Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

Universidad de Jaén

Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

REMIXING ESPACIAL EN REALIDAD AUMENTADA

Alumno: Jose Antonio Valero Martínez

Tutores: Prof. D. Julio José Carabias Orti y Raúl Mata Campos

Departamento de Ingeniería de Telecomunicación

2024

Universidad de Jaén

Escuela Politécnica Superior de Linares

Escuela Politécnica Superior de Linares
Grado en Ingeniería de Tecnologías de Telecomunicación
Trabajo Fin de Grado

REMIXING ESPACIAL EN REALIDAD AUMENTADA

Alumno	Tutor	Tutor
Jose Antonio Valero Martínez	Prof. D. Julio José Carabias Orti	Prof. D. Raúl Mata Campos
VALERO MARTINEZ JOSE ANTONIO - 77432562A Firmado digitalmente por VALERO MARTINEZ JOSE ANTONIO - 77432562A Fecha: 2024.09.05 06:31:24 +02'00'	CARABIAS ORTI JULIO JOSE - 53591108G Digitally signed by CARABIAS ORTI JULIO JOSE - 53591108G Date: 2024.09.05 12:39:47 +02'00'	MATA CAMPOS RAUL - 26480125H Firmado digitalmente por MATA CAMPOS RAUL - 26480125H Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-26480125H, givenName=RAUL, sn=MATA CAMPOS, cn=MATA CAMPOS RAUL - 26480125H Fecha: 2024.09.05 12:05:43 +02'00'

Índice:

Índice de Figuras

Índice de Fragmentos de Código

Índice de Ecuaciones

Índice de Tablas

1 INTRODUCCIÓN

La separación de fuentes sonoras y la realidad aumentada han sido, en los últimos tiempos, dos de las tecnologías que más han avanzado. Es por esto que en este trabajo se construye una relación simbiótica entre ambas tecnologías proporcionando así unos resultados novedosos e impactantes.

En los últimos años, la inteligencia artificial (IA) y, en particular, el Deep Learning, han experimentado un desarrollo exponencial. La IA, tecnología que en sus inicios se limitaba a la programación de reglas explícitas, ha evolucionado hasta llegar al punto en el que actualmente, contamos con modelos que pueden "*aprender*" y tomar decisiones a partir de grandes volúmenes de datos.

Estos grandes avances han sido consecuencia de las mejoras introducidas en los algoritmos de Deep Learning implementados. Técnicas como las redes neuronales convolucionales (CNN) así como las redes neuronales recurrentes (RNN) y todas sus variantes, como son las redes LSTM o las redes GRU, han revolucionado sus respectivos campos de aplicación.

Es gracias a estas mejoras en las redes neuronales que cada vez, y de una forma más fiel, se está consiguiendo recrear alguna de las capacidades y habilidades del ser humano, como son la capacidad de redacción, de identificar elementos en imágenes, la capacidad de producir imágenes o incluso la habilidad de identificar fuentes sonoras dentro de una pista de audio.

Este trabajo tiene como objetivo abordar el uso de redes neuronales con el fin de implementar distintas técnicas y procesos que permitan la obtención de cada una de las distintas fuentes sonoras que componen una mezcla sonora.

Para ello se han analizado las propuestas existentes de sistemas basados en redes neuronales, así como su rendimiento para, posteriormente, implementar una arquitectura de separación automática de fuentes musicales. Empleando las fuentes obtenidas para, en un entorno de RA, generar una experiencia interactiva que permitirá al usuario reproducir cada una de estas fuentes de manera independiente y controlada por él mismo.

1.1 Objetivos

Los principales objetivos que este trabajo pretende cubrir son los siguientes:

1. Realizar un estudio del estado del arte de las principales técnicas de separación de fuentes sonoras, con un enfoque particular en la separación de instrumentos musicales.
2. Analizar y comprender uno de los principales algoritmos de separación de fuentes musicales open source existentes, usando como base Open-Unmix.

3. Desarrollar el código en Python necesario para implementar un modelo convolucional que realice la separación automática de las fuentes sonoras que componen una mezcla.
4. Evaluar objetivamente los resultados obtenidos utilizando un conjunto de datos de prueba.
5. Implementar un sistema de remezcla de las fuentes separadas en un entorno de realidad aumentada (AR), creando un escenario interactivo.

1.2 Estructuración de la memoria

La memoria del trabajo se estructura en siete capítulos, descritos a continuación:

1. **Introducción:** En este primer capítulo, se ofrece una breve introducción al tema del trabajo, se explican los objetivos y se proporciona un resumen de la estructura de la memoria que permita al lector comprender de manera gradual la temática estudiada.
2. **Fundamentos Teóricos:** El segundo capítulo se centra en los fundamentos teóricos necesarios para entender el trabajo realizado. Se introducen conceptos relacionados con las redes neuronales, la representación y el enmascarado de audio. Además de esto se presentan las métricas utilizadas para realizar la evaluación objetiva de la separación de fuentes sonoras implementada.
3. **Estado del Arte:** En el tercer capítulo, se realiza un estudio del estado del arte enfocado principalmente en la separación de fuentes sonoras.
4. **Sistema de AR:** En el cuarto capítulo se explica y expone la información relacionada con la implementación en AR que ha sido desarrollada en este trabajo. Se expone una breve introducción, así como los distintos componentes y tecnologías empleados.
5. **Metodología:** El quinto capítulo detalla la metodología empleada en el trabajo, incluyendo los requisitos y recursos que han sido necesarios para el desarrollo de este trabajo, así como las arquitecturas de redes neuronales seleccionadas.
6. **Entrenamiento de Modelos:** En el sexto capítulo, se discuten los detalles del entrenamiento del modelo convolucional desarrollado. En este capítulo, además, se incluyen los códigos y herramientas empleados para realizar este entrenamiento, así como consideraciones prácticas del proceso de entrenamiento.
7. **Resultados y Evaluación:** El séptimo capítulo presenta los resultados obtenidos. En este apartado se exponen los resultados obtenidos de la implementación en AR así como se comparan y exponen los valores de evaluación objetivos obtenidos de las arquitecturas de redes neuronales empleadas, comparando el modelo original, Open-Unmix, con el modelo implementado Convolutional-Unmix.
8. **Conclusiones y Futuras Líneas de Investigación:** En el octavo y último capítulo, se recogen las conclusiones del trabajo y se sugieren posibles líneas de investigación futura.

1.3 Estructura del trabajo

Este trabajo fin de grado está conformado por dos grandes bloques, la separación de fuentes sonoras mediante redes neuronales y el remixing de las mismas en realidad aumentada (AR) como podemos apreciar en el esquema recogido en la Figura 1.

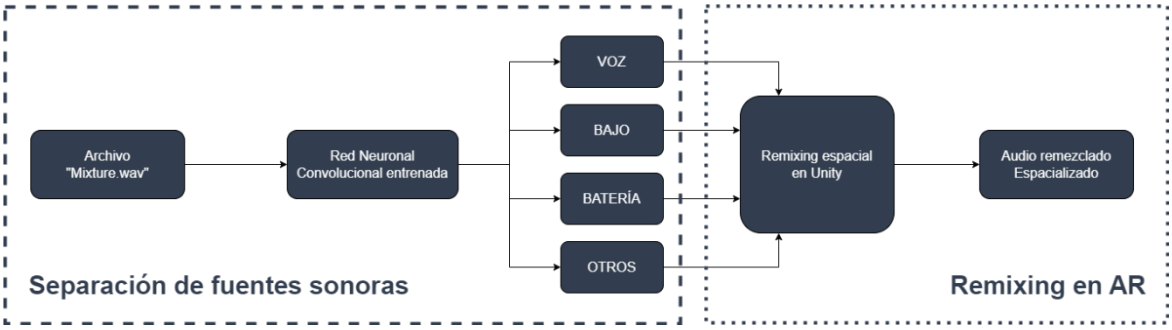


Figura 1 Diagrama de Flujo del trabajo

1.3.1 Separación de fuentes sonoras

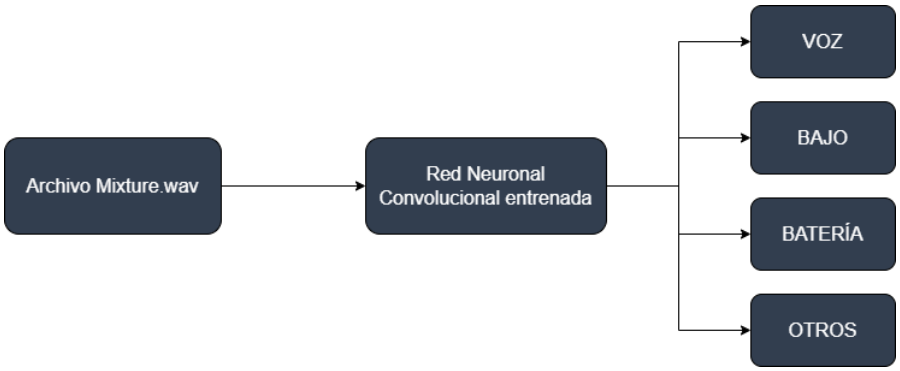


Figura 2: Esquema de la separación de fuentes

Para implementar la separación de las fuentes sonoras se ha empleado una red neuronal convolucional la cual al proporcionarle un archivo de audio de una composición (mixture) proporciona como salida cada una de las fuentes que componen la composición.

1.3.2 Remixing en AR

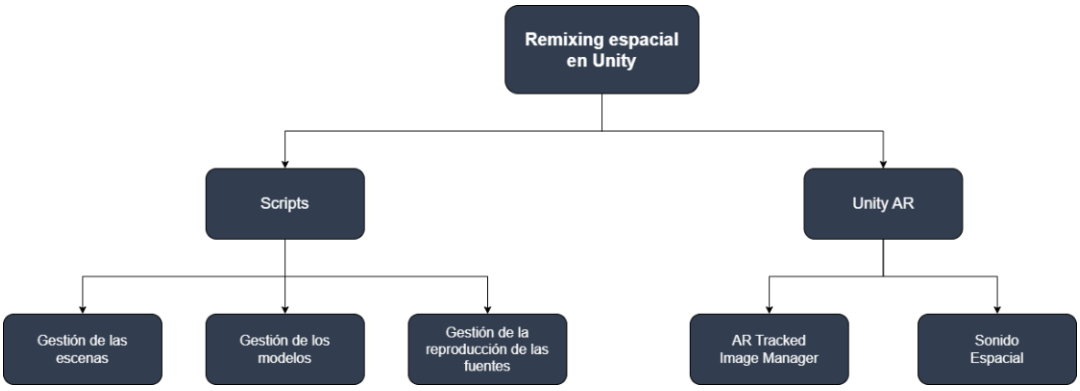


Figura 3 Componentes del módulo AR

El remezclado de las fuentes, o remixing en inglés, consiste en la fusión de varias fuentes sonoras en una sola pista de audio.

En este trabajo se ha querido ir más allá y proporcionar una experiencia inmersiva e interactiva con la que poder comprender e interactuar con la separación de las fuentes realizada.

Esta experiencia no consiste únicamente en el remix de las diferentes pistas obtenidas, sino que, además, se aplican efectos de espacialidad, permitiendo al usuario obtener la sensación de que las fuentes se encuentran un espacio tridimensional en el que el usuario es capaz de identificar e interactuar con las posiciones de las fuentes.

2 FUNDAMENTOS TEÓRICOS

Este capítulo se centrará en explicar aquellos conceptos fundamentales y necesarios para comprender el funcionamiento y comportamiento de las redes neuronales implementadas en el sistema de separación automática de fuentes sonoras.

Debido a la complejidad de los métodos y procesos de separación utilizados, se detallarán únicamente aquellos componentes y parámetros que desempeñan un papel crucial y son utilizados en este trabajo.

Comenzaremos con una introducción a los principios básicos del Machine Learning y las redes neuronales. Para, posteriormente, abordar los aspectos principales de la representación de señales de audio. Continuando así con la introducción del concepto de enmascaramiento en el dominio tiempo-frecuencia, la cual es una técnica utilizada para la separación de fuentes sonoras. Para, finalmente, acabar con la explicación de las métricas y parámetros objetivos empleados para evaluar el modelo implementado.

2.1 Machine Learning

El Machine Learning, o aprendizaje automático en español, es una rama de la inteligencia artificial (IA) la cual dota a las máquinas de la capacidad de "*aprender*" a partir de un conjunto de datos especificados.

La principal diferencia respecto a la programación tradicional, en la que se deben especificar instrucciones detalladas, es que los algoritmos de Machine Learning extraen conocimiento de forma autónoma a partir de grandes conjuntos de datos, identificando patrones y produciendo relaciones complejas.

La capacidad de generar aprendizaje con la que cuentan estos sistemas se apoya en métodos computacionales enfocados a mejorar su rendimiento de forma recursiva, de manera que, cuantos más datos sean analizados y más ejecuciones se realicen mejores resultados proporcionará el sistema. Además, estos sistemas cuentan con una alta capacidad de adaptabilidad ya que el comportamiento de estos sistemas cambia con cada ejecución en función de los datos introducidos.

El Machine Learning permite no tener que implementar algoritmos y ecuaciones destinadas exclusivamente para una tarea específica, si no que, son las propias máquinas las que “*aprenden*” y generan conocimiento a partir de los datos de entrenamiento introducidos de manera que, conforme aumenta el número de muestras a la entrada y el número de ejecuciones realizadas, mejores serán los resultados proporcionados.

2.1.1 Clasificación de los algoritmos

Los algoritmos utilizados para el entrenamiento de sistemas Machine Learning pueden clasificarse en tres tipos en función de los datos proporcionados a la entrada y la salida esperada:

1. **Aprendizaje Supervisado:** los algoritmos son entrenados a partir de un conjunto de datos etiquetados, donde cada dato tiene una etiqueta asociada que indica su categoría o valor correcto, es decir, el valor que debería de obtenerse a la salida del mismo. El objetivo del algoritmo es aprender a relacionar y referenciar los datos de entrada con las salidas esperadas. Algunos ejemplos de algoritmos de aprendizaje supervisado incluyen:
    - **Regresión:** Se utiliza para predecir valores numéricos continuos.
    - **Árboles de decisión:** Se utiliza para crear modelos de decisión que se puedan interpretar fácilmente y que permiten comprender la toma de decisiones del algoritmo.
    - **Clasificación:** Se utiliza para segmentar y organizar datos en una serie de categorías definidas por unos parámetros o características.
2. **Aprendizaje No Supervisado:** los algoritmos aprenden a partir de un conjunto de datos no etiquetados. El objetivo del algoritmo es descubrir patrones o estructuras ocultas en los datos. Algunos ejemplos de algoritmos de aprendizaje no supervisado incluyen:
    - **Agrupación:** Se utiliza para unir datos en grupos con características similares.
    - **Reducción de dimensionalidad:** Se utiliza para reducir el número de variables en un conjunto de datos sin perder información relevante.
    - **Detección de anomalías:** Se utiliza para separar aquellos datos que no siguen el patrón del resto de los datos en el conjunto.
3. **Aprendizaje por Refuerzo:** los algoritmos aprenden a partir de la interacción con un entorno definido. El objetivo del algoritmo es encontrar la secuencia de acciones que maximiza una recompensa (aciertos) y minimiza las penalizaciones (errores). Algunos ejemplos de algoritmos de aprendizaje por refuerzo incluyen:
    - **Aprendizaje Q:** Se utiliza para aprender una función de valor la cual asignará un valor para cada uno de los estados posibles del entorno.
    - **Aprendizaje con diferencias temporales (TD):** Se utiliza para aprender una función de valor la cual asigna un valor a cada estado posible en el entorno y a cada acción posible.
    - **Aprendizaje actor-crítico:** Se utiliza para aprender una serie de políticas las cuales determinan cual será la acción que debe tomar el agente en cada uno de los estados posibles.

El conjunto de datos con el que trabajan estos algoritmos suele dividirse en los siguientes subconjuntos:

- **Datos de entrenamiento:** Este subconjunto, el más numeroso ya que su finalidad es entrenar el modelo. Estos datos son los que utiliza el algoritmo para aprender a identificar patrones y relaciones, ajustando sus parámetros (pesos y umbrales) para minimizar el error de predicción. La calidad y el tamaño del conjunto de entrenamiento son cruciales para el éxito del modelo.
- **Datos de validación:** Este subconjunto, de menor tamaño que el de entrenamiento, se utiliza para evaluar el rendimiento del modelo durante el entrenamiento. Permite ajustar los hiperparámetros del modelo, como la tasa de aprendizaje o la arquitectura neuronal, para optimizar su generalización a datos nuevos. Los datos de validación no deben utilizarse para entrenar el modelo, ya que esto podría llevar al sobreajuste.
- **Datos de prueba**: Este subconjunto, el más pequeño de los tres, se utiliza para la evaluación final del modelo una vez que ha sido entrenado y validado. Los datos de prueba no deben haber sido utilizados en ninguna etapa previa del proceso.

2.1.2 Deep Learning

El Deep Learning, o aprendizaje profundo en español, se presenta como una evolución natural del Machine Learning.

Si entendemos una red neuronal artificial como un complejo laberinto de neuronas artificiales interconectadas entre sí, de forma análoga al funcionamiento del cerebro humano, podremos generar un proceso de entrenamiento exhaustivo donde estas neuronas "*aprenden*" a identificar patrones complejos en grandes cantidades de datos.

A diferencia del Machine Learning tradicional, que requiere una definición previa de las características relevantes, el Deep Learning opera de forma autónoma. Las redes neuronales profundas, compuestas por múltiples capas, son capaces de extraer automáticamente las características más significativas de los datos, permitiendo un análisis más profundo y preciso.

2.1.3 Redes neuronales

Uno de los modelos más importantes que pertenecen al Machine Learning son las Redes Neuronales o también conocidas como Redes Neuronales Artificiales (ANN).

Una red neuronal es una implementación de Deep Learning en la que se imita tanto en estructura como en funcionalidades el cerebro humano. Una red neuronal tiene como finalidad el aprendizaje y automatización de procesos, de manera que, una vez han sido entrenadas, se convierten en importantes herramientas de gestión de datos.

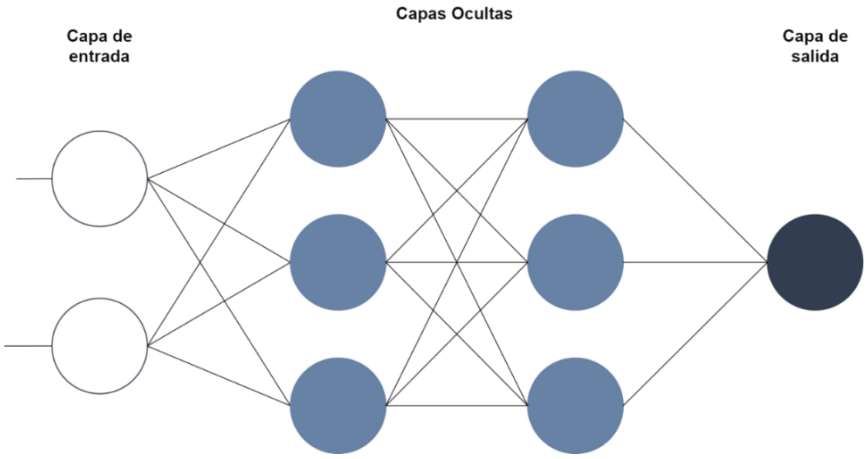


Figura 4 Diagrama de una red neuronal

Estas redes están constituidas por capas formadas por una serie de nodos interconectados entre sí, denominados neuronas artificiales. En la Figura 4 Diagrama de una red neuronal se muestra la estructura de una red neuronal básica, esta red cuenta con una capa de entrada, dos capas ocultas y una capa de salida. En este ejemplo la red contendría dos variables de entrada, tres neuronas en cada una de las dos capas ocultas y una en la capa de salida.

2.1.3.1 Neuronas artificiales

Una neurona artificial es la unidad básica de cálculo que conforma una red neuronal y reciben este nombre dada la similitud que presentan respecto a las neuronas humanas.

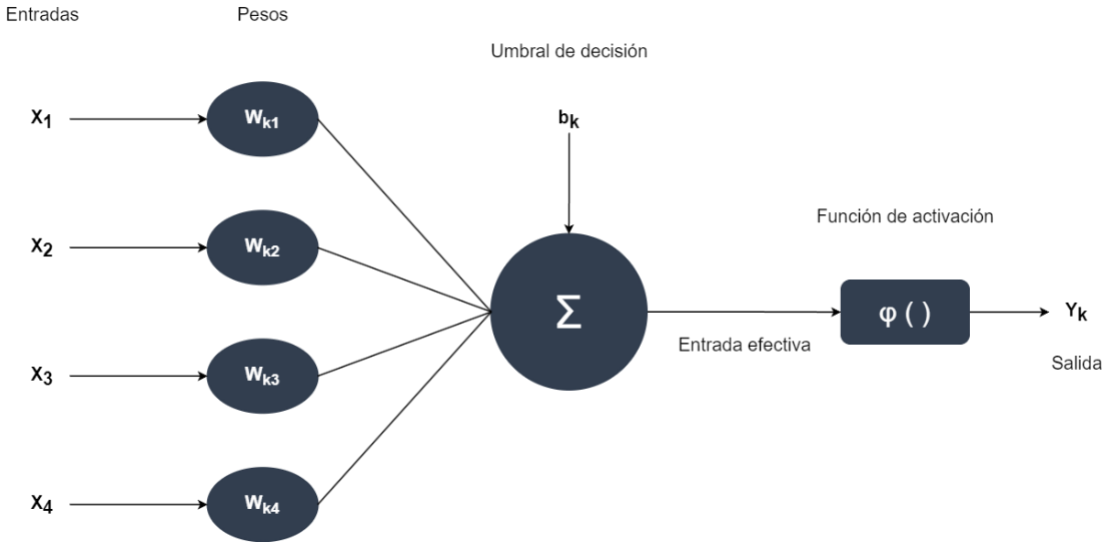


Figura 5 Diagrama de una neurona artificial

2.1.3.1.A *Salida* (Y_k):

La salida de una neurona artificial es un valor numérico, valor el cual se obtiene como el resultado del procesamiento de la entrada efectiva $\left(\sum_j w_{kj}x_j + b_k\right)$ a través de la función de activación $\varphi(\,)$ como se puede apreciar en la Ecuación 1.

$$Y_k = \varphi\left(\sum_j w_{kj} x_j + b_k\right)$$

Ecuación 1: Salida de una neurona artificial

En la ecuación se aprecian varios parámetros, los cuales se explicarán a continuación:

2.1.3.1.B *Entradas* (x_j)*:*

Las entradas representan aquellos datos que son proporcionados para ser procesados por la neurona. Estos datos pueden provenir de otras neuronas intermedias o ser obtenidos directamente de la entrada de la red.

2.1.3.1.C *Pesos* (W_{kj})*:*

Factor de importancia o influencia aplicado a cada una de las entradas de la neurona. Los pesos actúan como multiplicadores, ajustando la magnitud de la señal que se transmite de una neurona a otra. Un peso positivo indica una influencia excitatoria, mientras que un peso negativo indica una influencia inhibitoria.

Durante el entrenamiento de la red neuronal, se ajustan los pesos de forma iterativa buscando minimizar el error en la salida de la red. Este ajuste se basa en algoritmos de optimización como el descenso del gradiente, que buscan encontrar la combinación de pesos que optimice el rendimiento de la red.

Además, proporcionan información valiosa sobre la importancia de cada característica de entrada en la toma de decisiones de la red neuronal ya que un peso alto para una entrada indica que la neurona es sensible a esa característica y esta afecta en mayor medida en su proceso de activación

2.1.3.1.D *Umbral de decisión* (b_k)*:*

El umbral de decisión, también conocido como bias, es un valor constante que se suma a la entrada ponderada de una neurona artificial antes de aplicar la función de activación produciendo así la **entrada efectiva** (Entradas ponderadas + Umbral).

Este umbral permite ajustar la sensibilidad de la neurona a las entradas donde un umbral mayor necesitará unas entradas más fuertes para activarse, mientras que un umbral menor permitirá entradas más débiles.

2.1.3.1.E *Función de activación* $(\varphi(\,))$*:*

La función de activación en una neurona artificial es un componente crucial en las redes neuronales ya que produce la salida de la neurona a partir de la entrada efectiva.

Esta función introduce no linealidad en el modelo, lo que permite que la red neuronal aprenda y desarrolle relaciones complejas entre las entradas y las salidas de la misma. Sin una función de activación, la red neuronal se comportaría como un sistema lineal, limitando significativamente su capacidad para resolver problemas complejos.

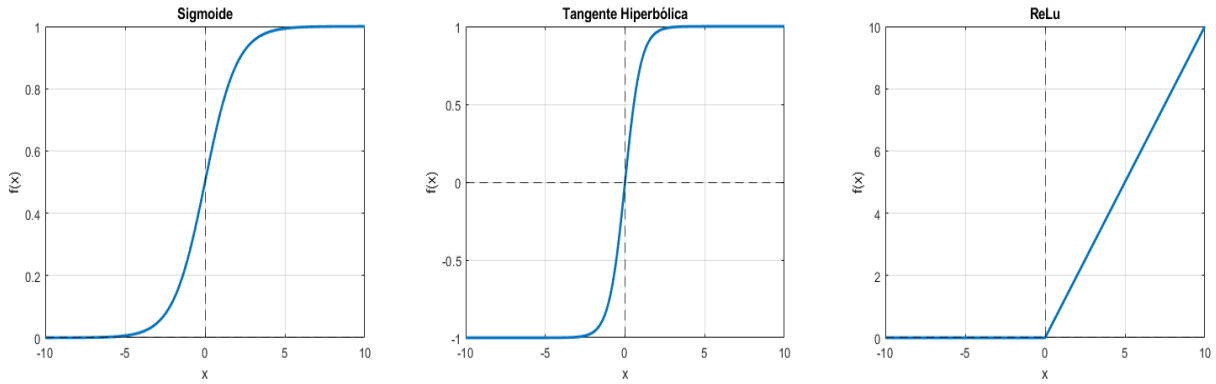


Figura 6: Funciones de Activación

E.1 Función Sigmoide $\sigma(x)$:

- **Rango:** (0, 1)
- **Ventajas:** Es una función suave y diferenciable lo que facilita el cálculo del gradiente durante el entrenamiento.
- **Desventajas:** Para valores extremos de x en los que la pendiente es extremadamente plana se produce el desvanecimiento del gradiente.

