Reducción de ruido en señales de audio basada en una Red Neuronal Convolucional

Adrián López Mora

Grado en Tecnologías de la Telecomunicación

Aplicaciones multimedia basadas en el procesamiento de la señal

Consultora: Lourdes Meler Corretjé

PRA: David García Solórzano

12 de junio de 2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	Reducción de ruido en señales de audio basada en una Red Neuronal Convolucional
Nombre del autor:	Adrián López Mora
Nombre del consultor/a:	Lourdes Meler Corretjé
Nombre del PRA:	David García Solórzano
Fecha de entrega:	06/2019
Titulación o programa:	Grado de Tecnologías de la Telecomunicación
Área del Trabajo Final:	Aplicaciones multimedia basadas en el procesamiento de la señal.
Idioma del trabajo:	Castellano
Palabras clave	Reducción de ruido, CNN, Audio

Resumen

En este TFG se implementa un sistema de reducción de ruido en señales de habla basado en una red neuronal convolucional (CNN). Un módulo de transformación de características extrae la magnitud y la fase de la señal de audio a partir del cálculo de la STFT. La CNN mapea la magnitud del espectro de la señal de entrada con ruido a una magnitud de espectro con ruido reducido. Un módulo de reconstrucción de la señal reconstruye la señal de audio con ruido reducido a partir del cálculo inverso de la STFT. Se utiliza como base de datos para el entrenamiento y test de la CNN el *corpus* catalán de Mozilla Common Voice. Para obtener la señal con ruido se añade AWGN con SNR = 0 dB a las señales de audio limpias. Todo el sistema se implementa sobre MATLAB. Para la medida de la eficacia del sistema se utilizan las métricas PESQ y STOI. Las pruebas sobre el sistema muestran resultados positivos para relaciones SNR

iguales a las utilizadas durante el entrenamiento, pero el aumento de la distorsión para relaciones SNR mayores disminuye la inteligibilidad general.

Abstract

This project describes a speech enhancement system implementation based on a convolutional neural network (CNN). A feature transform module computes the STFT and extracts spectral phase and magnitude from the speech signal. The CNN maps the spectrum magnitude of an input noisy speech signal to an output enhanced spectrum. A reconstruction module computes inverse STFT to recover the speech enhanced audio signal. Mozilla Common Voice database, in its Catalan corpus version, is used to perform training and testing. Noisy audio samples are obtained adding AWGN with 0 dB SNR to clean speech signals. PESQ and STOI objective metrics are used to measure system performance. System evaluation shows positive results when using SNR levels as in training, while overall intelligibility deteriorates when using higher SNR levels due to phase distortion.

Resum

En aquest TFG s'implementa un sistema de reducció de soroll en senyals de parla basat en una xarxa neuronal convolucional (CNN). Un mòdul de transformació de característiques extrau la magnitud i la fase del senyal d'àudio a partir del càlcul de la STFT. La CNN mapeja la magnitud de l'espectre del senyal d'entrada amb soroll a una magnitud d'espectre amb soroll reduït. Un mòdul de reconstrucció del senyal reconstrueix el senyal d'àudio amb soroll reduït a partir del càlcul invers de la STFT. S'utilitza com a base de dades per a l'entrenament i test de la CNN el *corpus* català de Mozilla Common Voice. Per obtenir el senyal amb soroll s'afegeix AWGN amb SNR = 0 dB als senyals d'àudio nets. Tot el sistema s'implementa sobre MATLAB. Per a la mesura de l'eficàcia del sistema s'utilitzen les mètriques PESQ i STOI. Les proves sobre el sistema mostren resultats positius per a relacions SNR iguals a les utilitzades durant l'entrenament, però l'augment de la distorsió per a relacions SNR majors disminueix la intel·ligibilitat general.

A Vio

Tabla de contenido

Lista de figuras

Lista de tablas

Lista de siglas y acrónimos

ANN Artificial Neural Network

AWGN Additive White Gaussian Noise

CNN Convolutional Neural Network

DL Deep Learning

DNN Deep Neural Network

FNN Feedforward Neural Network

GUI Graphic User Interface

ISTFT Inverse Short-Time Fourier Transform

LTSM Long Short-Term Memory

ML Machine Learning

MLP Multilayer Perceptron

MMSE Minimum Mean Square Error

PESQ Perceptual Evaluation of Speech Quality

ReLu Rectified Lineal unit

RNN Recurrent Neural Network

SNR Signal to Noise Ratio

STFT Short-Time Fourier Transform

STOI Short-Time Objective Intelligibility

Capítulo 1 Introducción

Este TFG se enmarca en el campo de las aplicaciones del aprendizaje automático al procesamiento de la señal. Se explora y propone la implementación de un sistema de reducción de ruido y reconstrucción en señales de audio. El objetivo es desarrollar una herramienta que, ante una señal con ruido en la entrada, permita obtener una representación en la salida lo más fiel posible a la señal original. Para ello, se propone el uso de modelos de aprendizaje automático supervisado basados en redes neuronales. En concreto, se desarrolla una red neuronal convolucional con el fin de mapear las características deseadas de la señal de entrada con ruido a una señal de salida con ruido reducido.

1.1 Justificación y relevancia

La comunicación es un proceso inherente al ser humano, y el habla es la forma más cómoda y directa de llevarlo a cabo, es un elemento flexible e intuitivo. Así, es natural que, a partir del rápido crecimiento, la investigación y la popularización de los sistemas de comunicación y de las interfaces de control mediante voz, hayan surgido dispositivos que facilitan la comunicación como smartphones, *tablets*, sistemas de teleconferencia, … y herramientas como el reconocimiento automático del habla que permiten la comunicación en circunstancias en las que no existía: televisiones inteligentes, hogares inteligentes, altavoces inteligentes, …

En un escenario ideal, no deberían alterarse la calidad y la inteligibilidad de la señal original recibida respecto a la emitida. En la práctica, sin embargo, la señal se degrada por la presencia de ruido de fondo y reverberaciones. Además, aunque la libertad y la autonomía que proporcionan las tecnologías anteriores han permitido la comunicación fuera de entornos controlados, esto mismo también ha generado nuevos desafíos para mantener la inteligibilidad de la comunicación, que puede afectar también a los sistemas de codificación y reconocimiento del habla. En este contexto surge el *speech enhancement*.

El objetivo del *speech enhancement* es mejorar la inteligibilidad y la calidad percibida de las señales degradadas utilizando técnicas de procesamiento de la señal. El campo más

importante en el que se centra es el de la reducción de ruido, estudiando las naturalezas y las características variables de los elementos que corrompen la señal. Las aplicaciones más importantes de las técnicas de reducción de ruido son las comunicaciones móviles, los sistemas de teleconferencia, el reconocimiento del habla y los audífonos [1]. En la mayoría de los sistemas de comunicación y de procesado de voz utilizados en entornos ruidosos es necesario incorporar módulos de reducción de ruido para que puedan funcionar correctamente. Por ejemplo, en sistemas de reconocimiento automático del habla, la calidad del audio de entrada tiene un papel importante en la capacidad de reconocimiento, independientemente del funcionamiento del resto del sistema. Por otra parte, en audífonos la amplificación del sonido ambiente supone un problema de inteligibilidad si no se cuenta con estrategias de reducción de ruido.

Durante las últimas décadas, se han desarrollado diferentes métodos para la reducción de ruido. Ejemplos son la sustracción espectral, filtrado Wiener, *minimum mean square error* (MMSE) y sus variantes [2], aunque estos métodos eran susceptibles de producir objetos espurios en forma de "ruido musical" [3] o no eran eficientes al eliminar ruido no estacionario. En general, estas técnicas están basadas en algoritmos de filtrado, sustracción espectral, métodos basados en modelos o wavelet [4].

Sin embargo, el resurgimiento del interés en las redes neuronales a partir del final del siglo XX ha favorecido la investigación y aplicación de técnicas de aprendizaje automático al problema de la mejora de la calidad de la señal [2], [5], [6], [7]. Además, la filosofía de código abierto, la facilidad de acceso a conjuntos de datos extensos preparados para el entrenamiento de las redes y a la publicación de modelos como Wavenet [8], han permitido la creación de proyectos como Magenta [9], que abren nuevos horizontes sobre los que explorar en el procesado del audio y al mismo tiempo sirven como base de nuevas soluciones para problemas más antiguos.

1.2 Motivación personal

Este trabajo final se enmarca en el contexto de las aplicaciones del procesamiento de la señal de audio y aúna y utiliza los conocimientos adquiridos a lo largo de todo el grado. La temática elegida intenta satisfacer el mismo conjunto de intereses que motivaron en buena medida el inicio de los estudios y la elección del itinerario de sistemas audiovisuales.

Por una parte, el procesado de audio. Relacionado con hobbies personales y posteriormente con vida profesional, ya sea desde un punto de vista *amateur* o algo más en profundidad. Tratado en varias asignaturas del grado, se pretende con este trabajo ahondar en ese conocimiento.

Por otra parte, las redes neuronales aplicadas al audiovisual. Han generado mucha expectación en los últimos años, sobre todo las relacionadas con los modelos generativos. Este trabajo servirá para conocer e investigar diferentes modelos, su funcionamiento y la forma de implementarlos.

Además, existe la motivación de conocer, aprender y desarrollar aplicaciones en nuevos lenguajes de programación, como puede ser Python, y los *frameworks* y demás facilidades de computación *open source* que existen en el entorno del modelado del aprendizaje automático, o el *toolbox* Deep Learning de MATLAB.

Supone también una motivación practicar la investigación y búsqueda de información general de fuentes de referencia científicas y técnicas.

Por último, otro motivador del trabajo elegido es que supondrá el primer proyecto relacionado con aprendizaje automático aplicado al procesamiento de audio presentado en los estudios de grado de la UOC y podrá servir de referencia e inspiración a otros alumnos que deseen adentrarse en la temática.

1. 3 Objetivos

Se plantean tres objetivos principales:

- Implementar y entrenar una red neuronal que permita mejorar la inteligibilidad de una señal de audio deteriorada por el ruido.
- Cuantificar la eficacia del resultado.
- Comprender los fundamentos del funcionamiento de las redes neuronales artificiales.

Y como objetivo secundario:

- Implementar una GUI sencilla que permita aplicar la red neuronal resultante a un conjunto de audios de test y obtener el resultado de la reducción del ruido.

1.4 Metodología

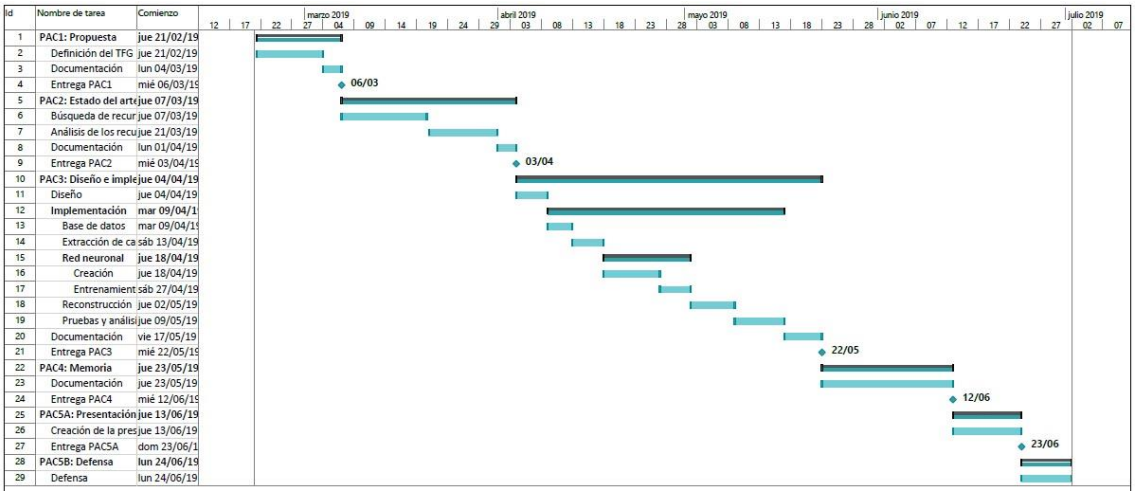
En este proyecto se seguirá la arquitectura propuesta en [10], donde se desarrolla una red neuronal convolucional aplicada a la reducción de ruido.

Constará de cuatro etapas. En la primera etapa se creará una base de datos a partir del corpus CommonVoice de Mozilla [11] y se generarán los datos de entrenamiento añadiendo ruido blanco gaussiano. En la segunda etapa se implementará un módulo de extracción de magnitud. En la tercera etapa se implementará el modelo de red neuronal convolucional para mapear estas características a las deseadas. En la cuarta etapa se implementará un módulo de reconstrucción de la señal a partir de la nueva magnitud y la fase original.

Se utilizará MATLAB para la codificación y construcción de todos los elementos en todas las etapas por familiaridad en el uso y por potencia en el tratamiento y computación de redes neuronales.

1.5 Planificación

El siguiente diagrama de Gantt sigue la planificación del proyecto, basada en el plan de entregas de pruebas de evaluación.



1.6 Sumario de productos obtenidos

Los productos obtenidos en este TFG son:

Implementación del sistema

Conjunto de funciones en lenguaje M e interfaz gráfica.

Memoria del proyecto

Documentación de la teoría básica utilizada, diseño e implementación del proyecto, pruebas sobre el sistema y conclusiones.

1.7 Descripción del resto de capítulos

Los capítulos restantes están estructurados de la siguiente manera:

Capítulo 2

Este capítulo presenta nociones teóricas básicas sobre Redes Neuronales Artificiales y *speech enhancement.*

Capítulo 3

Este capítulo presenta el diseño del sistema de reducción de ruido basado en una Red Neuronal Convolucional.

Capítulo 4

Este capítulo presenta la implementación del sistema utilizando el diseño desarrollado en el capítulo 3.

Capítulo 5

Este capítulo presenta los resultados de las pruebas realizadas sobre el sistema implementado y el análisis de su rendimiento.

Capítulo 6

Este capítulo presenta las conclusiones del proyecto basadas en los resultados obtenidos y relacionadas con los objetivos establecidos en el capítulo 1.

