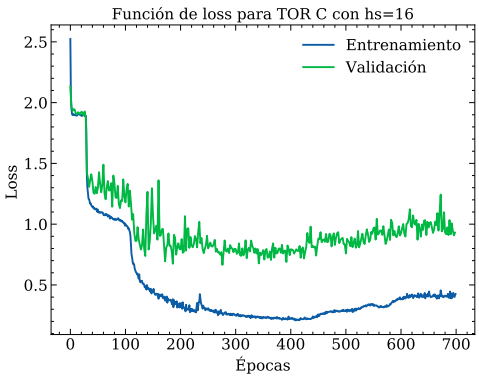


(c) Función de *loss* para C con *hs*=8.

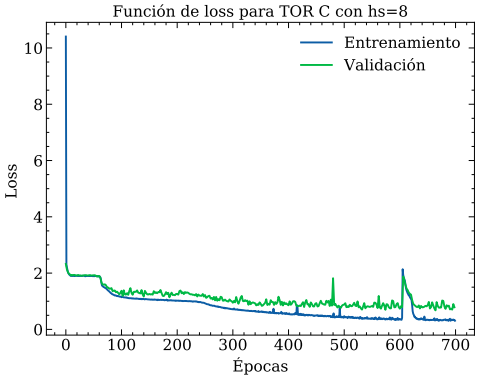
Figura A.9: Funciones de *loss* en entrenamiento y validación para variantes de modelo C.



(a) Función de *loss* para TOR C con *hs*=64.

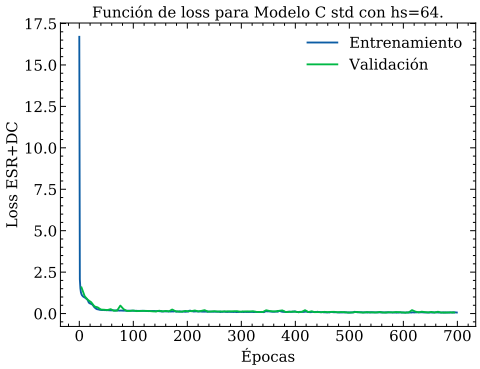


(b) Función de *loss* para TOR C con *hs*=16.

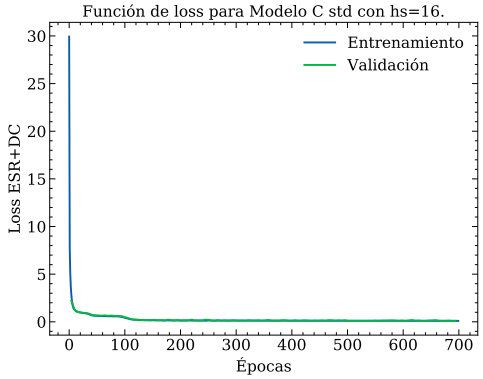


(c) Función de *loss* para TOR C con *hs*=8.

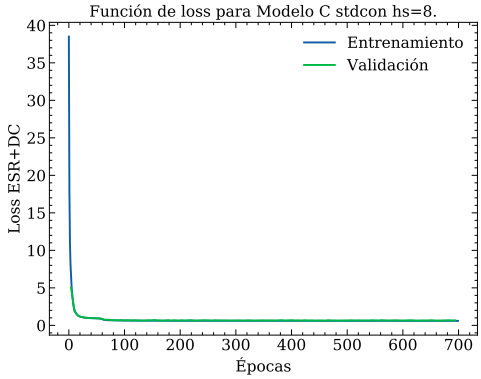
Figura A.10: Funciones de *loss* en entrenamiento y validación para KD TOR desde modelo B a C.



(a) Función de *loss* para C std con *hs*=64.

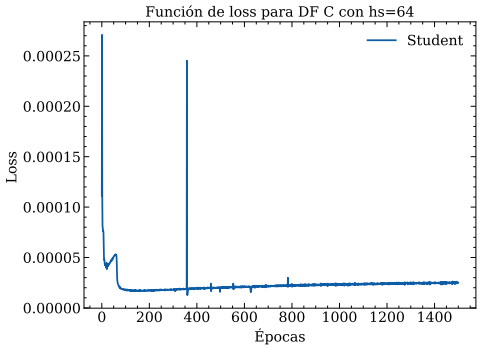


(b) Función de *loss* para C std con *hs*=16.

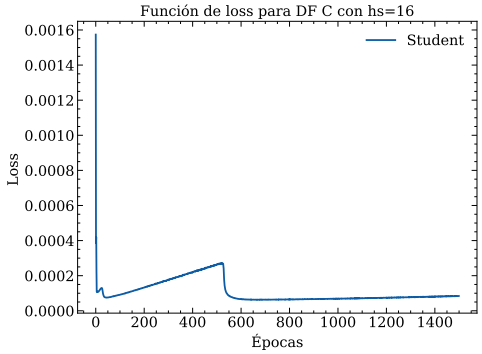


(c) Función de *loss* para C std con *hs*=8.

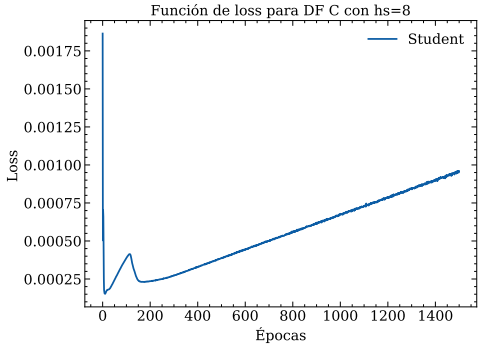
Figura A.11: Funciones de *loss* en entrenamiento y validación para variantes estandarizadas de modelo C.



(a) Función de *loss* para DF C con *hs*=64.



(b) Función de *loss* para DF C con *hs*=16.



(c) Función de *loss* para DF C con *hs*=8.

Figura A.12: Funciones de *loss* en entrenamiento y validación para KD DF desde modelo B a C.