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

Ecuación 2: Sigmoide

E.2 Tangente Hiperbólica $tanh(x)$:

- **Rango:** (-1, 1)
- **Ventajas:** Es muy similar a la Sigmoide salvo porque está centrada en 0, lo que puede ayudar a que el algoritmo de aprendizaje converja más rápidamente.
- **Desventajas:** Esta función también sufre de problemas de desvanecimiento del gradiente.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Ecuación 3: Tangente Hiperbólica

E.3 ReLu $f(x)$:

- **Rango:** [0, ∞)

- Ventajas: Es una función simple, computacionalmente eficiente y consigue mitigar el problema del desvanecimiento del gradiente.
- Desventajas: Puede causar la "*muerte*" de neuronas durante el entrenamiento si estas quedan atrapadas en la región donde X es siempre negativo.

$$f(x) = \max(0, x)$$

Ecuación 4: ReLu

2.1.3.2 Entrenamiento de la red neuronal

El entrenamiento de una red neuronal artificial es un proceso iterativo que busca optimizar los parámetros de la red para que pueda realizar una tarea específica de manera precisa y eficiente.

El proceso de entrenamiento se lleva a cabo a través de varios pasos, realizados de forma iterativa:

1. **Propagación hacia adelante (Forward):** Se alimenta la red con los datos del subconjunto destinados al entrenamiento de la red de manera que se recorre la red neuronal completa obteniendo así la salida de la red.
2. **Función de pérdida (Loss Function):** Se evalúan los resultados obtenidos a la salida de la red con los valores reales obteniendo así una medida del error cometido por la red.
3. **Propagación hacia atrás (Backward):** Se utiliza el algoritmo de retropropagación (backpropagation) con el fin de calcular el gradiente de la función de pérdida respecto a cada uno de los pesos y umbrales utilizados en la red.
4. **Actualización de los parámetros:** Se actualizan los pesos y umbrales en función de los gradientes obtenidos con el objetivo de reducir el error obtenido a la salida. Para realizar esta actualización de los parámetros se hará uso de un algoritmo de optimización como es el algoritmo de **descenso de gradiente de mini lotes**.

Estos pasos son repetidos múltiples veces hasta completar lo que se conoce como una época.

Se dice que se ha completado una época cuando todos los datos del subconjunto de entrenamiento seleccionado han sido introducidos en la red. Durante el entrenamiento de una red neuronal tienen lugar varias épocas hasta conseguir los resultados deseados, momento en el que el error converge en un valor deseado.

2.1.3.2.A *Problemas de Sobreajuste:*

Al entrenar una red neuronal no se busca un valor nulo en el cálculo del error ya que esto produciría un sobreajuste (overfitting) en la red neuronal.

Este sobreajuste puede dar lugar a situaciones en las que, aunque la red presenta un error de entrenamiento bajo, se obtienen errores de prueba bastante elevados los cuales son producido como consecuencia de un modelo que ha memorizado el ruido, así como los patrones de los datos usados para el entrenamiento, consiguiendo resultados muy precisos para este subconjunto, aunque es incapaz de generalizar y proporcionar unos resultados correctos para el subconjunto de datos de prueba.

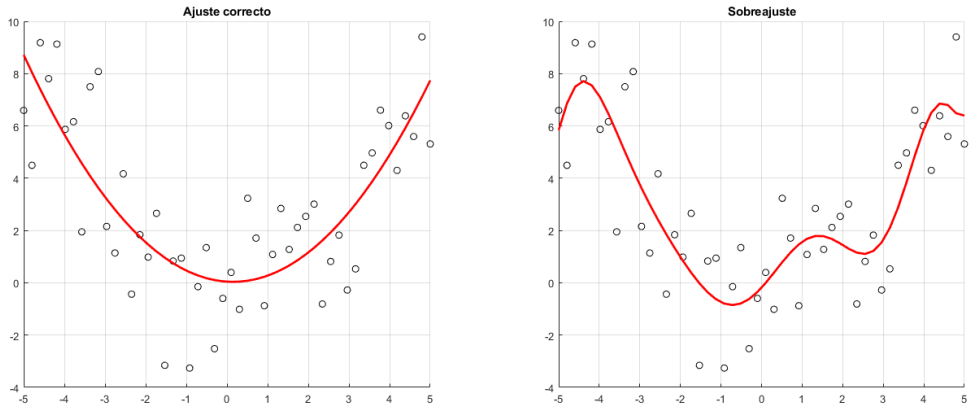


Figura 7: Efecto del sobreajuste

Con el fin de evitar el problema del sobreajuste se pueden implementar varias técnicas entre las más destacadas se encuentran:

- **Aumento de datos de entrenamiento:** Se utilizan más ejemplos de entrenamiento creados a partir de transformaciones como rotaciones, escalados o traslaciones a las entradas originales, generando así nuevas muestras similares. Consiguiendo así mejorar la capacidad del modelo para generalizar.
- **Empleo de parámetros compartidos:** Al compartir parámetros en diferentes partes de la red, se reduce el número total de parámetros del modelo que deben calcularse y optimizar. Esto simplifica el modelo y nos permite evitar el sobreajuste al limitar la capacidad del modelo para memorizar patrones específicos en los datos de entrenamiento.
- **Parada temprana (Early Stopping):** Durante el entrenamiento, se monitoriza el error en un conjunto de validación. Si este error comienza a aumentar de forma no deseada, se detendrá el entrenamiento de la red y se seleccionarán aquellos pesos del modelo que nos proporcionasen la menor pérdida en la validación.
- **Normalización por lotes (Batch Normalization):** Se añade un paso adicional entre las neuronas y la función de activación para normalizar las salidas. Para cada iteración, se calcula la media y la varianza de cada una de las salidas para el mini lote con el fin de ajustar las salidas de manera que estas tengan media de cero y una desviación uno. Esta regularización además de estabilizar el aprendizaje también acelera la convergencia.

- **Dropout:** Durante el entrenamiento de la red un porcentaje de las neuronas que forman cada una de las capas se "*matan*" aleatoriamente. Esto obliga a la red a no depender de ninguna neurona en particular, evitando así la memorización de patrones en los datos, así como una mejora de su capacidad de generalización al aprender características más robustas.

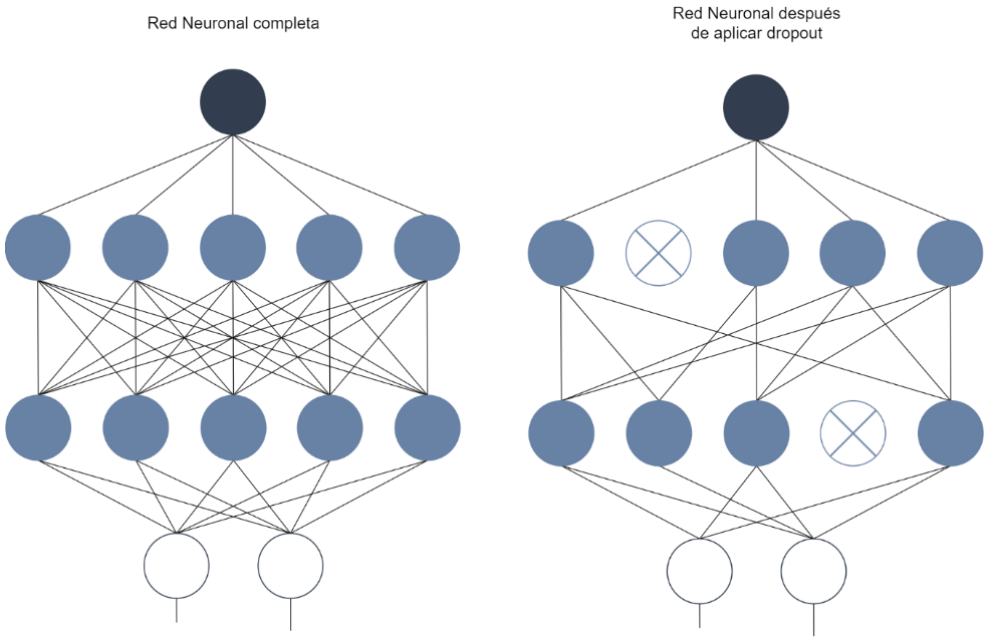


Figura 8: Ejemplo gráfico del DropOut

Estas técnicas, aplicadas de manera adecuada, ayudan a mejorar el rendimiento de los modelos de redes neuronales y a prevenir el sobreajuste, permitiendo una mejor generalización.

A.1 Descenso de gradiente de mini lotes

Esta implementación, en lugar de actualizar los parámetros de toda la red utilizando la información de error de todo el conjunto de entrenamiento en cada iteración, utiliza subconjuntos más pequeños de datos, llamados mini lotes. De manera que se realizan las actualizaciones en cada uno de esos mini lotes consiguiendo así calcular el gradiente y actualizar los parámetros de forma más computacionalmente eficiente.

2.1.3.3 Redes neuronales convolucionales (CNN)

Las Redes Neuronales Convolucionales (CNN) destacan frente a otras redes neuronales por su efectividad y mayor rendimiento a la hora de identificar, reconocer y clasificar datos bidimensionales como son imágenes, señales de voz o señales de audio.

Las redes neuronales convolucionales están compuestas principalmente por tres tipos de capas:

2.1.3.3.A *Capas Convolucionales*

Las capas convolucionales son el bloque principal de estos tipos de redes. Es en estas capas donde se realiza la mayor parte del procesado de los datos.

Una capa convolucional implementa un detector de características denominado filtro, máscara o Kernel en inglés. Este filtro recorrerá de manera secuencial los datos bidimensionales en búsqueda de una característica específica. El desplazamiento de la máscara a través del conjunto completo de los datos bidimensionales se denomina convolución.

Con cada capa convolucional que se implementa en la red, se aumenta gradualmente la complejidad y capacidad de detección de la CNN. Donde las primeras capas se centran en características simples como puede ser detectar bordes, cambios bruscos de niveles de intensidad entre otras características y, conforme avanzan los datos a través de las siguientes capas, la red va reconociendo elementos o patrones más grandes y complejos hasta que finalmente poder identificar el elemento deseado.

Los parámetros que definen una capa convolucional y determinan su comportamiento son los siguientes:

- **Tamaño de la máscara o kernel:** Determina el alto y ancho del filtro que se aplicará a la entrada. Los filtros suelen ser cuadrados con unas dimensiones impares (3x3, 5x5) lo que permite tener un elemento central.
- **Paso:** Determina la distancia de desplazamiento entre muestras, cuanto mayor sea este valor, mayor será la compresión aplicada a los datos de entrada.
- **Padding o relleno:** Determina el comportamiento o actuación en los bordes de los datos, ya que dado que contamos con un número de muestras finitas que tienen unas dimensiones limitadas. El padding nos permite definir como se comportará la máscara en los bordes, podemos añadir ceros de manera que se mantengan las dimensiones originales o no usar padding, lo que producirá una distorsión de aquellos datos que se encuentren en los bordes.
- **Dilatación:** Determina el espacio de separación que se mantiene entre los elementos del filtro. La implementación de dilatación puede permitirnos identificar características a diferentes escalas.

En los gráficos que se aprecian a continuación encontramos en la Figura 8 un ejemplo de convolución con padding, mientras que, en la Figura 9 podemos apreciar un ejemplo de convolución con dilatación.

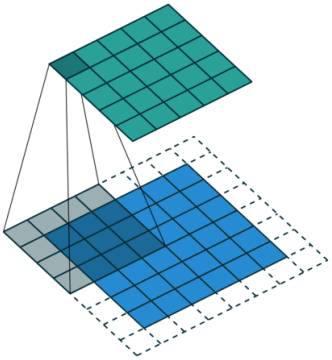


Figura 9: Convolución con padding [1]

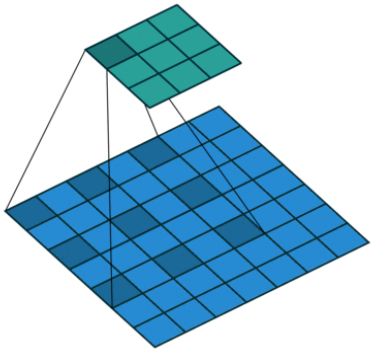


Figura 10: Convolución con dilatación [1]

2.1.3.3.B *Capas de agrupación*

Las capas de agrupación, también conocidas como capas de submuestreo, permiten reducir las dimensiones de los datos mediante la reducción del número de parámetros de entrada. Esta capa permite reducir la complejidad de la red, limita el riesgo de sobreajuste y mejora la eficiencia, aunque supone una alta tasa de pérdida de información.

Una capa de agrupación actúa de manera análoga a las capas convolucionales, se recorren todos los datos de entrada con una máscara, sin embargo, en este caso la máscara usada no tiene aplicada ningún peso. En su lugar, al aplicar la máscara se ejecuta una función de agregación a los valores de entrada completando así la matriz de salida.

Hay dos tipos principales de agrupación:

- **Agrupación máxima:** conforme el filtro recorre los datos de entrada, selecciona el valor máximo, valor que será enviado a la matriz de salida.
- **Agrupación media:** conforme el filtro recorre los datos de entrada, este calcula el valor medio, el cual, será enviado a la matriz de salida.

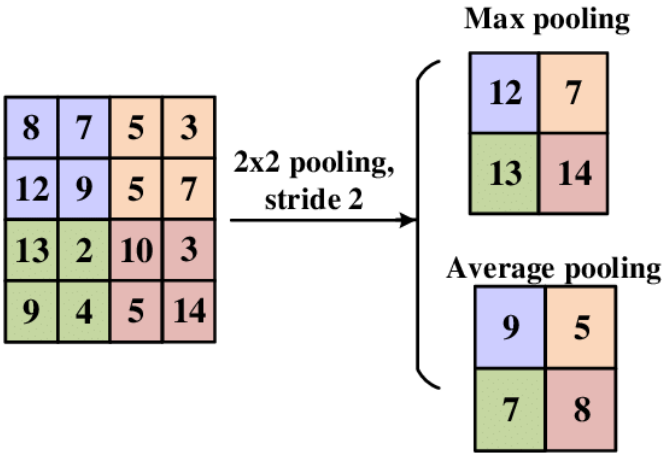


Figura 11: Agrupación Máxima y agrupación Media [2]

2.1.3.3.C *Capa Completamente Conectadas*

Esta capa es la encargada de realizar la clasificación de los datos basándose en las características obtenidas en las capas anteriores.

Mientras que las capas convolucionales y de agrupación suelen utilizar funciones ReLu, las capas totalmente conectadas suelen usar la función de activación softmax para clasificar correctamente las entradas y generar así una probabilidad entre 0 y 1.

2.1.3.4 Redes neuronales recurrentes (RNN)

Una red neuronal recurrente (RNN) es un tipo de red neuronal que utiliza datos secuenciales. Estas redes se diferencian de las redes neuronales tradicionales porque estas redes cuentan con conexiones recurrentes que permiten la persistencia de información, lo que produce que estas redes sean especialmente útiles para tareas en las que el contexto y la ordenación temporal son importantes.

Esto se consigue gracias a la capacidad que presentan estas redes para compartir los valores de los parámetros entre las distintas capas que conforma la red, consiguiendo así obtener información de entradas anteriores para obtener nuevos valores que podrán influir en la entrada y salida actuales.

En el ámbito al que está dirigido este trabajo, la separación de fuentes sonoras, las capas recurrentes tienen un papel fundamental ya que estas actúan de manera que el audio es incorporado de manera dinámica a medida que este evoluciona a lo largo del tiempo. Lo que permite, por ejemplo, a la hora de analizar un espectrograma en la capa recurrente, incorporar de forma dinámica una columna cada vez hasta obtener el espectro completo.

Las RNN implementan un algoritmo llamado *propagación hacia atrás a través del tiempo (BPTT)* para calcular los gradientes de error de la red.

Las capas recurrentes más utilizadas en la separación de fuentes son las de Memoria a Largo y Corto Plazo (LSTM). Estas capas tienen celdas especiales con tres puertas: una de entrada, una de salida y una puerta de olvido, que regulan el flujo de información dentro de la red para prever la salida de manera efectiva.

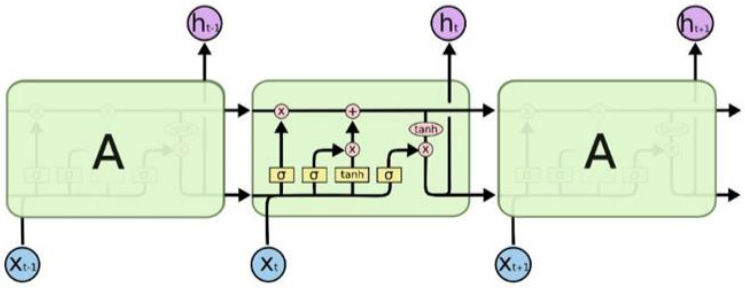


Figura 12: Implementación de un módulo LST [3]

2.2 Representación de las señales de audio

2.2.1 Forma de onda

La forma de onda es la representación gráfica más usada para describir la variación de una señal a lo largo del tiempo. Una forma de onda permite identificar las variaciones de presión o amplitud del sonido en función del tiempo.

La forma de onda puede describir tanto señales periódicas, como una onda sinusoidal que representa un tono puro, como señales más complejas que describen sonidos más ricos y variados, como la música o la voz humana.

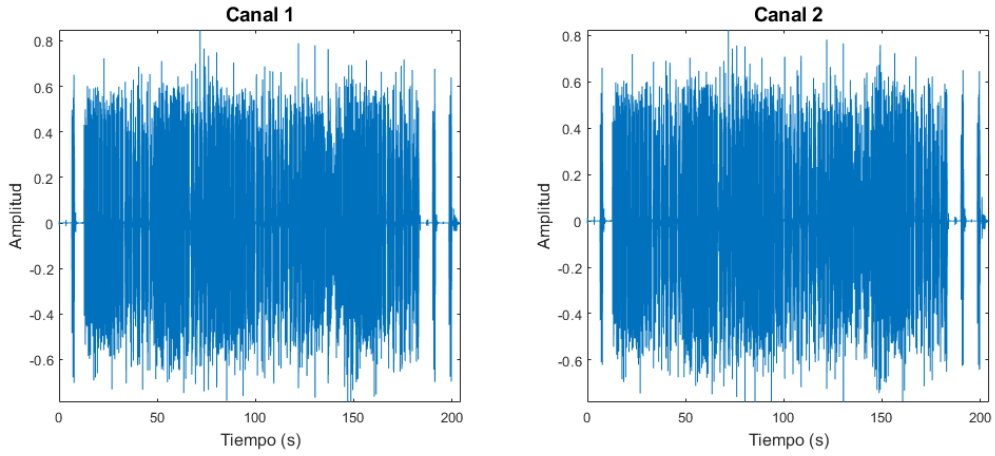


Figura 13: Forma de Onda de una señal

En la representación de la forma de onda de cualquier señal se encuentra toda la información de la misma, sin embargo, su interpretación directa no es una tarea sencilla y puede resultar compleja. Es por este motivo por el cual, para obtener un enfoque que nos permita una interpretación más directa, se opta por trabajar en el dominio frecuencial generando representaciones tiempo-frecuencia.

2.2.2 Representación en tiempo-frecuencia

Una representación en tiempo-frecuencia se define como una matriz bidimensional en la cual se describe el contenido frecuencial de una señal de audio a lo largo del tiempo.

Esta representación es la más usada para los distintos enfoques utilizados en el proceso de separación de fuentes. En particular, la representación tiempo-frecuencia denominada Transformada de Fourier de tiempo corto (STFT, por sus siglas en inglés: Short-Time Fourier Transform).

2.2.3 Transformada de Fourier de tiempo corto (STFT)

La Transformada de Fourier de Tiempo Corto (STFT) es una técnica utilizada en el procesamiento de señales para analizar cómo varían las frecuencias de una señal a lo largo del tiempo en una señal.

La STFT se obtiene a partir de fragmentos de tiempo de la señal completa de manera que, se calcula la Transformada Discreta de Fourier (DFT) para cada uno de estos segmentos de tiempo determinados al aplicar una ventana $w[n]$ sobre la señal original, la cual cuenta con un número de muestras N. Esta ventana se va desplazando a lo largo de la señal. La expresión matemática que describe este proceso es la siguiente:

$$X[n,k] = \sum_{m=-\infty}^{\infty} x[m]w[m-n]e^{-j\frac{2\pi km}{N}}$$

Ecuación 5: Cálculo de la STFT

Los coeficientes obtenidos del cálculo de la STFT, constituyen una matriz con valores complejos, es decir, son valores que cuentan con una magnitud y fase. Ambas componentes serán necesarias para realizar la conversión de la señal a su forma de onda.

A partir de las componentes de magnitud obtenidos de la STFT se obtiene una representación gráfica denominada espectrograma. El espectrograma representa la señal en los ejes tiempo-frecuencia, siendo el eje de abscisas la representación temporal, y el de ordenadas la representación frecuencial. Además de estas dos dimensiones un espectrograma cuenta con el valor absoluto de la STFT el cual se representa en cada punto mediante una escala de color que refleja la intensidad de señal en un punto específico, es decir, para una frecuencia y tiempo determinados. Esta escala se representa normalmente en decibelios (dB).

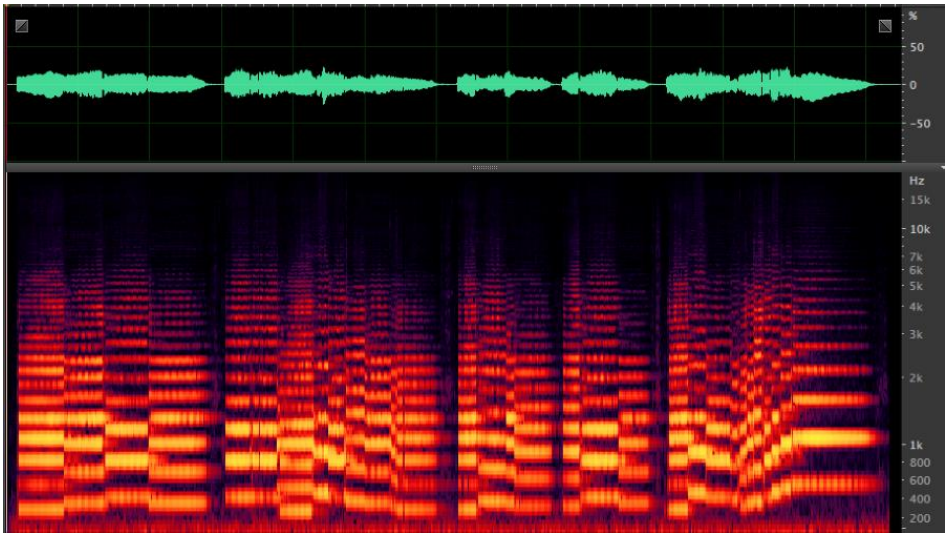


Figura 14: Ejemplo de un espectrograma [4]

2.2.3.1 Parámetros que afectan en el cálculo de la STFT

En el cálculo la STFT encontramos dos parámetros que son los más influyentes como son el enventanado aplicado a la señal y el tamaño de salto.

2.2.3.1.A *Propiedades del enventanado*

La ventana que seleccionemos para el cálculo de la STFT distorsionará el espectro obtenido de la señal, bien produciendo picos en frecuencias incorrectas (fuga) debido a los lóbulos secundarios del espectro de la ventana, o bien, produciendo pérdidas de resolución en frecuencia a causa del ancho del lóbulo principal de la ventana.

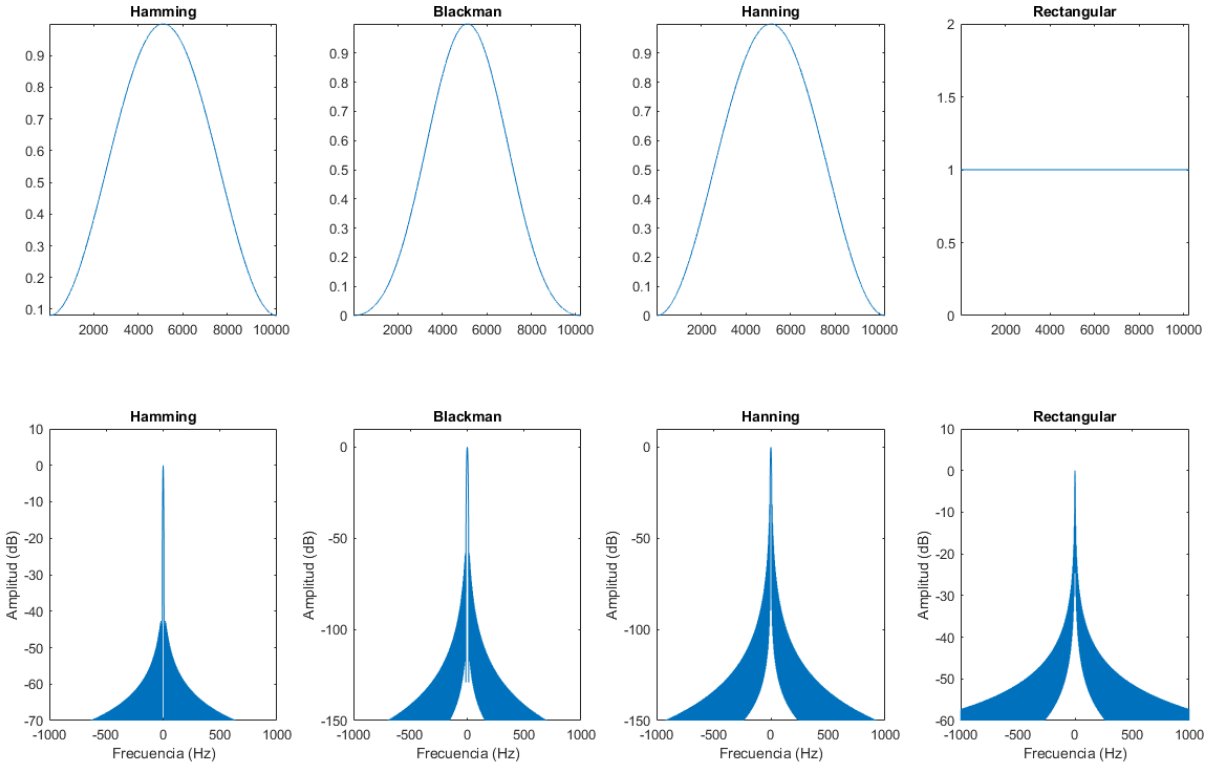


Figura 15: Ventanas empleadas

Con el fin de disminuir las distorsiones espectrales que producen el enventanado, se han estudiado varios tipos de ventanas de las cuales cada una ofrece un rendimiento distinto. La Ventanas empleadas Figura 15 muestra los cuatro tipos de ventana más comunes en el cálculo de la STFT para la separación de fuentes.