Capítulo 7

Este capítulo presenta posibles líneas de trabajo futuro.

Anexo

El anexo contiene la implementación en MATLAB del sistema de reducción de ruido desarrollado en este TFG.

Capítulo 2 Estado del arte

La aplicación de sistemas de aprendizaje automático basados en redes neuronales al procesamiento de señales de audio es un campo amplio que persigue diferentes objetivos. Aun así, existe un marco teórico básico necesario. Se expone en este capítulo el núcleo del funcionamiento de estos sistemas, la neurona artificial y los algoritmos y diferentes arquitecturas que permiten el aprendizaje. A continuación, se realiza una breve descripción del Deep Learning y de sus implementaciones más habituales. Se finaliza con una introducción al *speech enhancement*, y se pone en común con las redes neuronales profundas más habituales.

2.1 Redes Neuronales Artificiales

Las redes neuronales artificiales (ANN) son modelos computacionales pertenecientes al campo del *Machine Learning* (ML). Surgen a partir de la explicación conexionista del sistema cognitivo humano [12] y se inspiran en el funcionamiento de las Redes Neuronales Biológicas. Pueden definirse como estructuras formadas por elementos simples adaptativos capaces de procesar datos y representar conocimiento de grandes cantidades de datos de forma paralela [13]. Las ANN realizan un mapeo de datos de entrada a datos de salida con relaciones no lineales, de forma que pueden ser entrenadas para realizar tareas de clasificación y regresión. Existe un gran número de arquitecturas ANN, pero todas tienen en común un bloque básico: la neurona artificial.

2.1.1 Neurona artificial

La neurona artificial se divide en varios componentes: las entradas, la salida, los pesos y la función de activación [13]. La neurona recibe conexiones desde la entrada de la ANN o desde la salida de otros nodos, dependiendo de la arquitectura. Cada una de las conexiones tiene asociado un peso que determina la aportación de la información recibida. La suma ponderada de todas las conexiones se acumula en la neurona. La salida de la neurona, regularizada por un factor *bias*, se computa pasando esta suma a través de una función de activación.

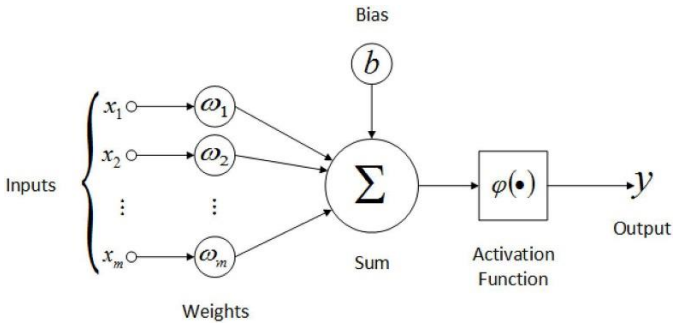


Figura 2.1 Esquema de una neurona artificial [14].

De esta forma, la salida de la neurona queda definida por [15]:

$$y = \varphi\left(\sum_{i=1}^{N} \omega_i \, x_i + \mathrm{b}\right)$$

Las funciones de activación más habituales son la función sigmoide, la función $tanh$ y la función $softmax$ [15].

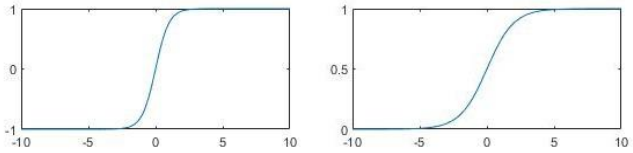


Figura 2.2. Función tanh (izquierda) y función sigmoide (derecha).

La aplicación a una neurona artificial de un algoritmo de aprendizaje da lugar al perceptrón [13], [16].

2.1.2 Perceptrón

El perceptrón es la arquitectura *Feedfordward Neural Network* (FNN) más simple. Si se toma como referencia el modelo de neurona artificial (Figura 2.1) y definimos $\bar{x}$ como el vector de entrada, $\overline{w}$ como el vector de pesos y d como la salida deseada, el algoritmo de aprendizaje que determina $\overline{w}$ es:

1. Inicialización aleatoria de $\overline{w}$.

2. Se introducen los datos de entrada $\bar{x}$ y se presenta d a la salida.

3. Si en la salida del perceptrón $y \neq d$, se actualizan los pesos.

$$w_i \leftarrow w_i + \alpha e_i x_i$$

   Donde $e = d - y$ y α es el ratio de aprendizaje.

4. Se repiten los pasos 2 – 3 hasta lograr un error definido.

Esta arquitectura presenta una limitación principal. Será convergente sólo para clasificaciones linealmente separables [17]. Por tanto, será incapaz de implementar la función XOR.

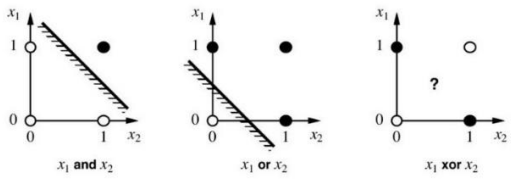


Figura 2.3. La función XOR no es linealmente separable [18].

2.1.3 Perceptrón multicapa

Una arquitectura que permite clasificar datos linealmente no separables es el perceptrón multicapa (MLP). Aquí, varias neuronas artificiales se combinan para formar una red organizada en capas. El MLP más sencillo está compuesto por una capa de entrada, una capa de salida y una capa oculta (Figura 2.4).

Cada capa i presenta una matriz de pesos $W^{(i)}$ y un vector *bias* $b^{(i)}$ [15], de forma que, para un nodo determinado de la capa oculta,

$$h^{(1)} = \varphi\left(W^{(1)T}x + b^{(1)}\right)$$

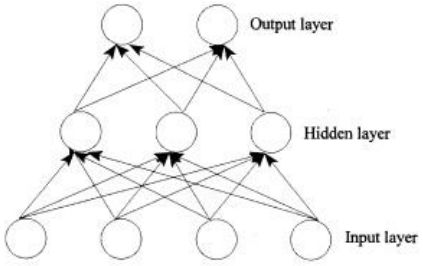


Figura 2.4. Perceptrón multicapa básico [13].

Y para la capa de salida,

$$y = \varphi\left(W^{(2)T} h^{(1)} + b^{(2)}\right)$$

Así, los valores en los nodos de la capa de salida dependen de la función de activación y de las salidas de los nodos de la capa oculta, que a su vez dependen de su función de activación y de los pesos y valor del vector de entrada de la capa de entrada. Este tipo de configuración no permite aplicar el algoritmo de aprendizaje mostrado para el perceptrón, definido sólo para la actualización de pesos en la capa de salida.

2.1.4 Aprendizaje en Redes Neuronales Multicapa

Se ha visto que es necesario un algoritmo de aprendizaje que permita actualizar los valores de los parámetros en capas ocultas. Se introducen a continuación la propagación hacia atrás, o *backpropagation*, y dos conceptos necesarios para su funcionamiento, la función de coste y la optimización.

2.1.4.1 Función de coste

Durante el entrenamiento, la función de coste determina la medida del error de la red neuronal. El objetivo es minimizarla. Las funciones de coste más habituales son la función error cuadrático medio [15], [19]

$$J = \sum_{i=1}^{M} \frac{1}{2}(d_i - y_i)^2$$

Y la entropía cruzada,

$$J = \sum_{i=1}^{M} \{-d_i \ln y_i - (1 - d_i) \ln(1 - y_i)\}$$

Donde M es el número de nodos de salida.

2.1.4.2 Optimización

Para encontrar los valores de los parámetros de la red que minimizarán la función de coste, es necesario un algoritmo de optimización [19, p. 271]. Los algoritmos de optimización utilizan el gradiente de la función de coste para hallar un mínimo [20]. En este sentido, habitualmente se emplean algoritmos basados en el gradiente descendente.

Gradiente descendente *batch* computa el gradiente de la función de coste para todos los parámetros de todo el conjunto de datos de entrenamiento para cada actualización.

Gradiente descendente estocástico computa el gradiente de la función de coste para un parámetro de uno de los pares entrada/objetivo por actualización.

Gradiente descendente *mini-batch* computa el gradiente para un conjunto de parámetros por actualización

Además, es posible implementar optimizaciones de estos métodos, como *momentum* [2] y Adam [10].

2.1.4.3 Retropropagación

El algoritmo que utilizan los sistemas de gradiente descendente para calcular el gradiente de la función de coste es el algoritmo de retropropagación o *backpropagation*. En primer lugar, durante el entrenamiento de la red, el error se calcula propagando la entrada a través

de los nodos para encontrar la salida que definen los parámetros para un determinado par entrada/objetivo. A continuación, el error se propaga en sentido contrario, hacia la capa de entrada, para encontrar en qué medida los diferentes parámetros han contribuido a ese error.

2.2 Aprendizaje profundo

El término aprendizaje profundo o *Deep Learning* (DL) se utiliza para describir el sistema de aprendizaje en arquitecturas profundas. En el campo de las redes neuronales, estas arquitecturas se caracterizan por presentar dos o más capas ocultas y se suelen denominar *Deep Neural Networks* (DNN). Las arquitecturas profundas son capaces de proporcionar múltiples niveles de abstracción de los datos de entrada, de forma similar al córtex humano [21] y permiten representar funciones altamente variables y no lineales [22]. Dentro de las DNN, existen dos modelos principales, las Redes Neuronales Convolucionales (CNN) y las Redes Neuronales Recurrentes (RNN).

2.2.1 Redes Neuronales Convolucionales

Las CNN son un tipo de DNN especializadas en procesar datos matriciales, como matrices de una dimensión con la representación del muestreo de una señal de audio o matrices de dos dimensiones con píxeles [19].

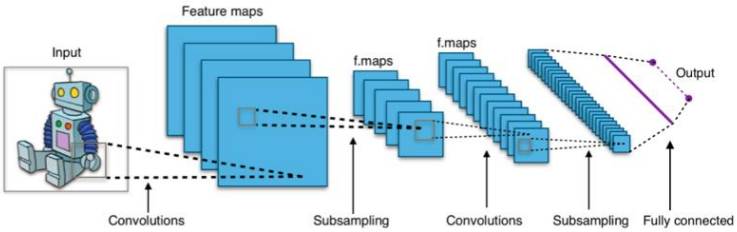


Figura 2.5. Arquitectura de una CNN aplicada a imágenes. Fuente: https://ca.wikipedia.org/wiki/Fitxer:Typical_cnn.png

Las capas en las CNN se dividen en tres etapas. En la primera etapa, la capa aplica una serie de convoluciones para producir un conjunto de activaciones lineales. En la segunda etapa, cada una de estas activaciones pasan por una función de activación no lineal, como

ReLu [23]. En la tercera etapa se utiliza una función *pooling* de reducción de dimensiones y promediado que permite forzar un patrón de conectividad entre elementos adyacentes y reducir la carga computacional [19].

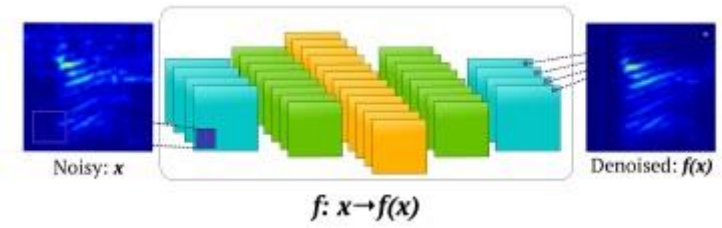


Figura 2.6. Arquitectura de una CNN aplicada a audio [10].

2.2.2 Redes Neuronales Recurrentes

Las Redes Neuronales Recurrentes (RNN) son un tipo de redes neuronales que operan sobre datos secuenciales y, en oposición a las FNN, incorporan conexiones a valores temporales previos. En una RNN, dada una secuencia de entrada $x = (x_1, \dots, x_T)$, se computan la secuencia del vector oculto $h = (h_i, \dots, h_T)$ y la secuencia del vector de salida $y = (y_1, \dots, y_T)$, iterando desde $t = 1$ hasta T [24],

$$h_T = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

Donde W denota las matrices de pesos, b los vectores *bias* y $\mathcal{H}$ es la función de activación de la capa oculta.

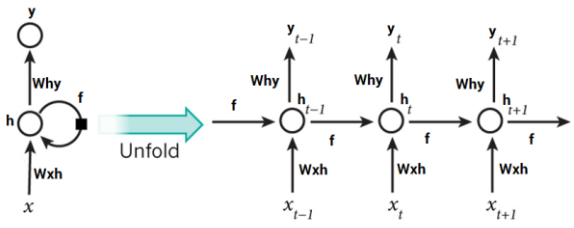


Figura 2.7. Conexión de nodos en una RNN. Desarrollo de la red durante el entrenamiento. Fuente: https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/

Durante el entrenamiento, las RNN se desarrollan en FNN, lo que genera estructuras muy profundas y complejas. El cálculo del gradiente para estas redes desarrolladas puede incluir varias multiplicaciones de un peso por sí mismo. Esto provoca un problema de desvanecimiento de gradiente, el error no llega a propagarse por toda la red [19]. Con el fin de solucionar este problema se implementan las Long Short-Term Memory (LTSM) [25]. Las redes recurrentes LTSM están organizadas en celdas, con conexiones hacia sí mismas controladas por compuertas, que almacenan el estado temporal de la red.

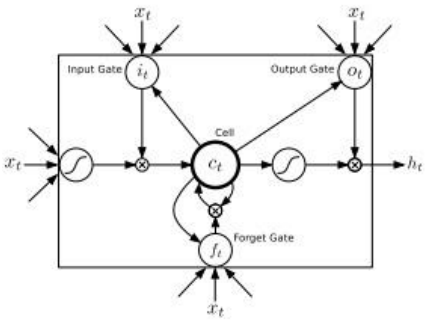


Figura 2.8. Celda en una red LTSM [24].