Al enventanar la señal original, la ventana seleccionada afectará a la fidelidad con la que se obtendrán los resultados de la STFT. Como podemos apreciar en la *Figura 16*, cada una de las ventanas genera una señal diferente, efecto producido por la forma de la ventana en el dominio temporal.

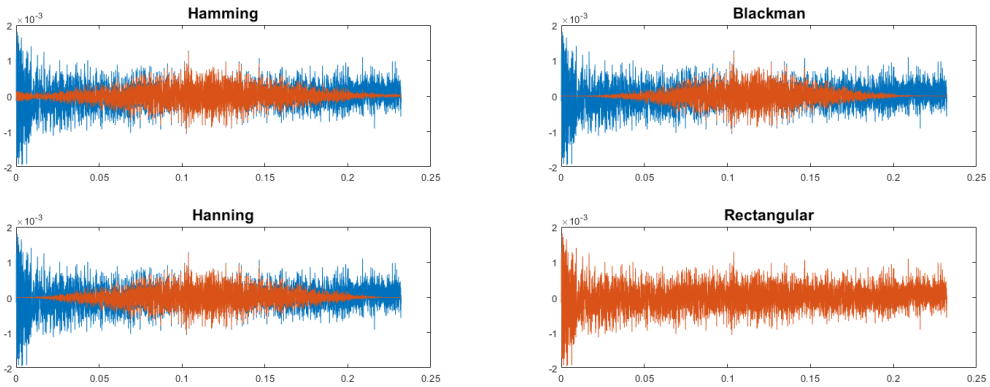


Figura 16: Efectos del enventanado en la señal

Además de su forma, el tamaño de la ventana también afecta en los resultados obtenidos ya que este determina el número de muestras que serán procesadas, así como la resolución del eje frecuencial de la STFT. Existe una relación inversa entre la resolución temporal y frecuencial de manera que, cuanto mayor sea el tamaño de la ventana, es decir, menor resolución en el dominio temporal, mayor será la resolución en el dominio frecuencial y viceversa.

2.2.3.1.B *Tamaño de salto*

El tamaño de salto es un parámetro que nos permite definir la distancia en muestras que hay entre dos ventanas consecutivas.

Si este tamaño de salto es menor que el tamaño de la ventana empleada se producirá solapamiento en el cálculo de la STFT. Este solapamiento puede ser una ventaja si implementamos técnicas como el overlap-add.

Esta técnica es empleada en el procesamiento de señales para reconstruir una señal de una duración elevada a partir de fragmentos procesados individualmente, es decir, esta técnica nos permite calcular las FFT de cada una de las señales enventanadas y combinarlas de manera que se obtenga la STFT de la señal completa.

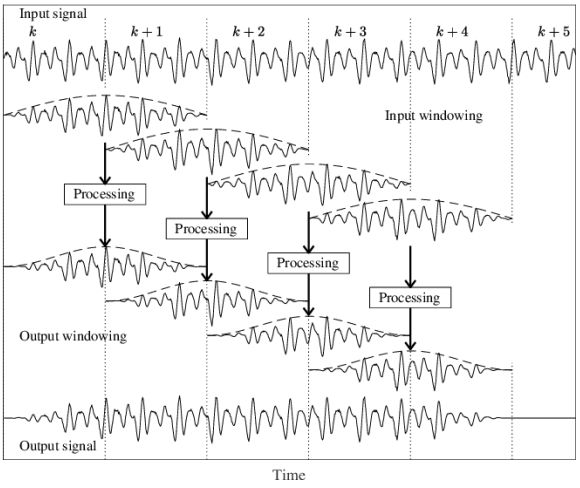


Figura 17: Overlap-add [5]

2.3 Máscaras tiempo-frecuencia

La representación tiempo-frecuencia que hemos obtenido es una herramienta crucial en la separación de las fuentes sonoras, esta representación nos permite implementar de forma simple la separación mediante máscaras tiempo-frecuencia.

Estas máscaras se aplican sobre los espectrogramas de manera que se produce un filtrado de la señal original conservando a la salida únicamente una de las fuentes deseadas.

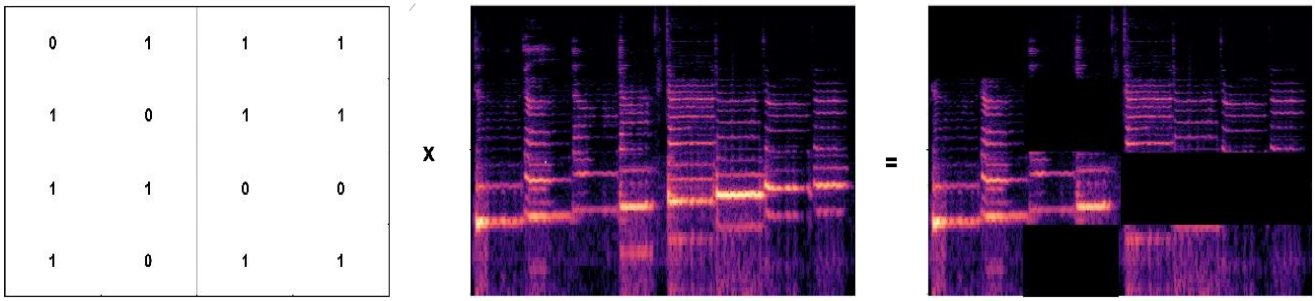


Figura 18: Aplicación de una máscara tiempo-frecuencia [6]

Las máscaras que podemos aplicar se dividen principalmente en 2 tipos:

- **Máscara binaria:** Este tipo de máscara solamente toman valores enteros. Se utiliza un valor 1 cuando se quiere mantener la sección correspondiente en la cual, la fuente a separar, tiene una presencia elevada, y se utiliza un valor 0 en caso contrario.
- **Máscara gradual:** Este tipo de máscara pueden toman valores decimales dentro del intervalo [0.0, 1.0] lo que permite obtener resultados más flexibles y con una menor distorsión.

El cálculo de las máscaras se realizará mediante una estimación de las fuentes individuales en el dominio de tiempo-frecuencia, tarea que puede llevarse a cabo mediante el uso de una red neuronal que prediga los valores óptimos de las mismas.

2.4 Medidas de evaluación objetiva

Las medidas de evaluación objetiva son métricas utilizadas para cuantificar el rendimiento del algoritmo de separación. Estas nos permiten obtener una evaluación replicable, lo que permite definir así una objetividad en las evaluaciones de la calidad y precisión de los algoritmos implementados.

Para realizar estas medidas se ha realizado una descomposición de la fuente estimada $\hat{S}_j$ en varias señales:

- S_{target}: Versión de la fuente original afectada por una distorsión considerada.
- $e_{interferencia}$: Interferencia procedente de las fuentes no deseadas.
- e_{ruido}: Ruido de la electrónica empleada para la captura de la mezcla.
- e_{artif}: Error propio del algoritmo de separación.

$$\hat{S}_j = S_{target} + e_{interferencia} + e_{ruido} + e_{artif}$$

Ecuación 6: Descomposición de la fuente estimada

A continuación, se presentan y describen algunas de las medidas ampliamente más utilizadas para la evaluación objetiva:

2.4.1 Relación Señal a Distorsión (SDR):

$$SDR = 10\ Log_{10}\ \frac{\left\|S_{target}\right\|^2}{\left\|e_{interferencia} + e_{ruido} + e_{artif}\right\|^2}$$

Ecuación 7: Cálculo de la relación señal a distorsión

2.4.2 Relación Señal a Interferencia

$$SIR = 10\ Log_{10}\ \frac{\left\|S_{target}\right\|^2}{\left\|e_{interferencia}\right\|^2}$$

Ecuación 8: Cálculo de la relación señal a interferencia

2.4.3 Relación Señal a Artefactos

$$SAR = 10Log_{10}\ \frac{\left\|e_{interferencia} + e_{ruido} + e_{artif}\right\|^2}{\left\|e_{artif}\right\|^2}$$

Ecuación 9: Cálculo de la relación señal a artefactos

3 ESTADO DEL ARTE

3.1 Separación de fuentes sonoras

La separación de fuentes sonoras ha sido un desafío en el campo del procesamiento de señales durante las últimas décadas. Este es un proceso que nos proporciona la capacidad de obtener la descomposición de las distintas fuentes que componen una mezcla de audio. Este desafío ha sido tan ampliamente estudiado durante décadas, en parte, debido a que existen diferentes enfoques con los que abordar el problema. Diferentes enfoques que se ha propuesto a lo largo del tiempo los cuales serán tratados y descritos brevemente a lo largo de este capítulo.

3.1.1 Métodos Tradicionales

Antes de la adopción de redes neuronales, se utilizaron varias técnicas estadísticas y basadas en modelos, tales como:

- **Análisis de Componentes Independientes (ICA):** Técnica que asume que las fuentes no son gaussianas y cuentan con una independencia estadística, permitiendo la separación basándose en esta independencia estadística. Un ejemplo de esta implementación lo encontramos en el “simo-model-based” [7]
- **Factorización de Matrices No Negativas (NMF):** Método que descompone un gran conjunto de datos como son, en el ámbito de la separación de fuentes, los espectrogramas, en atributos o factores representativos no negativos, reduciendo así la dimensionalidad de los datos, aunque se mantiene la misma información. Un ejemplo de esta implementación es *“Blind Separation of Positive Sources”* [8]

3.1.2 Redes Neuronales Profundas

Con la evolución de las redes neuronales, se han desarrollado nuevos enfoques que superan a los métodos tradicionales en precisión y capacidad generalizadora. Entre los más destacados se encuentran:

1. **Redes Neuronales Convolucionales (CNNs):**
   - **U-Net [9]:** Modelo originalmente utilizado para la segmentación de imágenes biomédicas desarrollado en la universidad de Friburgo. Este modelo fue adaptado para implementar una red capaz de realizar la separación de las fuentes de audio que componen una mezcla utilizando una estructura de autoencoders con capas de convolución y deconvolución.
   - **Wave-U-Net [10]:** Este modelo surge como una variante de U-Net, destacando esta versión respecto de su predecesor es el dominio de operación ya que esta versión trabaja directamente en el dominio del tiempo, remuestreando repetidamente los mapas de características para calcular y combinar las características en diferentes escalas de tiempo sobre la forma de onda de la mezcla, lo que permite evita las transformaciones espectrales además de proporcionar una representación temporal más precisa.

2. **Redes Neuronales Recurrentes (RNNs):**
   - **LSTM (Long Short-Term Memory) y GRU (Gated Recurrent Units):** Estas arquitecturas son capaces de capturar dependencias temporales a largo plazo, siendo especialmente útiles para la separación de señales en secuencias temporales, especialmente útil para aplicaciones de audio.
   - **BLSTM (Bidirectional LSTM):** Utiliza información tanto del pasado como del futuro para las estimaciones, lo que permite mejorar la predicción y separación de señales.
3. **Transformers:** Adaptaciones de modelos de transformers han sido exploradas recientemente, utilizando mecanismos de atención para capturar dependencias a largo plazo en la señal de audio, permitiendo una separación más precisa.
4. **Modelos Basados en Espectrogramas:** Muchos métodos modernos emplean redes neuronales que operan sobre representaciones de espectrograma de la señal de audio, permitiendo capturar tanto características temporales como frecuenciales de manera eficiente.

3.1.3 Técnicas Complementarias

- **Aumento de Datos (Data Augmentation):** Se aplican transformaciones a las señales de audio para aumentar la diversidad de los datos de entrenamiento, mejorando la robustez del modelo.
- **Regularización:** Técnicas como Dropout y Normalización por Lotes (Batch Normalization) son utilizadas para evitar el sobreajuste, mejorando la generalización de los modelos.
- **Métodos Híbridos:** Combinación de diferentes arquitecturas, como CNNs y RNNs, para aprovechar las ventajas de ambas en la separación de fuentes.

El uso de redes neuronales ha revolucionado el campo de la separación de fuentes sonoras, ofreciendo herramientas potentes y flexibles que superan a los métodos tradicionales. La continua investigación en arquitecturas y técnicas de aprendizaje profundo promete aún más mejoras, acercándonos cada vez más a la capacidad humana de distinguir y separar fuentes sonoras de manera efectiva.

4 SISTEMA DE AR

4.1 Fundamentos de AR

La Realidad Aumentada (AR), es una tecnología que permite al usuario vivir una experiencia inmersiva basada en el enriquecimiento del entorno real añadiendo capas de información virtual que permiten mantener la conexión con la realidad física.

4.1.1 Cronología de AR: Momentos Clave y Avances Tecnológicos

El concepto de AR tiene sus raíces en el campo de la computación gráfica y la interacción humano-computadora, con antecedentes que datan de la década de 1960.

Año	Tecnología	Descripción
1968	Ivan Sutherland - "*The Sword of Damocles*"	Ivan Sutherland, un pionero de la informática gráfica, desarrolló el primer sistema de visualización de RA con un casco HMD (Head-Mounted Display), llamado "*The Sword of Damocles*". Este fue uno de los primeros ejemplos de realidad mixta en la informática gráfica.
1974	Myron Krueger - "*Videoplace*"	Myron Krueger desarrolló el concepto de un "*ambiente artificial*" interactivo llamado Videoplace, donde los usuarios podían interactuar con objetos virtuales en un espacio digital.
1990	Tom Caudell y David Mizell - Término "*Realidad Aumentada*"	Tom Caudell y David Mizell acuñaron el término "*Realidad Aumentada*" mientras trabajaban en Boeing. Utilizaron RA para guiar a los trabajadores de fábrica en el ensamblaje de cables.
1992	Primera aplicación de RA en la industria	Louis Rosenberg desarrolló el sistema virtual llamado "*Virtual Fixtures*" en el Armstrong Labs de la Fuerza Aérea de EE. UU. Este sistema permitió a los usuarios manipular objetos físicos en un entorno aumentado.
1998	Primera transmisión de deportes con RA (NFL)	La NFL introdujo la primera línea amarilla superpuesta para señalar el primer Down durante los juegos televisados, utilizando tecnología de RA.
2000	Bruce Thomas - Sistema RA móvil (ARQuake)	Bruce Thomas desarrolló el primer sistema de RA móvil en un juego llamado ARQuake. Los jugadores interactuaban con entornos físicos y objetos virtuales.

2009	Aparición de aplicaciones de AR en smartphones	Aparecieron aplicaciones móviles como Wikitude y Layar, que utilizaron las cámaras de los teléfonos inteligentes para superponer información digital sobre el mundo real.
2012	Google Glasses	Google presentó Google Glasses, unas gafas de RA que permitían superponer información digital en el campo de visión del usuario, aunque el dispositivo fue controvertido y tuvo problemas de adopción masiva.
2016	Niantic - Pokémon GO	Pokémon GO popularizó el uso de la RA entre el público general. El juego usaba la cámara del teléfono móvil para permitir que los jugadores capturaran Pokémon en el mundo real.
2019	Microsoft - HoloLens 2	Microsoft lanzó el HoloLens 2, una versión mejorada de sus gafas de RA que permitía a los usuarios interactuar con hologramas en 3D de manera más intuitiva, mejorando las aplicaciones industriales y médicas.
2021	IKEA Place	Ikea presenta una aplicación que te permite probar los muebles del catálogo de IKEA directamente en tu salón, sin necesidad de haberlos comprado, transportado hasta tu casa y montado, con sólo apuntar la cámara de tu smartphone y pulsar un botón
2022	Google Live View	Google presenta Live View, una nueva manera de obtener indicaciones a la hora de seguir indicaciones en un trayecto definido en Google Maps en el mundo real.
2024	Apple Vision Pro	Apple lanza las Apple Vision Pro, un nuevo dispositivo que ofrece un revolucionario sistema basado en una interfaz espacial muy intuitiva y un sistema de entrada que permite moverse por los contenidos usando los ojos, las manos y la voz

Tabla 1: Cronograma con algunos de los momentos más importantes de la tecnología de AR

4.1.2 Evolución Tecnológica

La evolución que han sufrido los dispositivos móviles en los últimos años, la combinación de varias cámaras en un mismo dispositivo, la inclusión de sensores más precisos, procesadores potentes y pantallas de una mayor calidad han conseguido que la tecnología AR haya tenido una mayor inclusión en aplicaciones del público general.

Hoy en día cada vez son más las aplicaciones que incorporan esta tecnología en alguna de sus funcionalidades. Sin embargo, la aplicación más importante y la que introdujo esta tecnología al público general fue PokémonGO. Esta aplicación se popularizó por la integración que realizaba de los elementos virtuales, los Pokémons, en entornos reales gracias al uso de la cámara de los teléfonos móviles.

Aunque esta tecnología no ha sido una de las principales ramas de investigación y desarrollo son muchas las empresas tecnológicas que llevan años desarrollando software y dispositivos especializados y enfocados en mejorar las experiencias de VR y AR, como son las Microsoft HoloLens o las Apple Vision Pro, que ofrecen unas experiencias mucho más inmersiva que la que pueden proporcionar los teléfonos móviles al emplear visores encargados de proyectar las imágenes directamente en el campo de visión del usuario.

4.1.3 Fundamentos Técnicos de AR

Los sistemas de AR se construyen a partir de una simbiosis entre el hardware del dispositivo y el software de gestión del mismo.

El hardware utilizado para las implementaciones son sensores de cámaras, acelerómetros y giroscopios, así como pantallas, procesadores y otros dispositivos de entrada que permiten la obtención de información del entorno.

Información que el software deberá procesar mediante algoritmos de visión por ordenador y técnicas de procesamiento de imágenes con el fin de identificar características clave del entorno real, como pueden ser superficies, objetos o marcadores que serán utilizados como elementos de referencia para colocar objetos virtuales en el espacio.

Las librerías "*ARKit*" de Apple o "*ARCore*" de Google permiten a los desarrolladores crear aplicaciones de AR de una forma sencilla al emplear estas librerías ya construidas en sus desarrollos.

4.1.4 Aplicaciones y Futuro de la AR

La tecnología de AR tiene un gran potencial en la sociedad actual por su gran adaptabilidad y versatilidad. Ya que esta tecnología puede ser aplicada a diversos campos desde el entretenimiento o el marketing hasta la educación y la medicina.

En la industria del entretenimiento, el uso de AR ha supuesto una revolución en los videojuegos y aplicaciones interactivas, permitiendo una participación del usuario más directa e inmersiva.

Por otro lado, en la educación los estudiantes pueden interactuar con modelos 3D de estructuras complejas, como órganos, piezas mecánicas o edificios históricos, directamente en sus aulas y hogares.

A medida que los dispositivos se vuelvan más ligeros, potentes y asequibles, y los algoritmos de reconocimiento espacial continúen mejorando veremos como la AR se convierte en una herramienta cada vez más común en diversas áreas de la vida diaria, incluyendo la comunicación o el trabajo.

Esta tecnología tiene un futuro prometedor ya que será una tecnología clave en la creación de experiencias inmersivas que combinen lo digital con lo físico permitiendo así vivir nuestra realidad y entorno de una forma vitaminada.

4.2 Posibles tecnologías para la implementación

En este apartado se comentan varias de las tecnologías existentes actualmente para la implementación de un sistema de AR.

4.2.1 Unity con AR Foundation:

Unity es una de las plataformas más utilizadas, con casi 1500 millones de creadores activos y apareciendo en más del 50% del catálogo de videojuegos para dispositivos móviles. Además, Unity se utiliza en el 90% aplicaciones relacionadas con la realidad aumentada y virtual de hoy en día.

Es una plataforma que ofrece una gran versatilidad y facilidad de uso ya que con el módulo "*AR Foundation*", Unity permite a los desarrolladores construir experiencias de AR y exportarlas de forma sencilla a dispositivos iOS y Android al aprovechar las capacidades de las librerías de ARKit (iOS) y ARCore (Android). Unity además ofrece potentes herramientas gráficas y un entorno amigable con una interfaz Drag-and-Drop muy útil tanto para desarrolladores expertos como principiantes.

4.2.2 AR Core

ARCore es el kit de desarrollo (SDK) de realidad aumentada de Google que ofrece APIs multiplataforma para crear nuevas experiencias envolventes en Android, iOS, Unity y la Web. Facilita el rastreo de movimiento, la comprensión ambiental y la estimación de la luz, lo que permite a los desarrolladores anclar objetos virtuales en el mundo real.

4.2.3 ARKit

ARKit es el marco de desarrollo de Apple para la realidad aumentada en dispositivos iOS. Ofrece herramientas avanzadas como detección de superficies, integración de objetos 3D y seguimiento facial. ARKit está diseñado para crear experiencias AR avanzadas y está profundamente integrado con los dispositivos Apple.

4.2.4 Vuforia

Vuforia es una de las plataformas más antiguas y sólidas para la AR, y se destaca por su capacidad de rastrear imágenes, objetos 3D y modelos. Vuforia ofrece soporte tanto para aplicaciones industriales como comerciales, y es ampliamente utilizado para experiencias AR interactivas en publicidad y educación.

4.2.5 Microsoft Mixed Reality Toolkit 3 (MRTK)

MRTK3 es la tercera generación de Microsoft Mixed Reality Toolkit para Unity. Es un proyecto de código abierto controlado por Microsoft basado en Unity XR Interaction Toolkit y Unity Input System.

MRTK3 proporciona un sistema de entrada multiplataforma, así como los bloques de creación para las interacciones espaciales y la interfaz de usuario necesaria para llevar a cabo una implementación de AR. Además, permite la creación rápida de prototipos gracias a la simulación en el editor lo que consigue visualizar los cambios realizados en tiempo real.

Sin embargo, la compatibilidad de esta plataforma es más limitada ya que está más dirigida a dispositivos de realidad mixta como son las HoloLens2, Meta Quest, SteamVR, Rifts Rift en OpenXR, Lenovo ThinkReality A3.

4.2.6 Tabla Comparativa de Tecnologías de AR

Tecnología	Compatibilidad	Lenguaje Utilizado	Principales Características
Unity AR	Android, iOS, HoloLens2, Magic Leap, Meta Quest…	C#	Amplio soporte multiplataforma, fácil integración con ARKit/ARCore
ARCore	Android	Java, C++, C#	Preciso rastreo de movimiento y comprensión ambiental
ARKit	iOS	Swift, Objective-C	Integración profunda con el hardware de Apple
Vuforia	Android, iOS, UWP	C#, Java, C++	Fuerte soporte para el reconocimiento de imágenes y objetos
Microsoft Mixed Reality Toolkit	HoloLens2, Meta Quest, SteamVR, Rifts Rift (OpenXR), Lenovo ThinkReality A3	C#	Herramientas para realidad mixta, soporte para HoloLens

4.3 Esquema de funcionamiento de la aplicación AR

El comportamiento de la aplicación desarrollada puede dividirse en dos secciones, una primera sección destinada a la gestión de las escenas y una segunda destinada a la gestión de los marcadores detectados.

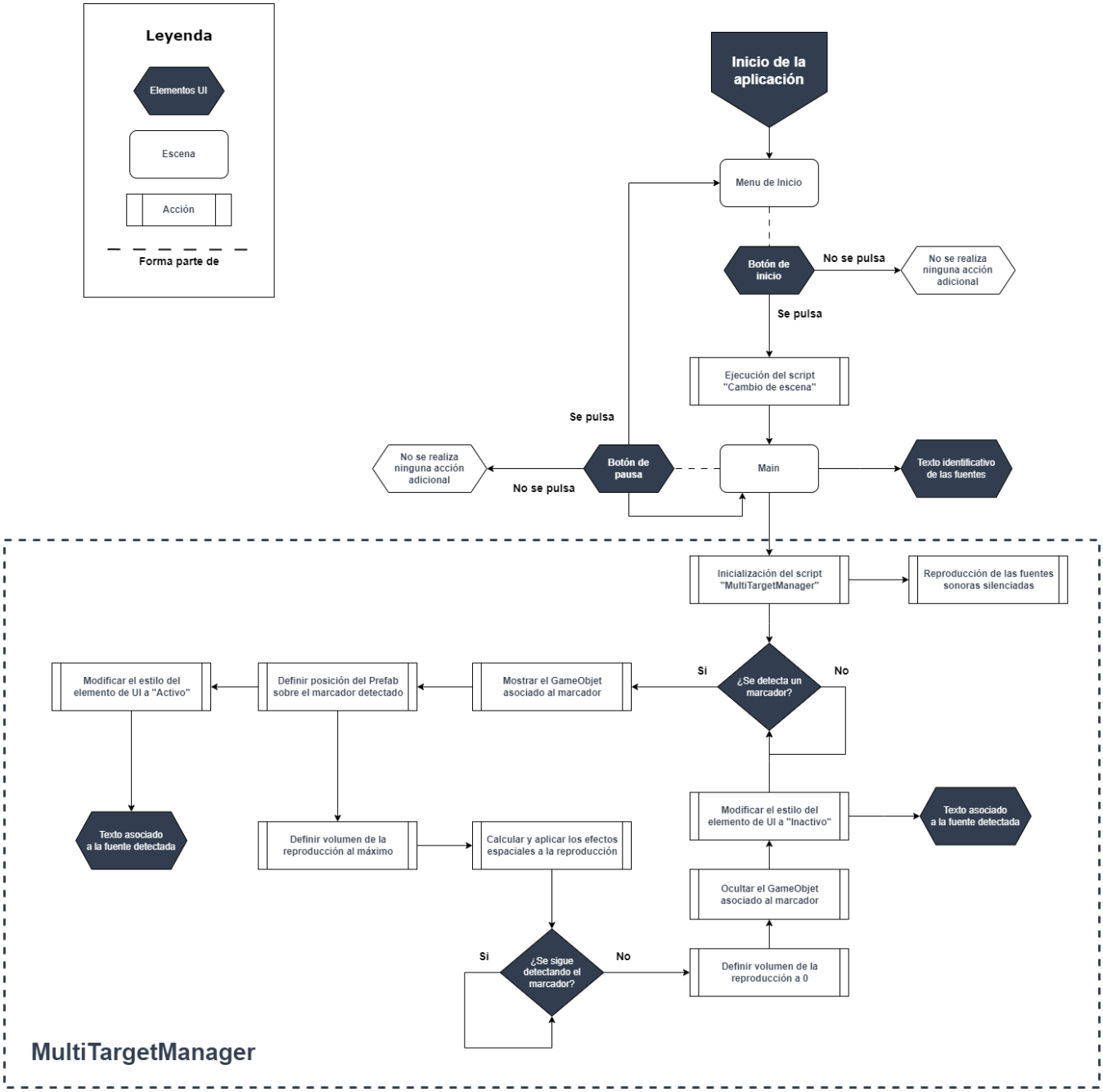


Figura 19: Esquema lógico global del funcionamiento de la aplicación

4.3.1 Cambio de escena

La gestión del cambio de escena se realiza mediante el script "*CambioDeEscena*", el cual sigue la ejecución indicada en el esquema.

Con el inicio de la aplicación se muestra la primera escena "*Menú de Inicio*", la aplicación permanecerá en esta escena hasta que sea pulsado el botón de inicio, momento en el cual se ejecutará el script "*CambioDeEscena*" y se mostrará la escena "*Escena principal*".

Al cargarse esta escena se inicia el script "*MultiTargetManager*" e inicia su comportamiento, en el que se destaca el inicio de la reproducción de las fuentes de forma silenciada.

Esta escena se mantendrá activa hasta que el usuario pulse el botón de pausa, momento en el que se vuelve a llamar al script "*CambioDeEscena*" y se muestra de nuevo el menú de inicio, deteniendo en este cambio de escena la reproducción de las fuentes.

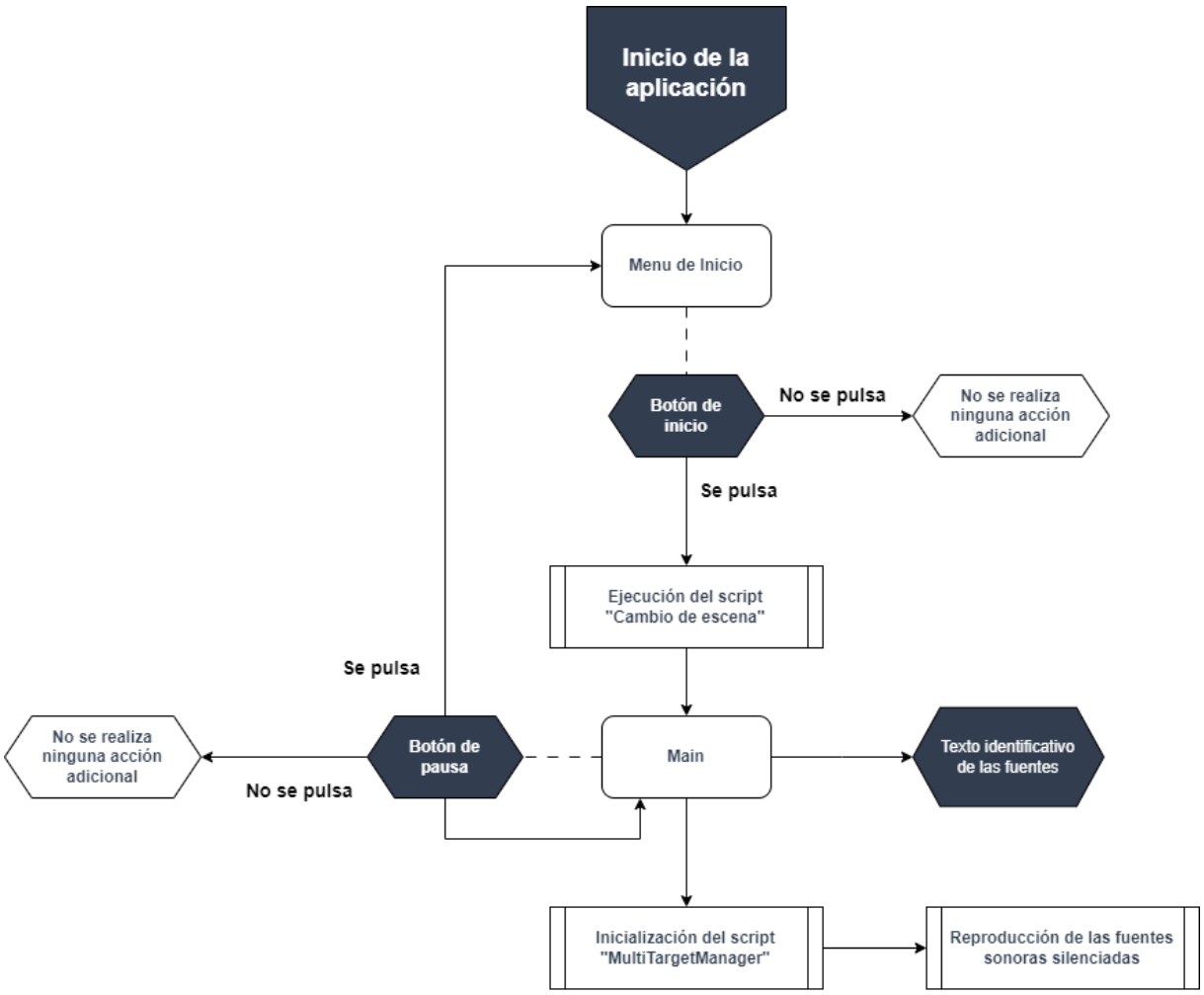


Figura 20: Esquema lógico del comportamiento de la aplicación al realizar un cambio de escena

4.3.2 Gestión del comportamiento de los marcadores

La gestión de los marcadores se realiza siguiendo el esquema mostrado de manera que, cuando un marcador es detectado se realizarán todas las acciones definidas para su activación como son:

1. Mostrar en la escena el GameObject asociado a este marcador.
2. Modificar sus coordenadas espaciales para que coincidan con las coordenadas del marcador detectado.
3. Modificar el elemento de texto asociado a la fuente con la que se corresponde el marcador detectado, asignando el estilo "*Activo*".
4. Modificar el volumen de la reproducción, la cual fue iniciada al cargarse la escena en silencio, para asignar el 100%
5. Aplicar los efectos de espacialidad configurados para la fuente sonora los cuales serán aplicados en función de la posición del oyente respecto del GameObject.

Una vez aplicados todas estas modificaciones se mantendrá una monitorización del estado del marcador de manera que, en el momento en el que deje de ser detectado se realizarán las acciones para su desactivación las cuales son:

1. Modificar el volumen de la reproducción al 0%.
2. Ocultar de la escena el GameObject asociado al marcador que ha dejado de detectarse.
3. Modificar el elemento de texto asociado a la fuente con la que se corresponde el marcador que ha dejado de ser detectado, asignando el estilo "*Inactivo*".

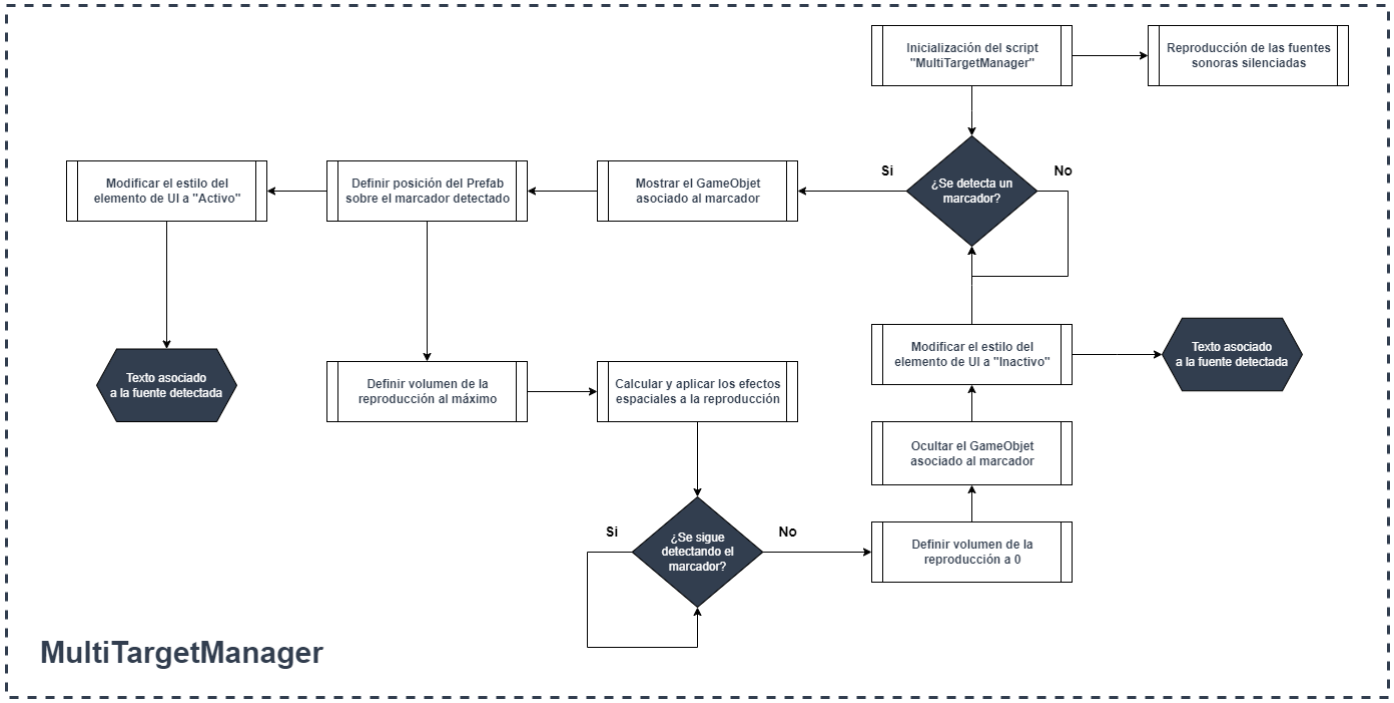


Figura 21: Esquema lógico del comportamiento de la aplicación en la gestión de los marcadores

4.4 Sistema implementado

Para la producción de un entorno interactivo en el que poder llevar a cabo la combinación o remix de las fuentes sonoras obtenidas de la red neuronal se ha implementado un entorno interactivo mediante un sistema de realidad aumentada (AR). El desarrollo de este entorno se realizado mediante la herramienta Unity AR Foundation.

El entorno interactivo ha sido construido como una aplicación para dispositivos móviles, la cual permite, en el entorno físico que rodea al usuario de manera visual e interactiva, realizar el remixing de las distintas fuentes sonoras.

Esta aplicación hace uso de varios "*targets*" o marcadores que sirven como referencias físicas para la misma. Cada uno de estos marcadores cuenta con un prefab asociado, donde cada uno de estos ha sido asociado a una fuente sonora específica.

Esta configuración consigue que, al ser detectado el marcador en el entorno físico, este sea identificado y sea mostrado, en el entorno virtual, el prefab correspondiente a dicho marcador, elemento encargado de actuar como fuente sonora que reproduce la pista correspondiente al marcador detectado.

4.4.1 Unity y Unity AR Foundation

Para entender la explicación y el desarrollo de la aplicación realizado es importante conocer algunos conceptos básicos de Unity [11] como son los conceptos de "*Escena*" o "*GameObject*".

Una Escena en Unity es un entorno o nivel en el que se desarrolla parte de la aplicación. Es un espacio donde se colocan los diferentes objetos a emplear, luces, cámaras y otros elementos que forman parte de la aplicación de AR. Cada escena puede contener diferentes elementos lo que permite utilizar cada escena para una situación específica como puede ser un menú principal o una sección propia de la aplicación.

Por otro lado, uno de los conceptos más importantes es el de GameObject ya que es el componente más básico de Unity. Un GameObject no es más que un objeto en la escena que puede ser definido para representar cualquier cosa, puede representar una luz, una fuente sonora, una cámara o un simple marcador. Un GameObject no cuenta con la capacidad de realizar ninguna acción por sí mismos, sin embargo, se les pueden agregar diferentes componentes como pueden ser scripts, físicas, elementos gráficos, fuentes de audio, efectos visuales, etc. que permitan incorporar al GameObject un comportamiento o apariencia más complejo.

Podría decirse que un GameObject no es más que un contenedor vacío al que pueden agregarse funcionalidades para que este cumpla un propósito dentro de la escena.

4.4.2 Targets o marcadores

Los "*targets*" o marcadores son los elementos gráficos que son utilizados como referencias en el plano físico por la aplicación para su correcto funcionamiento.

4.4.2.1 Niveles de detección

Es importante tener en cuenta que no cualquier cosa puede realizar la función de marcador ya que para que un marcador sea correctamente detectado y reconocido debe cumplir con un número mínimo de puntos de interés, en inglés "*keypoints*" [12].

Los keypoints, como su propio nombre indica, son puntos dentro del marcador que tienen cierta importancia visual para los algoritmos de visión máquina. Estos keypoints pueden ser calculados mediante diferentes algoritmos implementados por las librerías de AR utilizadas, como son ARCore o ARKit entre otras.

Estos algoritmos son algoritmos privados lo que produce que no exista una definición exacta sobre qué parámetros o requisitos se deben cumplir en una imagen para producir un keypoint en la misma. Uno de los métodos usados para la obtención de estos keypoints podría ser un algoritmo de detección de bordes en el que se tenga en cuenta el contraste entre píxeles vecinos en una imagen, sin embargo, esto es solo una hipótesis.

Este desconocimiento produjo cierta ambigüedad y desconocimiento a la hora de diseñar o seleccionar las imágenes que se usarían como marcador ya que, para confirmar si este es válido al contar con un nivel suficiente de keypoints como para ser detectada correctamente, debe comprobarse empíricamente con las herramientas que proporcionan las librerías.

La herramienta que se usó para la evaluación de los marcadores fue la proporcionada por Google, relativa a su ARCore denominada arcoreimg.exe [13]. Esta herramienta puede ser descargada desde su repositorio oficial de GitHub [14].

```
G:\Mi unidad\TFG Drive\Targets>arcoreimg.exe eval-img --input_image_path="Icono Bajo.jpg"
50

G:\Mi unidad\TFG Drive\Targets>arcoreimg.exe eval-img --input_image_path=Bajo.jpg
100
```

Figura 22: Ejemplo del comportamiento de la herramienta de evaluación ARCoreimg.exe

4.4.2.2 Primera selección de marcadores

Durante la etapa más temprana del desarrollo de la aplicación para la realización de diversas pruebas y experimentos se utilizaron códigos QR como marcadores. Estos códigos proporcionaban un nivel de detección bastante elevado, aunque, en varias situaciones como pueden ser situaciones de poca iluminación o ángulos extremos la aplicación no terminada de detectarlos correctamente. Sin embargo, aunque el nivel de detección era elevado, el nivel de identificación no lo era y el cambio de un marcador a otro

lo llegaba a producir que se detectasen falsamente los dos marcadores simultáneamente cuando realmente solo se encontraba uno en escena.

Tras estas pruebas se decidió optar por unos marcadores que fuesen más diferentes entre sí, además de seleccionar unos marcadores que fuesen fácilmente identificables permitiendo así diferenciarlos visualmente unos de otros con el fin de identificar cada una de las fuentes asociadas a los mismos.

Tomando como base la naturaleza y el tema de este TFG se optó por buscar unos targets que permitiesen referenciar el concepto de sonido, fuentes instrumentales y música.

De manera que, al obtener de la separación implementada mediante redes neuronales 4 fuentes: "*vocals*", "*bass*", "*drums*" y "*others*"; se intentó elegir unos marcadores que referenciaran cada uno de estos grupos, sin embargo, todas las imágenes encontradas que eran visualmente identificables y que hacían referencia al tipo de fuente asociada, no cumplían con los niveles necesarios para la correcta detección.

Al no conseguir encontrar ningún conjunto de marcadores que cumpliesen con los niveles de detección necesarios, mantuviesen una concordancia visual y cumpliesen los requisitos anteriormente mencionados, se optó por diseñarlos de manera autónoma.

4.4.2.3 Decisiones de diseño de los marcadores

Al crear de forma autónoma los marcadores estos podrán ser definidos de manera que cumplan con los requisitos impuestos para la correcta detección e identificación de estos. Además, al haber sido diseñados especialmente para esta aplicación y ser una creación propia se podrán tomar todas las libertades creativas deseadas, permitiendo, entre otras cosas, mantener una concordancia visual además de ser visualmente diferenciables unos de otros.

Para el diseño de los marcadores finalmente se optó por definir una forma geométrica que incluya un patrón y sobre el que se colocará cada símbolo identificativo de cada una de las fuentes.

Los patrones diseñados son patrones monocromáticos que emplean valores absolutos de blanco (100% de luminosidad) y negro (0% de luminosidad). Esta decisión permite conseguir generar el mayor contraste posible dentro de los marcadores. Además, la selección de un patrón monocromático consigue que, aunque los patrones sean diferentes para cada uno de los marcadores, se mantenga una armonía visual entre todos ellos al mismo tiempo que se consigue esa diferenciación deseada.

4.4.2.4 Proceso de diseño de los marcadores

En un primer intento de generar los marcadores se optó por la utilización de formas circulares con fondo negro y relleno blanco como podemos apreciar en la Figura 23.

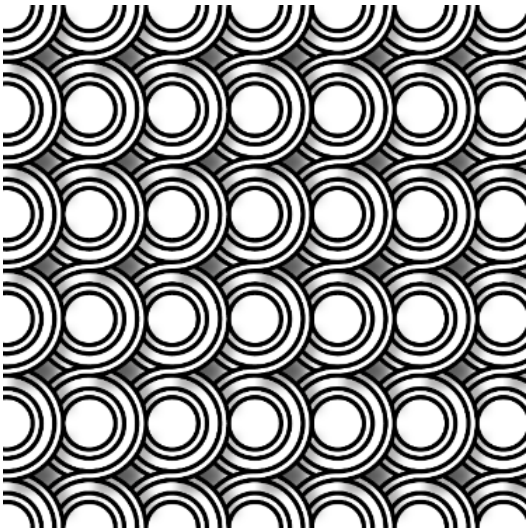


Figura 23: Patrón original usado como fondo

Tomando este patrón como base se diseñaron 4 marcadores con formas diversas, sobre estas formas se aplicaron y realizaron una serie de modificaciones y transformaciones con el fin de obtener 4 patrones más diferenciados entre sí, obteniendo así la primera versión de los marcadores.

Figura 24: Marcador A V 0.5

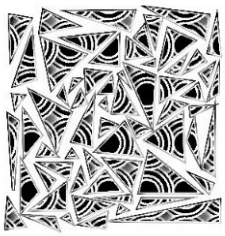


Figura 25: Marcador B V 0.5

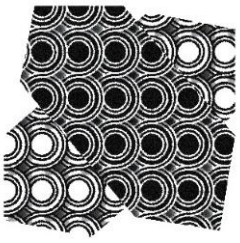


Figura 26: Marcador C V 0.5

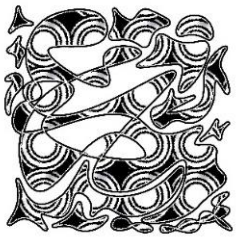


Figura 27: Marcador D V 0.5

Sin embargo, estas primeras versiones no conseguían los niveles de detección e identificación deseados. Es por esto que se volvió al concepto inicial de patrón negro sobre fondo blanco inspirado en los códigos QR utilizados al inicio, generando una nueva versión con un patrón común en los marcadores y añadiendo los símbolos de cada una de las fuentes.

Figura 28: Marcador A V 1

Figura 29: Marcador c V 1

Figura 30: Marcador b V 1

Figura 31: Marcador d V 1

Aunque esta nueva versión mejoró notablemente los niveles de detección, el nivel de identificación seguía sin ser el deseado ya que, aunque contasen con los símbolos al emplear todos los marcadores el mismo patrón de fondo provocaba falsos positivos en la detección. Es por esto que se decidió diseñar una nueva versión con patrones más variados y únicos para cada uno de los marcadores. Además, en esta nueva versión se añadió el nombre de la fuente asociada a cada una de los marcadores.

Figura 32: Fuente A Micrófono

Figura 33: Fuente B Batería

Figura 34: Fuente C Bajo

Figura 35: Fuente D Micrófono/Voz

Con estas nuevas versiones se conseguían detecciones e identificaciones mucho más precisas, consolidando así los marcadores que serán utilizados para la aplicación AR.

```
G:\Mi unidad\TFG Drive\Targets>arcoreimg.exe eval-img --input_image_path=Altavoz.jpg
100

G:\Mi unidad\TFG Drive\Targets>arcoreimg.exe eval-img --input_image_path=Bajo.jpg
100

G:\Mi unidad\TFG Drive\Targets>arcoreimg.exe eval-img --input_image_path=Bateria.jpg
100

G:\Mi unidad\TFG Drive\Targets>arcoreimg.exe eval-img --input_image_path=Microfono.jpg
100
```

Figura 36: Evaluación de las versiones finales de los marcadores mediante ARCore

4.4.1 Prefabs en Unity

Un Prefab es una plantilla o conjunto de objetos que han sido guardados como una unidad reutilizable. Esta plantilla puede contener GameObjects con propiedades específicas, incluyendo componentes, configuraciones y comportamientos personalizados.

4.4.1.1 Selección de Prefabs

De manera similar a lo que ocurrió con los marcadores no se encontró ningún conjunto de prefabs que cumpliesen con las expectativas deseadas por lo que se optó por diseñarlos. Para realizar el diseño se hizo uso de un software gratuito denominado Onshape.

Para los modelos 3D se tomaron como referencia los iconos empleados para los marcadores, de manera que los modelos diseñados, que serían incorporados y convertidos en prefabs dentro de Unity, mantuvieran una concordancia visual con los marcadores.

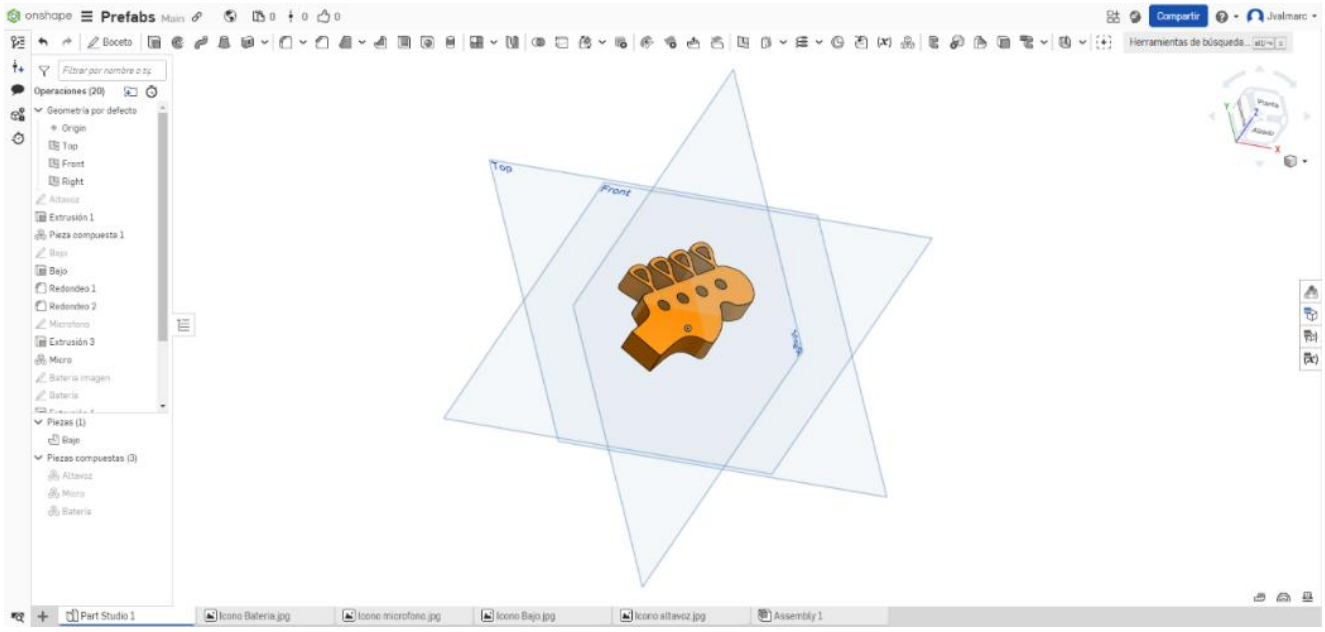


Figura 37: Herramienta de diseño en 3D usada para el diseño de los prefabs

4.4.1.2 Componente Audio Source

Como se ha mencionado anteriormente, un prefab es un objeto de Unity que permite almacenar propiedades y componentes.

El componente principal de los prefabs empleados en este trabajo es el componente AudioSource.

Este componente cuenta con una variedad de parámetros considerable que permiten una gran funcionabilidad y personalización.

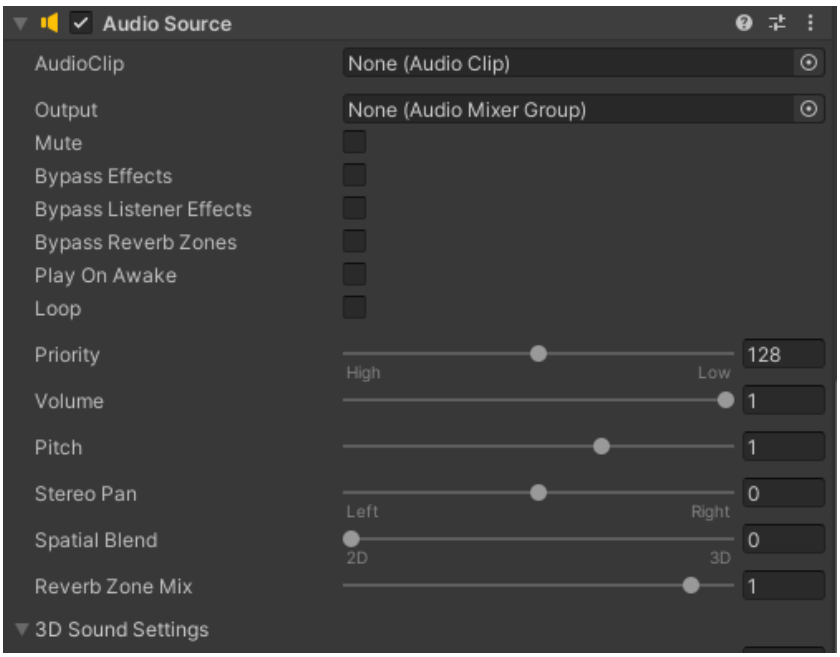


Figura 38: Parámetros del componente Audio Source

A continuación, se describen algunos de los parámetros más interesantes además que aquellos que se hayan modificado para este trabajo, como son:

- **AudioClip:** Referencia al archivo del clip de sonido que será reproducido.
- **Output:** De forma predeterminada, el clip se envía directamente al componente AudioListener, sin embargo, es posible definir en este campo un mezclador de audio que aplique efectos al clip de audio.
- **Mute:** Si está habilitado, el sonido se reproducirá, pero silenciado.
- **Play On Awake:** Si está habilitado, el sonido se reproducirá en el momento en que se inicie la escena. Mientras que, si está deshabilitado, la reproducción debe ser iniciada desde un script.
- **Loop:** Permite que el clip de audio se repita cuando llega al final.
- **3D Sound Settings:** Configuraciones específicas destinada a producir una Mezcla espacial.
    - **Doppler Level:** Determina cuánto efecto Doppler se aplicará a esta fuente de audio (si se establece en 0, no se aplica ningún efecto).
    - **Spread:** Establece el ángulo de propagación en sonido estéreo 3D o multicanal en el espacio del altavoz.
    - **Volume RollOff:** Indica la velocidad con la que se desvanece el sonido conforme nos alejamos del componente. Cuanto mayor sea el valor, más cerca debe estar el oyente para oír el sonido. Este parámetro se define mediante un gráfico. Integrado en Unity contamos con dos configuraciones definidas además de con la opción de definir una configuración propia.