2.3 Speech enhancement

El objetivo principal del *speech enhancement* es mejorar la inteligibilidad y la calidad subjetiva de una señal degradada utilizando técnicas de procesamiento de la señal. Tradicionalmente, los algoritmos empleados para lograrlo se clasifican en cuatro categorías [25]:

Algoritmos de sustracción de espectro, donde se utiliza la estimación del espectro del ruido durante periodos en los que el habla no está presente para poder sustraerlo durante los tramos hablados.

Algoritmos basados en modelos estadísticos, donde se aplica el principio de estimación estocástica.

Algoritmos de subespacio, donde se asume la señal limpia como un subespacio de la señal degradada.

Algoritmos de máscara binaria, donde se aplica una máscara binaria a la representación tiempo-frecuencia de la señal degradada para eliminar ciertos elementos frecuenciales aplicando un umbral al ratio Señal/Ruido (SNR).

2.3.1 Deep learning aplicado al speech enhancement

La capacidad de las ANN para modelar funciones de mapeo no lineales complejas las convierten en una herramienta ideal para el *speech enhancement* tanto en el mapeo en el dominio temporal como en el dominio transformado [26]. Los sistemas que aplican técnicas basadas en el dominio temporal asumen que las características en las capas ocultas adoptan formas que permiten extraer directamente la señal limpia de la señal degradada [27]. Los sistemas basados en el dominio transformado transforman previamente la señal del habla a un dominio que permita extraer de la señal características con mejores propiedades para el reconocimiento y entrenan la ANN de forma que pueda mapear las características deterioradas a las características limpias, para después transformarlas de nuevo en habla. Las características más habituales en este caso son las espectrales [2], [10], [28].

Existen varias aproximaciones al *speech enhancement* desde el punto de vista de la arquitectura de la red. Se han implementado sistemas basados en DNN [2], [27], [29], basados en *autoencoders* [7], [28]. En [10] se expone un sistema basado en CNN que permite reducir el número de parámetros de la red mientras se mantiene una eficiencia similar al resto de implementaciones.

2.3.2 Medida del speech enhancement

Como se ha comentado, el objetivo del *speech enhancement* es mejorar la calidad subjetiva de una señal de habla. Se hace necesario por tanto un método que permita cuantificar esta mejora. El *Perceptual Evaluation of Speech Quality* (PESQ) es un algoritmo de medida de la calidad propuesto por la ITU-T [30], [31], que fue implementado con el objetivo de evaluar la degradación en sistemas de telecomunicaciones. El PESQ se ha convertido en un estándar en la evaluación de la mejora subjetiva de la señal en sistemas *speech enhancement* [2], [10], [32], [33]. La computación del algoritmo requiere las señales limpia y degradada y se basa en la

comparación espectral de las dos teniendo en cuenta sonoridad y efectos de enmascaramiento.

Capítulo 3 Diseño

En este capítulo se define el diseño del sistema de reducción de ruido implementado. En el primer apartado se describen sus componentes y la relación entre ellos para sus diferentes fases. En el último apartado se contextualizan la base de datos y el entorno de desarrollo utilizados.

3.1 Sistema speech enhancement

Se diseña un sistema *speech enhancement* basado en el propuesto en [10]. Las redes neuronales convolucionales, por su diseño estructurado en capas de convolución, son el tipo de arquitectura *Deep Learning* que resulta más eficiente en las aplicaciones relacionadas con el procesamiento de imágenes y que, en general, presenta un menor número de parámetros necesarios en la implementación. La idea básica detrás del desarrollo propuesto en [10] es aprovechar estas características y utilizar la *imagen* de la magnitud del espectro de un audio como entrada de la red, en contraste con enfoques secuenciales propios de arquitecturas RNN o DNN [2], [32].

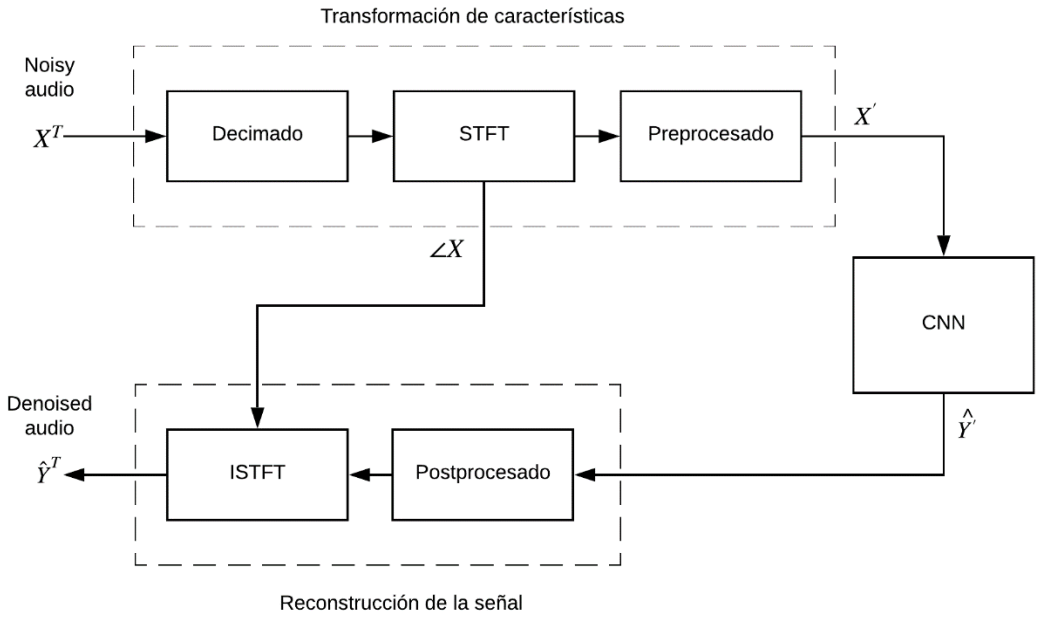


Figura 3.1. Esquema del sistema de reducción de ruido.

En concreto, una red neuronal convolucional mapea la magnitud del espectro de una señal de entrada con ruido a una magnitud de espectro de una señal de salida con ruido reducido. Un módulo de transformación de características extrae la magnitud y la fase del espectro de la señal a partir del cálculo de la *Short-Time Fourier Transform* (STFT) y la adapta a la entrada de la CNN. Un módulo de reconstrucción de la señal adapta la salida de la CNN para realizar el cálculo de la *Inverse Short-Term Fourier Transform* (ISTFT) a partir de la magnitud obtenida de la red y de la fase obtenida en el módulo de extracción.

Debido a que el sistema de percepción humano no es especialmente sensible a las diferencias de fase del espectro, esta característica se utiliza sólo en la reconstrucción de la señal y no en la predicción de la CNN.

Se describen en los siguientes subapartados los diferentes módulos que componen el sistema.

3.1.1 Módulo de transformación de características

La señal de entrada, con ruido, se remuestrea a 8 KHz para reducir la carga computacional aprovechando que la mayor parte de información en el habla se sitúa por debajo de los 4 KHz. A continuación, se obtienen los vectores fase y magnitud utilizando la STFT con un enventanado Hamming de 32 ms (256 muestras) con un encabalgamiento de 8 ms (64 muestras). Por simetría del espectro, sólo se utiliza la mitad del espectro resultante. Para proporcionar contexto acústico y mejorar la eficiencia del sistema, para cada vector magnitud se consideran los siete vectores anteriores. La señal de entrada a la red tiene por tanto un tamaño 129x8xNmuestras.

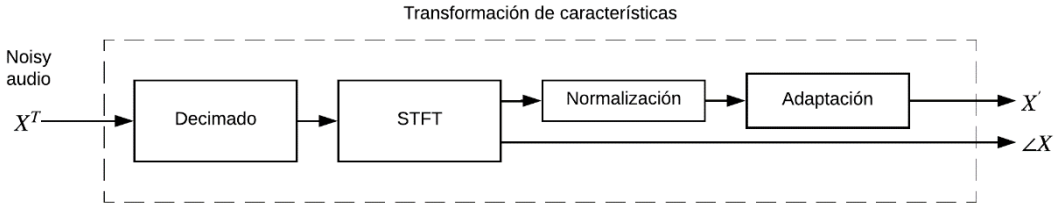


Figura 3.2. Esquema del módulo de transformación de características.

Se introduce también un submódulo de normalización para escalar la señal por la media y desviación típica de los datos con ruido de entrenamiento, de forma que las entradas a la red resulten homogéneas en términos de amplitud y variación, centradas en cero y con $\sigma = 1$:

$$Mag\_normalizada = \frac{mag - \overline{mag\_entrenamiento}}{\sigma_{mag\_entrenamiento}}$$

Por último, el submódulo de adaptación adapta las dimensiones del vector de entrada (3 dimensiones) a las esperadas por la capa de entrada de una CNN (4 dimensiones).

3.1.2 Módulo de reconstrucción de la señal

El esquema de la reconstrucción de la señal de audio a partir de la salida de la CNN se muestra en la Figura 3.3. El vector de salida de la CNN se escala a partir de la media y desviación típica obtenidas para el set de audios sin ruido en la fase de entrenamiento:

$$Magnitud_{escalada} = mag_{salida} \cdot \sigma_{mag\_entrenamiento} - \overline{mag\_entrenamiento}$$

A continuación, se reconstruye la parte simétrica de la magnitud y se incluye la fase obtenida para la señal con ruido durante la transformación de características. Se computa entonces la ISTFT para formar la señal de audio completa con reducción de ruido.

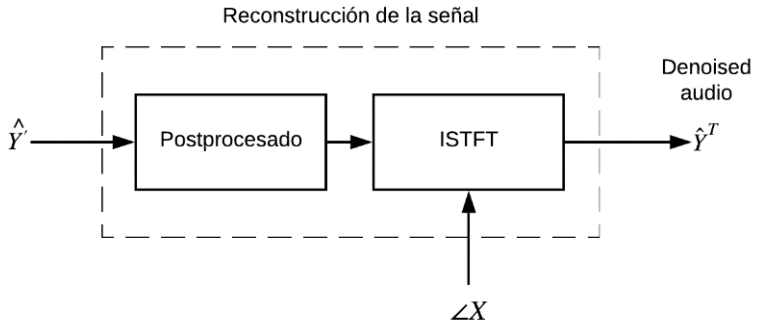


Figura 3.3. Esquema del módulo de reconstrucción de la señal.

3.1.3 Red Neuronal Convolucional (CNN)

El elemento central del sistema *speech enhancement* implementado es una CNN similar a la desarrollada en [10]. El objetivo, como se ha comentado, es relacionar una determinada magnitud espectral en la entrada con una magnitud en la salida asociada a una señal con ruido y la misma señal limpia, respectivamente. El diseño de la CNN se puede dividir en las dos etapas descritas a continuación: configuración y entrenamiento.

3.1.3.1 Diseño de la CNN

La CNN consta de 16 capas convolucionales, además de las capas de entrada y salida, formadas por 18/30/8 filtros de tamaño 9/5/9 y seguidas cada una de ellas por una capa de normalización y una capa de activación ReLu (Tabla 3.1). Se utiliza *backpropagation* como algoritmo de cálculo de la función de coste mediante el método gradiente descendente con un *mini-batch* de 64 parámetros y algoritmo de optimización Adam. Se define un ratio de aprendizaje $lr = 0.0015$ y los valores *gradient decay factor* $\beta_1 = 0.9$, *squared decay factor* $\beta_2 = 0.999$ y épsilon $\epsilon = 10^{-8}$ recomendados para Adam [34].

Capa	Tamaño	Número de filtros
Entrada	129x8	
Convolucional	9x8	18
Normalización		
ReLu		
Convolucional	5x1	30
Normalización		
ReLu		
Convolucional	9x1	8
Normalización		
ReLu		
Convolucional	9x1	18
Normalización		
ReLu		
Convolucional	5x1	30
Normalización		
ReLu		
Convolucional	9x1	8
Normalización		
ReLu		
Convolucional	9x1	18
Normalización		
ReLu		
Convolucional	5x1	30
Normalización		
ReLu		
Convolucional	9x1	8
Normalización		
ReLu		
Convolucional	9x1	18
Normalización		

ReLu		
Convolucional	5x1	30
Normalización		
ReLu		
Convolucional	9x1	8
Normalización		
ReLu		
Convolucional	9x1	18
Normalización		
ReLu		
Convolucional	5x1	30
Normalización		
ReLu		
Convolucional	9x1	8
Normalización		
ReLu		
Salida	129x1	1

Tabla 3.1. Capas de la CNN

Parámetro	Valor
Algoritmo coste	*Backpropagation*
Optimización	Adam
Batch	64
Learn rate	0.0015
β_1	0.9
β_2	0.999
ϵ	10^{-8}

Tabla 3.2. Parámetros de la CNN.

3.1.3.2 Fase de entrenamiento

El esquema de la fase de entrenamiento de la CNN se muestra en la Figura 2.4. La señal con ruido de entrada se obtiene añadiendo ruido blanco gaussiano (AWGN) con SNR = 0dB a la señal limpia que actúa como objetivo de la red. El módulo de transformación de características es idéntico al empleado en la fase *speech enhancement*.

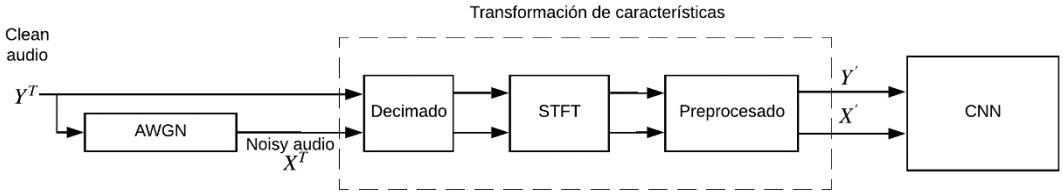


Figura 2.4. Esquema de la fase de entrenamiento de la CNN.

Parte del conjunto de audios de entrenamiento se destinan a validación para evitar *overfitting* en la red.

3.2 Corpus y entorno de desarrollo

Se expone a continuación una descripción de la base de datos utilizada y del software necesario para la implementación del sistema.