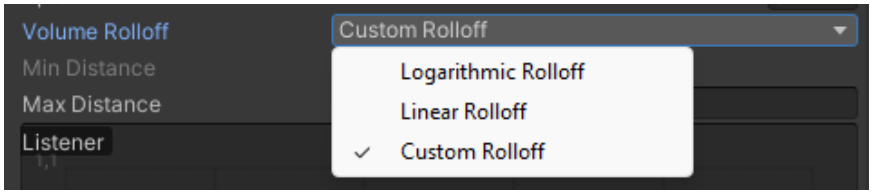


Figura 39: Configuraciones predefinidas de RollOff

    - **Min Distance:** Distancia que asegura que el volumen del sonido se mantendrá lo más alto posible. Una vez que se supera esta distancia el sonido comenzará a atenuarse.
    - **Max Distance:** Distancia a partir de la cual, el sonido deja de atenuarse. Más allá de este punto no se aplicará más atenuación a la fuente sonora, por lo que se mantendrá el valor de atenuación para la distancia máxima.

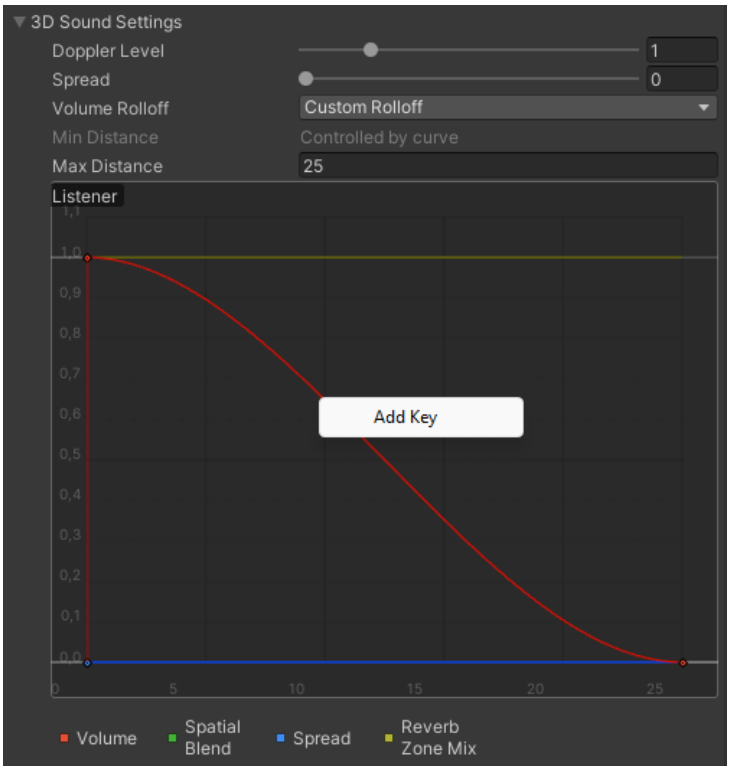


Figura 40: Gráfico de RollOff del componente Audio Source

4.4.2 Detección de los marcadores

La detección y gestión de los marcadores se ha implementado haciendo uso de las librerías de UnityAR Foundation.

4.4.2.1 Componente "AR Tracked Image Manager"

La detección de los marcadores se implementa mediante el componente "*AR Tracked Image Manager*", componente de Unity encargado de detectar las imágenes definidas dentro de una librería de imágenes de referencia ("*Reference Image Library*") y mostrar un prefab sobre el marcador detectado.

Sin embargo, este componente cuenta con una limitación a la hora de presentar los prefabs. Ya que, aunque permite definir varios marcadores de referencia dentro de la librería, solamente permite definir un único prefab a mostrar para todos los marcadores. Esta limitación supone un problema para la implementación deseada debido a que buscamos poder mostrar 4 prefabs diferentes, uno para cada uno de las marcadores y fuentes que hemos obtenido del audio original.

Para solventar esta limitación se decidió desarrollar un script que permite, haciendo uso del componente de Unity, detectar varios marcadores de una librería y mostrar, para cada uno de ellos, un prefab específico.

4.4.3 Librería de marcadores (Target Library)

Las imágenes que serán utilizadas como marcadores se ha definido dentro de una "*XR Reference Image Library*". En esta se han añadido cada una de las imágenes que serán usadas como marcadores, designando y definiendo para cada una de ellas un nombre identificativo: Altavoz, Bajo, Batería y Micrófono.

La nomenclatura empleada para las imágenes que forman la librería es importante por cómo está implementada la asignación de los prefabs respecto de los marcadores.

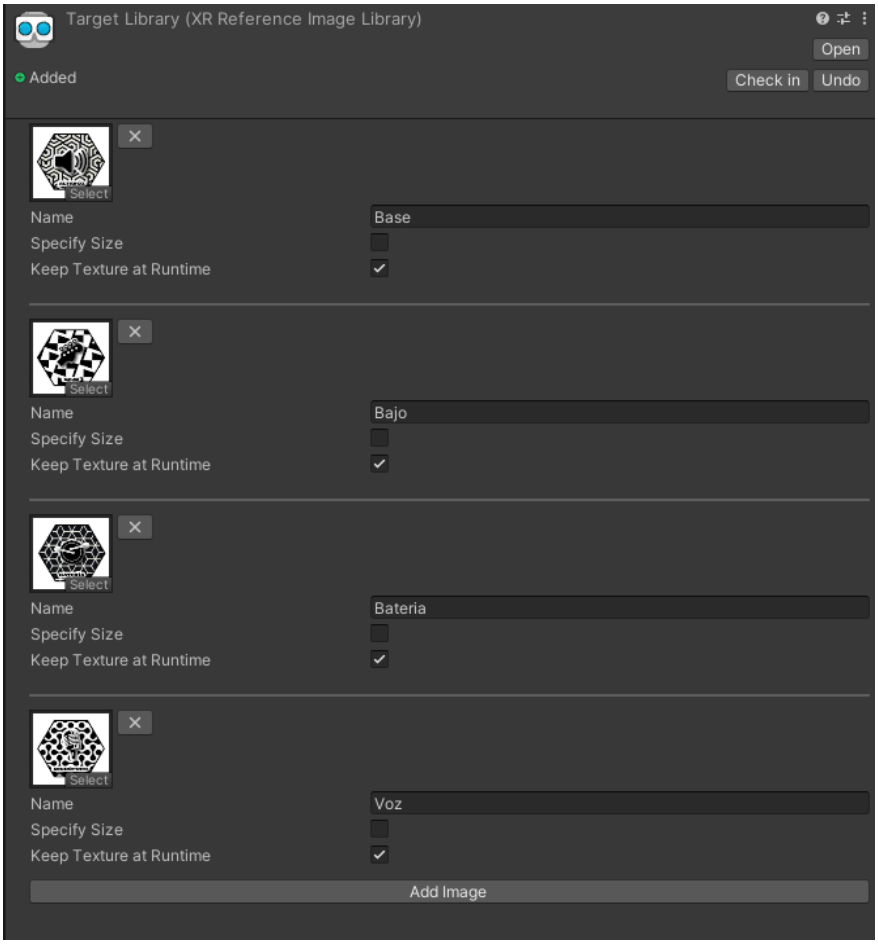


Figura 41: Configuración de la librería de marcadores en Unity

4.4.4 Escenas de la aplicación

La aplicación que ha sido desarrollada implementa su funcionamiento haciendo uso de 2 escenas de Unity: "*Menú de inicio*" y "*Escena principal*" las cuales serán explicadas a continuación.

4.4.4.1 Menú de inicio

La primera escena denominada "Menú de inicio" contiene únicamente los elementos visuales que son mostrados al iniciar la aplicación entre los que se encuentran el nombre de la aplicación, el nombre del alumno y un botón de inicio como se puede apreciar en la Figura 42.

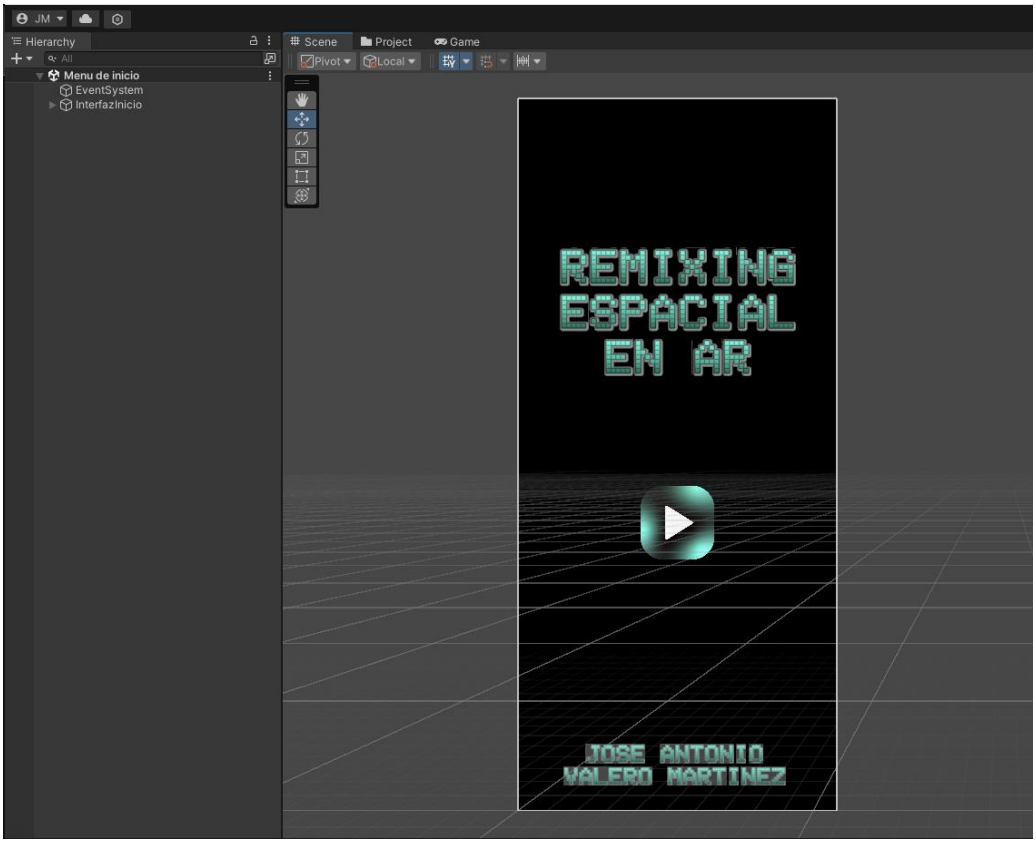


Figura 42: Escena "Menú de Inicio"

El botón de inicio cuenta con un script asociado diseñado para realizar el cambio entre escenas, de manera que, al pulsarse este botón se cambia de esta primera escena a la "Escena principal", la cual ha sido referenciada en el evento "*On Click*" del botón como se puede apreciar en la Figura 43.

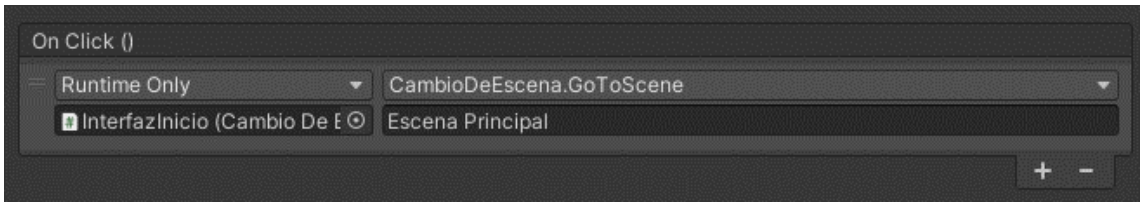


Figura 43: Asociación del Script "Cambio de Escena" al botón de inicio

4.4.4.2 Escena principal

En esta segunda escena se pueden identificar más elementos visuales en la GUI. Entre los que destaca el Script encargado de la gestión del comportamiento de los prefabs, el "*Multi Target Manager Script*". Script que se encuentra asociado al componente *"XR Origin"* de la escena, componente perteneciente al paquete de Unity AR Foundation.

Como se puede apreciar en la Figura 44, este script cuenta con varios objetos referenciados como son el elemento "*AR Tracked Image Manager*", el diccionario de imágenes conformado por los marcadores, el vector con los prefabs de los modelos, así como los elementos de texto de las fuentes que se encuentran en la GUI.

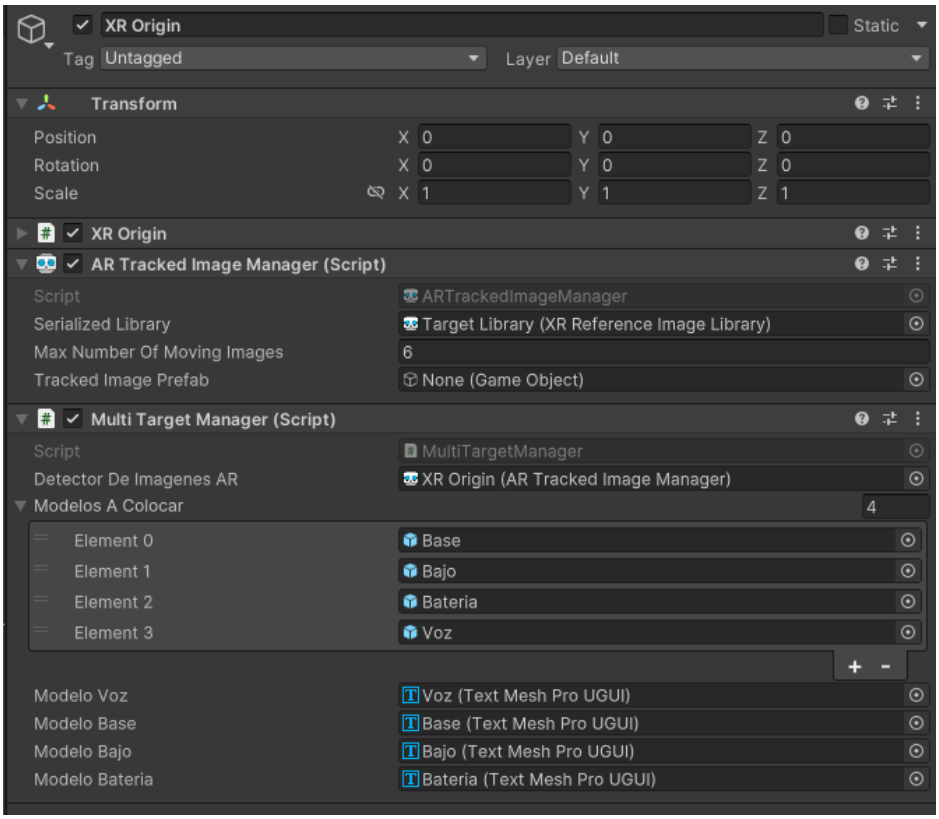


Figura 44: Componente XR Origin

En esta escena, como se pueden apreciar en la Figura 45, se pueden identificar 6 elementos principales, 4 elementos de texto en la parte superior y un elemento de texto y un botón de pausa en la parte inferior.

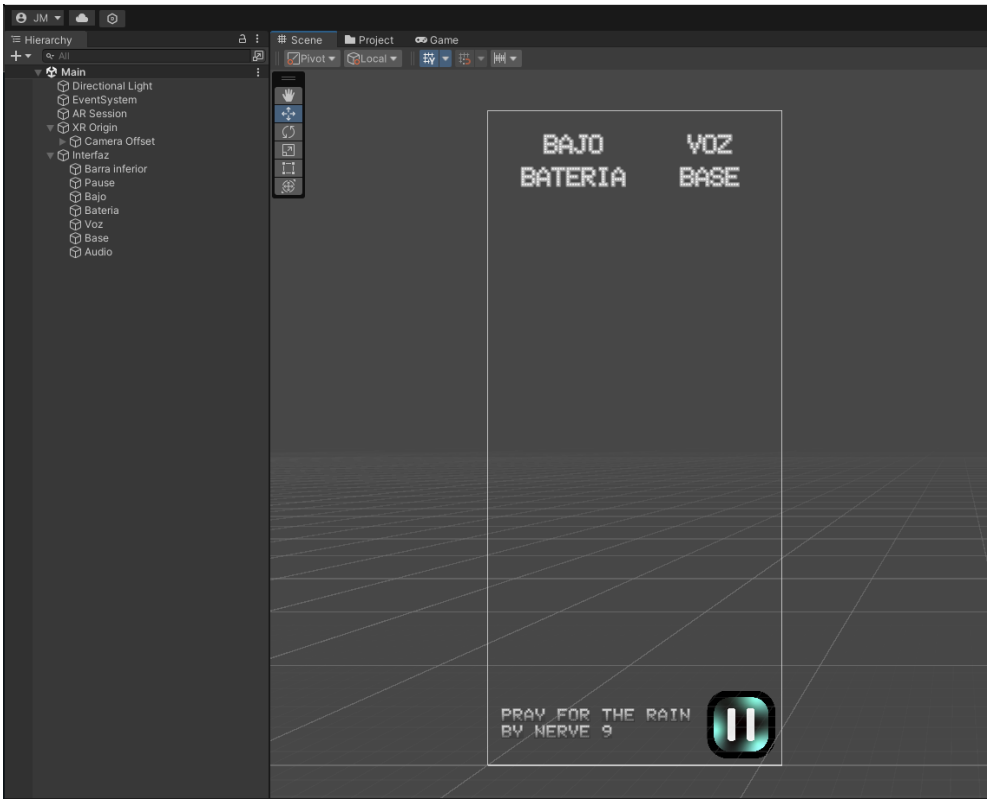


Figura 45: Escena principal

Como se ha mencionado anteriormente, en la parte superior de la escena se encuentran los "*Rótulos de fuentes*", cuatro elementos de texto que han sido diseñados y configurados para proporcionar, de una manera visual e intuitiva, información sobre el estado de las fuentes, de manera que se pueda conocer cuales están siendo reproducidas.

Figura 46: "Rótulos de fuentes"

En la parte inferior de la escena se encuentra el "*Rótulo de canción*", un cuadro de texto que contiene el nombre de la canción que se usará para realizar la especialización de las fuentes separadas.

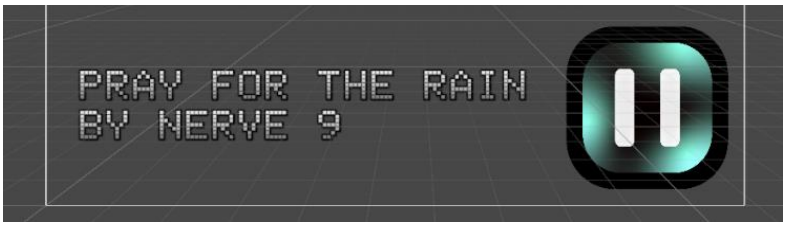


Figura 47: Texto identificativo de la canción usada y botón de pausa

Además de este rótulo se encuentra el botón de pausa, botón que, al igual que se ha configurado para el botón de inicio, cuenta con el script "CambioDeEscena" asociado el cual se encarga de realizar el cambio de escena como se puede apreciar en la Figura 48.

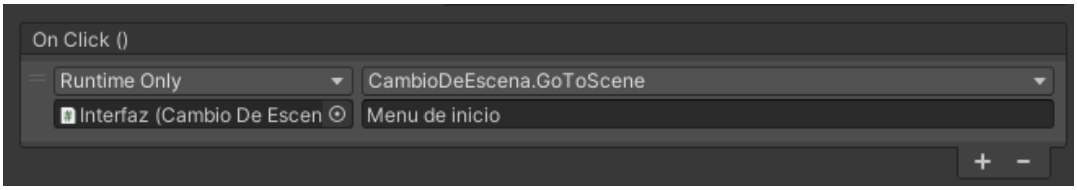


Figura 48: Asociación del Script "Cambio de Escena" al botón de pausa

4.4.5 Script Cambio De Escena

Como se ha mencionado anteriormente, este script se encarga de realizar un cambio de escena, desde la escena actual, hasta aquella escena cuyo nombre haya sido proporcionado como parámetro de entrada de la función "*GoToScene*".

```
public class CambioDeEscena : MonoBehaviour {
    public void GoToScene(string NombreEscena)
    {
        SceneManager.LoadScene(NombreEscena);
    }
}
```

Código 1: Script "Cambio De Escena"

4.4.6 Multi Target Manager Script

Este script ha sido desarrollado en C#, lenguaje utilizado dentro de Unity, implementando y utilizando como base funciones y librerías propias de Unity.

Por la forma en la que se ha desarrollado el script es necesario que los nombres de las imágenes que componen la librería sean idénticos a los nombres de los prefab a los que están referenciados.

Aunque con esta primera versión del script se consiguió poder identificar y asociar cada uno de los prefabs a sus marcadores correspondientes, así como reproducir la pista de audio asociada a los prefabs no era una solución válida para el funcionamiento buscado ya que cada vez que un target era detectado iniciaba la reproducción de la pista de audio desde el inicio, lo que provocaba que se perdiese la continuidad y sincronización en la reproducción al encontrarse cada una de las fuentes en un momento diferente de la pista.

Es por esto, entre otras funcionalidades necesarias que no habían sido implementadas. que se decidió desarrollar una versión más compleja de este Script.

4.4.7 Multi Target Manager Script V2

En esta nueva versión se incorporó la gestión de la reproducción de las pistas de audio de las fuentes además de añadir varios elementos informativos en la GUI.

Estos elementos se tratan de cuadros de texto destinados a mostrar el nombre de la fuente que esté siendo reproducida asociada al marcador que esté siendo detectado.

4.4.7.1 Gestión de la reproducción de las pistas

Para implementar la gestión de la reproducción de las fuentes se contaba con dos posibles opciones:

4.4.7.1.A *Pausar la reproducción al dejar de detectar el marcador*

En una primera opción se optó por implementarla mediante "*time stamps*". Al iniciar la aplicación se almacenaría el tiempo absoluto de inicio, de manera que, al detectar un marcador, se calculase la diferencia de tiempo desde el instante en el que se inicia la aplicación y el momento en el que se ha detectado el marcador, utilizando esta diferencia temporal como un "*delay*" que aplicar en la reproducción del audio.

Sin embargo, esta implementación no era eficiente ya que si una fuente se perdía por un instante y volvía a ser detectada generaba un desfase respecto de las demás fuentes que acababa provocando la pérdida de la sincronización entre las fuentes.

4.4.7.1.B *Silenciar la reproducción al dejar de detectar el marcador*

Dado que el pausar y reanudar la reproducción se generaba un delay se optó por silenciar las fuentes una vez el marcador asociado deja de ser detectado.

Esta implementación conlleva una modificación en el inicio de la escena ya que, al iniciarse la escena, debe iniciase la reproducción de todas las fuentes con un nivel de volumen definido en 0, es decir, silenciadas. Esta configuración consigue que, una vez es detectado el marcador, se establece el volumen de reproducción al 100%, al igual que cuando deja de ser detectado, se vuelve a silenciar estableciendo el volumen al 0%.

4.4.7.2 Parámetros de entrada del Script

El Script cuenta con 3 tipos de parámetros de entrada: el componente *"AR Tracked Image Manager"* de Unity denominado internamente en el script como **"*Detector De Imágenes*"**, el vector formado por los prefabs de las fuentes denominado **"*Modelos A Colocar*"** y los elementos informativos de la GUI denominados como ***"Modelo"*** seguido de la fuente como se puede apreciar en la Figura 49.