3.2.1 Mozilla Common Voice

Los audios utilizados para el entrenamiento y las pruebas del sistema se extraen del corpus de voz Mozilla Common Voice [11]. El Common Voice es un proyecto de Mozilla creado con el objetivo de desarrollar una base de datos de frases habladas en diferentes idiomas proporcionadas y validadas por los usuarios. Common Voice facilita los archivos de datos con licencia CC-0. El corpus catalán está compuesto por un total de 92 horas de audio divididas en 33000 locuciones de 1639 hablantes diferentes. Los archivos de audio están muestreados a 16 KHz.

3.2.2 MATLAB

El sistema se implementa sobre MATLAB R2019a. Se utiliza la versión de demostración de dos *Toolbox* no incluidos en la licencia académica:

Audio *Toolbox*

Conjunto de herramientas para el procesado de audio, análisis del habla y mediciones acústicas. Habilita el uso de estructuras *Audio Datastore*, que permiten manejar colecciones de archivos de audio de tamaño superior a la memoria.

Deep Learning Toolbox

Proporciona los elementos necesarios para la construcción e implementación de redes neuronales convolucionales y redes LSTM y el uso, junto a la *Parallel Computing Toolbox*, de GPUs para el entrenamiento.

Capítulo 4 Implementación

En este capítulo se detalla la implementación de los módulos definidos en el diseño del sistema. La implementación consta de dos fases: fase de entrenamiento y fase de test. Durante la fase de entrenamiento se establecen los parámetros que la CNN utilizará para mapear una magnitud de espectro de audio ruidoso a una magnitud de audio limpio a partir de esos mismos dos audios. En la fase de test, la CNN utiliza los parámetros aprendidos para inferir la magnitud del espectro de la señal limpia a partir de la señal con ruido [35].

Para la carga en el sistema de los audios de la base de datos se utiliza una estructura AudioDatastore generada a partir de la función `audioDatastore`. Este tipo de elementos permiten ser almacenados en memoria de forma parcial en *Tall Arrays* y optimizar la evaluación de funciones sobre ellas con el comando `gather`. En este sentido, se define la función `getAudioData(set)` que devuelve un AudioDatastore con los elementos del set 'train' o 'test'.

4.1 Fase de entrenamiento

El esquema de implementación de la fase de entrenamiento se muestra en la Figura 4.1. El conjunto de audios de entrenamiento es procesado por el bloque de transformación de características, donde se obtiene la señal ruidosa y se computan los vectores magnitud, y por el bloque de adaptación a la red, donde se normalizan los valores de todo el set y se adaptan las dimensiones a las esperadas por la red. A continuación, una vez obtenidos los elementos de entrada y objetivo se inicia el entrenamiento de la red CNN.

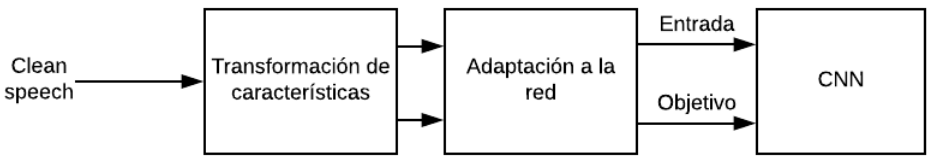


Figura 4.1. Esquema de la implementación de la fase de entrenamiento.

A efectos de entrenamiento se utilizará un subconjunto de 1250 muestras de audio de la base de datos, de los cuales un 20% (250 muestras) estarán destinadas a validación. La fase de entrenamiento queda definida por la siguiente función:

Función	Descripción
`[denoisingCNN, inputsMean, inputsStd, targetsMean, targetsStd] = trainCNN ()`	Devuelve un objeto CNN entrenado a partir del set de entrenamiento.

Tabla 4.1. Función trainCNN.

4.1.1 Transformación de características

El bloque de transformación de características está compuesto por tres subprocesos: remuestreo, AWGN y obtención de vectores magnitud.

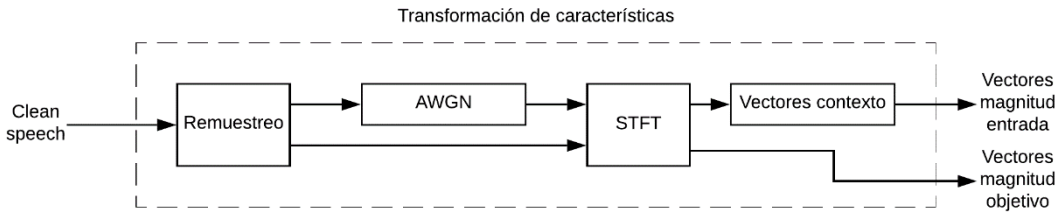


Figura 4.2. Esquema de la implementación del bloque de transformación de características.

Remuestreo

Aplica un decimado a la señal para obtener un muestreo de 8000 Hz tras asegurar que la longitud del audio es proporcional a factor de decimado.

AWGN

Genera el audio con ruido blanco gaussiano con SNR = 0dB que se utilizará para el entrenamiento.

Obtención de vectores magnitud

Se obtienen los vectores magnitud y fase para la señal limpia y los vectores magnitud con contexto acústico para la señal con ruido. En ambos casos se elimina la mitad simétrica redundante del espectro, por lo que quedan vectores de tamaño 129xNmuestras y 129x8xNmuestras, respectivamente.

Para este bloque se definen las siguientes funciones:

Función	Descripción
`[audio] = audioLengthExtend (audio)`	Aumenta la duración de la señal de audio de forma que el número de muestras sea proporcional al decimado.
`[speechMag, speechPhase] = computeStft (speech, window, shift, windowLength, nMagnitude)`	Devuelve fase y magnitud de la STFT de la señal
`[inputVectors] = getInputVectors (noisySpeechSTFT, nInputVectors, nMagnitude)`	Genera los vectores magnitud con contexto acústico de la entrada.

Tabla 4.2. Funciones del bloque de transformación de características.

4.1.2 Adaptación a la red

Como se introduce en los apartados de diseño, es necesario normalizar los datos de entrada a la red para optimizar el entrenamiento y adaptar sus dimensiones de acuerdo a las esperadas por la capa de entrada. El bloque de adaptación se sitúa justo antes de la entrada a la red y está compuesto por dos subprocesos: normalización y adaptación de dimensiones.

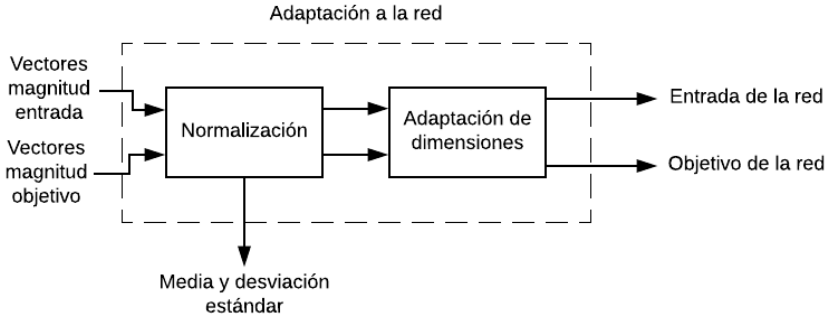


Figura 4.3. Esquema de la implementación del bloque de adaptación a la red.

Normalización

Normaliza el set de vectores magnitud por su media y desviación típica. Los valores se almacenan en variables para el posterior escalado en la fase de test.

Adaptación de dimensiones

Aumenta las dimensiones de los vectores magnitud de forma que coincida con las de la capa de entrada de la CNN.

En este bloque se definen las siguientes funciones:

Función	Descripción
`[mags, inputsMean, inputsStd] = magNormalization (mags)`	Normaliza el set de entrada y devuelve su media y desviación típica.
`[netInputs] = adaptInputShape (inputs)`	Adapta las dimensiones de los vectores de entrada a las de la red.
`[netTargets] = adaptTargetShape (targets)`	Adapta las dimensiones de los vectores objetivo a las de la red.

Tabla 4.3. Funciones del bloque de adaptación a la red.

4.1.3 Selección de los datos de validación y configuración

Los datos de validación permiten detectar en qué punto del entrenamiento comienza a producirse *overfitting*. Es habitual destinar un 80% de los datos de entrenamiento a validación [10]. Justo antes del inicio del entrenamiento, se obtienen los datos de validación y se define la configuración de la CNN con los parámetros definidos en el diseño del sistema. Además, se establece un máximo de 3 *epochs* debido a las limitaciones en hardware y al aumento proporcional en el tiempo de entrenamiento que conllevaría un valor más alto. Sin embargo, se observa tras el entrenamiento que este valor es suficiente para la convergencia de la red.

Función	Descripción
`[trainInputs, trainTargets, validationInputs, validationTargets] = splitTrainToValidation (inputs, targets)`	Divide el set en muestras de entrenamiento y muestras de validación.
`[layers, options] = configureCNN (trainInputs, validationInputs, validationTargets)`	Configura las capas y los parámetros de la red.

Tabla 4.4. Funciones para la selección de datos y la configuración

4.2 Fase de test

El esquema de implementación de la fase de test se muestra en la Figura 4.4. El audio de entrada a la red se procesa de la misma forma que en la fase de entrenamiento. A la salida de la CNN se obtiene el vector magnitud de la predicción para la reducción de ruido y se procesa a través de un bloque de escalado por los valores media y desviación típica del set de entrenamiento obtenidos en la fase de entrenamiento. A continuación, un bloque de restauración de espectro devuelve al vector a sus dimensiones normales tras el paso por la CNN, duplica y centra el espectro y añade la información de fase extraída en el bloque de transformación de características. Por último, se reconstruye la señal de audio calculando la ISTFT.

La fase de test queda definida por la siguiente función:

Función	Descripción
`[denoisedAudio] = denoiseSpeech (noisySpeech, cnn, targetsMean, targetsStd, noisyMean, noisyStd)`	Devuelve un audio restaurado a partir del audio con ruido utilizando la CNN entrenada.

Tabla 4.5. Función de la fase de test.

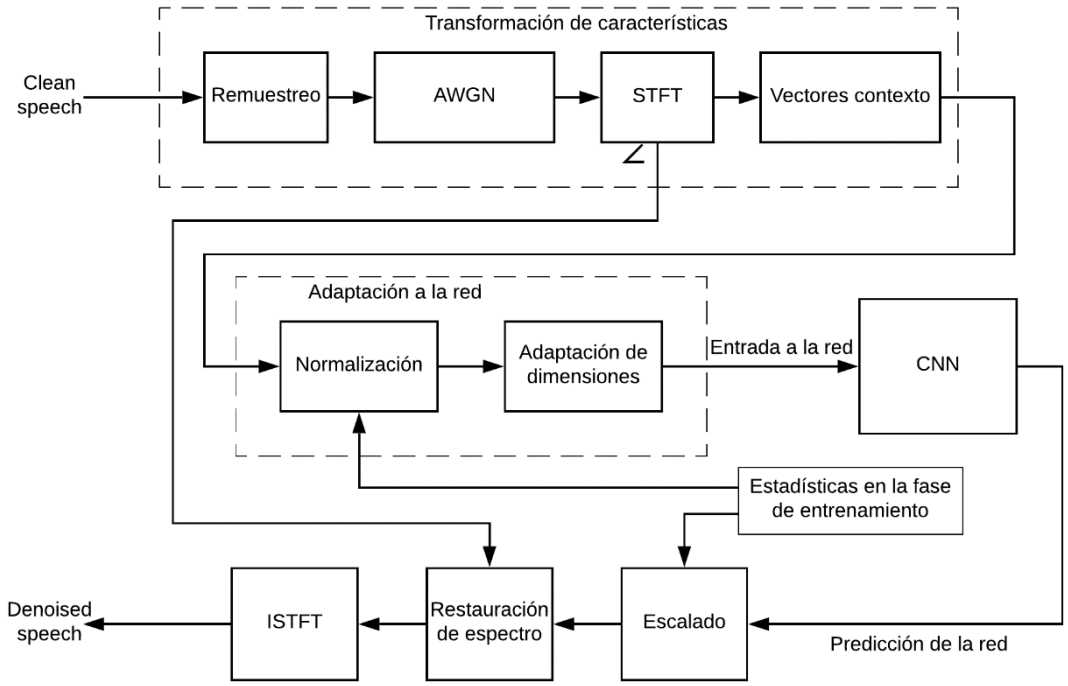


Figura 4.4. Esquema de la implementación de la fase de test.

4.2.1 Escalado

El bloque de escalado devuelve los vectores magnitud a sus valores nominales teóricos tras la normalización en el bloque de adaptación a la red. Para este bloque se define la siguiente función:

Función	Descripción
`[scaledOutput] = scaleOutput (outputStft, targetMean, targetStd)`	Escala la salida de la red por la media y la desviación típica del conjunto de entrenamiento de entrada.

Tabla 4.6. Función del bloque de escalado.

4.2.2 Restauración del espectro

Se reconstruye el espectro completo reflejando la parte simétrica obtenida a la salida de la CNN y se eliminan las dimensiones generadas para adaptar los vectores a la red. Para este bloque se define la siguiente función:

Función	Descripción
`[restoredStft] = restoreStft (outputStft, inputPhase)`	Restaura la parte simétrica de la magnitud e incorpora la información de fase.

Tabla 4.7. Función del bloque de restauración de espectro

4.2.3 ISTFT

Se genera la señal de audio con reducción de ruido calculando la *Inverse Short-Time Fourier Transform.* Este bloque queda definido por la siguiente función:

Función	Descripción
`[speech] = computeIstft (speechSTFT, window, shift, windowLength)`	Devuelve la señal de audio para una STFT dada.

Tabla 4.8. Función del bloque ISTFT

4.3 Interfaz gráfica

Como parte de los objetivos del proyecto y con el fin de agilizar el proceso de reducción de ruido y la visualización y análisis de resultados, se desarrolla una interfaz gráfica sencilla utilizando la herramienta *AppDesigner* de MATLAB.

La interfaz permite cargar [Get Audio] una pareja de audios limpio/con ruido con una determinada relación SNR introducida por el usuario. Una vez cargados, se visualizan las dos señales en formato forma de onda y espectrograma. La función [Denoise] ejecuta el sistema sobre la señal con ruido y reproduce la forma de onda y el espectrograma de la señal con reducción de ruido. Se muestran los resultados del algoritmo STOI para los dos últimos en relación con la señal limpia.