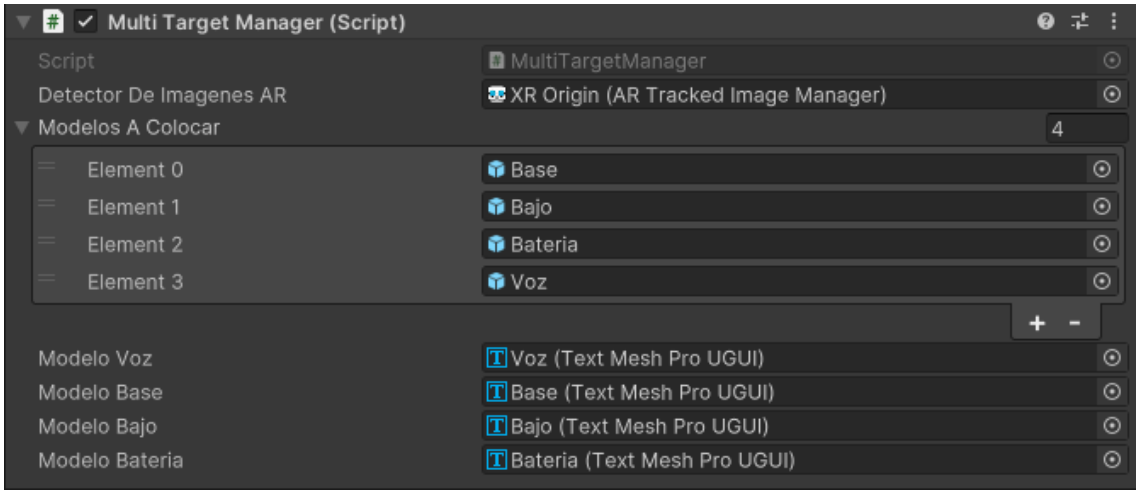


Figura 49: Parámetros de entrada del Script Multi Target Manager

```
[SerializeField] private ARTrackedImageManager DetectorDeImagenesAR;
[SerializeField] private GameObject[] ModelosAColocar;

[SerializeField] private TextMeshProUGUI ModeloVoz;
[SerializeField] private TextMeshProUGUI ModeloBase;
[SerializeField] private TextMeshProUGUI ModeloBajo;
[SerializeField] private TextMeshProUGUI ModeloBateria;
```

Código 2: Definición de parámetros de entrada de Multi Target Manager Script

4.4.7.3 Diccionarios empleados

La gestión de los prefabs se ha realizado mediante el uso de dos diccionarios:

- El diccionario **"*DiccionarioModelosAR*"** almacena y asocia los GameObjects, en este caso los prefabs de las fuentes, con los nombres de los mismos.
- El diccionario **"*EstadosDeLosModelos*"** actúa como un diccionario booleano almacenando el estado del modelo.

```
private Dictionary<string, GameObject> DiccionarioModelosAR = new
Dictionary<string, GameObject>();
private Dictionary<string, bool> EstadosDeLosModelos = new
Dictionary<string, bool>();
```

Código 3: Diccionarios empleados en Multi Target Manager Script

4.4.7.4 Variables globales definidas

Se han definido, de forma global, dos vectores que son usados a lo largo del código, estos vectores son usados para aplicar un escalado al prefab como se explicará más adelante.

```
Vector3 EscaladoDetectado = new Vector3(-1f, 1f, -1f);
Vector3 EscaladoPerdido = new Vector3(0f, 0f, 0f);
```

Código 4: Vectores de escalado

4.4.7.5 Start()

Al iniciarse la escena en la que se encuentra el script se invoca de forma inmediata la función "*Start()*". Esta función tiene como objetivo la instanciación de los prefabs dentro de la escena. Así como iniciar la reproducción de las fuentes asociadas

```
void Start() {
        foreach (var ModeloAR in ModelosAColocar) {
            GameObject NuevoModeloAR = Instantiate(ModeloAR, Vector3.zero,
Quaternion.identity);
            NuevoModeloAR.SetActive(true);

            NuevoModeloAR.name = ModeloAR.name;
            NuevoModeloAR.transform.localScale = EscaladoPerdido;

            DiccionarioModelosAR.Add(NuevoModeloAR.name, NuevoModeloAR
            EstadosDeLosModelos.Add(NuevoModeloAR.name, false);

            AudioSource FuenteSonora =
NuevoModeloAR.GetComponent<AudioSource>();
            FuenteSonora.Play(0);
            FuenteSonora.volume = 0.0f;
        }
    }
```

Código 5: Función Start()

4.4.7.6 OnEnable() y OnDisable()

Estas son funciones que pertenecen a los MonoBehaviour de Unity y son métodos especiales que Unity llama automáticamente cuando el objeto que contiene el script se activa *"OnEnable"* o se desactiva *"OnDisable"*.

La función "*OnEnable*", se suscribe al evento "*.trackedImagesChanged*" del componente "*DetectorDeImagenesAR*", de manera que cuando se detecta un cambio en las imágenes detectadas se ejecutará la función "*ImageFound*".

La función "*OnDisable*" realiza el comportamiento contrario, deteniendo la ejecución de la función "ImageFound".

```
    private void OnEnable() {
        DetectorDeImagenesAR.trackedImagesChanged += ImageFound;
    }
    private void OnDisable() {
        DetectorDeImagenesAR.trackedImagesChanged -= ImageFound;
    }
```

Código 6: Funciones OnEnable y OnDisable

4.4.7.7 Función CambioColorTexto

Esta función tiene como finalidad modificar el estilo visual de los elementos de texto de la GUI para designarles el color RGBA (141, 255, 231, 255) mientras que el marcador correspondiente al contenido del elemento de texto sea detectado, de igual manera que asignará el color RGBA (255, 255, 255, 255) cuando este deje de ser detectado.

```
private void CambioColorTexto(string MarcadorDetectado, bool Estado) {
    Color32 ColorTexto;

    if (Estado) {
        ColorTexto = new Color32(141, 255, 231, 255);
    }
    else {
        ColorTexto = new Color32(255, 255, 255, 255);
    }

    switch (MarcadorDetectado)
    {
        case "Voz":
            ModeloVoz.color = ColorTexto;
            break;
        case "Bajo":
            ModeloBajo.color = ColorTexto;
            break;
        case "Bateria":
            ModeloBateria.color = ColorTexto;
            break;
        case "Base":
            ModeloBase.color = ColorTexto;
            break;
    }
}
```

Código 7: Función CambioColorTexto

4.4.7.8 ImageFound()

En esta función se diferencian dos tipos de eventos dentro del evento iniciador "*ARTrackedImagesChanged*" ya que la actuación que debe realizarse no es la misma si la imagen detectada, ha sido detectada por primera vez, o si es una imagen que ya había sido detectada anteriormente, pero ha sido actualizado su estado de seguimiento.

4.4.7.8.A *Imágenes detectadas por primera vez*

Para esta primera situación se ha definido un bucle que recorre la lista de eventos "*eventData.added*", lista que contiene las imágenes que han sido detectadas por primera vez por el sistema AR.

En cada iteración del bucle se llama al método ShowARModel() proporcionando como entrada del mismo la variable "*ImagenDetectada*".

4.4.7.8.B *Imágenes actualizadas*

Si la imagen ya había sido detectada, se recorre la lista de eventos "*eventData. updated*", lista que contiene las imágenes cuyo estado de seguimiento ha cambiado.

Dentro del bucle, el código evaluará el estado de rastreo (trackingState) de cada imagen. Se distinguen dos posibles estados:

- **TrackingState.Tracking:** Si la imagen está siendo detectada correctamente, se llamrá al método ShowARModel(ImagenDetectada).
- **TrackingState.Limited:** Si la detección de la imagen es limitado, por ejemplo, la imagen no se puede ser detectada correctamente debido a condiciones de iluminación, movimiento o ya no se encuentra en el campo de visión, se llamará al método HideARModel(ImagenDetectada).

```
private void ImageFound(ARTrackedImagesChangedEventArgs eventData) {

    foreach (var ImagenDetectada in eventData.added) {
        ShowARModel(ImagenDetectada);
    }

    foreach (var ImagenDetectada in eventData.updated) {
        if (ImagenDetectada.trackingState == TrackingState.Tracking) {
            ShowARModel(ImagenDetectada);
        }
        else if (ImagenDetectada.trackingState == TrackingState.Limited) {
            HideARModel(ImagenDetectada);
        }
    }
}
```

Código 8: Función ImageFound()

4.4.7.9 ShowARModel()

En primer lugar, se accede al diccionario "*EstadoDeLosModelos*" y se comprueba el estado del prefab correspondiente al marcador detectado. En función del estado distinguimos dos situaciones:

- **Modelo Inactivo:** Si este prefab no se encontraba activo se realizarán las siguientes actuaciones:
  1. Se obtiene el modelo desde el diccionario "*DiccionarioModelosAR*.
  2. Se posiciona el modelo en el mismo lugar que la imagen detectada en el entorno real, "*ImagenDetectada.transform.position*", al prefab.
  3. Se activa el prefab mediante el método SetActive(true), lo que lo hace visible en la escena y se aplica una escala de 0.02 en las 3 dimensiones.
  4. Se actualiza el estado del modelo en el diccionario "*EstadosDeLosModelos"*.
  5. Se define, en el elemento de texto "*ModeloDetectado*" el nombre del prefab.
  6. Se modifica el volumen de la reproducción asignando el volumen al 100%.
  7. Se llama a la función CambioColorTexto proporcionándole el estado del modelo y el nombre del prefab.
- **Modelo Activo:** Si este prefab ya se encuentra activo únicamente se actualiza su posición en el entorno real.

```
   private void ShowARModel(ARTrackedImage ImagenDetectada) {
        bool EstadoActual =
EstadosDeLosModelos[ImagenDetectada.referenceImage.name];

        if (!EstadoActual) {
            GameObject ModeloAR =
DiccionarioModelosAR[ImagenDetectada.referenceImage.name];

            ModeloAR.transform.localScale = EscaladoDetectado;
            ModeloAR.transform.position = ImagenDetectada.transform.position;
            ModeloAR.transform.rotation = ImagenDetectada.transform.rotation;
            EstadosDeLosModelos[ImagenDetectada.referenceImage.name] = true;

            AudioSource FuenteSonora = ModeloAR.GetComponent<AudioSource>();
            FuenteSonora.volume = 1f;

            CambioColorTexto(ModeloAR.name, true);
        }
        else {
            GameObject ModeloAR =
DiccionarioModelosAR[ImagenDetectada.referenceImage.name];

            ModeloAR.transform.position = ImagenDetectada.transform.position;
            ModeloAR.transform.rotation = ImagenDetectada.transform.rotation;
```

```
            AudioSource FuenteSonora = ModeloAR.GetComponent<AudioSource>();
            FuenteSonora.volume = 1f;

            CambioColorTexto(ModeloAR.name, true);
        }
    }
```

Código 9: Función ShowARModel()

4.4.7.10 HideARModel()

Al igual que en la función ShowARModel(), en primer lugar, se accede al diccionario "*EstadoDeLosModelos*" y se comprueba el estado del prefab correspondiente al marcador detectado. Si el estado no es "*activado*" no se realizará ninguna actuación, mientras que, si el estado del prefab es "*activado*" se realizarán las siguientes actuaciones:

1. Se obtiene el modelo desde el diccionario "*DiccionarioModelosAR*.
2. Se desactiva el prefab mediante el método SetActive(false), lo que elimina el prefab de la escena y se aplica una escala de 0 en las 3 dimensiones.
3. Se actualiza el estado del modelo en el diccionario "*EstadosDeLosModelos*".
4. Se define, en el elemento de texto "*ModeloDetectado*" el valor "*Ninguno*".
5. Se modifica el volumen de la reproducción asignando el volumen al 0%.
6. Se llama a la función CambioColorTexto proporcionándole el estado del modelo y el nombre del prefab.

```
 private void HideARModel(ARTrackedImage ImagenDetectada) {
        bool EstadoActual =
EstadosDeLosModelos[ImagenDetectada.referenceImage.name];

        if (EstadoActual) {
            GameObject ModeloAR =
DiccionarioModelosAR[ImagenDetectada.referenceImage.name];

            ModeloAR.transform.localScale = EscaladoPerdido;
            EstadosDeLosModelos[ModeloAR.name] = false;

            AudioSource FuenteSonora = ModeloAR.GetComponent<AudioSource>();
            FuenteSonora.volume = 0f;

            CambioColorTexto(ModeloAR.name, false);
        }
    }
```

Código 10: Función HideARModel()

5 METODOLOGÍA

5.1 Recursos utilizados para la implementación AR

5.1.1 Hardware

CPU	AMD Ryzen 7 7700X 8-Core @4.50 GHz
RAM	32,0 GB DDR5 @5.5 GHz
Almacenamiento	1 TB SSD
GPU	NVIDIA RTX 4070
Sistema Operativo	Windows 11

5.1.2 Software

- Unity Hub en su versión 3.7.0
- Unity en su versión 2022.3.14f1
- Adobe Ilustrator
- Onshape

5.2 Recursos utilizados para el desarrollo de las Redes Neuronales

5.2.1 Hardware

Entrenar un modelo de Machine Learning requiere de una gran capacidad de cálculo computacional y, aunque el modelo Open-Unmix, no requiere de una gran capacidad computacional ya que puede ser entrenado mediante la CPU, esta opción no es eficiente y prolonga el tiempo de entrenamiento. Es por ello que se ha hecho uso de GPUs, hardware diseñado para realizar una alta cantidad de operaciones simultáneamente.

En este TFG se ha usado un sistema con las siguientes características:

CPU	Bullx-R xeon E5-2699v4 22c 2.2GHz-9.6GT-55M-145W
RAM	8x bullx-R 32GB DDR4-2400 ECC RDIMM (1x32GB) DR 1.2V
Almacenamiento	bullx-R 480GB 2.5" 7mm SATA3 Server SSD
GPU	TESLA P100 GPU card 16GB. EDU Customer
Sistema Operativo	Ubuntu 22.04 Server LTS

5.2.2 Software

El software usado en este proyecto ha sido Python a través de Miniconda. Se han utilizado distintas versiones para cada uno de los modelos empleados.

DeepConvSep, modelo convolucional original, se desarrolla en Python 2.7, utiliza Theano y Lasagne para su implementación de las redes neuronales. Además, utiliza librerías como Numpy o SciPy.

La versión modificada de Open-Unmix, modelo modificado para implementar un modelo convolucional, se ha desarrollado en Python 3.9. Este modelo emplea librerías como Pytorch, NumPy, Scikit-learn, FFMpeg así como CudaToolkit entre otras.

5.2.2.1 Python

Python es un lenguaje de programación ampliamente utilizado debido a sus numerosas ventajas.

Es un lenguaje multiplataforma, muy eficiente y fácil de aprender que cuenta con una sintaxis clara, similar al inglés y orientado a objetos.

Su naturaleza interpretada y de alto nivel permite una rápida prototipación y desarrollo de software. Además, una de las principales ventajas de Python es su vasta colección de bibliotecas especializadas, bibliotecas que proporcionan herramientas y funciones preconstruidas para diseñar y construir infinidad de sistemas.

Python cuenta con una comunidad activa de millones de desarrolladores en todo el mundo. Comunidad que contribuye continuamente con nuevas herramientas, recursos, bibliotecas y soporte, lo que agiliza la resolución de problemas y el desarrollo de nuevas técnicas. La abundancia de documentación, tutoriales y foros de discusión también ayuda a los desarrolladores a abordar desafíos complejos de manera efectiva.

Es por todo esto que la combinación de su sintaxis intuitiva, bibliotecas especializadas, la gran comunidad de soporte y capacidad de integración con la que cuenta Python que es la opción más usada para el desarrollo de redes neuronales.

5.2.2.2 Librerías usadas

A continuación, se describirán brevemente algunas de las librerías destacadas que se han comentado anteriormente:

5.2.2.2.A *Theano (Inactivo):*

Theano era una biblioteca de computación numérica poderosa enfocada en el Deep Learning. Ofrecía capacidades de diferenciación simbólica, haciéndola adecuada para definir y optimizar arquitecturas complejas de redes neuronales. Además, contaba con la capacidad de ejecutarse en GPUs, con la consiguiente aceleración del entrenamiento de los modelos que esto suponía.

5.2.2.2.B *Lasagne (Basado en Theano):*

Lasagne era una biblioteca de alto nivel construida sobre Theano, la cual proporcionaba una interfaz simple para construir y entrenar redes neuronales, centrándose en la modularidad y la reutilización del código.

5.2.2.2.C *PyTorch:*

PyTorch es una biblioteca de computación numérica poderosa con un fuerte enfoque en el aprendizaje profundo. Al igual que Theano cuenta con la capacidad de ejecutarse en GPUs.

5.2.2.2.D *NumPy:*

NumPy es la base de muchas bibliotecas científicas de Python. Es una librería especializada en el cálculo numérico y el análisis de datos ya que proporciona potentes capacidades de manipulación de arrays y matrices, operaciones de álgebra lineal y generación de números aleatorios entre otras funciones. Los arrays de NumPy son eficientes para cálculos numéricos y se utilizan a menudo como bloque de construcción para el procesamiento de datos arrays ya que permiten representar colecciones de datos de un mismo tipo en múltiples dimensiones además de contar con funciones muy eficientes para su manipulación.

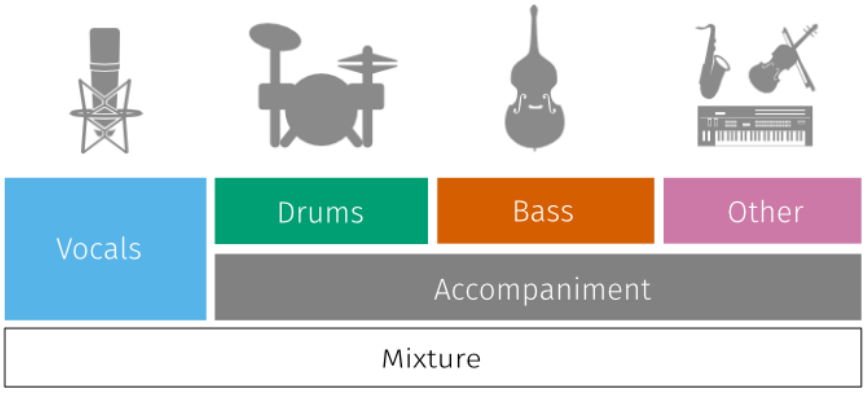
5.2.2.2.E *SciPy:*

SciPy complementa a NumPy al ofrecer una colección de algoritmos para diversas tareas de computación científica. Incluye funciones para optimización, integración, interpolación, transformadas de Fourier, procesamiento de señales y más.

5.2.2.2.F *Scikit-Learn:*

Scikit-Learn es una biblioteca de alto nivel construida sobre NumPy y SciPy. Proporciona una colección de algoritmos bien probados para diversas tareas de aprendizaje automático, que incluyen versiones de algoritmos de clasificación, regresión, agrupación, reducción de dimensionalidad y selección de modelos. Scikit-Learn ofrece una interfaz fácil de usar para construir y evaluar modelos de aprendizaje automático.

5.3 Dataset (MusDB18)



MusDB18 [15] es un conjunto de datos público de música de alta calidad ampliamente utilizado para la investigación en separación de fuentes musicales. Contiene 150 mezclas estereofónicas completas de diversos géneros, junto a estas mezclas se proporcionan además sus pistas aisladas, que son grabaciones separadas de la batería, el bajo, la voz y los acompañamientos.

La distribución de géneros musicales que podemos encontrar en el dataset, está predominada por el pop y el rock, tal y como podemos observar en la siguiente tabla:

Género	Cantidad de Canciones
Singer/Songwriter	13
Pop/Rock	83
Country	3
Rock	15
Rap	7
Reggae	2
Electronic	7
Jazz	3
Pop	12
Heavy Metal	11

Tabla 2: Distribución de las canciones por género en MUSDB18

El dataset MusDB18 consta de dos subconjuntos de datos. Uno de ellos destinado al entrenamiento de modelos, compuesto por 100 canciones y el otro, destinado a la realización de pruebas una vez el modelo ha sido entrenado, compuesto por 50 canciones.

Se debe de tener en cuenta que para este trabajo se ha empleado el dataset MusDB18-HQ, el cual contiene las mismas canciones que el dataset original, sin embargo, el formato de los archivos de audio es diferente siendo en esta versión HQ (High-Quality) archivos WAV sin compresión.

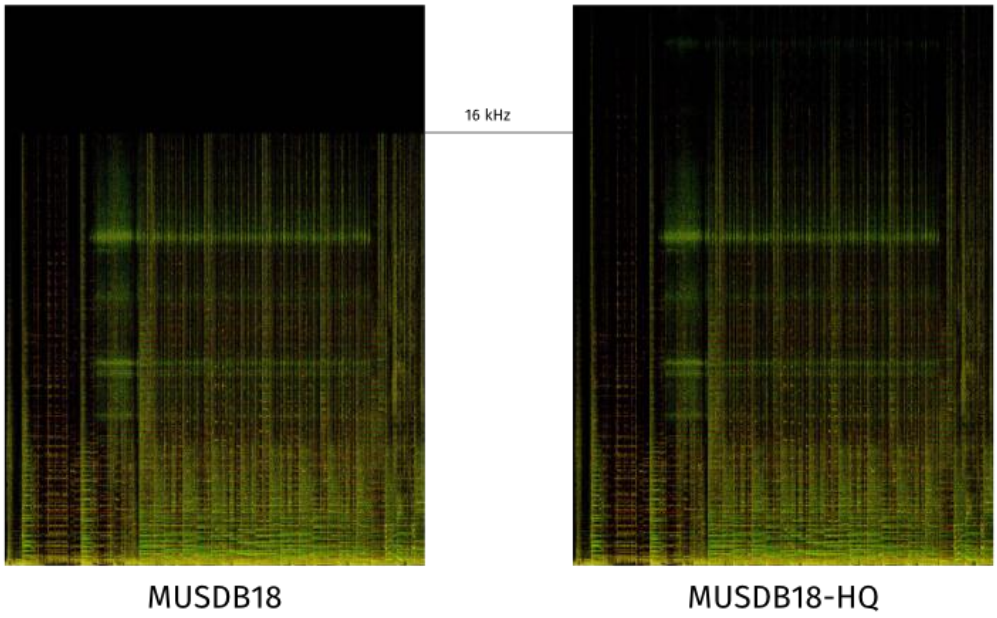


Figura 50: Comparativa entre MusDB18 y MusDB18-HQ [15]

5.4 Modelos de separación de fuentes musicales empleados

5.4.1 DeepConvSep

Este modelo de separación automático de fuentes musicales fue propuesto por Pritish Chadna y Marius Miron en el año 2017.

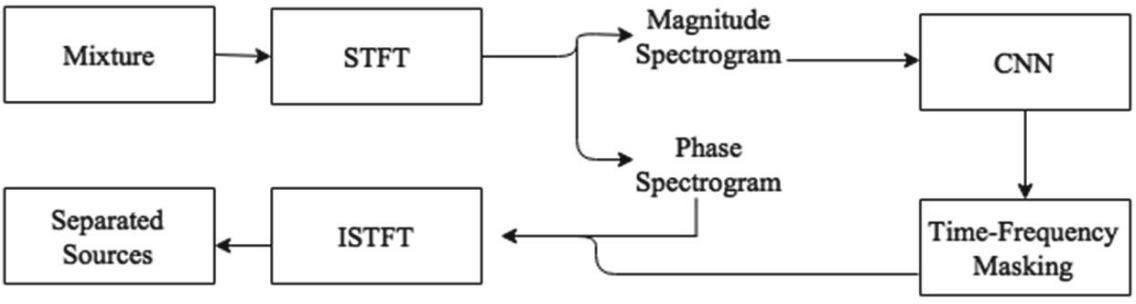


Figura 51: Diagrama de bloques DeepConvSep [16]

Este modelo hace uso de la STFT (Transformada de Fourier de Tiempo Corto) para calcular el espectrograma de magnitud de la mezcla, el cual será proporcionado como entrada a la red neuronal convolucional (CNN) con el fin de obtener a la salida una estimación de cada una de las fuentes que conforman la mezcla.

A partir de la estimación obtenida se genera una máscara tiempo-frecuencia para cada una de las fuentes, de manera que, al aplicar la máscara sobre el espectrograma se obtiene el espectrograma de cada una de las fuentes. Este espectrograma modificado junto a la fase, se empleará para obtener las señales de audio correspondientes a las distintas fuentes sonoras separadas.

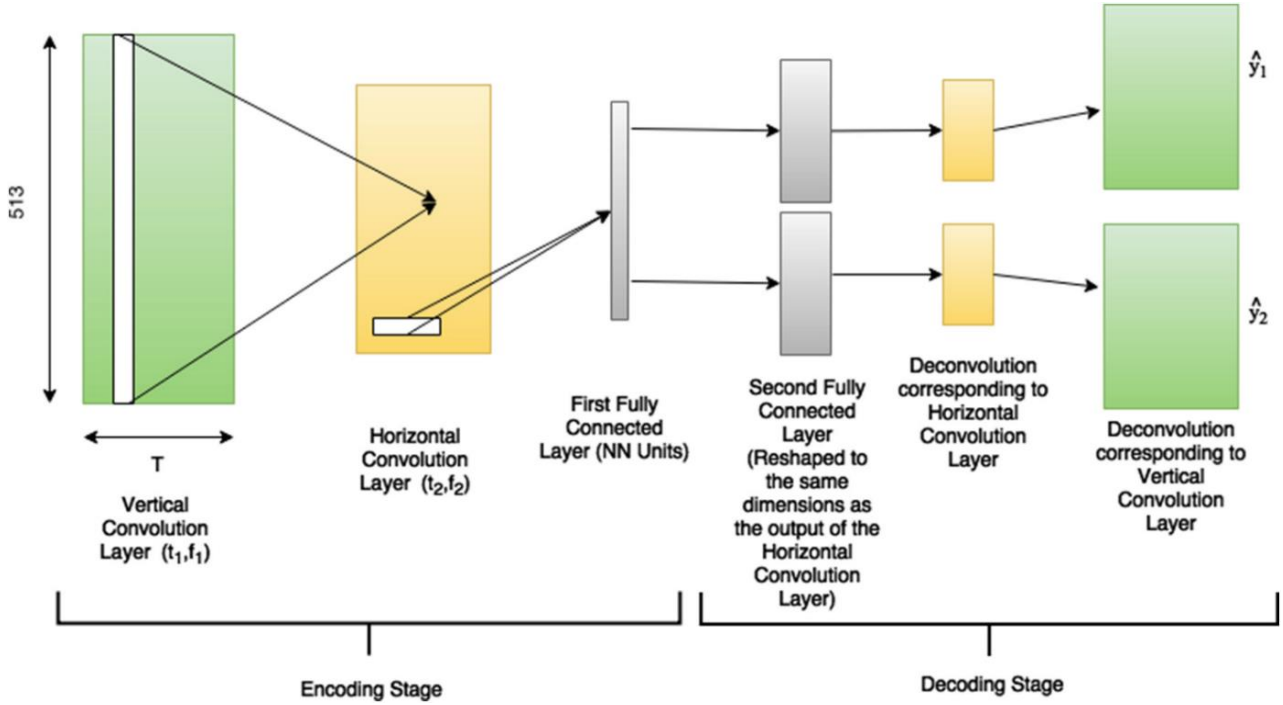


Figura 52: Arquitectura del modelo DeepConvSep [16]

En el esquema de la Figura 52 se puede identificar la arquitectura de la red neuronal de este modelo. Aunque en este caso únicamente se obtienen dos salidas $\hat{y}_1$ e $\hat{y}_2$, el funcionamiento es escalable a las cuatro salidas deseadas en la separación de fuentes.

Además, se puede observar que la red está formada principalmente por una etapa de codificación seguida de una etapa de decodificación en las que participan las diversas capas que forman la red:

- **Capa de Convolución Vertical:** Se realiza una convolución vertical con el objetivo de obtener una representación de las características de frecuencia de la mezcla proporcionada como entrada de la red.
- **Capa de Convolución Horizontal:** Esta segunda convolución modela la evolución temporal de los diferentes instrumentos a partir de las características extraídas en la capa anterior.
- **Primera Capa completamente conectada:** Es la última capa de la fase de codificación, y tiene como objetivo facilitar la obtención de características no lineales a partir de las salidas anteriores. Esta es una capa compartida por todas las salidas de las fuentes y actúa como cuello de botella de la red. En esta capa se implementa la función de activación ReLu.

- **Segunda Capa completamente conectada:** Esta capa tiene como finalidad aplicar un "*filtrado*" de manera que se obtengan las características que son idóneas para determinar la salida de cada una de las fuentes.
- **Capa de Deconvolución Horizontal:** En esta capa se realiza el proceso inverso al que es implementado en la Capa de Convolución Horizontal.
- **Capa de Deconvolución Vertical:** Esta capa proporciona la salida final de la red, esta salida puede interpretarse como las máscaras tiempo-frecuencia de las fuentes.

5.4.2 Modelo Open-Unmix

Open-Unmix es un modelo de código abierto diseñado para la separación de fuentes musicales basado en PyTorch. El modelo es capaz de descomponer una mezcla de música en sus componentes individuales, voz, batería, bajo, y otros instrumentos.

Open-Unmix es un modelo basado en redes neuronales recurrentes, las cuales, al igual que pasaba en DeepConvSep, trabajan en el dominio tiempo-frecuencia, de manera que realiza la separación de las fuentes operando sobre el espectrograma de la señal. La señal de entrada se divide en segmentos garantizando así un coste computacional que no pueda verse comprometido, de manera que el espectrograma de entrada se limita a bandas de 16 KHz las cuales son normalizadas usando la media global y la desviación estándar.

Además, se aplica una normalización por lotes en distintas etapas del modelo, lo que permite reducir el riesgo de sobreajuste durante el entrenamiento.

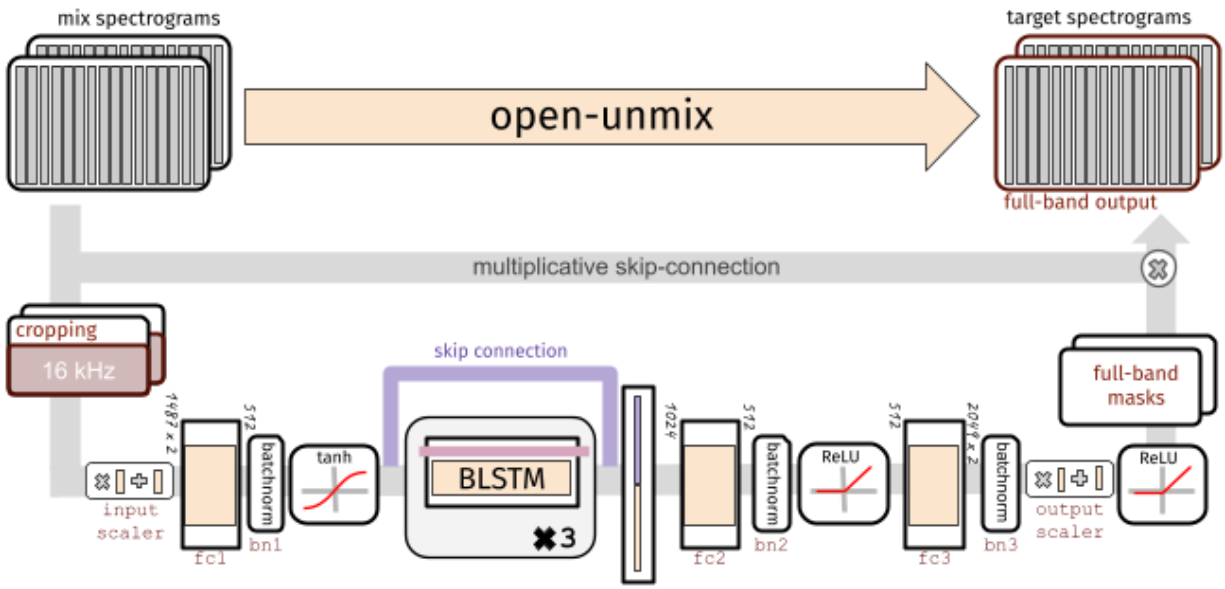


Figura 53: Arquitectura del modelo Open-Unmix [17]

Open-Unmix cuenta con un diseño modular que permite la modificación y personalización de las distintas secciones que lo conforman permitiendo así una gran adaptabilidad frente a las distintas necesidades que puedan plantearse.

La red neuronal que conforma el modelo está implementada mediante tres capas LSTM bidireccionales. El uso de este tipo de capas le permite a la red tener contexto completo de la señal al procesarse los datos tanto hacia adelante como hacia atrás, lo que permite al modelo capturar dependencias temporales tanto del pasado como del futuro simultáneamente. Esto provoca que sea un modelo que no puede ser implementado en tiempo real.

El uso de capas LSTM y el análisis bidireccional que estas proporcionan resultan muy útil, específicamente, para las implementaciones destinadas al procesamiento de audio ya que se utiliza la información futura y pasada de la señal para determinar los valores intermedios e influir en la predicción de salida.

Dado que la LSTM no es capaz de funcionar con la resolución original del espectrograma de entrada la red aprende a comprimir el eje de frecuencias del modelo reduciéndose la redundancia del modelo y consiguiendo así que este converja con mayor rapidez, lo que se traduce en tiempos de entrenamiento más reducidos.

Para llevar a cabo esta compresión de los datos, se usan capas completamente conectadas.

Tras su paso por la red LSTM la señal es decodificada a su tamaño original. La salida de la red se multiplica por el espectrograma de entrada obteniendo así el espectrograma de la fuente deseada que será finalmente empleado para la síntesis de la señal de salida mediante la STFT inversa.

En la última etapa del modelo, para llevar a cabo la inferencia, la señal es procesada mediante la implementación de un filtro Wiener multicanal.

Las funciones de activación implementadas en este modelo son ReLU, tanh y sigmoidea a lo largo del mismo cuyo funcionamiento y cometido fue descrito anteriormente en la introducción teórica de esta memoria.

5.4.3 Modelo propuesto: Convolutional-Unmix

Una vez han sido estudiados ambos modelos de separación de fuentes musicales, se propone un nuevo modelo inspirado en la arquitectura de la red neuronal implementada en el modelo DeepConvSep, utilizando el software de código abierto Open-Unmix.

El objetivo es evaluar el rendimiento de las redes convolucionales en un software más actualizado y moderno implementando además la separación mediante redes neuronales convolucionales. Una de las principales limitaciones del modelo DeepConvSep es su implementación, ya que este modelo está implementado en una versión Python 2.7, una versión que ya no recibe actualizaciones. Es por esto que se decidió utilizar como base el modelo Open-Unmix debido a su diseño modular, que facilita modificaciones y ampliaciones.

5.4.3.1 Arquitectura del modelo

Como se ha mencionado anteriormente este modelo está basado en el modelo DeepConvSep, el cual está formado por un sistema que aplica un codificador el cual nos permite comprimir los datos de entrada, en este caso las señales con la mezcla, seguido de un decodificador.

El modelo original implementa 4 decodificadores destinando uno para cada uno de las fuentes de la mezcla, sin embargo, en este modelo desarrollado utilizando como base Open-Unmix se ha implementado un único decodificador ya que se entrenarán las distintas fuentes por separado debido a la limitación de Open-Unmix en el entrenamiento de la red.

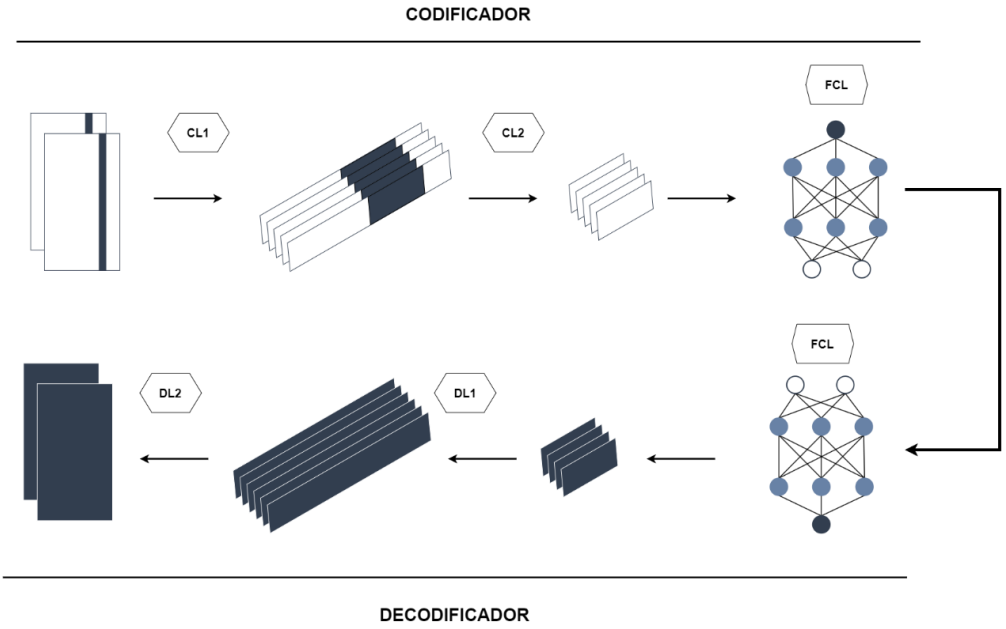


Figura 54: Arquitectura del modelo de red neuronal propuesto

5.4.3.1.A *Codificación*

Como se puede observar en la Figura 54 el codificador está definido mediante dos capas convolucionales en serie.

La primera capa del compresor, ***CL1***, es una capa de convolución vertical, la cual cuenta con 50 filtros cuyas dimensiones son 2049x1. Esta capa tiene como objetivo capturar patrones espectrales dentro del espectrograma de la mezcla.

Segunda capa del compresor, ***CL2***, se trata de una capa de convolución horizontal, la cual cuenta con 30 filtros cuyas dimensiones son 1x15. Esta capa tiene como objetivo obtener la evolución temporal de los patrones espectrales obtenidos en la primera capa.

La salida obtenida de esta segunda capa, ***CL2***, se proporciona como entrada de una capa completamente conectada ***FCL*** de 512 neuronas con el fin de producir un espacio latente que identifique y represente las características más relevantes. Con esta capa se completa el proceso de codificación.

5.4.3.1.B *Decodificación*

La salida de esta ***FCL*** se introduce ahora en el decodificador el cual tiene como finalidad obtener a partir del espacio latente producido el espectrograma de la fuente que deseamos separar de la mezcla.

De manera análoga a la implementación realizada en el codificador se ha implementado el decodificador, invirtiendo el orden de las capas empleadas.

La primera capa del decodificador es una ***FCL*** de 512 neuronas seguida de una capa convolucional transpuesta horizontal, ***DL1***, cuya salida alimenta otra capa convolucional transpuesta vertical, ***DL2*** capas en las que se lleva a cabo la deconvolución y nos proporcionan a la salida el espectrograma aislado de la fuente deseada.

5.4.3.2 Código del modelo

El código del modelo podrá encontrarse en el repositorio de GitHub creado:

Convolutional-Unmix

6 ENTRENAMIENTO DE LOS MODELOS

Este capítulo tiene como finalidad explicar todos los detalles del entrenamiento del modelo de rede neuronal propuesto.

6.1 Training API

Para realizar el entrenamiento del modelo desarrollado se ha hecho uso de la API de entrenamiento con la que cuenta el modelo base de Open-Unmix. Esta API no permite el entrenamiento simultáneo de las fuentes por lo que se ha entrenado cada una de las fuentes por separado empleando en todos los casos los mismos parámetros.

Parámetro	Descripción	Default
--is-wav	Carga los archivos en formato WAV decodificados para una carga de datos más rápida.	False
--root	Ruta donde se encuentra la base de datos usada para el entrenamiento	-
--target <str>	Fuente para la que se desea realizar el entrenamiento.	vocals
--output <str>	Ruta donde guardar el modelo entrenado, así como los checkpoints.	./open-unmix
--checkpoint <str>	Ruta al checkpoint del modelo para reanudar el entrenamiento.	-
--model <str>	ruta o cadena al objetivo entrenado previamente para ajustar el modelo	-
--no_cuda	Deshabilita el uso de CUDA incluso si está disponible.	-
--epochs <int>	Número de épocas a entrenar.	1000
--batch-size <int>	Tamaño del lote de entrenamiento influye en el rendimiento del entrenamiento.	16
--seq-dur <int>	Duración de la secuencia, en segundos, de los fragmentos extraídos del conjunto de datos.	6
--hidden-size <int>	Tamaño oculto de capas densas.	512
--nfft <int>	Longitud de la ventana STFT-FFT en muestras.	4096
--weight-decay <float>	Decaimiento de peso para regularización	0.00001
--nb-workers <int>	La cantidad de trabajadores o hilos de procesamiento paralelos.	0

Tabla 3: Parámetros de entrenamiento Open-Unmix [17]

6.2 Entrenamiento del modelo Convolutional-Unmix

El entrenamiento del modelo propuesto se ha realizado con los parámetros designados en el fichero de entrenamiento proporcionado en el modelo Open-Unmix train.py. A continuación, se adjunta el extracto del mismo donde se puede apreciar la definición de los parámetros implementados.

```
# Training Parameters

  parser.add_argument("--epochs", type=int, default=1000)
  parser.add_argument("--batch-size", type=int, default=16)
  parser.add_argument("--lr", type=float, default=0.001)
  parser.add_argument("--patience", type=int, default=140)
  parser.add_argument("--lr-decay-patience", type=int, default=80)
  parser.add_argument("--lr-decay-gamma", type=float, default=0.3)
  parser.add_argument("--weight-decay", type=float, default=0.00001)
  parser.add_argument("--seed", type=int, default=42, metavar="S")

# Model Parameters

  parser.add_argument("--seq-dur", type=float, default=6.0)
  parser.add_argument("--unidirectional", action="store_true", default=False)
  parser.add_argument("--nfft", type=int, default=4096)
  parser.add_argument("--nhop", type=int, default=1024)
  parser.add_argument("--hidden-size",type=int,default=512)
  parser.add_argument("--bandwidth", type=int, default=16000)
  parser.add_argument("--nb-channels", type=int, default=2)
  parser.add_argument("--nb-workers", type=int, default=0)
  parser.add_argument("--debug", action="store_true", default=False)
```

Código 11: Parámetros de entrenamiento del modelo

```
python train.py --root /mnt/BBDD/Audio/musdb18HQ/ --target XXXXX --nb-workers
32 --output /home/Modelo --is-wav --checkpoint /home/Modelo/
```

Código 12: Comando ejecutado para iniciar el entrenamiento

Además de los parámetros definidos en el fichero train.py, se han definido otros parámetros adicionales, parámetros definidos con el fin de mejorar y acelerar el proceso de entrenamiento de cada una de las fuentes. Estos parámetros han sido especificados en la ejecución del entrenamiento, durante la llamada del archivo train.py:

- –nb-workers 32
- –output /home/Modelo
- –is-wav
- –checkpoint /home/Modelo/

6.2.1 Modificaciones en los parámetros de entrenamiento

Al emplear para el entrenamiento del modelo diferentes tamaños de lotes (batch-size), se observa que se modifica el proceso de entrenamiento. Al aumentarse el tamaño de los lotes se consigue reducir el tiempo de procesamiento de cada época. Sin embargo, esta modificación no se puede emplear de manera indefinida ya que valores demasiado elevados provocan un aumento en la disonancia de los valores de pérdida de entrenamiento y pérdida de prueba.

Otro parámetro que se puede incrementar es el decaimiento de peso (weight_decay). Aumentar este parámetro produce una reducción de la diferencia entre la pérdida de entrenamiento y la de prueba dentro de unos márgenes limitados. Ha de tenerse en cuenta que, valores demasiado altos de este parámetro producen una reducción drástica de los valores de SDR.

6.2.2 Registro del entrenamiento

La API de entrenamiento proporciona un fichero de registro en formato JSON con todo el proceso de entrenamiento para cada una de las fuentes.

En estos archivos se pueden identificar los detalles de la configuración utilizada para el entrenamiento, así como un registro de las pérdidas de entrenamiento ***"train_loss_history"***, la validación de cada una de las épocas entrenadas ***"valid_loss_history"***, la época para la que se obtuvo el mejor resultado ***"best_epoch"***, el número de épocas entrenadas ***"epochs_trained"*** entre otros datos.


```
{
    "args": {
        …
    },
    "best_epoch": 656,
    "best_loss": 1.536461889743805,
    "epochs_trained": 796,
    "num_bad_epochs": 143,
    "train_loss_history": [
        1.7953364954784858,
        1.5226527086870616…
    ],
    "train_time_history": [
        5591.39773273468,
        5707.818798542023…
    ],
    "valid_loss_history": [
        2.3273858853748868,
        2.200346257005419,…
    ]
}
```


Código 13: Fragmento del archivo de entrenamiento de una fuente

7 RESULTADOS

7.1 Remixing espacial en AR

De la implementación realizada se ha obtenido una aplicación compatible con Android, IOS y otros entornos de virtualización como pueden ser gafas de realidad virtual capaz de proporcionar un entorno en AR interactivo. Siendo este un entorno interactivo que le permite al usuario controlar cada una de las fuentes mediante elementos físicos como son los marcadores.

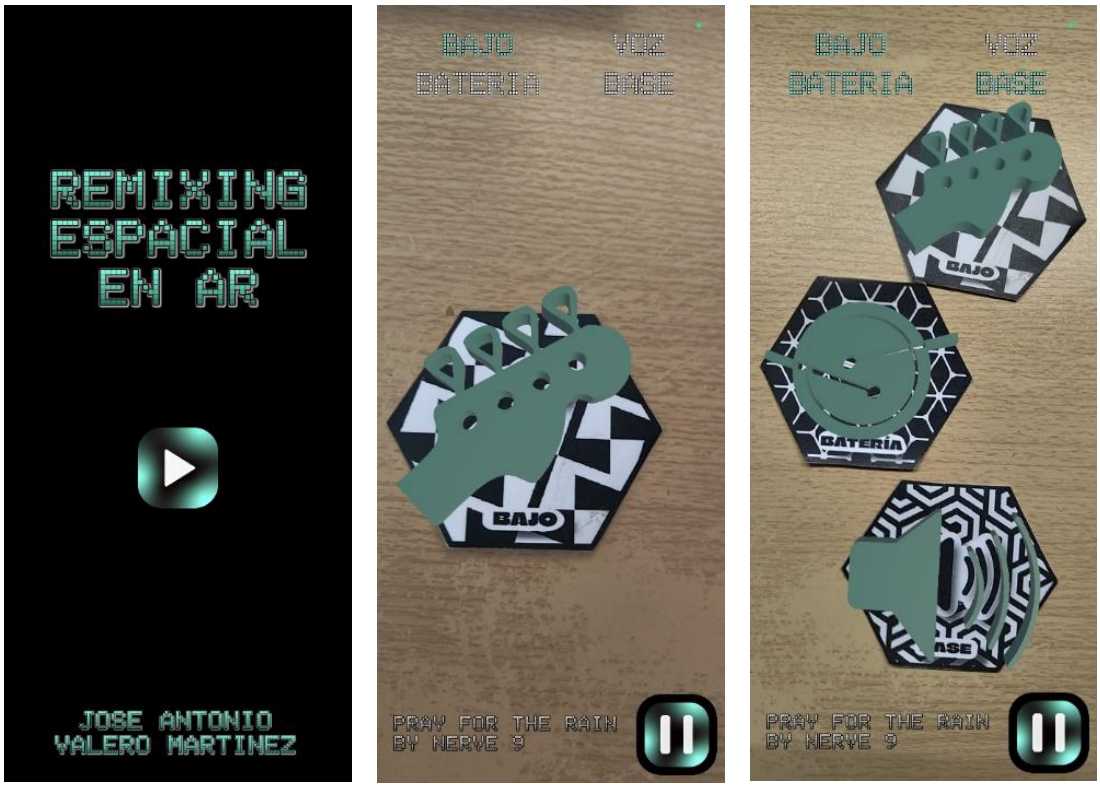


Figura 55: Imágenes de la Aplicación AR en funcionamiento

Demo gráfica del funcionamiento de la aplicación

7.2 Separación de fuentes

Una vez entrenado el modelo propuesto se ha realizado una evaluación objetiva de los resultados obtenidos de cada uno de los modelos de separación de fuentes. Este capítulo tiene como finalidad proporcionar los resultados obtenidos.

7.2.1 Obtención de las fuentes separadas

Para obtener las fuentes que componen los mixtures se han empleado los comandos que se muestran a continuación.

7.2.1.1 Comando y parámetros empleados en la separación de las fuentes

```
umx mixture.wav
```

Código 14: Comando para realizar la separación de fuentes

Aunque este sea el comando que permite ejecutar la separación de las fuentes debemos indicarle algunos parámetros adicionales para obtener los resultados deseados al utilizar el modelo convolucional entrenado.

Parámetro	Descripción
--model	Ruta en la que se encuentran los archivos del modelo entrenado.
--target <str>	Fuente para la que se desea realizar el entrenamiento.
-- outdir <str>	Ruta donde guardar las fuentes separadas.

Código 15: Parámetros adicionales en la separación de fuentes

```
umx /mnt/BBDD/Audio/musdb18HQ/test/*/mixture.wav --model /home/javm0009/Modelo
--target vocals drums bass other --outdir /home/javm0009/FuentesSeparadasConv/*
```

Código 16: Separación de fuentes empleando el modelo convolucional entrenado

```
umx /mnt/BBDD/Audio/musdb18HQ/test/*/mixture.wav --target vocals drums bass
other --outdir /home/javm0009/FuentesSeparadasOriginal/*
```

Código 17: Separación de fuentes empleando el modelo original

Los símbolos **“*”** empleados implican que la ruta mostrada es relativa ya que, dentro de la misma, sustituyendo el “*” existen varios subdirectorios diferentes con la misma estructura.

7.2.2 Obtención de los resultados

Para obtener estos valores se ha empleado la herramienta bss_eval [18]. Herramienta implementada en Matlab que nos proporciona los valores de SDR, SIR y SAR de las fuentes separadas.

Para ello se ha desarrollado un pequeño script que será ejecutado para cada una de las mezclas separadas obtenidas, modificando, en cada ejecución, la ruta desde la que se obtienen los archivos de audio a comparar.

7.2.2.1 Script de evaluación

```
cd /home/javm0009/FuentesSeparadasConv/*/mixture

[x_other, fs] = audioread('other.wav');
x_other = x_other(:, 1);
[x_drums] = audioread('drums.wav');
x_drums = x_drums(:, 1);
[x_bass] = audioread('bass.wav');
x_bass = x_bass(:, 1);
[x_vocals] = audioread('vocals.wav');
x_vocals = x_vocals(:,1);

X = [x_vocals, x_drums, x_bass, x_other];

cd /mnt/BBDD/Audio/musdb18HQ/test/*

[y_other, fs] = audioread('other.wav');
y_other = y_other(:, 1);
[y_drums] = audioread('drums.wav');
y_drums = y_drums(:, 1);
[y_bass] = audioread('bass.wav');
y_bass = y_bass(:, 1);
[y_vocals] = audioread('vocals.wav');
y_vocals = y_vocals(:,1);

Y = [y_vocals, y_drums, y_bass, y_other];

cd('/home/javm0009/EvaluacionMatlab')

[SDR,SIR,SAR,perm] = bss_eval_sources(X', Y');

Datos = [SDR, SIR, SAR, perm]

writematrix(Datos, '/home/javm0009/EvaluacionMatlab/DatosEvaluacion.csv',
'WriteMode', 'append');
```

Código 18: Script en Matlab desarrollado para la Evaluación

Los valores obtenidos de este script son almacenados en un archivo CSV denominado "*DatosEvaluacion.csv*" el cual tiene el siguiente formato:

```
SDR, SIR, SAR, perm (1- x_vocals 2- x_drums 3- x_bass 4- x_other)
-2.0288, 5.51258, -0.111727, 1
9.3174, 17.1472, 10.1826, 2
2.8843, 8.02208, 5.10875, 3
1.9585, 4.09847, 7.48603, 4
```

Código 19: Formato de los datos de evaluación

7.2.3 Asociación de los audios con su ID

ID	Nombre	ID	Nombre
1	Al James - Schoolboy Facination	26	Mu - Too Bright
2	AM Contra - Heart Peripheral	27	Nerve 9 - Pray For The Rain
3	Angels In Amplifiers - I'm Alright	28	PR - Happy Daze
4	Arise - Run Run Run	29	PR - Oh No
5	Ben Carrigan - We'll Talk About It All Tonight	30	Punkdisco - Oral Hygiene
6	BKS - Bulldozer	31	Raft Monk - Tiring
7	BKS - Too Much	32	Sambasevam Shanmugam - Kaathaadi
8	Bobby Nobody - Stitch Up	33	Secretariat - Borderline
9	Buitraker - Revo X	34	Secretariat - Over The Top
10	Carlos Gonzalez - A Place For Us	35	Side Effects Project - Sing With Me
11	Cristina Vane - So Easy	36	Signe Jakobsen - What Have You Done To Me
12	Detsky Sad - Walkie Talkie	37	Skelpolu - Resurrection
13	Enda Reilly - Cur An Long Ag Seol	38	Speak Softly - Broken Man
14	Forkupines - Semantics	39	Speak Softly - Like Horses
15	Georgia Wonder - Siren	40	The Doppler Shift - Atrophy
16	Girls Under Glass - We Feel Alright	41	The Easton Ellises - Falcon 69
17	Hollow Ground - Ill Fate	42	The Easton Ellises (Baumi) - SDRNR
18	James Elder & Mark M Thompson - The English Actor	43	The Long Wait - Dark Horses
19	Juliet's Rescue - Heartbeats	44	The Mountaineering Club - Mallory
20	Little Chicago's Finest - My Own	45	The Sunshine Garcia Band - For I Am The Moon
21	Louis Cressy Band - Good Time	46	Timboz - Pony
22	Lyndsey Ollard - Catching Up	47	Tom McKenzie - Directions
23	M.E.R.C. Music - Knockout	48	Triviul feat. The Fiend - Widow
24	Moosmusic - Big Dummy Shake	49	We Fell From The Sky - Not You
25	Motor Tapes - Shore	50	Zeno - Signs

Tabla 4: Asignación de los identificadores a los audios

7.2.4 Resultados del modelo Open-Unmix

Para la evaluación del modelo se han empleado los audios obtenidos al realizar la separación de fuentes haciendo uso de la versión preentrenada del modelo de Open-Unmix que puede ser descargada desde su repositorio oficial.