En la Figura 4.5 se muestra la interfaz y se resaltan en rojo los elementos que permiten la interacción con el sistema, numerados en orden de ejecución, y en azul los elementos que muestran la información resultante de la ejecución. A continuación se describe cada uno de ellos.

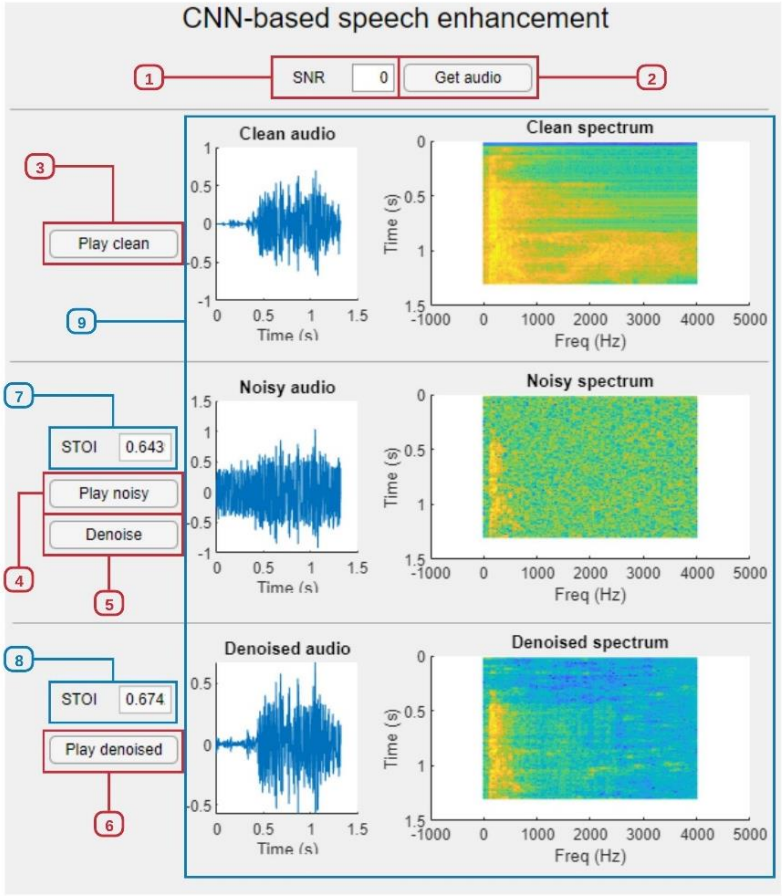


Figura 4.5. Interfaz gráfica para la aplicación del sistema

1. **SNR**. Define la relación SNR que tendrá el audio ruidoso.

2. **Get audio**. Carga en el sistema un audio del conjunto de audios de test, genera el audio con ruido correspondiente añadiendo AWGN con el SNR definido en el paso 1 y muestra en la interfaz las formas de onda y magnitudes de espectro de los dos audios. Para el audio con ruido, se muestra también el STOI en 7.
   Ejecuta las funciones `getTestAudio` y `stoi`.

3. **Play clean**. Reproduce el audio limpio cargado en el paso 2.

4. **Play noisy**. Reproduce el audio con ruido generado en el paso 2.

5. **Denoise**. Procesa el audio con ruido generado en el paso 2 con el sistema de reducción de ruido. Muestra en 8 el STOI resultante para el audio con reducción de ruido obtenido y en 9 su forma de onda y su magnitud de espectro.
   Ejecuta las funciones `denoiseSpeech` y `stoi`.

6. **Play denoised**. Reproduce el audio procesado por el sistema obtenido en el paso 5.

7. Muestra el STOI de la señal con ruido

8. Muestra el STOI de la señal con reducción de ruido.

9. Formas de onda y magnitudes de espectro de las señales.

Capítulo 5 Pruebas y análisis

Se utilizan dos métricas principales para la evaluación de resultados: la *Perceptual Evaluation of Speech Quality* (PESQ) y la *Short-Time Objective Intelligibility* (STOI). La STOI es una medida de inteligibilidad objetiva que asume que la inteligibilidad está relacionada con la similitud entre unidades tiempo-frecuencia adyacentes en una señal limpia y unidades tiempo-frecuencia en una señal procesada con ruido [36].

El funcionamiento del sistema se analiza para cuatro valores SNR: 0dB, 5dB, 10dB y 15dB. Para cada valor se generan diez audios con AWGN y se aplica la reducción de ruido. A continuación, se calculan las métricas PESQ y STOI de las señales con ruido y con reducción de ruido en relación a la señal limpia original. La Tabla 5.1 muestra los resultados de estas medidas.

0dB	**STOI**		**PESQ**	
Test	noisy	denoised	noisy	denoised
1	0,7301	0,8049	1,107	2,02
2	0,6193	0,7202	1,871	2,335
3	0,5883	0,6778	1,5	1,949
4	0,6768	0,7795	1,087	1,934
5	0,5648	0,6208	1,484	1,515
6	0,6132	0,6744	1,987	2,257
7	0,4437	0,5283	1,78	2,114
8	0,5725	0,6254	1,668	1,768
9	0,6983	0,728	1,151	1,766
10	0,7336	0,7955	2,006	2,357
5dB	**STOI**		**PESQ**	
Test	noisy	denoised	noisy	denoised
1	0,8628	0,8598	1,637	2,189
2	0,7851	0,7572	1,893	2,428
3	0,7998	0,7878	1,455	2,02
4	0,6054	0,6898	1,643	2,355
5	0,8986	0,7934	1,458	1,821
6	0,7375	0,8082	1,225	2,21
7	0,7828	0,7898	1,41	2,165
8	0,8135	0,866	1,777	2,451
9	0,8562	0,7677	2,153	2,275
10	0,8672	0,821	1,476	1,99
10dB	**STOI**		**PESQ**	
Test	noisy	denoised	noisy	denoised

1	0,8625	0,7468	1,318	1,566
2	0,8956	0,861	2,437	2,776
3	0,9103	0,8826	2,498	2,526
4	0,8085	0,7491	1,635	1,507
5	0,961	0,8652	2,009	1,985
6	0,9308	0,917	1,817	2,318
7	0,9022	0,8856	1,785	2,29
8	0,848	0,7789	1,667	2,153
9	0,9118	0,8546	2,375	2,908
10	0,8715	0,8519	1,592	2,12
15dB	**STOI**		**PESQ**	
Test	noisy	denoised	noisy	denoised
1	0,9585	0,8815	2,453	2,527
2	0,9295	0,8533	1,974	2,266
3	0,9405	0,8066	2,17	1,941
4	0,8768	0,653	2,453	1,958
5	0,9697	0,9201	2,088	1,891
6	0,9346	0,8615	2,173	2,305
7	0,8666	0,7348	1,516	1,437
8	0,8836	0,7992	1,887	2,293
9	0,9583	0,8594	2,082	2,127
10	0,9063	0,8041	2,333	2,471

Tabla 5.1. Resultados PESQ y STOI para relaciones SNR 0dB, 5dB, 10dB y 15dB. Resaltados en verde los valores en los que la métrica mejora.

El sistema mejora la inteligibilidad objetiva de forma notable para todas las muestras con SNR = 0dB, la relación con la que ha sido entrenada la red. Sin embargo, para valores SNR mayores los resultados tienden a empeorar linealmente (Figura 5.1).

Para SNR = 5dB las medidas PESQ siguen siendo positivas, pero la mayoría de resultados STOI indican una merma en la inteligibilidad. A partir de SNR = 10dB, aunque para algunas medidas la PESQ se mantiene favorable, la caída en la STOI es generalizada.

0 dB	STOI	PESQ	**5 dB**	STOI	PESQ
Noisy	0.62	1.53	Noisy	0.80	1.59
Denoised	0.69	1.98	Denoised	0.79	2.18
10 dB	STOI	PESQ	**15 dB**	STOI	PESQ
Noisy	0.89	1.88	Noisy	0.92	2.09
Denoised	0.84	2.17	Denoised	0.81	2.1

Tabla 5.2. Valores medios de STOI y PESQ para el conjunto de prueba.

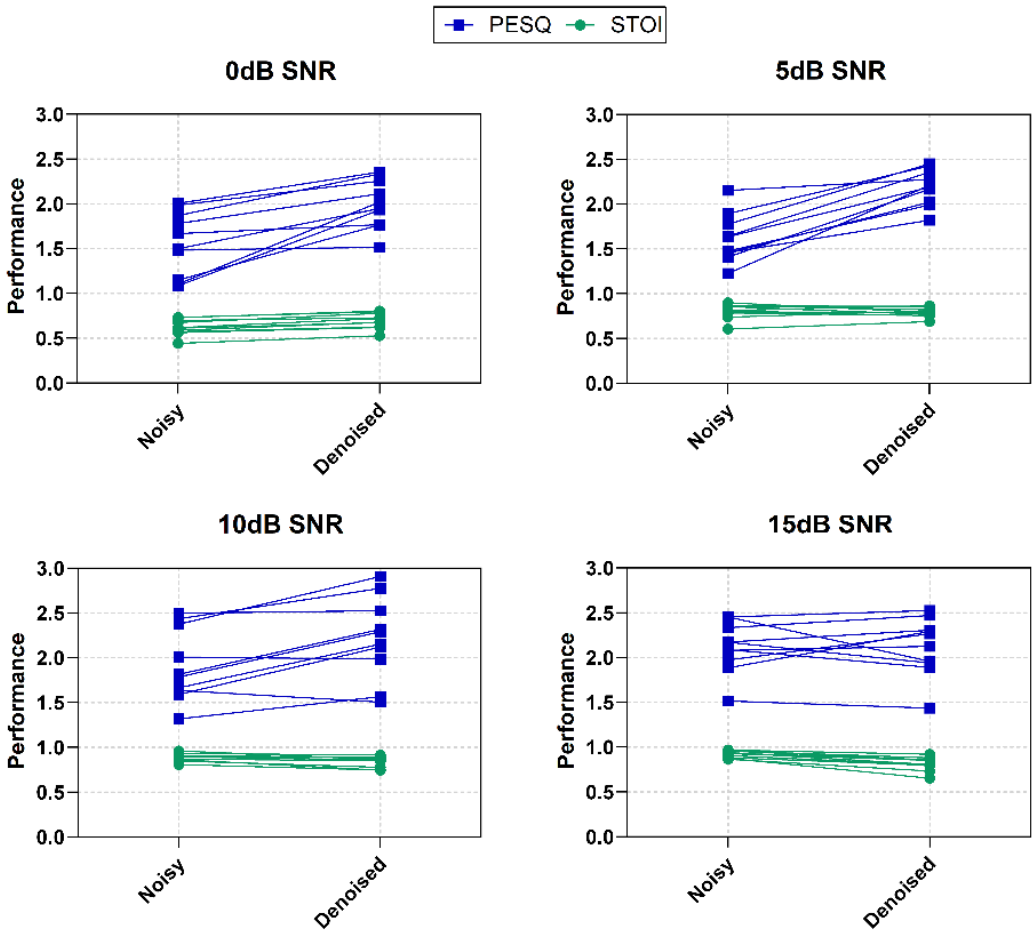


Figura 5.1. Evolución para PESQ y STOI en el conjunto de prueba.

La base de datos de voz utilizada en la implementación, Common Voice, está construida sobre aportaciones de usuarios grabadas con diferentes calidades y en entornos muy variables. La presencia de ruido en las grabaciones limpias puede haber llevado a la red a interpretar ciertas activaciones causadas por el ruido como deseables. Esta circunstancia provoca distorsiones que PESQ y STOI, cuando la señal de voz tiene más potencia que el AWGN, detectan como una merma en la inteligibilidad.

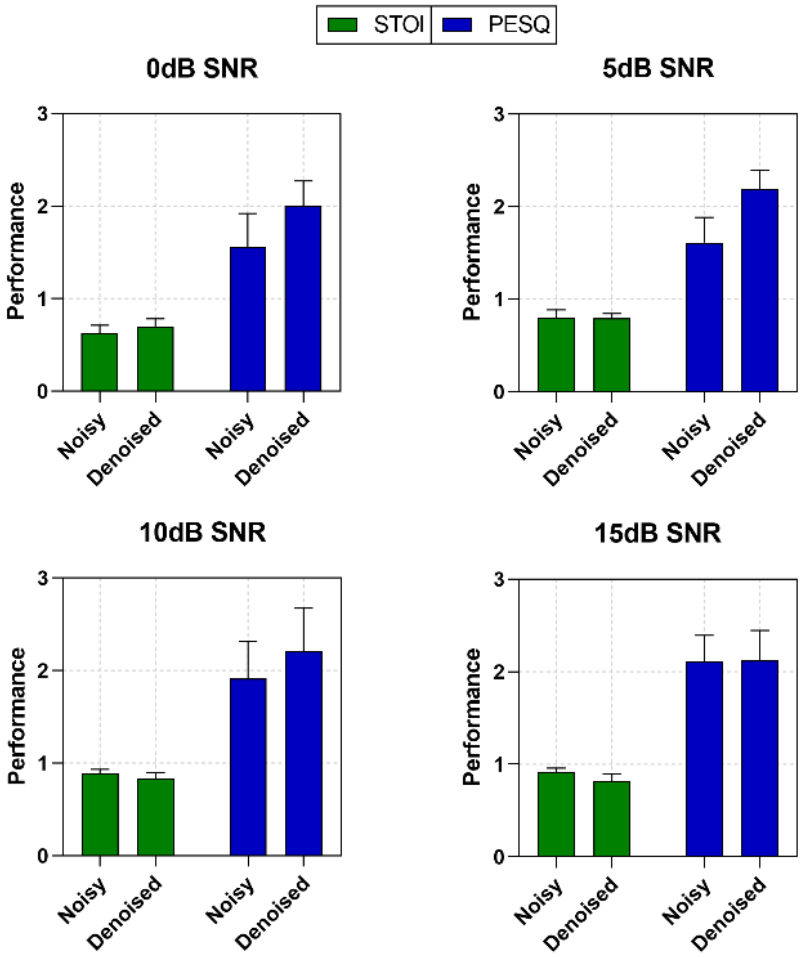


Figura 5.2. Evolución de los valores medios de PESQ y STOI para el conjunto de prueba.