ID	SDR			SIR			SAR		
	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO
1	7,2	4,51	-3,64	16,73	11,06	6,28	7,9	5,23	-3,19
2	6,41	3,44	5,52	10,85	4,08	16,36	5,17	1,18	2,3
3	8,33	5,32	4,46	10,53	9,01	20,41	6,63	4,08	1,17
4	5,73	5,5	6,21	-0,33	-8,36	19,42	12,68	11,41	-10,24
5	2,75	8,65	6,54	-3,91	-2,58	30,67	0,66	1,57	0,09
6	9,17	6,07	9,04	3,16	-12,56	13,03	4,05	0,17	1,36
7	2,45	6,3	6,7	18,13	16,06	16,54	2,5	5,95	6,44
8	7,39	5,49	5,56	22,17	16,05	17,81	6,63	4,72	5,32
9	1,34	7,36	6,18	-1,54	4,48	13,44	0,26	4,58	2,41
10	4,62	3,62	-1,18	21,98	11,23	1,91	3,24	3,07	-0,35
11	8,37	8,97	4,28	1,68	7,1	31,03	2,99	5,81	0,08
12	6,89	4,99	0	12,65	16,99	1,36	7,6	5,73	-0,05
13	8,42	0,95	3,43	27,19	-5,04	-1,29	0,22	0,16	1,06
14	5,53	8,4	4,67	25,03	17,59	19,74	4,97	8,91	2,96
15	4,11	6,44	6,52	12,67	14,21	14,26	3,08	5,89	5,34
16	3,96	6,35	2,72	24,27	12,37	10,12	3,35	6,81	2,99
17	6,44	4,54	0,69	19,94	15,11	11,67	6,19	3,95	-0,85
18	5,43	5,42	4,94	3,91	-0,37	24,24	2,54	1,91	0,17
19	7,13	9,63	1,64	23,91	24,24	17,3	7,27	9,83	-0,12
20	9,19	1,8	9,14	17,64	20,33	22,49	9,37	8,27	9,48
21	6,48	7,09	5,65	8,61	7,79	22,37	4,66	5	0,41
22	9,93	4,63	4,44	22,97	11,33	14,51	9,15	3,62	4,42
23	8,1	4,29	6,4	3,1	-6,08	5,11	3,59	0,39	1,61
24	8,08	7,67	4,24	18,86	17,78	11,96	7,41	8,31	3,53
25	3,45	5,3	9,28	24,94	18,5	16,39	2,38	5,29	8,52
26	8,56	12,1	6,84	2,93	7,15	24,28	3,79	6,95	0,55
27	5,15	7,99	4,8	8,88	4,65	22,59	3,96	4,17	0,29
28	-11,06	10,85	0	-30,82	16,56	16,63	-0,49	13,02	-1,87
29	-9,91	5,28	1,09	-25,71	16,09	20,99	2,35	8,69	1,96
30	5,74	6,33	4,57	13,9	14,64	17,69	6,38	7,5	5,84
31	4,12	4,49	5,83	-1,37	-10,71	6,54	1,03	0,21	2,18
32	10,09	3,22	8,01	24,61	15,74	16,58	9,79	1,74	8,5
33	5,06	3,82	3,73	22,5	16,32	17,01	4,43	2,83	2,3
34	6,22	5,13	9,49	18,85	15,72	18,31	6,08	4,87	10,21

35	10,68	10,49	6,79	15,66	21,65	17,31	11,62	12,49	6,42
36	7,76	2,67	-0,33	14,21	-6,53	13,58	6,33	0,15	-0,01
37	0	5,77	9,32	10,6	15,82	14,76	-0,4	5,64	9,63
38	2,34	4,38	10,35	7,23	17,59	20,54	3,5	6,19	10,55
39	5,65	3,12	11,94	22	22,16	28,29	5,59	3,62	12,03
40	4,87	6,82	1,14	6,19	-0,04	9,93	3,57	1,57	-0,27
41	2,76	7,14	2,48	13,32	16,71	19,8	1,81	7,81	2,42
42	2,49	7,42	6,27	16,68	13,36	16,49	0,9	9,03	5,02
43	7,08	4,77	4,9	24,11	15,58	14,96	6,75	3,88	3,58
44	7,84	4,15	8,23	24,87	15	14,85	8,4	1,85	8,98
45	9,35	0,5	5,33	18,52	14,74	6,56	9,61	-1,49	6,8
46	2,82	3	1,63	12,35	12,33	7,65	1,9	1,31	0,13
47	7,1	11,93	4,28	0,44	-4,31	17,86	2,78	1,38	0,08
48	6,98	3,8	9,72	16,14	11,61	16,3	6,87	3,52	11,98
49	4,26	9,25	1,24	15,41	14,6	13,8	3,22	9,23	-1,23
50	5,66	6,53	1,09	25,07	15,5	17,14	5,37	6,76	-3,6

Tabla 5: Resultados modelo Open-Unmix

7.2.5 Resultados del modelo Convolutional-Unmix

Para la evaluación del modelo se han empleado los audios obtenidos al realizar la separación de fuentes empleando la versión del modelo de modificado de Open-Unmix, Convolutional-Unmix.

ID	SDR			SIR			SAR		
	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO	VOZ	BATERÍA	BAJO
1	3,92	-8,08	3,76	9,68	5,55	9,56	5,70	2,48	5,53
2	5,33	3,10	-5,03	8,88	7,06	-3,18	8,39	0,61	4,44
3	0,71	2,38	0,39	14,53	31,75	47,35	8,17	4,58	3,65
4	2,47	-2,09	1,47	9,97	1,35	3,11	0,37	2,92	8,24
5	5,41	0,40	3,85	8,78	9,52	8,79	1,22	5,84	6,06
6	3,07	7,34	0,47	44,50	14,11	12,86	4,88	8,53	0,56
7	6,31	3,84	0,48	11,62	11,48	31,35	8,12	4,96	0,86
8	4,40	1,42	2,60	10,02	38,68	4,35	6,20	2,98	8,75
9	-2,03	9,32	2,88	9,89	17,15	34,98	-11,17	10,18	51,09
10	1,99	2,87	-3,36	9,95	9,17	1,77	0,32	0,05	4,50
11	5,70	9,01	1,71	12,93	16,32	16,78	6,83	10,00	4,54
12	24,14	3,24	-15,94	37,88	11,82	-81,11	3,91	4,16	-6,42
13	0,64	-3,48	-1,36	12,29	42,32	-57,85	7,88	-1,29	9,76
14	1,63	8,26	-1,29	12,79	13,95	40,39	2,20	9,79	-2,49
15	2,86	4,20	4,61	6,77	11,08	10,12	2,75	5,52	6,45

16	4,35	3,32	5,91	5,62	6,68	6,02	3,06	0,68	3,03
17	2,80	1,12	-5,12	10,69	36,87	-46,08	39,26	26,70	0,10
18	4,03	4,97	3,30	12,59	11,49	49,25	4,92	6,36	4,97
19	4,72	6,63	-0,28	13,46	52,41	20,56	5,54	9,96	9,79
20	-20,35	60,22	-22,70	-17,83	47,14	9,89	0,11	9,32	-4,02
21	1,63	4,35	2,93	24,11	1,21	38,96	3,95	5,42	4,95
22	10,08	3,82	0,36	15,45	12,58	44,62	11,70	46,75	55,99
23	5,56	2,72	9,53	8,66	9,99	2,76	9,04	4,03	5,33
24	3,69	5,79	9,02	17,44	10,54	35,65	8,11	7,94	2,45
25	3,24	2,35	6,56	10,38	14,27	53,43	4,56	2,80	9,71
26	3,84	0,09	1,04	46,51	13,44	12,83	58,44	10,48	38,11
27	3,69	5,96	1,09	42,19	14,79	41,35	5,86	6,71	5,48
28	-68,19	99,01	-27,93	-44,80	11,55	16,68	27,90	1,52	15,83
29	-7,01	5,67	1,57	-3,56	38,11	19,23	7,54	9,64	1,70
30	2,18	1,87	1,45	11,70	35,00	16,98	0,58	8,51	1,66
31	-3,03	7,56	-7,91	53,03	10,20	-25,13	57,06	11,37	-1,98
32	0,64	3,51	4,53	12,93	7,78	47,57	7,72	1,88	6,78
33	4,08	1,16	-5,28	0,99	8,86	1,46	5,81	0,25	6,17
34	2,83	3,33	6,65	31,28	12,09	10,63	0,53	4,21	9,22
35	7,11	9,52	2,22	10,04	16,19	37,62	10,62	10,68	40,59
36	5,77	6,01	-2,04	11,64	7,91	21,03	7,36	0,21	4,80
37	-1,82	2,41	7,95	34,34	22,71	13,48	-6,90	5,20	9,57
38	-6,03	1,76	7,40	10,85	34,40	14,12	1,64	3,60	8,60
39	35,12	54,45	11,17	23,96	54,46	16,51	69,61	14,88	12,76
40	3,40	5,72	-4,50	10,98	49,78	-17,47	4,57	8,54	0,60
41	-0,83	0,59	3,83	2,78	11,12	12,70	8,50	7,78	4,66
42	0,02	6,10	3,63	38,77	10,94	46,74	8,27	8,16	5,54
43	4,22	3,20	2,10	10,55	15,26	44,56	5,73	2,63	7,20
44	0,57	0,35	5,31	11,81	14,52	33,24	7,25	3,98	9,51
45	6,94	-1,81	-1,29	9,55	6,16	-3,82	10,85	-11,95	91,43
46	-4,68	-3,11	-8,75	1,01	4,43	-0,59	-7,76	2,80	1,35
47	62,98	11,14	5,44	11,66	20,74	32,25	80,78	11,68	56,02
48	0,36	-2,69	7,98	27,52	5,96	14,10	6,85	2,38	9,36
49	-3,03	7,56	-7,91	33,03	10,20	-25,13	57,06	11,37	-1,98
50	2,13	4,01	-0,45	6,40	9,22	9,64	5,06	6,06	-5,57

Tabla 6: Resultados modelo Convolutional-Unmix

7.2.6 Comparación de los modelos

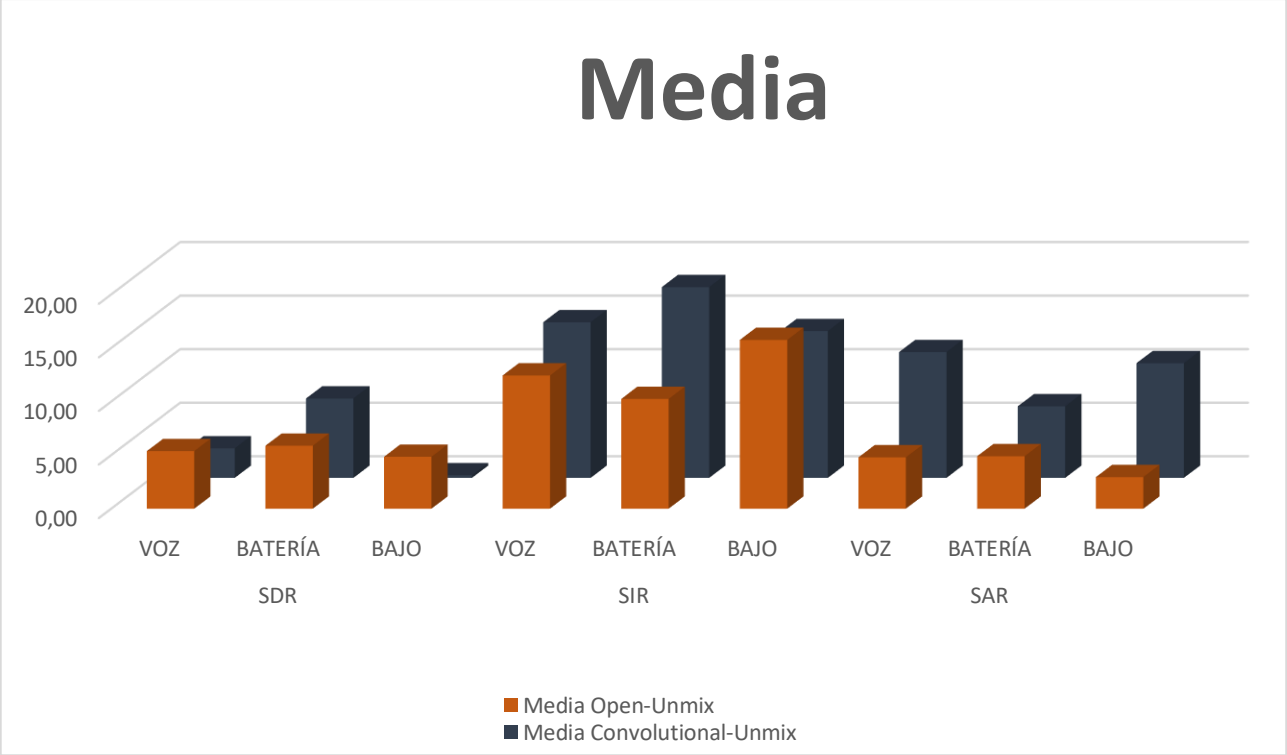


Figura 56: Media de los valores de ambos modelos

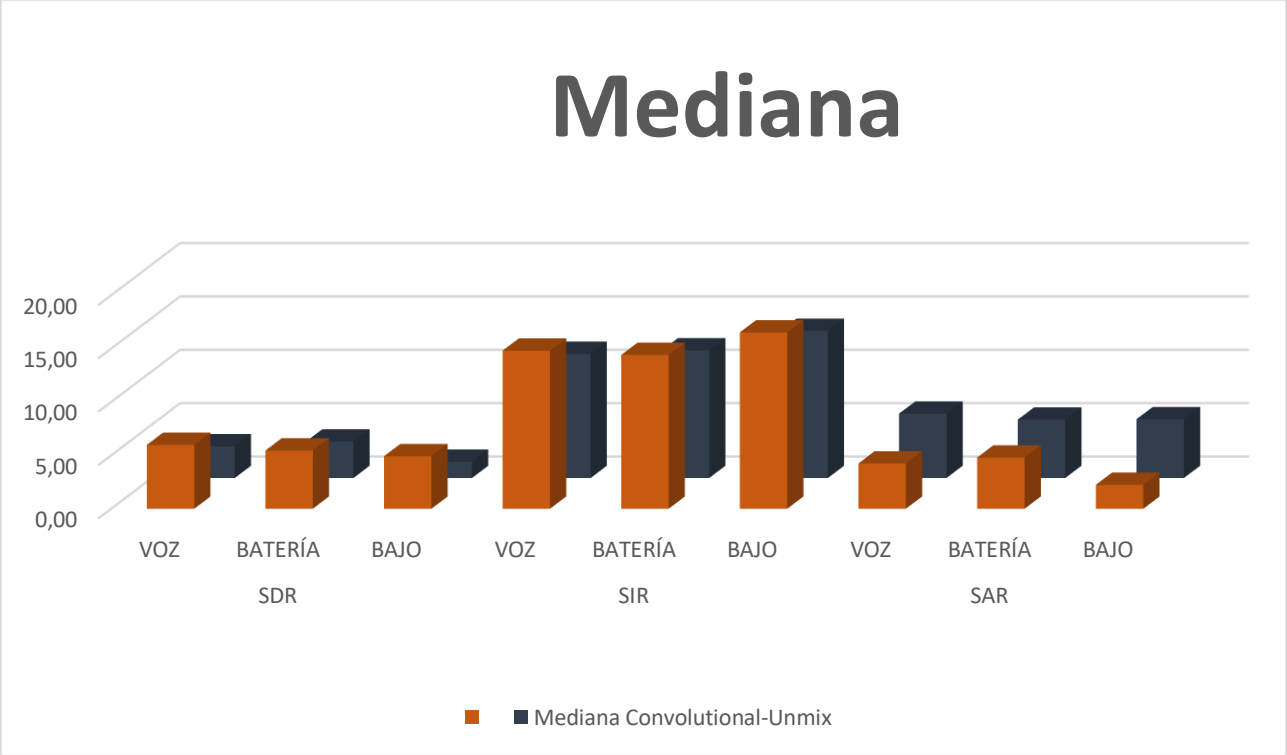


Figura 57: Mediana de los valores de ambos modelos

7.2.7 Discusión de los resultados

Aunque el modelo convolucional propuesto se trata de un modelo más sencillo que el modelo original de Open-Unmix, ha conseguido unos valores comparables como los obtenidos del modelo original.

7.2.7.1 Media

Como se puede apreciar en la Figura 56, los valores medios del modelo original, son más constantes en las 3 fuentes estudiadas, mientras que, en el modelo convolucional implementado, son algo más heterogéneas. Se puede apreciar qué, tanto para el SDR como para el SIR, la fuente que proporciona los resultados más elevados es la batería, mientras que en la SAR es la que peores valores proporciona, siendo estos aún así superiores a los obtenidos del modelo original.

7.2.7.2 Mediana

Como se puede apreciar en la Figura 57, los valores de la mediana de ambas implementaciones son más similares, obteniendo unos valores más elevados para el SDR y el SIR el modelo original respecto del convolucional implementado. Así como se obtienen unos valores más elevados en la SAR en el modelo convolucional.

7.2.7.3 Discusión general

A pesar de las diferencias que encontramos entre la media y la mediana de ambos modelos, se puede considerar que esta diferencia puede ser reducida mejorados mediante un entrenamiento más complejo y prolongado de la red, evitando el posible sobreajuste sufrido durante el entrenamiento causante de los buenos resultados obtenidos en la separación de la batería del resto de la mezcla. Además, se podría realizar el entrenamiento haciendo de una base de datos con un número mayor de composiciones y con más géneros musicales.

8 CONCLUSIONES Y LÍNEAS DE FUTURO

Este último capítulo tiene como finalidad exponer las conclusiones del trabajo que se han sido obtenidas durante la realización del mismo. Así como proponer algunas líneas de investigación futuras.

8.1 Conclusiones

8.1.1 Separación de Fuentes mediante redes neuronales

Durante el desarrollo de este trabajo fin de grado, se ha analizado tanto la arquitectura como el funcionamiento y el rendimiento de dos de los modelos más destacados en la separación automática de fuentes musicales como son DeepConvSep y Open-Unmix.

Después de analizar el modelo DeepConvSep, modelo que utiliza redes convolucionales para la separación de fuentes, se decidió explorar otro modelo diferente que empleara un tipo diferente de redes neuronales con el fin de comparar el rendimiento que pueden ofrecer diferentes arquitecturas. Para ello se optó por compararlo con Open-Unmix debido a su gran facilidad de reproducción, modificación y ampliación, además de ofrecer unos resultados bastante buenos en la separación de las fuentes implementada mediante redes LSTM.

Sin embargo, el estudio, así como la realización de pruebas para DeepConvSep, fue un proceso muy complicado, llegando a no poder realizarse pruebas de ejecución con este modelo debido, principalmente, a problemas con las versiones de las librerías empleadas ya que este es un modelo que fue desarrollado como un proyecto con cierta antigüedad, que fue publicado en 2018 y está implementado en la versión 2.7 de Python, versión que se encuentra desactualizada hoy en día.

Es por esto que se optó por la creación y desarrollo de un nuevo modelo convolucional basado en las redes implementadas en el modelo DeepConvSep e implementado a través del software de código abierto de Open-Unmix. Denominando a este nuevo modelo como **Convolutional-Unmix**.

Este nuevo modelo, como se ha comentado en la sección destinada a los resultados, ha conseguido unos valores bastante competitivos. Este modelo además permite al emplear redes convolucionales proporcionar una nueva perspectiva y camino que poder seguir con el fin de obtener unos resultado más cercanos o incluso mejores que los del modelo original ya que las redes convolucionales son más potentes para este tipo de implementaciones además de tener menor latencia, para posibles implementaciones en tiempo real de esta tecnología.

8.1.2 Remixing espacial en AR

Durante el desarrollo de la implementación en AR se ha analizado, probado y empleado todas aquellas configuraciones y funciones que han permitido proporcionar la mejor experiencia de usuario y la mayor cantidad de funcionalidades con el fin de proporcionar una versión final y completa de la aplicación.

Sin embargo, la versión de la aplicación proporcionada no es una versión que podría considerarse definitiva. Esto es debido a que la versión proporcionada cuenta con varias situaciones en los que no funciona correctamente o genera situaciones irreales como detectar los marcadores sin que estos estén en cámara o desplazar el GameObject sin que el marcador asociado al mismo esté siendo desplazado.

Estas situaciones fueron detectadas durante la fase de pruebas. En la que se han obtenido, como se ha mencionado anteriormente situaciones no previstas además de ciertas ejecuciones en las que la aplicación no ha sido capaz de detectar los marcadores hasta que no se ha reiniciado la escena principal, es decir, se ha vuelto a la escena de inicio e iniciado de nuevo la escena principal mediante el botón de inicio, momento tras el cual la aplicación ha funcionado correctamente y se han detectado los marcadores correctamente.

Es cierto que este tipo de aplicaciones son muy atractivas visualmente para los usuarios menos acostumbrados a las interacciones virtuales ya que proporcionan una interacción real con los elementos virtuales, sin embargo, el correcto funcionamiento de este tipo de aplicaciones, al depender de factores externos a su desarrollo como pueden ser la cámara del dispositivo empleado, la potencia del procesador del mismo o la iluminación del lugar en el que está ejecutando entre otros hacen que no siempre se obtengan los resultados de funcionamiento deseados.

Enlace al APK de la aplicación

8.2 Líneas de futuro

8.2.1 Separación de Fuentes mediante redes neuronales

Para poder obtener resultados en la separación de fuentes de mayor calidad mediante el uso de algoritmos de Machine Learning, es necesario disponer de una cantidad de datos mayor.

Es por esto que, aunque no sea una continuación directa de este trabajo, otros trabajos futuros podrían centrarse en la creación de nuevas pistas para entrenar los modelos. Estas nuevas pistas permitirían ampliar la cantidad de ficheros con las que contamos actualmente en bases como MUSDB18 o incluso generar bases independientes nuevas, las cuales pudiesen estar centradas en otros estilos musicales diferentes lo que

permitiría a los modelos especializarse en estos géneros o incluso conseguir modelos mucho más complejos y potentes.

8.2.2 Remixing espacial en AR

El desarrollo para este TFG se ha limitado al uso de una única mezcla de audios en un entorno sin modificaciones ni aplicación de efectos de sonido en las diferentes fuentes. Es por esto que una línea a futuro de esta aplicación podría realizarse implementado la opción de utilizar varios audios diferentes precargados en la aplicación de manera que fuese el usuario el encargado de seleccionar cuál de las mezclas disponibles será usada para la ejecución.

Además de esta mejora, podría desarrollarse un sistema de efectos mediante marcadores, de manera que al presentar un marcador específico se apliquen efectos a las fuentes activas, llegando incluso a producir una mesa de mezclas en RA que permita aplicar diferentes efectos a las fuentes separadas, así como controlar su volumen y panning.

9 BIBLIOGRAFÍA

[1] Francesco. (2019). GitHub. Obtenido de https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md

[2] Ali, I. (s.f.). Research Gate. https://www.researchgate.net/figure/Pooling-layer-operation-oproaches-1-Pooling-layers-For-the-function-of-decreasing-the_fig4_340812216/actions#reference

[3] Ali, M. (2024). Introducción a las funciones de activación en las redes neuronales. Obtenido de https://www.datacamp.com/es/tutorial/introduction-to-activation-functions-in-neural-networks

[4] Albertcastan. (s.f.). Wikipedia. https://es.wikipedia.org/wiki/Espectrograma#/media/Archivo:Especrtograma_harm%C3%B2nic.png

[5] Bäckström, T. (2019). Research Gate. https://www.researchgate.net/figure/Illustration-of-input-and-output-windowing-with-overlap-add-synthesis-in-a-speech_fig1_332791055

[6] Ethan Manilow, P. S. (s.f.). Open-Source Tools & Data for Music Source Separation.

[7] Daniel Stoller, Sebastian Ewert, and Simon Dixon. Wave-u-net: A multi-scale neural network for end-to-end audio source separation. Technical Report 1806.03185, arXiv, 2018.

[8] Stoller, D., Ewert, S., & Dixon, S. (2018, June 8). Wave-U-Net: a Multi-Scale neural network for End-to-End audio source separation. arXiv.org. https://arxiv.org/abs/1806.03185

[9] T. Takatani, T. Nishikawa, H. Saruwatari, and K. Shikano, "SIMO-model-based independent component analysis for high-fidelity blind separation of acoustic signals," Proc. International Symposium on ICA and BSS (ICA2003), pp.993-998, Apr. 2003.

[10] Oja, Erkki & Plumbley, Mark. (2003). Blind Separation of Positive Sources using Non-Negative PCA.

[11] Unity. (2024). Unity User Manual https://docs.unity3d.com/Manual/index.html

[12] Andreas Jakl (2018). Basics of AR: Anchors, Keypoints & Feature Detection. Obtenido de https://www.andreasjakl.com/basics-of-ar-anchors-keypoints-feature-detection/

[13] Google. (2024). “La herramienta arcoreimg” https://developers.google.com/ar/develop/augmented-images/arcoreimg?hl=es-419#windows

[14] Google-AR (2018) ARCore SDK for Android Github https://github.com/google-ar/arcore-android-sdk/tree/main/tools/arcoreimg

[15] Rafii, Zafar and Liutkus, Antoine and Fabian-Robert Stöter and Mimilakis, Stylianos Ioannis and Bittner, Rachel (2017). The MUSDB18 corpus for music separation. Obtenido de https://sigsep.github.io/datasets/musdb.html. https://doi.org/10.5281/zenodo.1117372

[16] Chandna, P (2017). P. Chandna, M. Miron, J. Janer, and E. Gomez, “Monoaural audio source separation using deep convolutional neural networks” International Conference on Latent Variable Analysis and Signal Separation, 2017.

[17] Open-Unmix: Stöter et al., (2019). Open-Unmix- A Reference Implementation for Music Source Separation. Journal of Open-Source Software, 4(41), 1667. https://doi.org/10.21105/joss.01667)

[18] E. Vincent, R. Gribonval, and C. Févotte, "Performance measurement in blind audio source separation" (https://hal.inria.fr/inria-00544230/document), IEEE Transactions on Audio, Speech, and Language Processing, 14(4), pp 1462-1469, 2006.