Un análisis a nivel espectral revela el funcionamiento adecuado del sistema (Figura 5.3). Se analizan las cuatro relaciones SNR anteriores: 0dB, 5dB, 10dB y 15dB. En todas ellas es visible la reducción de ruido, aunque siempre acentuada en frecuencias superiores a las frecuencias fundamentales de la voz.

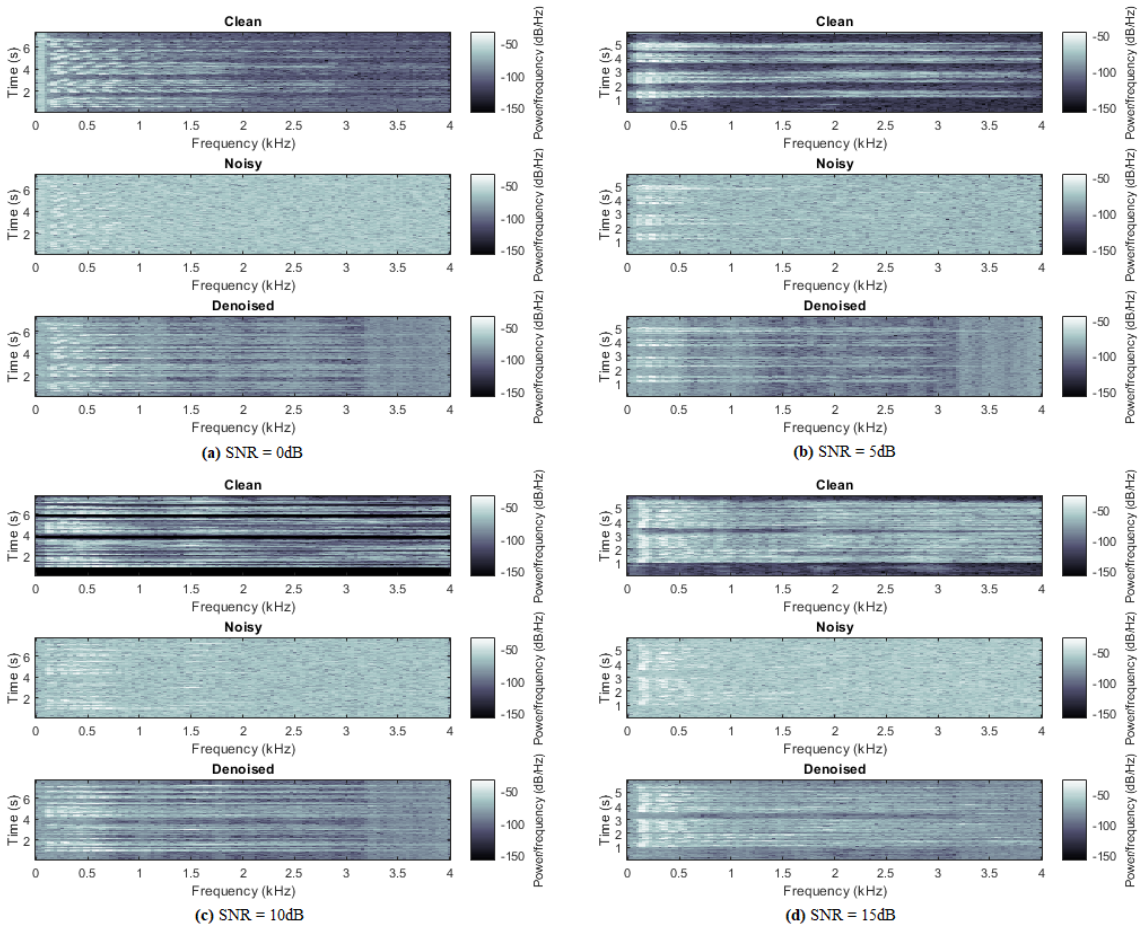


Figura 5.3. Espectrograma de señal limpia, con ruido y con reducción de ruido para los SNR 0dB, 5dB, 10dB y 15dB.

El análisis auditivo presenta características similares a las arrojadas por los resultados PESQ y STOI (Figura 5.4). Para relaciones SNR de 0dB las ventajas de la mejora en la inteligibilidad debida a la reducción de ruido son subjetivamente mayores a la merma causada por la introducción de distorsión. Sin embargo, para relaciones SNR mayores el ruido sigue siendo eliminado, pero la inteligibilidad general queda reducida por la distorsión.

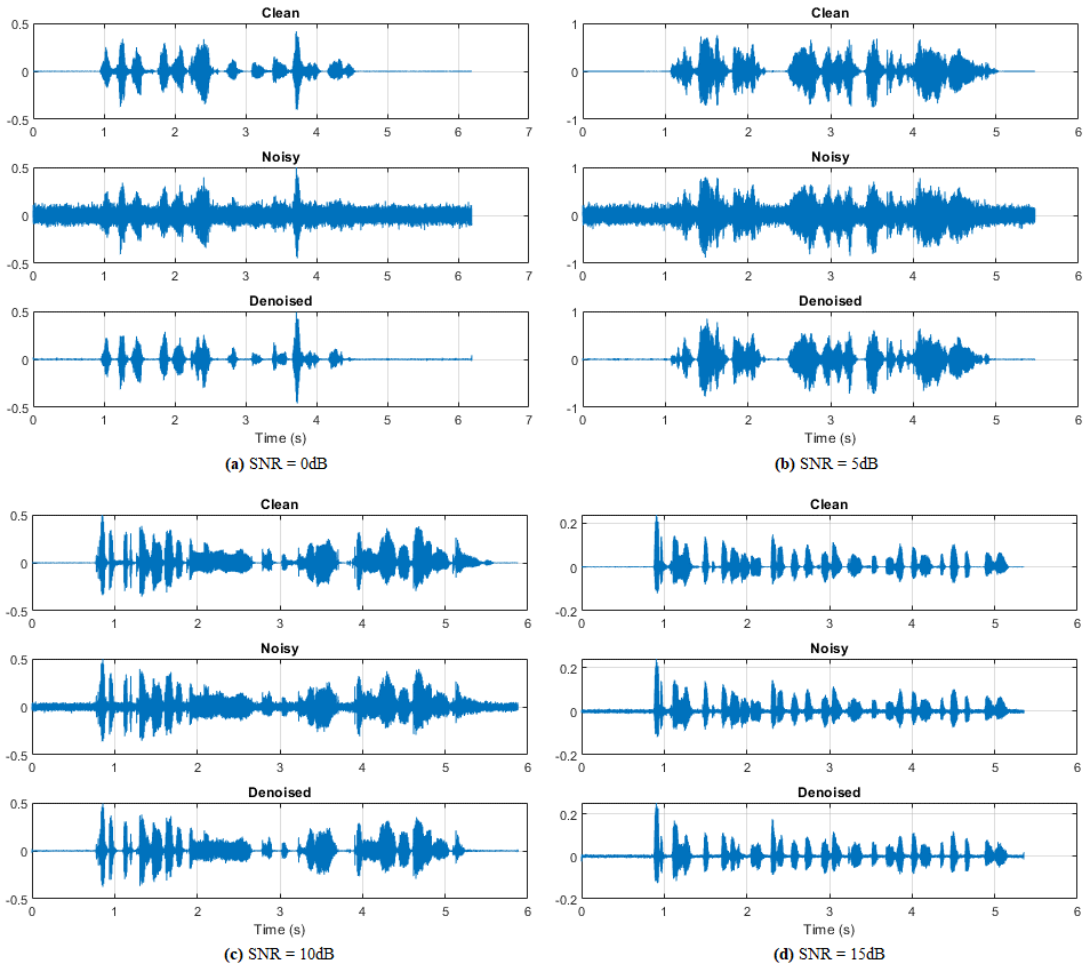


Figura 5.4. Señal temporal para relaciones SNR 0dB, 5dB, 10dB y 15dB.

Capítulo 6 Conclusiones

En este Trabajo de Fin de Grado se ha investigado el uso de sistemas *Deep Learning* en el contexto del *speech enhancement*, motivado por el resurgimiento del interés en las redes neuronales y la reciente aplicación de estas a sistemas destinados a la mejora de la inteligibilidad en señales de audio.

En concreto, se ha implementado un sistema de reducción de ruido basado en una red neuronal convolucional. Se han utilizado como características las magnitudes de los espectros de las señales de audio, a partir del cálculo de las STFT., incluyendo los 7 vectores magnitud anteriores para introducir contexto acústico. Se ha configurado la red con 16 capas convolucionales siguiendo los parámetros establecidos en [10]. El conjunto de datos de entrenamiento y test se ha obtenido del *corpus* catalán de Mozilla Common Voice y consiste en 92 horas de audio divididas en 33000 locuciones, de las cuales 1000 han sido utilizadas para entrenamiento y 250 para validación. Por último, se ha implementado una interfaz gráfica que permite analizar el funcionamiento del sistema añadiendo AWGN a las señales de audio de test, ejecutando el proceso de reducción de ruido y mostrando la señal resultante.

Las pruebas realizadas sobre el conjunto de audios de test con AWGN añadido han arrojado resultados mixtos. Por una parte, para señales a las que se ha añadido ruido con la misma relación SNR con la que se ha entrenado la red la mejora de la inteligibilidad ha sido apreciable, de forma que, considerando los efectos globales sobre la señal, la aplicación del sistema ha sido positiva. En cambio, cuando la señal introducida ha presentado relaciones SNR superiores a las usadas en el entrenamiento, la relación entre distorsión introducida por el sistema y la reducción de ruido ha resultado demasiado grande y los beneficios del enhancement se han perdido. Una mayor cantidad de recursos de computación y de tiempo de entrenamiento habrían permitido mejorar este último aspecto.

En cuanto a los objetivos planteados para este TFG, en el apartado 1.3 se establecían tres principales y uno secundario que se reproducen a continuación:

Principales:

- Implementar y entrenar una red neuronal que permita mejorar la inteligibilidad de una señal de audio deteriorada por el ruido.
- Medir la eficacia del resultado.
- Comprender los fundamentos del funcionamiento de las redes neuronales artificiales.

Secundario:

- Implementar una GUI sencilla que permita aplicar la red neuronal resultante a un conjunto de audios de test y obtener el resultado de la reducción del ruido.

Por tanto, se puede concluir que se han alcanzado todos los objetivos marcados, matizando únicamente el primer punto debido a las limitaciones en la mejora de la inteligibilidad, condicionadas por el SNR de la señal.

Capítulo 7 Líneas de trabajo futuro

Existen varias opciones de trabajo futuro e investigación basadas en los resultados de este TFG. La más obvia es incrementar el tamaño del conjunto de audios utilizados para el entrenamiento y el número de iteraciones de la red durante esa fase. En la práctica, esto implica ampliar la capacidad de computación del entorno en el que se implementa el sistema, ya sea a nivel local con procesadores gráficos (la GPU de Nvidia Titan V es un estándar en la investigación DL) o recurriendo al *Cloud Computing* con servicios como Amazon Web Services o Google Cloud.

A nivel de arquitectura, y con el objetivo de atenuar los efectos de la distorsión de fase en los resultados obtenidos, se puede introducir un módulo *Phase Aware Scaling* como el utilizado en [10], con el que se evitan diferencias grandes entre la fase de los audios limpio y con ruido codificando la magnitud del espectro del primero de la siguiente forma:

$$s_{phase\ aware} = s_{clean} \cos(\theta_{clean} - \theta_{noisy})$$

Por otra parte, como se comenta en el capítulo 5, los resultados de la ejecución del sistema son muy dependientes de la calidad y las condiciones con las que ha sido captado el conjunto de datos de entrenamiento, por lo que sería interesante evaluar el funcionamiento para un *corpus* diferente, como TIMIT [37]. En este sentido, teniendo en cuenta la variación de características y duración del ruido en aplicaciones reales, otro elemento a considerar sería la introducción de ruidos diferentes a AWGN, como los incluidos en la QUT-NOISE Database [38].

Además, sería adecuado comparar los resultados obtenidos con los de otros sistemas de reducción de ruido en el habla, como los presentados en el apartado 2.3 y con otras arquitecturas DNN, de forma que pueda existir una evaluación más objetiva del sistema desarrollado en este TFG.

Por último, en un plano más conceptual y relacionado con la aplicación de la reducción de ruido en los sistemas de comunicación descritos en el apartado 1.1 y la predominancia actual de los dispositivos móviles, sería interesante aprovechar el bajo número de parámetros requerido por la CNN implementada en comparación con otras arquitecturas e investigar su posible aplicación en un sistema de reducción de ruido en tiempo real.

Bibliografía

[1] M. Parchami, W.-P. Zhu, B. Champagne y E. Plourde, «Recent Developments in Speech Enhancement in the Short-Time Fourier Transform Domain,» *IEEE Circuits and Systems Magazine,* vol. 16, nº 3, pp. 45-77, 2016.

[2] Y. Xu, J. Du, L. R. Dai y C.-H. Lee, «A Regression Approach to Speech Enhancement Based on Deep Neural Networks,» *IEEE/ACM Transactions on Audio, Speech, and Language Processing,* vol. 23, nº 1, pp. 7-19, 2014.

[3] P. Scalart y J. Filho, «Speech enhancement based on a priori signal to noise estimation,» de *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings,* Atlanta, GA, USA, 1996.

[4] A. Chaundhari y S. Dhonde, «A Review on Speech Enhancement Techniques,» de *2015 International Conference on Pervasive Computing (ICPC),* Pune, India, 2015.

[5] Y. Wang y D. Wang, «A deep neural network for time-domain signal reconstruction,» de *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* Brisbane, QLD, Australia, 2015.

[6] A. Kumar y D. Florencio, «Speech Enhancement In Multiple-Noise Conditions using Deep Neural Networks,» de *Proc. of the Int. Speech Communication Association Conf. (INTERSPEECH),* 2016, pp. 3738-3742.

[7] X. Lu, Y. Tsao, S. Matsuda y C. Hori, «Speech enhancement based on deep denoising autoencoder,» de *INTERSPEECH,* Lyon, France, 2013, pp. 436-440.

[8] «Wavenet,» [En línea]. Available: https://deepmind.com/blog/wavenet-generative-model-raw-audio/. [Último acceso: 05 March 2019].

[9] «Magenta,» [En línea]. Available: https://magenta.tensorflow.org/. [Último acceso: 05 March 2019].

[10] S. Park y J. Lee, «A Fully Convolutional Neural Network for Speech Enhancement,» [En línea]. Available: https://arxiv.org/abs/1609.07132.

[11] «Mozilla Common Voice,» [En línea]. Available: https://voice.mozilla.org.

[12] M. Van Gerven, «Computational Foundations of Natural Intelligence,» *Frontiers in Computational Neuroscience*, vol. 11, p. 112, 2017.

[13] I. Basheer y M. Hajmeer, «Artificial neural networks: fundamentals, computing, design, and application,» *Journal of Microbiological Methods*, vol. 43, nº 1, pp. 3-31, 2000.

[14] J. Bapu Ahire, «The Artificial Neural Networks Handbook: Part 4,» [En línea]. Available: https://medium.com/@jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e.

[15] C. M. Bishop, Pattern Recognition and Machine Learning, Cambridge: Springer Science, 2006.

[16] F. Rosenblatt, «The Perceptron: A probabilistic Model for Information Storage and Organization in the Brain,» *Psychological Review*, vol. 65, nº 6, pp. 65-386, 1958.

[17] R. Lippmann, «An Introduction to Computing with Neural Nets,» *IEEE ASSP Magazine*, vol. 4, nº 2, pp. 4-22, 1987.

[18] R. Chandradevan, «Radial Basis Functions Neural Networks—All we need to know,» [En línea]. Available: https://towardsdatascience.com/radial-basis-functions-neural-networks-all-we-need-to-know-9a88cc053448.

[19] I. Goodfellow, Y. Bengio y A. Courville, Deep Learning, MIT Press, 2016.

[20] S. Ruder, «An overview of gradient descent optimization algorithms,» [En línea]. Available: https://arxiv.org/abs/1609.04747v2.

[21] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich y T. Poggio, «A quantitative theory of immediate visual recognition,» *Progress in Brain Research*, vol. 165, pp. 33-58, 2007.

[22] H. Larochelle, Y. Bengio, J. Louradour y P. Lamblin, «Exploring Strategies for Training Deep Neural Networks,» *Journal of Machine Learning Research*, vol. 1, pp. 1-40, 2009.

[23] V. Nair y G. E. Hinton, «Rectified linear units improve restricted boltzmann machines,» de *ICML'10 Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010.

[24] A. Graves, A.-r. Mohamed y G. Hinton, «Speech Recognition with Deep Recurrent Neural Networks,» *IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2013.

[25] S. Hochreiter y J. Schmidhuber, «Long Short-Term Memory,» *Neural Computation*, vol. 9, nº 8, pp. 1735-1780, 1997.

[26] P. Gurunath Shivakumar y P. Georgiou, «Perception Optimized Deep Denoising AutoEncoders for Speech Enhancement,» *INTERSPEECH*, pp. 3743-3747, 2016.

[27] Y. Wang y D. Wang, «A Deep Neural Network for Time-domain Signal Reconstruction,» de *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brisbane, Australia, 2015.

[28] L. Pandey, A. Kumar y V. Namboodiri, «Monoaural Audio Source Separation Using Variational Autoencoders,» de *INTERSPEECH*, 2018.

[29] J. Xu, J. Du, L.-R. Dai y C.-H. Lee, «An Experimental Study on Speech Enhancement Based on Deep Neural Networks,» *IEEE Signal Processing Letters*, vol. 21, nº 1, pp. 65-68, 2014.

[30] ITU-T, «Perceptual evaluation of Speech Quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs,» 2001.

[31] A. Rix, J. Beerends, M. Hollier y A. Hekstra, «Perceptual evaluation of speech quality (PESQ)-a new method for speech quality assessment of telephone networks and codecs,» de *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings*, Salt Lake City, USA, 2001.

[32] A. Maas, Q. V. Le, T. O'Neil, O. Vinyals y P. Nguyen, «Recurrent Neural Networks for Noise Reduction in Robust ASR,» de *INTERSPEECH*, 2012.

[33] X. lu, Y. Tsao, S. Matsuda y C. Hori, «Speech enhancement based on deep denoising Auto-Encoder,» de *INTERSPEECH*, 2013.

[34] D. Kingma y J. Ba, «Adam: A Method for Stochastic Optimization,» de *International Conference on Learning Representations*, 2014.

[35] Matlab, «Denoise Speech Using Deep Learning Networks,» [En línea]. Available: https://es.mathworks.com/help/audio/examples/denoise-speech-using-deep-learning-networks.html.

[36] C. H. Taal, R. C. Hendriks, R. Heusdens y J. Jensen, «An Alorithm for Intelligibility Prediction of Time-Frequency Weighted Noisy Speech,» *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, nº 7, pp. 2125-2136, 2011.

[37] J. S Garafolo, L. F Lamel, W. M Fisher, J. G Fiscus y D. S pallett, «DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1,» *NASA STI/Recon Technical Report*, vol. 93, 1993.

[38] «QUT-NOISE Database,» [En línea]. Available: https://research.qut.edu.au/saivt/databases/qut-noise-databases-and-protocols/.

Anexo

A continuación se muestra la implementación en lenguaje M del sistema de reducción de ruido basado en CNN de este TFG.

```
function [netInputs] = adaptInputShape (inputs)
%%
% inputShapeAdapt: Adapta los sets al formato esperado por la CNN.
% La red espera matrices filas*columnas*canales*imágenes.
%
%   inputs  : magnitudes normalizadas de los audios con ruido

netInputs  =
reshape(inputs,size(inputs,1),size(inputs,2),1,size(inputs,3));
```

```
function [netTargets] = adaptTargetShape (targets)
%%
% inputShapeAdapt: Adapta los sets al formato esperado por la CNN.
% La red espera matrices filas*columnas*canales*imágenes.
%
%   targets : magnitudes normalizadas de los audios limpios

netTargets = reshape(targets,size(targets,1),1,1,size(targets,2));
```

```
function [audio] = audioLengthExtend (audio)
% audioLengthExtend: Aumenta la duración de la señal de audio de
forma que
% el número de muestras sea proporcional al factor de decimado.
%
%   audio: Señal de audio.

D     = 16/8; % Factor de decimado.
L     = floor(numel(audio)/D); % Número entero de muestras tras el
decimado.
audio = audio(1:D*L);
```

```
function [speech] = computeIstft (speechSTFT, window, shift,
windowLength)
% computeIstft: Devuelve la señal de audio para una fft dada.
%
%   speechSTFT: fft de la señal de audio.
%   window: ventana de la stft.
%   shift: número de puntos de encabalgamiento.
%   windowLength: puntos de la ventana.
%   nMagnitude: número de vectores magnitud.
```

```
speech = istft(speechSTFT, 'Window', window, 'OverlapLength', shift,
...
    'FFTLength', windowLength,'ConjugateSymmetric',true);
```

```
function [speechMag, speechPhase] = computeStft (speech, window,
shift, windowLength, nMagnitude)
% getStft: Computa la STFT de la señal y devuelve fase y magnitud.
%
%   speech: señal de audio.
%   window: ventana de la STFT.
%   shift: número de puntos de encabalgamiento.
%   windowLength: puntos de la ventana.
%   nMagnitude: número de vectores magnitud.

speechSTFT  = stft(speech, 'Window', window, 'OverlapLength',
shift,...
    'FFTLength', windowLength);

% Cálculo de fase y magnitud. Se elimina la mitad simétrica del
espectro.
speechPhase = angle(speechSTFT(nMagnitude-1:end,:));
speechMag   = abs(speechSTFT(nMagnitude-1:end,:));
```

```
function [layers, options] = configureCNN (trainInputs,...
    validationInputs, validationTargets)
% configureCNN devuelve los parámetros de configuración de la CNN.
%
%   trainInputs : set de entrenamiento de entrada
%   validationInputs : set de validación de entrada
%   validationTargets : set de validación de objetivo

miniBatchSize = 64;

% Configuración de capas. 18 filtros de entrada [9 8] + 5* (8
filtros [9 1]
% 18 filtros [5 1] 30 filtros [9 1]) + salida [129 1].
layers = [imageInputLayer([129,8])
          convolution2dLayer([9 8],18,"Stride",[1
100],"Padding","same")
          batchNormalizationLayer
          reluLayer

          convolution2dLayer([5 1],30,"Stride",[1
100],"Padding","same")
          batchNormalizationLayer
          reluLayer

          convolution2dLayer([9 1],8,"Stride",[1
100],"Padding","same")
          batchNormalizationLayer
          reluLayer
```

```
        convolution2dLayer([9 1],18,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([5 1],30,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([9 1],8,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([9 1],18,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([5 1],30,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([9 1],8,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([9 1],18,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([5 1],30,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([9 1],8,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([9 1],18,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([5 1],30,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
        reluLayer

        convolution2dLayer([9 1],8,"Stride",[1
100],"Padding","same")
        batchNormalizationLayer
```

```
          reluLayer

          convolution2dLayer([129 1],1,"Stride",[1
100],"Padding","same")

          regressionLayer
          ];

% Configuración de opciones.
options = trainingOptions("adam", ...
    "MaxEpochs",3, ...
    "InitialLearnRate",0.0015,...
    "MiniBatchSize",miniBatchSize, ...
    "Shuffle","every-epoch", ...
    "Plots","training-progress", ...
    "Verbose",false, ...

"ValidationFrequency",floor(size(trainInputs,4)/miniBatchSize),...
    "LearnRateSchedule","piecewise",...
    "LearnRateDropFactor",0.9,...
    "LearnRateDropPeriod",1,...
    "ValidationData",{validationInputs,validationTargets});
```

```
function [denoisedAudio] = denoiseSpeech (noisySpeech, cnn,
targetsMean, targetsStd, noisyMean, noisyStd)
% denoiseSpeech: Devuelve el audio restaurado a partir de una señal
ruidosa
% utilizando la CNN entrenada.
%
%   noisySpeech: Señal de audio con ruido.
%   cnn: Red neuronal convolucional.
%   targetsMean: Media de los audios limpios en la fase de
entrenamiento.
%   targetsStd: Desviación típica de los audios limpios en la fase
de entrenamiento.
%   noisyMean: Media de los audios con ruido en la fase de
entrenamiento.
%   noisyStd: Desviación típica de los audios con ruido en la fase
de entrenamiento.

% Parámetros de la STFT.
WindowLength  = 256;
window        = hamming(WindowLength,'periodic');
Shift         = round(0.75 * WindowLength); % Encabalgamiento
ventanas
nMagnitude    = WindowLength/2 + 1; % Vectores magnitud
nInputVectors = 8; % N° vectores mag consecutivos en input

% Se extraen los vectores magnitud y fase del audio.
[noisyMag, noisyPhase] = computeStft(noisySpeech, window, Shift,
WindowLength, nMagnitude);
noisyMag = getInputVectors (noisyMag, nInputVectors, nMagnitude);

% Se normaliza la magnitud en función de los valores con los que la
red ha
% sido entrenada.
noisyMag(:) = (noisyMag(:) - noisyMean) / noisyStd;
```

```
% Se dimensiona la magnitud para adaptarla al formato de entrada de
la red.
input = adaptInputShape(noisyMag);

% Predicción de los vectores magnitud de la señal restaurada.
denoisedStft = predict(cnn, input);

% Escalamos la salida en función de los valores con los que la red
ha sido
% entrenada.
denoisedStft(:) = scaleOutput(denoisedStft, targetsMean,
targetsStd);

% Se restaura la parte simétrica de la magnitud y se añade la
información
% de fase.
denoisedStft = restoreStft(denoisedStft, noisyPhase);

% Cálculo de la señal de audio restaurada.
denoisedAudio = computeIstft(denoisedStft, window, Shift,
WindowLength);
```

```
function [inputProcessedAudio, targetProcessedAudio, speechPhase] =
featureTransform (speech, src)
% Preprocess: devuelve las STFT del audio y la matriz 129*8 del
audio+ruido
%
%   speech: Señal de audio de entrada
%   src: Conversor de frecuencia de muestreo

% Parámetros de la STFT.
WindowLength  = 256;
window        = hamming(WindowLength,'periodic');
Shift         = round(0.75 * WindowLength); % Encabalgamiento
ventanas
nMagnitude    = WindowLength/2 + 1; % Vectores magnitud
nInputVectors = 8; % N° vectores mag consecutivos en input

% Asegura que la longitud del audio es proporcional al factor de
decimado
speech = audioLengthExtend(speech);

% Conversión fs
speech = src(speech);
reset(src);

% Se añade AWGN con SNR=0 a la señal de audio.
noisySpeech = awgn(speech, 0, 'measured');

% Cálculo de la STFT para la señal original
[speechSTFT, speechPhase] = computeStft(speech, window, Shift,
WindowLength, nMagnitude);
targetProcessedAudio = speechSTFT;

% Cálculo de la STFT para la señal con ruido y obtención de la
matriz de
```

```
% predicción.
noisySpeechSTFT = computeStft(noisySpeech, window, Shift,
WindowLength, nMagnitude);
inputProcessedAudio = getInputVectors (noisySpeechSTFT,
nInputVectors, nMagnitude);
```

```
function [datastore] = getAudioData (set)
% getAudioData: Devuelve un objeto Audio Datastore con los audios de
la
% categoría especificada en la entrada en orden aleatorio.
%
%   set: cadena de caracteres con valor 'test' o 'train'

if strcmpi(set, 'test')
    datastore = audioDatastore(fullfile('..', 'test'));
end

if strcmpi(set, 'train')
    datastore = audioDatastore(fullfile('..', 'train'));
end

datastore = shuffle(datastore);
```

```
function [inputVectors] = getInputVectors (noisySpeechSTFT,
nInputVectors, nMagnitude)
% getInputVectors: Devuelve una matriz 129*8*(puntosSTFT) con los
vectores de entrada.
% Cada paso en la dim3 contiene los 7 vectores STFT anteriores y el
actual.
%
%   noisySpeechSTFT: magnitud del audio input.
%   nInputVectors: número de vectores magnitud consecutivos en input
%   nMagnitude: número de vectores magnitud

% Cómputo de las matrices target 129*8
noisySpeechSTFT = [noisySpeechSTFT(:,1:nInputVectors - 1), ...
    noisySpeechSTFT]; % Expansión por la izquierda

inputVectors = zeros(nMagnitude, nInputVectors , ...
    size(noisySpeechSTFT,2) - nInputVectors + 1);
for index = 1:size(noisySpeechSTFT,2) - nInputVectors + 1
    inputVectors(:,:,index) = (noisySpeechSTFT(:,index:index + ...
       nInputVectors - 1));
end
```

```
function [clean, noisy] = getTestAudio (snr)
% getTestAudio: Devuelve un par audio limpio/audio con ruido awgn a
partir
% del set de test.
%
%   snr: Relación SNR para el audio con ruido.
```

```
% Se obtiene un audio del set de test.
ds = getAudioData('test');
clean = read(ds);

% Asegura que la longitud del audio es proporcional al factor de
decimado.
clean = audioLengthExtend(clean);

% Se remuestrea el audio a 8000Hz.
src = dsp.SampleRateConverter("InputSampleRate",16000, ...
                              "OutputSampleRate",8000, ...
                              "Bandwidth",7920);
clean = src(clean);
reset(src);

% Se añade awgn a la señal limpia.
noisy = awgn(clean, snr, 'measured');
```

```
function [mags, inputsMean, inputsStd] = magNormalization (mags)
%%
% magNormalization: devuelve la media, la desviación típica y el set
% centrado en 0 y con desviación típica 1. normX = (x -
mean(x))/std(x).
%
%   mags  : magnitudes de los audios

mags          = cat(ndims(mags{1}),mags{:});
inputsMean    = mean(mags(:));
inputsStd     = std(mags(:));

mags(:) = (mags(:) - inputsMean)/inputsStd;
```

```
function [restoredStft] = restoreStft (outputStft, inputPhase)
% restoreStft: Restaura la parte simétrica de la magnitud e
incorpora la
% información de fase.
%
%   outputStft: Salida de la red.
%   inputPhase: Fase extraida en el preprocesamiento.

% Se eliminan las dimensiones de tamaño 1 creadas al adaptar y se
incorpora
% la fase.
outputStft = squeeze(outputStft) .* exp(1j*inputPhase);

% Se restaura la parte simétrica.
restoredStft = [conj(outputStft(end-1:-1:2,:)) ; outputStft];
```

```
function [scaledOutput] = scaleOutput (outputStft, targetMean,
targetStd)
% scaleOutput: Escala la salida de la red por los parámetros media y
% desviación estándar de la entrada target.
```

```
%
%   outputStft: magnitud salida de la red.
%   targetMean: media del set limpio de entrenamiento.
%   targetStd: desviación estándar del set limpio de entrenamiento.

scaledOutput(:) = targetStd * outputStft(:) + targetMean;
```

```
function [trainInputs, trainTargets, validationInputs, ...
    validationTargets] = splitTrainToValidation (inputs, targets)
% splitTrainToValidation: Divide el set preprocesado y normalizado
en sets
% de entrenamiento y validación.
%
%   inputs  : magnitudes normalizadas de los audios con ruido
%   targets : magnitudes normalizadas de los audios limpios

TrainAssign = 0.99; % Se asigna el 1% a validación

inds              = randperm(size(inputs,4));
testLength        = round(TrainAssign * size(inputs,4));

trainInputs       = inputs(:,:,:,inds(1:testLength));
trainTargets      = targets(:,:,:,inds(1:testLength));
validationInputs  = inputs(:,:,:,inds(testLength+1:end));
validationTargets = targets(:,:,:,inds(testLength+1:end));
```

```
function d = stoi(x, y, fs_signal)
%   d = stoi(x, y, fs_signal) returns the output of the short-time
%   objective intelligibility (STOI) measure described in [1, 2],
where x
%   and y denote the clean and processed speech, respectively, with
sample
%   rate fs_signal in Hz. The output d is expected to have a
monotonic
%   relation with the subjective speech-intelligibility, where a
higher d
%   denotes better intelligible speech. See [1, 2] for more details.
%
%   References:
%      [1] C.H.Taal, R.C.Hendriks, R.Heusdens, J.Jensen 'A Short-
Time
%      Objective Intelligibility Measure for Time-Frequency Weighted
Noisy
%      Speech', ICASSP 2010, Texas, Dallas.
%
%      [2] C.H.Taal, R.C.Hendriks, R.Heusdens, J.Jensen 'An
Algorithm for
%      Intelligibility Prediction of Time-Frequency Weighted Noisy
Speech',
%      IEEE Transactions on Audio, Speech, and Language Processing,
2011.
%
%
% Copyright 2009: Delft University of Technology, Signal &
Information
```

```
% Processing Lab. The software is free for non-commercial use. This
program
% comes WITHOUT ANY WARRANTY.
%
%
%
% Updates:
% 2011-04-26 Using the more efficient 'taa_corr' instead of 'corr'

if length(x)~=length(y)
    error('x and y should have the same length');
end

% initialization
x           = x(:);                             % clean speech
column vector
y           = y(:);                             % processed speech
column vector

fs          = 10000;                            % sample rate of
proposed intelligibility measure
N_frame     = 256;                              % window support
K           = 512;                              % FFT size
J           = 15;                               % Number of 1/3
octave bands
mn          = 150;                              % Center frequency
of first 1/3 octave band in Hz.
H           = thirdoct(fs, K, J, mn);           % Get 1/3 octave
band matrix
N           = 30;                               % Number of frames
for intermediate intelligibility measure (Length analysis window)
Beta        = -15;                              % lower SDR-bound
dyn_range   = 40;                               % speech dynamic
range

% resample signals if other samplerate is used than fs
if fs_signal ~= fs
    x   = resample(x, fs, fs_signal);
    y   = resample(y, fs, fs_signal);
end

% remove silent frames
[x y] = removeSilentFrames(x, y, dyn_range, N_frame, N_frame/2);

% apply 1/3 octave band TF-decomposition
x_hat       = stdft(x, N_frame, N_frame/2, K);  % apply short-time
DFT to clean speech
y_hat       = stdft(y, N_frame, N_frame/2, K);  % apply short-time
DFT to processed speech

x_hat       = x_hat(:, 1:(K/2+1)).';            % take clean single-
sided spectrum
y_hat       = y_hat(:, 1:(K/2+1)).';            % take processed
single-sided spectrum

X           = zeros(J, size(x_hat, 2));         % init memory for
clean speech 1/3 octave band TF-representation
Y           = zeros(J, size(y_hat, 2));         % init memory for
processed speech 1/3 octave band TF-representation
```

```
for i = 1:size(x_hat, 2)
    X(:, i) = sqrt(H*abs(x_hat(:, i)).^2);      % apply 1/3 octave
bands as described in Eq.(1) [1]
    Y(:, i) = sqrt(H*abs(y_hat(:, i)).^2);
end

% loop al segments of length N and obtain intermediate
intelligibility measure for all TF-regions
d_interm    = zeros(J, length(N:size(X, 2)));
% init memory for intermediate intelligibility measure
c           = 10^(-Beta/20);
% constant for clipping procedure

for m = N:size(X, 2)
    X_seg   = X(:, (m-N+1):m);
% region with length N of clean TF-units for all j
    Y_seg   = Y(:, (m-N+1):m);
% region with length N of processed TF-units for all j
    alpha   = sqrt(sum(X_seg.^2, 2)./sum(Y_seg.^2, 2));
% obtain scale factor for normalizing processed TF-region for all j
    aY_seg  = Y_seg.*repmat(alpha, [1 N]);
% obtain \alpha*Y_j(n) from Eq.(2) [1]
    for j = 1:J
        Y_prime             = min(aY_seg(j, :), X_seg(j, :)+X_seg(j,
:)*c); % apply clipping from Eq.(3)
        d_interm(j, m-N+1)  = taa_corr(X_seg(j, :).', Y_prime(:));
% obtain correlation coeffecient from Eq.(4) [1]
    end
end

d = mean(d_interm(:));
% combine all intermediate intelligibility measures as in Eq.(4) [1]

%%
function  [A cf] = thirdoct(fs, N_fft, numBands, mn)
%   [A CF] = THIRDOCT(FS, N_FFT, NUMBANDS, MN) returns 1/3 octave
band matrix
%   inputs:
%       FS:         samplerate
%       N_FFT:      FFT size
%       NUMBANDS:   number of bands
%       MN:         center frequency of first 1/3 octave band
%   outputs:
%       A:          octave band matrix
%       CF:         center frequencies

f               = linspace(0, fs, N_fft+1);
f               = f(1:(N_fft/2+1));
k               = 0:(numBands-1);
cf              = 2.^(k/3)*mn;
fl              = sqrt((2.^(k/3)*mn).*2.^((k-1)/3)*mn);
fr              = sqrt((2.^(k/3)*mn).*2.^((k+1)/3)*mn);
A               = zeros(numBands, length(f));

for i = 1:(length(cf))
    [a b]                   = min((f-fl(i)).^2);
    fl(i)                   = f(b);
    fl_ii                   = b;

    [a b]                   = min((f-fr(i)).^2);
```

```
    fr(i)                   = f(b);
    fr_ii                   = b;
    A(i,fl_ii:(fr_ii-1))    = 1;
end

rnk         = sum(A, 2);
numBands    = find((rnk(2:end)>=rnk(1:(end-1))) &
(rnk(2:end)~=0)~=0, 1, 'last' )+1;
A           = A(1:numBands, :);
cf          = cf(1:numBands);

%%
function x_stdft = stdft(x, N, K, N_fft)
%   X_STDFT = X_STDFT(X, N, K, N_FFT) returns the short-time
%   hanning-windowed dft of X with frame-size N, overlap K and DFT
size
%   N_FFT. The columns and rows of X_STDFT denote the frame-index
and
%   dft-bin index, respectively.

frames      = 1:K:(length(x)-N);
x_stdft     = zeros(length(frames), N_fft);

w           = hanning(N);
x           = x(:);

for i = 1:length(frames)
    ii              = frames(i):(frames(i)+N-1);
    x_stdft(i, :)   = fft(x(ii).*w, N_fft);
end

%%
function [x_sil y_sil] = removeSilentFrames(x, y, range, N, K)
%   [X_SIL Y_SIL] = REMOVESILENTFRAMES(X, Y, RANGE, N, K) X and Y
%   are segmented with frame-length N and overlap K, where the
maximum energy
%   of all frames of X is determined, say X_MAX. X_SIL and Y_SIL are
the
%   reconstructed signals, excluding the frames, where the energy of
a frame
%   of X is smaller than X_MAX-RANGE

x       = x(:);
y       = y(:);

frames  = 1:K:(length(x)-N);
w       = hanning(N);
msk     = zeros(size(frames));

for j = 1:length(frames)
    jj      = frames(j):(frames(j)+N-1);
    msk(j)  = 20*log10(norm(x(jj).*w)./sqrt(N));
end

msk     = (msk-max(msk)+range)>0;
count   = 1;

x_sil   = zeros(size(x));
y_sil   = zeros(size(y));
```

```
for j = 1:length(frames)
    if msk(j)
        jj_i            = frames(j):(frames(j)+N-1);
        jj_o            = frames(count):(frames(count)+N-1);
        x_sil(jj_o)     = x_sil(jj_o) + x(jj_i).*w;
        y_sil(jj_o)     = y_sil(jj_o) + y(jj_i).*w;
        count           = count+1;
    end
end

x_sil = x_sil(1:jj_o(end));
y_sil = y_sil(1:jj_o(end));

%%
function rho = taa_corr(x, y)
%   RHO = TAA_CORR(X, Y) Returns correlation coeffecient between
column
%   vectors x and y. Gives same results as 'corr' from statistics
toolbox.
xn      = x-mean(x);
xn      = xn/sqrt(sum(xn.^2));
yn      = y-mean(y);
yn      = yn/sqrt(sum(yn.^2));
rho     = sum(xn.*yn)
```

```
function [denoisingCNN, inputsMean, inputsStd, targetsMean,
targetsStd] = trainCNN ()
% trainCNN: Devuelve una CNN entrenada a partir de un set de
entrenamiento.


% Se crea un datastore para almacenar los audios de entrenamiento
trainDs = getAudioData('train');

% Se utiliza un subconjunto del set de 1000 audios.
trainDs = subset(trainDs, 1:1250);

% Se crea un tall array para importar de forma parcial los audios en
% memoria.
t = tall(trainDs);

% Se define el conversor de fs
src = dsp.SampleRateConverter("InputSampleRate",16000, ...
                              "OutputSampleRate",8000, ...
                              "Bandwidth",7920);

% Preprocesamiento de los audios.
[inputs,targets] = cellfun(@(x)featureTransform(x,
src),t,"UniformOutput",false);
[inputs, targets] = gather(inputs, targets);

% Normalización de los vectores magnitud.
[inputs, inputsMean, inputsStd]  = magNormalization(inputs);
[targets, targetsMean, targetsStd] = magNormalization(targets);

% Adaptación al formato de entrada de la cnn.
```

```
inputs  = adaptInputShape(inputs);
targets = adaptTargetShape(targets);

% Se divide el set de entrenamiento en entrenamiento y validación.
[trainInputs, trainTargets, validationInputs, validationTargets] =
...
    splitTrainToValidation(inputs, targets);

% Se definen los parámetros de la cnn.
[layers, options] = configureCNN (trainInputs, validationInputs,
validationTargets);

% Entrenamiento de la cnn.
denoisingCNN = trainNetwork(trainInputs, trainTargets, layers,
options);
```